Efficient Algorithms for Processing Preference Queries

Marcos Roberto Ribeiro
Federal University of Uberlândia and Federal Institute of Minas Gerais
Uberlândia, Brazil and Bambuí, Brazil
marcos.ribeiro@ifmg.edu.br

Fabíola Souza F. Pereira
Federal University of Uberlândia
Uberlândia, Brazil
fabfernandes@comp.ufu.br

Vinícius Vitor S. Dias
Federal University of Minas Gerais
Belo Horizonte, Brazil
viniciusvdias@dcc.ufmg.br

ABSTRACT

The widespread of applications like e-commerce and recommendation systems requires the use of efficient customization techniques and user preferences handling. In database community a lot of research on this topic has been focused on extending standard SQL with preference facilities in order to provide personalized query answering. More specifically, user preferences are an essential ingredient of personalized database applications. In this paper, we consider the *conditional preference queries* (cp-queries) of CPrefSQL query language where user preferences are specified by a set of conditional rules. We propose new algorithms based on a technique we called *preference partition* that outperform the state-of-the-art algorithms for CPrefSQL preference operators significantly decreasing the number of scans over database.

CCS Concepts

•**Information systems** → **Query languages;** *Structured Query Language;*

Keywords

Query Language, Preferences, SQL Extension, Relational Algebra

1. INTRODUCTION

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 research work has been dedicated to this topic in recent years [13, 8, 14, 9]. The main interest is essentially focused on enhancing standard query languages with new operators enabling the query engine to return the most preferred tuples according to a given preference ordering.

An important work in this scenario is the CPrefSQL language, an extension of SQL able to express *conditional preference queries* (cp-queries) where user's preferences are taken into account in the query answering process [10, 15]. A cp-query incorporates the usual *hard constraints* (WHERE clause) as well as *soft constraints* specified by conditional preference rules (*cp-rules*). The cp-queries allow several types of applications return more customized answers to users. The Example 1 illustrates a typical scenario of preference application.

Example 1 (Motivating Example). *Let us suppose we are given a database relation Travels storing information about travel packages with attributes D (Destination), P (Price), Du (Duration) and I (Itinerary). The following statements express user preferences about trips: (1) between two travel options with the same price, I prefer cruise than beach itineraries; (2) between two travel options with the same duration, I prefer beach than urban itineraries; (3) for cruise itineraries, I prefer options costing less than $2500. These preference statements are declared through the* `CREATE PREFERENCES` *instruction below:*

```
CREATE PREFERENCES MyPrefs FROM Travels AS
(I='cruise') > (I='beach') [D, Du] AND
(I='beach') > (I='urban') [P, D] AND
IF (I='cruise') THEN
    (P < 2500) > (P >= 2500) [D,Du];
```

Next, the user can ask for the three travel options that are not ecological and most fulfill his preferences MyPrefs.

```
SELECT * FROM Travels
WHERE I <> 'ecological'
ACCORDING TO PREFERENCES 3, MyPrefs;
```

In this query, the hard constraint is `I <> 'ecological'` *and the soft constraints are given by the rules MyPrefs. The list of attributes within square brackets in the rules are used to compute preferences under the ceteris paribus semantics. According to this semantics the attributes inside square brackets are indifferent, that is, when two tuples are compared the values for these attributes are not important. This matter will be detailed in Section 2. So, attributes neither inside brackets nor in the right side of the rules (called the ceteris paribus attributes) must have identical values. For instance, the ceteris paribus attribute of first rule is P.*

The CPrefSQL language extends SQL with two algebraic operators ***Select-Best*** and ***SelectK-Best***. The former cal-

SAC 2016,April 04-08, 2016, Pisa, Italy

 ISBN 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851659

culates the most preferred tuples with a semantics based on the BMO model (best-matches only) of [16]: the result is the set of tuples which are not dominated by others. The latter adjusts returned tuples by ***Select-Best*** in order to make up the k required tuples according to the preference hierarchy established by the preference rules. If $k \leq N$ where N is the total amount of most preferred tuples returned by ***Select-Best***, then we are done. If $k > N$, then the hierarchy of the preference order is taken into account to return the remaining amount of tuples required.

A baseline approach to implement the cp-queries is simply by translating them into a SQL query. It is well known that any operator whose semantics follows the BMO model does not enhance the expressive power of SQL, since it can be expressed in relational algebra [8], by means of a rewriting procedure responsible to translate a preference query into a standard SQL-compliant query [13]. However, an explicit definition of preference operators enables separating preference issues from the other aspects of the query, allowing to design specific and efficient algorithms to implement these operators.

As we are dealing with databases, an algorithm should always optimize how it scans the input avoiding I/O operations. So, the complexity of algorithms for processing cp-queries is mainly related with the number of tuples in the input relation. This is the key point that must be balanced to design efficient algorithms.

The state-of-the-art algorithms follow the lines of a basic BNL (block nested loop) technique. In [10], the BNL* algorithm was proposed for evaluating the **Select-Best** operator. It uses the technique introduced in [1] for transitive closure evaluation. Then, the work of [15] developed the algorithms BNL** and R-BNL**. They are coupled with a Datalog Program [5] responsible for deciding between two tuples which one is preferred according to a preference order inferred by transitivity. While BNL** implements the ***Select-Best operator***, R-BNL** (Ranked-BNL**) returns the top-k preferred tuples, following the ***SelectK-Best*** operator.

In this paper, we go a step beyond, by proposing more efficient algorithms to solve cp-queries. Our algorithms use a technique we called *preference partition* where the input is partitioned according to the values of *ceteris paribus* attributes. While BNL algorithms compare one tuple with all the others, here we split the tuples in partitions, so that only tuples in the same partition are compared. Moreover, our partition algorithms take into account a knowledge base that already contains the transitive closure, avoiding recursive calls present in the BNL** algorithms. The Figure 1 shows CPrefSQL algorithms evolution.

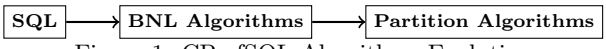


Figure 1: CPrefSQL Algorithms Evolution

Main contributions. The main contributions of this paper are summarized as follows: (1) development of the new and efficient *preference partition* technique to process cp-queries; (2) design of the algorithms Partition and R-Partition by using this new technique; (3) implementation of proposed algorithms following the *on top* approach over the PostgreSQL database system; (4) an exhaustive set of experiments comparing the state-of-the-art algorithms, the proposed algorithms and their corresponding SQL counterpart; (5) generalization of the preference model initially proposed in [15] to support more expressive preferences.

