Optimization of Preference Queries with Multiple Constraints

Markus Endres and Werner Kießling
University of Augsburg
Institute for Computer Science
Germany
{endres,kiessling}@informatik.uni-augsburg.de

ABSTRACT

Nowadays, the efficient integration of preference querying into standard database technology is an important issue. In some instances, preference queries challenge traditional query processing and optimization. In this paper we study preference database queries involving hard constraints over multiple attributes belonging to several relations. The main bottleneck for such queries is the computation of the cartesian product which may lead to high memory and computation costs. We develop algebraic optimization techniques to transform a preference query with hard constraints in order to enable its efficient processing by database engines. For this purpose, we show a dominance criterion and we introduce rewriting techniques to eliminate dominated tuples before building the cartesian product and therefore speed up the evaluation. These techniques lead to novel preference transformation laws and extend previous developed rules.

1. INTRODUCTION

Preferences are ubiquitous in everyday private and business life. They recently have received increased attention in the scope of personalized applications. In many cases they are designed for use in database search engines and e-commerce applications (e.g. [4, 7, 16]). In some instances, preference queries challenge traditional query processing and optimization, as illustrated by the following example.

EXAMPLE 1. *Consider a database storing nutritional information for single servings of different kinds of food relations like Soups, Meats and Beverages. A user, Mrs. Diet, wants to complete her diet sheet and therefore is interested in finding meals that satisfy nutritional requirements such as a restriction on the number of calories (cal), the amount of Vitamin C (Vc) and the amount of total lipid fat (abbr. fat). For example, the recommendations for a 30-year old female, who is moderately active, are at most 1100 calories, at least 38mg of Vitamin C and maximal 9g of fat for a main meal [24].*

However, in the context of a diet, each user has preferences concerning its meals. Mrs. Diet for example ***likes chicken soup*** *as starter. The main course* ***should be beef*** *and the* ***cholesterol*** *of the beef* ***should be as little as possible****. For beverage she* ***likes red wine****. These preferences are equally important for her (a Pareto preference). The complete meal* ***must fulfill*** *the restrictions of maximal* ***1100 calories****, at least* ***38g of vitamin C*** *and at most* ***9g fat*** *for her personalized diet sheet.*

Mrs. Diet wants find all such meals which fulfill the hard constraints and satisfy her preferences best possible.

Using Kießling's approach of modelling preferences as strict partial orders [13, 14], the above mentioned hard and soft constraints can be expressed by Preference SQL [16] as follows:

```
SELECT S.name, M.name, B.name
FROM   Soups S, Meats M, Beverages B
WHERE  S.cal + M.cal + B.cal ≤ 1100
  AND  S.Vc + M.Vc + B.Vc ≥ 38
  AND  S.fat + M.fat + B.fat ≤ 9
PREFERRING
  S.name IN ('Chicken soup')        AND
 (M.name IN ('Beef')
          AND M.Cholesterol LOWEST) AND
  B.name IN ('Red wine')
```

This query expresses Mrs. Diet's preferences after the keyword `PREFERRING`*. It is a Pareto preference (*`AND`*)* [1] *consisting of preferences on soups, meats and beverages.* `IN` *denotes a preference for members of a given set, a POS-preference. The whole preference is evaluated on the result of the hard sum constraints.*

The query in the example is a preference query containing multiple sum constraints in its condition and user preferences on some attributes. Therefore, we call such queries **Preference Queries with Multiple Constraints**.

Of course, those problems are not limited to planning tasks as in the example above. They also occur in the context of document retrieval [12, 8], multimedia data retrieval [6], geographic information systems [9], dynamic resource allocation on the grid [18] and e-commerce [22, 4].

Conventional approaches implement such queries by a set of binary join operators and evaluate the hard constraints. Afterwards the user preferences as soft selection combined with the Pareto operator (`AND`, $\otimes$) are evaluated by a skyline algorithm, e.g. [2, 21, 20], to retrieve all combinations that fulfill the preferences best possible.