Organization of the paper. In Section 2, we present the theoretical background underlying the cp-queries and related algebraic operators. In Section 3, we firstly discuss about the dominance test problem, the key task in the algorithm, and we present efficient algorithms using the preference partition technique. In Section 4, we discuss the experimental results. In Section 5, we discuss some related work. Finally, in Section 6 we conclude the paper and discuss some future work. For lack of space, the proofs of the results stated in this paper are given in the Appendix A.

2. CP-QUERIES

The user preferences are expressed by if-then rules composed by predicates over attributes. In the present work, we propose an extension of simple equality predicates defined in [15] by supporting generic predicates $A_i \theta a$, where $a \in \mathbf{Dom}(A_i)$ and $\theta \in \{<, \leq, =, \neq, \geq, >\}$. The generic predicates allow an improvement on the expressive power, for example, we can express preferences like: "for cruise itineraries, I prefer those with price *lower than* $2500".

Definition 1 (CP-Rules and CP-Theory). *Let a relational schema $R(A_1, ..., A_l)$. A conditional preference rule, or cp-rule, over R is an expression in the format $\varphi : C_\varphi \rightarrow Q^+_\varphi(A_\varphi) \succ Q^-_\varphi(A_\varphi)[W_\varphi]$, where:*

1) *The attribute $A_\varphi \in R$ is the preference attribute and $W_\varphi \subset R$ is the set of indifferent attributes such that $A_\varphi \notin W_\varphi$;*
2) *The predicates $Q^+_\varphi(A_\varphi)$ and $Q^-_\varphi(A_\varphi)$ represent the preferred and non preferred values for A_φ, respectively, such that $\{a \in \mathbf{Dom}(A_i) \mid a \models Q^+_\varphi(A_\varphi)\} \cap \{a \in \mathbf{Dom}(A_i) \mid a \models Q^+_\varphi(A_\varphi)\} = \emptyset$;*
3) *The preference condition C_φ is a conjunction $Q(A_1) \wedge ... \wedge Q(A_k)$ such that $Q(A_i)$ is a predicate over A_i and $\{A_1, ..., A_k\} \in R$ and $\{A_1, ..., A_k\} \cap (\{A_\varphi\} \cup W_\varphi) = \emptyset$;*

A conditional preference theory on R, or cp-theory, Γ is a finite set of cp-rules on R.

Example 2 (CP-Theory). *The preference statements of Example 1 are represented by cp-theory $\Gamma = \{\varphi_1, \varphi_2, \varphi_3\}$ where:*

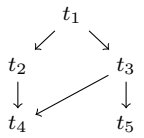
$\varphi_1 : \rightarrow (I = cruise) \succ (I = beach)[D, Du]$;
$\varphi_2 : \rightarrow (I = beach) \succ (I = urban)[P, D]$;
$\varphi_3 : (I = cruise) \rightarrow (P < 2500) \succ (P \geq 2500)[D, Du]$;

The processing of cp-queries have to work only over consistent cp-theories, since irreflexivity of the preference ordering is not a desirable situation in a database context. This notion of consistency has been addressed in [15]. Thus, when a set of cp-rules is declared by `CREATE PREFERENCES` command the consistency test is executed prior to registration of them into the system catalog.

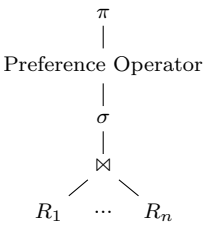
Semantics. Now we come to the task of how to compare two tuples according to a cp-theory. Let $\mathbf{Tup}(R)$ be the set of all possible tuples over a relational schema R. A

	Destination	Price	Duration	Itinerary
t_1	Angra	2000	4	cruise
t_2	Buzios	2000	5	beach
t_3	Salvador	2600	6	cruise
t_4	Belo Horizonte	2700	5	urban
t_5	Rio de Janeiro	2600	7	beach

(a)



(b)



(c)

Figure 2: (a) An instance of relation `Travels`; (b) The better-than graph for relation `Travels`; (c) The execution plan for a CPrefSQL block.

preference order over $\mathbf{Tup}(R)$ is a strict partial order over $\mathbf{Tup}(R)$, that is an irreflexive and transitive binary relation over $\mathbf{Tup}(R)$. In this paper, we use the ceteris paribus semantics ("everything else equal") [4] to compare tuples.

Given two tuples t and t' and a cp-rule φ. The tuple t is preferred to t' according to φ, denoted by $t \succ_\varphi t'$, if: (1) t and t' satisfy the cp-rule condition C_φ, (2) t and t' have identical values for the attributes not appearing in $W_\varphi \cup \{A_\varphi\}$ and (3) t satisfies $Q^+_\varphi(A_\varphi)$ and t' satisfies $Q^-_\varphi(A_\varphi)$. We can also infer a preference order by transitivity as addressed by Theorem 1.

THEOREM 1. *Let a cp-theory Γ and a relational schema R. Let $t, t' \in \mathbf{Tup}(R)$ be two tuples. The tuple t is preferred to t' according to Γ, denoted by $t \succ_\Gamma t'$, if and only if there exists tuples $t_1, ..., t_{m+1} \in \mathbf{Tup}(R)$ and cp-rules $\varphi_1, ..., \varphi_m \in \Gamma$ such that $t_1 \succ_{\varphi_1} ... \succ_{\varphi_m} t_{m+1}$, $t = t_1$ and $t' = t_{m+1}$ [17].*

Remark that, in this paper, we use the terms *dominates* and *is preferred to* as synonymous. The Example 3 illustrates the concept of preference order under ceteris paribus semantics. For more details please see [15].

EXAMPLE 3. *Let us consider the cp-rule φ_2 of Example 2 and the tuples t_2 and t_4 of Figure 2(a). We can affirm that $t_2 \succ_{\varphi_2} t_4$ since: (1) t_2 and t_4 satisfy the rule condition (empty conditions are satisfied by any tuple); (2) both tuples have the same value for ceteris paribus attribute Du; (3) for the preference attribute, t_2 has a preferred value (I = beach) and t_4 has a non preferred value (I = urban). Let us consider the instance of `Travels` described in Figure 2(a) and the cp-theory $\Gamma = \{\varphi_1, \varphi_2, \varphi_3\}$ given in Example 2. The order induced on `Travels` by the cp-theory Γ is illustrated by the better-than graph depicted in Figure 2(b). In this graph, a path from t to t' means that t dominates t' according to Γ.*

Operators. There are two algebraic operators developed to evaluate cp-queries: ***Select-Best*** [10] and ***SelectK-Best*** [15]. Given a cp-theory Γ and a relation instance r, the ***Select-Best*** operator selects the most preferred tuples of r (those which are not dominated by others) according to Γ, while the ***SelectK-Best*** gets the set of *top-k tuples* of r with respect to the preference hierarchy imposed by Γ. More precisely, the operation $\textbf{\textit{SelectK-Best}}_{\Gamma,k}(r)$ returns k tuples of r with the smallest levels (Definition 2).

DEFINITION 2 (LEVEL). *Let a cp-theory Γ over the relational schema R, a database instance r over R and a tuple $t \in r$. The level of t, denoted by $lv(t)$, according to Γ is inductively defined as follows:*

1) If $\nexists t' \in r$ such that $t' \succ_\Gamma t$ then $lv(t) = 0$;

2) Otherwise $lv(t) = max\{lv(t') \mid t' \succ_\Gamma t\} + 1$.

EXAMPLE 4. *Let us consider the preference ordering induced by Γ over the tuples in `Travels` stated on Example 3. Thus, we have: $lv(t_1) = 0$, $lv(t_2) = lv(t_3) = 1$, $lv(t_4) = max\{lv(t_1), lv(t_2), lv(t_3)\} + 1 = 2$, $lv(t_5) = 2$. Thus, the result of query of Example 1, computed by operation $\textbf{\textit{SelectK-Best}}_\Gamma(3, \texttt{Travels})$, is $\{t_1, t_2, t_3\}$.*

The CPrefSQL query language is an extension of the standard SQL with the two new preference operators ***Select-Best*** and ***SelectK-Best***. The simple query block of CPrefSQL is given below.

```
SELECT <attribute-list> FROM <tables>
--hard constraints
WHERE <conditions>
--soft constraints
ACCORDING TO PREFERENCES [k,] <cp-rules>;
```

The parameter `<cp-rules>` on clause `ACCORDING TO PREFERENCES` is a list of cp-rules declared by the `CREATE PREFERENCES` command (as illustrated in Example 1). The canonical execution plan associated to a CPrefSQL block (without aggregate constructors) is shown in Figure 2(c). The symbols σ, π and $\bowtie$ denote the usual relational algebra operators *Selection, Projection* and *Join*, respectively. The parameter k is a non-negative integer and it is optional. If k is not provided, the query is evaluated using the ***Select-Best***, otherwise the evaluation uses the ***SelectK-Best*** operator.

3. ALGORITHMS FOR CP-QUERIES EVALUATION

In [15] the algorithms BNL** and R-BNL** were proposed to evaluate the operators ***Select-Best*** and ***SelectK-Best***, respectively. These BNL algorithms are based on a Block Nested Looping strategy of [3], and they use a Datalog program [5] to perform the *dominance test*: *"given a cp-theory Γ and two tuples t and t', decide if $t \succ_\Gamma t'$"*.

In order to simplify the comparison between tuples, we propose a new dominance test based on a knowledge base where the transitive closure is already computed. Furthermore, we developed significantly more efficient algorithms by using this knowledge base.

3.1 Theoretical Foundations

An alternative to the Datalog dominance test is to use the strategy of building a *knowledge base*. The idea is to scan a cp-theory extracting all possible *comparisons* (including

those obtained by transitive closure) and storing them in a called *knowledge base*. In this way, to perform a dominance test, just a scan in the *knowledge base* is necessary, avoiding Datalog recursive runs. Remark that in this approach an additional preprocessing is done before storing user preferences in system catalog. The construction of a knowledge base starts by computing the set of *essential formulas* (Definition 3).

DEFINITION 3 (ESSENTIAL FORMULA). *Let a cp-theory Γ over a relational schema $R(A_1, ..., A_l)$. Let $Q_\Gamma(A_i)$ the set of predicates for attribute A_i present in Γ. An essential formula is inductively defined as follows:*

1) *A predicate $Q(A_i)$ is an essential formula such that $Q(A_i) \in Q_\Gamma(A_i)$ and $A_i \in R$;*
2) *If f is an essential formula and $A_i \in R$ does not appear in f, then $f \wedge Q(A_i)$ is an essential formula such that $Q(A_i) \in Q_\Gamma(A_i)$.*

The notation F_Γ represents all essential formulas over Γ.

EXAMPLE 5. *Let us consider the cp-theory Γ of Example 2. The set of essential formulas for this cp-theory is*

$$F_\Gamma = \left\{ \begin{array}{l} (I = cruise), (I = beach), (I = urban), (P < 2500), \\ (P \geq 2500), (I = cruise) \wedge (P < 2500), \\ (I = beach) \wedge (P < 2500), (I = urban) \wedge (P < 2500), \\ (I = cruise) \wedge (P \geq 2500), (I = beach) \wedge (P \geq 2500), \\ (I = urban) \wedge (P \geq 2500) \end{array} \right\}$$

Let $\mathbf{Att}(f)$ be the set of attributes appearing in f. Consider the essential formulas $f : g \wedge h \wedge w$ and $f' : g' \wedge h \wedge w'$ such that g, g', h, w and w' are conjunctions of predicates. The formula f is preferred to f' according to a cp-rule φ, denoted by $f \succ_\varphi f'$, if $g = C_\varphi \wedge Q^+_\varphi(A_\varphi)$ and $g' = C_\varphi \wedge Q^-_\varphi(A_\varphi)$ and $\mathbf{Att}(w) \subseteq W_\varphi$ and $\mathbf{Att}(w') \subseteq W_\varphi$. Once built the essential formulas, the next step is to identify all possible comparisons between them according to Γ (Definition 4). Intuitively, a comparison between two formulas means that a formula is, directly or indirectly, preferred to another formula according to a cp-theory.