Because the hard constraints refer to attributes from more than two relations, pair-wise join operators may fail to remove intermediate results based on these condition. For a hard constraint such

[1]Note that `AND` in the `WHERE`-clause means Boolean conjunction, whereas `AND` in the `PREFERRING`-clause denotes Pareto preference construction, i.e. all preferences are equally important.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '08, August 24-30, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

as $A_1 + ... + A_r \ \Theta \ c, \ \Theta \in \{<, >, \leq, \geq, =, \neq\}$ and c a constant, current join operators cannot test the satisfiability of an intermediate tuple until all variables have been determined. A consequence of this inability to remove intermediate tuples that will not lead to any results is that the query evaluation process must evaluate the cartesian product of all tuples of all join relations, which leads to **high memory and computation costs**, particularly if the relations are large.

The goal of this paper is to **eliminate tuples** from the relations which definitely can never be in the result set **before building the cartesian product**. This reduces the relation sizes and therefore the computation costs and needed memory for the cartesian product.

Section 4.1 of this paper is an extension of [5], where we only considered **one single sum constraint** over multiple attributes. But in real life, database queries are not restricted to only one hard constraint, but use several different constraints, e.g. as in example 1. Therefore, we must consider **multiple hard constraints** over multiple attributes and for such problems we present novel rewriting techniques in this paper. Further on, we present optimization techniques based on mathematical rewriting of the original constraints and add additional **selection operators** into the Preference SQL query, cp. section 4.2.

The rest of this paper is organized as follows: In the next section we take a look at related work. Afterwards we describe the background of our preference model in section 3. In section 4 we develop different optimization techniques and introduce new rewriting techniques for preference queries. Further on, we present experimental results in section 5, and finally we conclude with a summary and outlook.

2. RELATED WORK

Database queries with constraints over attributes belonging to several relations occur frequently in the real world, but have not been intensively researched in the database context in the past.

Agarwal et. al. [1] address queries with linear constraints, and Guha et. al. [10] address queries with aggregation constraints. However, their work is only valid for queries on one relation.

Ilyas et. al. [12] developed algorithms for top-k queries that can be extended to implement queries with a constraint on the value of a monotone function, but this is only valid for one constraint. Liu, Yang and Foster [17] integrated constraint-programming techniques with traditional database techniques to solve sum constraint queries by modifiying existing nested loop-join operators. Nestorov, Liu and Foster [19] use similar principles but can be implemented without modifying existing database engines, but not in combination with preferences.

Georgiadis et. al. [8] prevent the construction of tuples that cannot appear in the result of a preference query using appropriate linearizations, but not in combination with hard constraints.

Hafenrichter [11] and Chomicki [3] developed extensive transformation laws and rewriting techniques for various preference queries, e.g. 'push preference over hard constraint', but these laws are not applicable for the processing of preference queries with hard constraints over multiple attributes belonging to more than one relation.

Döring et. al. [4] present a first approach for processing of queries dealing with individual and global preferences of customers. Their preference query rewriting is a kind of query expansion to deliver all tuples that matches the original preferences of a user as well as cheaper alternatives in an online travel booking system.

In [5] we developed transformation laws to optimize preference queries with **one sum constraint** over a set of attributes belonging to several relations. But these laws are only valid for one constraint. In this paper we develop laws for **multiple constraints** and introduce further optimization techniques. Further on, we show how to integrate our new laws into the preference query optimizer implemented by Hafenrichter [11].

Note that our approach does not intend to replace other optimization methods. Instead, our method can be combined with other algorithms to allow more *Optimization of Preference Queries with Multiple Constraints* on relational database systems.

3. PREFERENCE ALGEBRA

In this section we want to give some background concerning the preference technology.

3.1 Preference Algebra

Preference frameworks tailored to standard database systems have been introduced in [13] and [3]. We depict the preference algebra from [13] which is a direct mapping to relational algebra and declarative query languages. This preference model is based on *strict partial orders* and is semantically rich, easy to handle and very flexible to represent user preferences which are ubiquitous in our life.