DEFINITION 4 (COMPARISON). *Let a cp-theory Γ. A comparison over Γ is a statement in the form $b : (f_b^+ \succ f_b^-)[W_b]$) where there exists essential formulas $f_1, ..., f_{m+1} \in F_\Gamma$ and cp-rules $\varphi_1, ..., \varphi_m \in \Gamma$ such that:*

1) *The formula f_b^+ is preferred to f_b^-, $f_b^+ = f_1$ and $f_b^- = f_{m+1}$ and $f_1 \succ_{\varphi_1} ... \succ_{\varphi_m} f_{m+1}$;*
2) *The set $W_b = (W_{\varphi_1} \cup ... \cup W_{\varphi_n}) \cup (\{A_{\varphi_1}\} \cup ... \cup \{A_{\varphi_n}\})$ are the indifferent attributes.*

Let b be a comparison over a cp-theory Γ. Two tuples t, t' can be compared using a comparison b, denoted by $t \succ_b t'$ if $t \models f_b^+$, $t' \models f_b^-$ and $t.A_i = t'.A_i$ for all $A_i \notin W_b$. The Lemma 1 guarantees the existence of a comparison when two tuples are directly comparable by a cp-rule (the proof is given in the Appendix A).

LEMMA 1. *Let Γ be a cp-theory and $\varphi \in \Gamma$ be a cp-rule. If $t \succ_\varphi t'$ then there exists a comparison b such that $t \succ_b t'$.*

We denote by K^*_Γ the set of all comparisons on Γ. Two tuples t, t' can be compared using K^*_Γ, denoted by $t \succ_{K^*_\Gamma} t'$, if there exists $b \in K^*_\Gamma$ such that $t \succ_b t'$. The Theorem 2 ensures that $t \succ_{K^*_\Gamma} t'$ if and only if $t \succ_\Gamma t'$ (the proof is given in the Appendix A).

THEOREM 2. *Let us consider two tuples t, t' and a cp-theory Γ. Then $t \succ_{K^*_\Gamma} t'$ if and only if $t \succ_\Gamma t'$.*

The set of comparisons K^*_Γ may contains unnecessary comparisons due to indifferent attributes. Consider, for instance, the comparisons $b : (I = cruise) \succ (I = beach)[D, I, P]$ and $b' : (I = cruise) \wedge (P \geq 2500) \succ (I = beach) \wedge (2000 \leq P < 2500)[D, I, P]$. Notice that $\succ_{b'} \subset \succ_b$, this happens because b is more generic than b'. We can verify if a comparison $b : (f^+ \succ f^-)[W_b] \in K^*_\Gamma$ is more generic than a comparison $b' : (g^+ \wedge h^+ \succ g^- \wedge h^-)[W_{b'}] \in K^*_\Gamma$ if one of the following conditions are satisfied:

1) $f^+ = g^+$, $f^- = g^-$, $h^+ = h^-$ and $W_{b'} \subseteq W_b$;
2) $f^+ = g^+$, $f^- = g^-$, $(\mathbf{Att}(h^+) \cup W_{b'}) \subseteq W_b$ and $(\mathbf{Att}(h^-) \cup W_{b'}) \subseteq W_b$.

Taking advantage of this property we build a knowledge base considering only the *essential* comparisons (Definition 5).

DEFINITION 5 (KNOWLEDGE BASE). *Let K^*_Γ be the set of all comparisons over a cp-theory Γ. A comparison $b \in K^*_\Gamma$ is essential if there no exists another comparison $b' \in K^*_\Gamma$ such that b' is more generic than b. The knowledge base K_Γ is the set of all essential comparisons in K^*_Γ.*

EXAMPLE 6. *Let us consider the cp-theory Γ and its essential formulas of Example 5. The knowledge base for this cp-theory is $K_\Gamma = \{b_1, ..., b_5\}$ where:*

b_1 : $(I = cruise) \succ (I = urban)[D, Du, I, P]$;
b_2 : $(I = cruise) \wedge (P < 2500) \succ I = beach) \wedge (P \geq 2500)[D, Du, I, P]$;
b_3 : $(I = beach) \succ (I = urban)[D, I, P]$;
b_4 : $(I = cruise) \succ (I = beach)[D, Du, I]$;
b_5 : $(I = cruise) \wedge (P < 2500) \succ (I = cruise) \wedge (P \geq 2500)[D, Du, P]$.

The Theorem 3 ensures that if $t \succ_{K^*_\Gamma} t'$ then $t \succ_{K_\Gamma} t'$ (the proof is given in the Appendix A). Thus, once the knowledge base is built, we can perform the dominance test "$t \succ_\Gamma t'$" by looking for an essential comparison b in K_Γ such that $t \succ_b t'$.

THEOREM 3. *Let a cp-theory Γ and two tuples t, t'. If $t \succ_{K^*_\Gamma} t'$ then $t \succ_{K_\Gamma} t'$.*

BNL Algorithms with Knowledge Base (BNL-KB).** In order to compare the two types of dominance test, we modify the BNL** algorithms proposed in [15] by replacing the Datalog dominance test by the knowledge base dominance test. The idea is to avoid recursive runs from Datalog for each dominance test, by providing to the algorithm a precomputed knowledge base already containing the transitive closure. In general, this modification optimizes the BNL** algorithms as will be discussed in the following complexity analysis. The Section 4 presents the experimental results of this comparison.

3.2 Partition Algorithms

The main result of this paper is the development of *partition algorithms* which use a new divide-and-conquer technique

called *preference partition*. The intuition of the preference partition technique is: for each comparison b, partitions are built according to attributes not present in W_b; these are the *ceteris paribus* attributes and must have the same values for all tuples. Then, inside a partition where all tuples have the same values for *ceteris paribus* attributes, we can identify the dominant, dominated and incomparable tuples according to essential formulas of b.

The Algorithm 1, **Partition**, evaluates the **Select-Best** operator by using the partition technique. This algorithm uses the procedure **BestPartition** (Algorithm 2) in an incremental way to select the dominant tuples. In fact, for each iteration, the procedure **BestPartition** returns the set of dominant tuples S^+ and the set of dominated tuples S^-. As the set S^- is not needed, it is discarded. At the end, the set S^+ has the dominant tuples of r according to Γ.

Algorithm 1: **Partition**(r, Γ)

1: $S^+ \leftarrow r$
2: **for all** $b \in K_\Gamma$ **do**
3: $\quad S^+, S^- \leftarrow$ **BestPartition**(S^+, b)
4: **return** S^+

The procedure **BestPartition** splits the set of tuples S into two sets: S^+ (dominant tuples) and S^- (dominated tuples). The first step is to build a hash table of partitions $P^\# = (t \in S \mapsto \{t/W_b\})$ over S by using the values of tuple attributes not present in W_b (*ceteris paribus* attributes). Next, for each partition set P in $P^\#$, the algorithm separates the tuples into three subsets: preferred tuples P^+, non preferred tuples P^- and incomparable tuples P^*. The tuples in P^* are always dominant because they cannot be compared by b. The preferred tuples satisfy the formula f_b^+ while non preferred tuples satisfy the formula f_b^-. As all tuples in the same partition have identical values for *ceteris paribus* attributes, then all tuples in P^+ dominate the tuples in P^-. After the subsets P^+, P^- and P^* are computed, the procedure check if P^+ is empty. If yes, all tuples in P are dominant because any tuple is dominated. If no, P^- has the dominated tuples and the dominant tuples are those in P^+ and in P^*.

Algorithm 2: **BestPartition**(S, b)

1: $P^\# \leftarrow HashTable(S, b)$
2: $S^+ \leftarrow \{\}; S^- \leftarrow \{\}$
3: **for all** $P \in P^\#$ **do**
4: $\quad P^+ \leftarrow \{t \in P \mid t \models f_b^+\}$
5: $\quad P^- \leftarrow \{t \in P \mid t \models f_b^-\}$
6: $\quad P^* \leftarrow \{t \in P \mid t \not\models f_b^+$ **and** $t \not\models f_b^-\}$
7: $\quad$ **if** $P^+ = \{\}$ **then**
8: $\quad\quad S^+ \leftarrow S^+ \cup P$
9: $\quad$ **else**
10: $\quad\quad S^+ \leftarrow S^+ \cup P^+ \cup P^*$
11: $\quad\quad S^- \leftarrow S^- \cup P^-$
12: **return** S^+, S^-

In the Figure 3(a) we can see the execution of **Partition** considering the relation `Travels` of Figure 2(a) and the knowledge base of Example 6. For the first comparison b_1, there are no *ceteris paribus* attributes and the algorithm builds only one partition containing all tuples. The tuple t_4 in bold is dropped because it is dominated by t_1 and t_3 according to b_1. Considering b_2, again there is just one partition and the tuple t_5 in bold is dropped. When b_3 is evaluated, the partitions are built according to attribute Du not present in W_{b_3}. There are three partitions: $\{t_1\}$ for $(Du = 4)$, $\{t_2\}$ for $(Du = 5)$ and $\{t_3\}$ for $(Du = 6)$, but any tuple is dominated. For comparison b_4, there are two partitions and t_2 is dropped. At the end, when the algorithm considers the comparison b_5, there is only one partition, and t_3 is dropped. Thus, the final result of the algorithm is $\{t_1\}$.

	Partitions
b_1	$\{t_1, t_2, t_3, \mathbf{t_4}, t_5\}$ P^+ P^* P^+ P^- P^*
b_2	$\{t_1, t_2, t_3, \mathbf{t_5}\}$ P^+ P^* P^* P^-
b_3	$\{t_1\}, \{t_2\}, \{t_3\}$ P^* P^+ P^*
b_4	$\{t_1, \mathbf{t_2}\}, \{t_3\}$ P^+ P^- P^+
b_5	$\{t_1, \mathbf{t_3}\}$ P^+ P^-

(a)

	Partitions
b_1	$\{t_2, t_3, \mathbf{t_4}, t_5\}$ P^* P^+ P^- P^*
b_2	$\{t_2, t_3, t_5\}$ P^* P^* P^-
b_3	$\{t_2\}, \{t_3, t_5\}$ P^+ P^* P^+
b_4	$\{t_2\}, \{t_3, \mathbf{t_5}\}$ P^- P^+ P^-
b_5	$\{t_2\}, \{t_3\}$ P^* P^-

(b)

Figure 3: (a) Partitions for all tuples; (b) Partitions for tuples $\{t_2, ..., t_5\}$.

The algorithm **R-Partition** (Algorithm 3) uses the preference partition technique to implement the operator **SelectK-Best**. This algorithm uses a list L to store the top-k tuples sorted by level. The algorithm stops when L has k tuples or all input tuples are in L. At the beginning, the set S^+ has all input tuples. Then, for each iteration of outer loop, the dominant tuples are kept on S^+ and dominated ones are moved to S^-. At the end of iteration, the algorithm appends the tuples of S^+ into L and checks if L has k tuples. In negative case, the dominated tuples are moved to S^+ and the algorithm proceeds to the next iteration. This process guarantees that if t appears first than t' in L then $lv(t) \leq lv(t')$. The Example 7 presents an execution of algorithm **R-Partition**.

Algorithm 3: **R-Partition**(r, Γ, k)

1: $S^+ \leftarrow r; L \leftarrow List()$
2: **while** $|L| < k$ **and** $S^+ \neq \{\}$ **do**
3: $\quad S^- \leftarrow \{\}$
4: $\quad$ **for all** $b \in K_\Gamma$ **do**
5: $\quad\quad S^+, S \leftarrow$ **BestPartition**(S^+, b)
6: $\quad\quad S^- \leftarrow S^- \cup S$
7: $\quad L.Append(S^+)$
8: $\quad S^+ \leftarrow S^-$
9: **return** first k tuples of L

EXAMPLE 7. *Consider the operation* ***SelectK-Best***$_\Gamma(3, \texttt{Travels})$ *of Example 4. This operation is evaluated by* ***R-Partition*** *algorithm as follows: (1) The first iteration of outer loop produces the same partitions of Figure 3(a), at this point* $L = \{t_1\}$ *and* $S^+ = \{t_2, ..., t_5\}$. *Another iteration is required because* $|L| < 3$ *and there are tuples in* S^+ *to be processed; (2) The Figure 3(b) shows the partitions of the second iteration. At this moment, the list* $L = \{t_1, t_2, t_3\}$, *then the algorithm stops and return the tuples in* L.

3.3 Complexity Analysis

The complexity analysis of algorithms takes into account the number of cp-rules in Γ (m), the number of attributes of input relation (l) and the number of input tuples (n). First, we analyze the building cost and size of the knowledge base (K_Γ) used in our proposed algorithms, then we compare the complexity BNL algorithms and partition algorithms.

Knowledge Base Complexity. First, we scan the cp-rules and take the set of predicates $Q_\Gamma(A_i)$ for each attribute A_i. Hence, these predicates are combined to form the essential formulas. In the worst case, all relation attributes are present in Γ, $|\mathbf{Att}(\Gamma)| = l$, and every attribute has a different predicate for each cp-rule, $|Q_\Gamma(A_i)| = m$. The computation of all essential formulas must to combine the m predicates of l attributes. Thus, in the worst case, the cost for obtaining the essential formulas is $O(m^l)$ and $|F_\Gamma| = O(m^l)$.

After the computation of essential formulas, the knowledge base is built by two tasks. First, we try to compare directly every pair of essential formulas using the cp-rules. In the worst case, all pairs of essential formulas are directly comparable and there are m^{2l} direct comparisons. Next, we can use the Floyd-Warsall algorithm [11] in order to calculate the transitive closure of direct comparisons in cubic time. Thus, the final building cost of K^*_Γ is $O(m^l + m^{2l} + (m^{2l})^3) = O(m^{6l})$. In the worst case, every comparison is essential, then $|K_\Gamma| = O(m^{4l})$.

Algorithms Complexity. The complexity of algorithm BNL** is $O(n^2 \times m^m)$. The factor $O(m^m)$ represents the cost of the Datalog dominance test and the term $O(n^2)$ is the cost of nested loops over the input tuples. In the case of algorithm BNL-KB, the cost of knowledge base dominance test is a scan in K_Γ. Furthermore, the knowledge base must be built before the execution of the algorithm. Thus, the complexity of BNL-KB is $O(m^{6l} + n^2 \times m^{4l})$.

The complexity of algorithm **Partition** is also related to the building cost and size of the knowledge base. The procedure **BestPartition**, called by every comparison in K_Γ, scans the input tuples two times and uses a hash table to store the partitions. For each tuple, the procedure must read its attributes to select the correct partition, so its cost is $O(ln)$. Thus, in the worst case, the complexity of **Partition** is $O(m^{6l} + ln \times m^{4l})$.

The complexity of algorithms R-BNL**, R-BNL-KB and R-Partition is the same complexity of their counterpart algorithms BNL**, BNL-KB and Partition, respectively, multiplied by the *depth level* of the cp-theory. The depth level involves the interaction between the preference rules through transitive closure. For instance, let $\Gamma = \{\varphi_1, \varphi_2\}$, where $\varphi_1 : C_{\varphi_1} \rightarrow (B = b_1) \succ (B = b_2)[W_{\varphi_1}]$ and $\varphi_2 : C_{\varphi_2} \rightarrow (B = b_2) \succ (B = b_3)[W_{\varphi_2}]$. In this case, the depth level of Γ is 2, since the maximum length of a path in its induced BTG is 2.

All algorithms are affected by the number of input tuples (n) and by the number of cp-rules (m). In addition, the use of knowledge base is impacted by the number of attributes (l). In practice, the number of input tuples causes the biggest impact on the algorithms complexity because n is millions of times greater than l and m. Thus, our main contribution is the reduction of the factor $O(n^2)$ in BNL algorithms to $O(n)$ in the partitions algorithms.

4. EXPERIMENTAL RESULTS

In our experiments we compare the performance and scalability of: (1) original BNL algorithms (BNL** and R-BNL**), (2) BNL algorithms with knowledge base dominance test (BNL-KB and R-BNL-KB), (3) partitions algorithms (**Partition** and **R-Partition**) and (4) correspondent cp-queries translated to recursive *SQL* – our baseline.

Datasets. The experiments have been performed using the TPC-H benchmark[1] that is a specific benchmark for *ad-hoc* querying workloads. The datasets generated from TPC-H in our experiments have sizes of 16MB, 32MB, 64MB, 128MB, 256MB, 512MB and 1024MB.

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; (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 Q_3, Q_5, Q_{10} and Q_{18} we have considered in the experiments enabled an expressive variation on the number of tuples submitted to the preference operators. We denote by q this number of tuples corresponding to the query Q executed over the default dataset of 32MB. More precisely: $q_3 = 698$, $q_5 = 7596$, $q_{10} = 18644$ and $q_{18} = 34263$.

Preferences. The preference rules have been built by taking into account two main features: the number of rules and the depth level. The number of rules used in the experiments varies from 6 to 40 rules. The depth level of the cp-theories we have used in our experiments varies from 1 to 6. The conditions of cp-rules considered in the experiments are a boolean composition of two atomic formulas $(C_\varphi : Q_1(A_1) \wedge Q_2(A_2))$. The choice for varying the features *number of rules* and *depth level* is related with the performance impact that they cause in our algorithms. While the latter is tied with the number of recursive calls, the former is responsible for the number of dominance test calls. Summarizing, all parameters and their variations are shown in Figure 4.

Parameter	Default	Variation
Dataset (MB)	32	16, 32, 64, 128, 256, 512 and 1024
Depth level	2	1, 2, 3, 4, 5 and 6
Number of rules	6	6, 10, 20, 30 and 40
Query	Q_5	Q_3, Q_5, Q_{10} and Q_{18}
k	-1	-1, 10, 100, 1000 and 5000

Figure 4: Parameters used for experiments

Performance analysis. In Figure 5(a), we present the results of the experiments varying the number of rules to compare the performance of algorithms. Note that the BNL** and BNL-KB performances decrease drastically as the number of rules increases when compared with the Partition implementation. As the number of rules increases more tuples can be compared and the dominance test is slower. This is not the case for the Partition algorithm that reads the knowledge base less times because the number of scans on database is reduced by the preference partition technique.

In Figure 5(b), we illustrate the behavior of query executions when varying the depth level of the cp-theories. This feature has more impact on the performance of the BNL-KB algorithm. As the depth level increases, the trend is that the the knowledge base also increases because there are

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

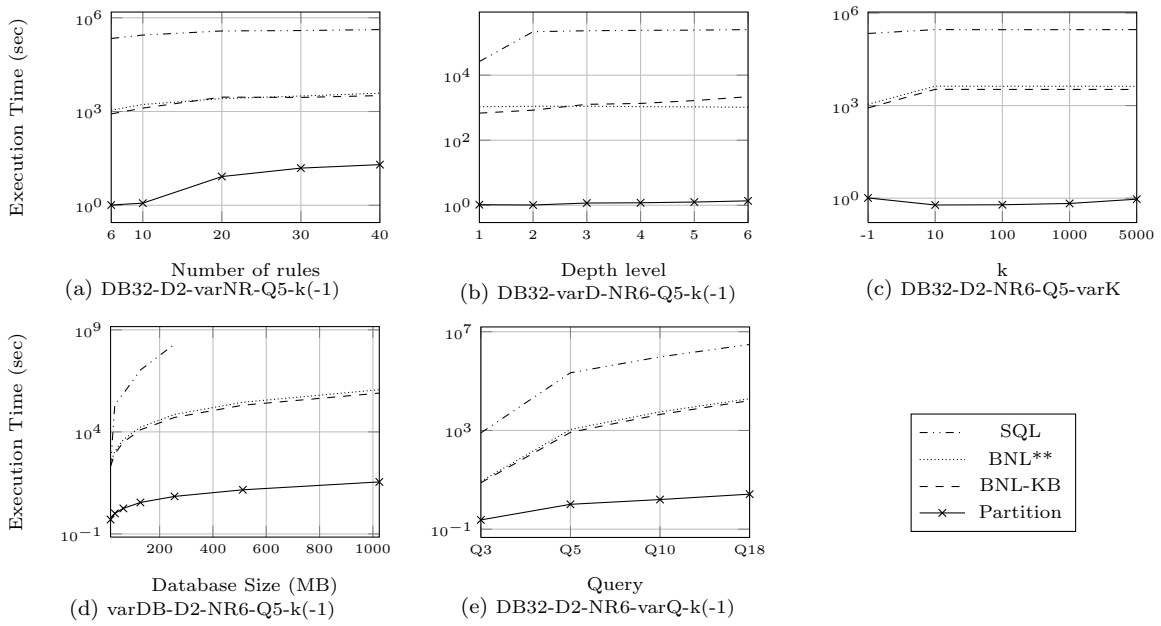


Figure 5: Performance and scalability results

more comparisons by transitivity.

The preference features considered in Figures 5(a) and 5(b) show that the performances of CPrefSQL algorithms depend on the selectivity factor of the rules, that is, the number of tuples returned. The number of rules and the depth level of the cp-theory have a considerable negative impact on BNL** and BNL-KB performances. On the other hand, the performance of Partition algorithm has not been strongly affected.

We have also tested the performance of cp-queries by varying the parameter k (the desired amount of preferred tuples) as shown in Figure 5(c). The performance of both BNL algorithms and Partition is practically unchanged for different values of $k > 0$. As expected, the performance is better for $k = -1$ (in this case the algorithms return the most preferred tuples), since for $k > 0$ they need extra time to process different levels.

Scalability results. Figure 5(d) shows how algorithms scale when the size of the database increases from 16MB to 1024MB. This experiment evidences the difference in the complexity between BNL algorithm and Partition algorithm. The bigger databases have more tuples and this causes more impact in the BNL algorithm than in Partition algorithm.

In Figure 5(e) we can see the behavior of algorithms when varying the reduction factor of the `WHERE` clause, i.e., the number of tuples directly submitted to the preference operator. Again, the performance of Partition algorithm is far better than BNL algorithms because the number of tuples to be processed.

5. RELATED WORK

The research literature on preference models and extension of SQL with preference is extensive. The approach of CP-Nets and TCP-Nets uses a graphical model which captures users qualitative *conditional preference* over tuples, under a *ceteris paribus* semantics [4, 2]. In this paper, we follow the logical framework for preference specification introduced in [17]. This framework is more expressive and generalizes the CP-Net and TCP-Net approaches.

Regarding extensions of SQL, the research of [13] introduces the query language Preference SQL which extends SQL with some built-in base preference constructors and accumulation constructors. The optimizer uses a rewriting procedure which transforms preference queries into standard SQL queries.

The topic of preference query evaluation has been extensively studied in the literature. In [3], the basic BNL (block-nested loop) algorithm has been introduced for evaluating *skyline queries*. In [7], the algorithm SFS (sort-filter-skyline) has been proposed which outperforms the BNL algorithm for skyline queries evaluation. The algorithm BNL$^+$ for *pareto query* evaluation has been introduced in [6]. In [16], 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 (a strict partial order with negative transitivity). In [12] the problem of multiple preference queries has been addressed, where different users may compete for the same object simultaneously. The authors proposed a Skyline-based algorithm to solve it.

6. CONCLUSION

In this paper we developed new algorithms using a preference partition technique for evaluating cp-queries in linear time with respect to number of tuples. The partition technique uses the concept of knowledge base where the direct and indirect comparisons of tuples are already computed. The new algorithms have been implemented following the *on top* approach over PostgreSQL query processor. Our experiments showed a considerable better performance and scalability of new algorithms with respect to state-of-art algorithms. We have also enhanced the syntax and semantics of the preliminary CPrefSQL version [15] by considering more

general preference rules.

We intend to follow three main directions of research in the future, namely: (1) *Optimization*: design of index-based algorithms for evaluating the cp-queries. (2) *Experiments on High Dimensionality*: analyze the behavior of algorithms in scenarios with high dimensionality. (3) *Continuous Queries*: development of incremental algorithms to update the most preferred tuples in applications dealing with data that evolve rapidly in time such as sensor networks and financial data analysis. In such applications, the queries are continuously submitted to the system and must be efficiently evaluated in real time. We are presently working on the latter direction of research.

Acknowledgments. The authors thanks to Research Agencies CNPq, CAPES and FAPEMIG for supporting this work. The authors are grateful to Professor Sandra de Amo for excellent advices.

7. REFERENCES

[1] R. Agrawal, A. Borgida, and H. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 253–262, 1989.

[2] T. Allen. Cp-nets: From theory to practice. In *Algorithmic Decision Theory*, volume 9346 of *Lecture Notes in Computer Science*, pages 555–560. Springer, 2015.

[3] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 421–430, 2001.

[4] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[5] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *Proc. of IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 1(1):146–166, 1989.

[6] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Efficient processing of skyline queries with partially-ordered domains. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 190–191, 2005.

[7] J. Chomicki. Preference formulas in relational queries. *Proc. of ACM Trans. on Database Systems (TODS)*, 28(4):427–466, December 2003.

[8] J. Chomicki. Logical foundations of preference queries. *IEEE Data Engineering Bulletin*, 34(2):4–11, 2011.

[9] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline queries, front and back. *ACM SIGMOD Record*, 42(3):6–18, 2013.

[10] S. de Amo and M. R. Ribeiro. CPref-SQL: A query language supporting conditional preferences. In *Proc. of ACM Symp. on Applied Computing (ACM SAC)*, pages 1573–1577, 2009.

[11] P. Höfner and B. Möller. Dijkstra, floyd and warshall meet kleene. *Formal Aspects of Computing*, 24(4-6):459–476, 2012.

[12] U. L. Hou, N. Mamoulis, and K. Mouratidis. Efficient evaluation of multiple preference queries. *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 1251–1254, 2009.

[13] W. Kießling, M. Endres, and F. Wenzel. The Preference SQL system: An overview. *IEEE Data Engineering Bulletin*, 34(2):11–18, 2011.

[14] R. Nedbal. Preference handling in relational query languages. In *Proc. of Int. Conf. on Application of Information and Communication Technologies (AICT)*, pages 1–7. IEEE, 2011.

[15] F. S. F. Pereira and S. de Amo. Evaluation of conditional preference queries. *Journal of Information and Data Management*, 1(3):503–518, 2010.

[16] T. Preisinger, W. Kießling, and M. Endres. The BNL++ algorithm for evaluating pareto preference queries. In *Multidisc. Workshop on Advances in Preference Handling*, pages 114–121, 2006.

[17] N. Wilson. Extending cp-nets with stronger conditional preference statements. In *Proc. of AAAI Nat. Conf. on Artificial Intelligence*, pages 735–741, 2004.

APPENDIX

A. PROOFS

PROOF OF LEMMA 1. Suppose by absurd that $t \succ_\varphi t'$ and there no exists a comparison b such that $t \succ_b t'$. Let f, f' be two essential formulas over Γ such that $f = C_\varphi \wedge Q^+_\varphi(A_\varphi) \wedge g \wedge w$ and $f' = C_\varphi \wedge Q^-_\varphi(A_\varphi) \wedge g \wedge w'$ and $\mathbf{Att}(w) \subseteq W_\varphi$ and $\mathbf{Att}(w') \subseteq W_\varphi$ and $t \models f$ and $t' \models f'$. By definition, $f \succ_\varphi f'$. Let the set $W' = (W_\varphi \cup \{A_\varphi\}$. Consider the comparison $b : (f \succ f')[W']$, by definition $t \succ_b t'$, which is a contradiction. □

PROOF OF THEOREM 2. In order to prove Theorem 2, we have to prove:

1) If $t \succ_\Gamma t'$ then $t \succ_{K^*_\Gamma} t'$;
2) If $t \succ_{K^*_\Gamma} t'$ then $t \succ_\Gamma t'$.

(Part 1) By Theorem 1, $t \succ_\Gamma t'$ if and only if $t = t_1 \succ_{\varphi_1} ... \succ_{\varphi_m} t_{m+1} = t'$. By definition, if $t \succ_{K^*_\Gamma} t'$ then there exists $b \in K^*_\Gamma$ such that $t \succ_b t'$. Thus, we must prove that if $t_1 \succ_{\varphi_1} ... \succ_{\varphi_m} t_{m+1}$ then there exists $b \in K^*_\Gamma$ such that $t \succ_b t'$. By Lemma 1, there exists comparisons $b_i : (f^+_{b_i} \succ f^-_{b_i})[W_{b_i}]$ such that $t_i \succ_{b_i} t_{i+1}$ and $f^+_{b_i} = C_{\varphi_i} \wedge Q^+_{\varphi_i}(A_{\varphi_i}) \wedge g_i \wedge w_i$ and $f^-_{b_i} = C_{\varphi_i} \wedge Q^-_{\varphi_i}(A_{\varphi_i}) \wedge g_i \wedge w'_i$ for all $i \in \{1, ..., m\}$. Thus, we have to show that there exists f_i such that $f_i = f^-_{b_i} = f^+_{b_{i+1}}$ for all $i \in \{1, ..., m-1\}$. Suppose by absurd that f_i does not exist. Consider $B' = \mathbf{Att}(f^-_{b_i}) \cap \mathbf{Att}(f^+_{b_{i+1}})$. Let h_i, h'_i sub-formulas of $f^-_{b_i}$ such that $\mathbf{Att}(h_i) = B'$ and $\mathbf{Att}(h'_i) = \mathbf{Att}(f^-_{b_i}) - B'$. Let h_{i+1}, h'_{i+1} sub-formulas of $f^+_{b_{i+1}}$ such that $\mathbf{Att}(h_{i+1}) = B'$ and $\mathbf{Att}(h'_{i+1}) = \mathbf{Att}(f^+_{b_{i+1}}) - B'$. As $\mathbf{Att}(h_i) = \mathbf{Att}(h_{i+1})$ and $t_i \models h_i$ and $t_i \models h_{i+1}$ then $h_i = h_{i+1}$. Consider $f'_i = h_i = h_{i+1}$ and $f_i = f'_i \wedge h'_i \wedge h'_{i+1}$. We can see that $f_i = f^-_{b_i}$ and $f = f^+_{b_{i+1}}$ such that $C_{\varphi_i} \wedge Q^-_{\varphi_i}(A_{\varphi_i}) \wedge g_i = f'_i \wedge h'_i$ and $w_i = h'_{i+1}$ and $C_{\varphi_{i+1}} \wedge Q^+_{\varphi_{i+1}}(A_{\varphi_{i+1}}) \wedge g_{i+1} = f'_i \wedge h'_{i+1}$ and $w_{i+1} = h'_i$. Thus, $f_i = f^-_{b_i} = f^+_{b_{i+1}}$ which is a contradiction.

(Part 2) Suppose by absurd that $t \succ_{K^*_\Gamma} t'$ and $t \not\succ_\Gamma t'$. If $t \succ_{K^*_\Gamma} t'$ then there exists $b \in K^*_\Gamma$ such that $t \succ_b t'$. By definition, if $t \succ_b t'$ then there exists formulas $f_1, ..., f_{m+1} \in F_\Gamma$ and cp-rules $\varphi_1, ..., \varphi_m \in \Gamma$ such that $f_1 \succ_{\varphi_1} ... \succ_{\varphi_m} f_{m+1}$ and $t \models f_1$ and $t' \models f_{m+1}$. Therefore, there exists tuples $t_1, ..., t_{m+1}$ such that $t_i, t_{i+1} \models f_i$ and $t = t_1$ and $t' = t_{m+1}$. Thus, $t_1 \succ_{\varphi_1} ... \succ_{\varphi_m} t_{m+1}$ and $t \succ_\Gamma t'$ which is a contradiction. □

PROOF OF THEOREM 3. Suppose by absurd that $t \succ_{K^*_\Gamma} t'$ and $t \not\succ_{K_\Gamma} t'$. This means there exists a comparison $b' \in K^*_\Gamma$ such that $t \succ_{b'} t'$ and there no exists an essential comparison $b \in K_\Gamma$ such that $t \succ_b t'$. If there is no a comparison more generic than b' then $b' \in K_\Gamma$ which is a contradiction. If no, then there exists another comparison $b \in K_\Gamma$ more generic than b' which also is a contradiction. □