DEFINITION 1. ***Preference***
Let $A = \{A_1, \ldots, A_n\}$ be a set of attribute names with corresponding domains of values $dom(A_i)$. The domain of A is defined as $dom(A) = dom(A_1) \times \cdots \times dom(A_n)$. Then a preference P is a strict partial order $P = (A, <_P)$, where $<_P \subseteq \ dom(A) \times dom(A)$.
The term $x <_P y$ is interpreted as "I like y more than x".

Having defined preferences as strict partial orders we provide a variety of intuitive and customizable **base preference constructors** for categorical and numerical domains which can intuitively be combined to build complex preferences still yielding partial orders. Formally, a base preference constructor has one or more arguments, the first characterizing the attributes A and the others the strict partial order $<_P$, referring to A.

For example, the POS-preference $POS(A, <_P)$ on an attribute A expresses that a special value of an attribute is preferred to all others (compare the `IN`-clause in example 1). There is also a NEG preference constructor, which is defined analogous to the POS-preference, but with a NEG-set. Moreover, it is possible to combine these preferences to *POS/NEG* or *POS/POS*.

If we want to focus on preferences where the domain is a numerical data type, e.g. decimal, which can be infinite, we can use a number of *numerical preference constructors* which are defined by [13], e.g. LOWEST, HIGHEST, AROUND, BETWEEN and the SCORE preference. LOWEST(cholesterol), for example, represents that a person prefers lower values for 'cholesterol' over higher values.

DEFINITION 2. ***Extremal preferences***
We define the LOWEST and HIGHEST preference that the desired value should be as low (high) as possible. Formally:

- *P is called a LOWEST preference, if: $x <_P y$ iff $x > y$*
- *P is called a HIGHEST preference, if: $x <_P y$ iff $x < y$*

There are further categorical and numerical base preference constructors. Detailed information on these preference constructors are given in [13].

In the following we will briefly discuss two **complex preference constructors**, namely the *Pareto preference* (`AND`, $\otimes$) and the *Prioritization* (`PRIOR TO, &`).

DEFINITION 3. ***Pareto preference:*** $P_1 \otimes P_2$
Given two preferences $P_1 = (A, <_{P_1})$ *and* $P_2 = (B, <_{P_2})$*, for* $x, y \in dom(A) \times dom(B)$ *we define*

$$x <_{P_1 \otimes P_2} y \quad \textit{iff}$$
$$(x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 = y_2)) \quad \vee$$
$$(x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 = y_1))$$

$P = (A \cup B, <_{P_1 \otimes P_2})$ *is called Pareto preference modelling* P_1 *and* P_2 *as* ***equally important****.*

DEFINITION 4. ***Prioritized preference:*** $P_1 \,\&\, P_2$
Given two preferences $P_1 = (A, <_{P_1})$ *and* $P_2 = (B, <_{P_2})$*, for* $x, y \in dom(A) \times dom(B)$ *we define*

$$x <_{P_1 \& P_2} y \quad \textit{iff}$$
$$x_1 <_{P_1} y_1 \vee (x_1 = y_1 \wedge x_2 <_{P_2} y_2)$$

$P1$ *is considered* ***more important*** *than* P_2*;* P_2 *is respected only where* P_1 *does not mind.*

A generalization of the Pareto preference constructor and the Prioritization to more than two preferences is obvious. An extended definition can be found in [21, 20]. Detailed information on all preference constructors are given in [13].

EXAMPLE 2. *As in example 1, beef and cholesterol as little as possible are equally important for Mrs. Diet. With preference algebra we describe her preferences as:*

$$P = POS(Meats, \{Beef\}) \otimes LOWEST(cholesterol)$$

3.2 The BMO Query Model

Given preferences over a set of attributes a central question is to determine an outcome that is preferentially optimal with respect to the preference statements. Whether preferences can be satisfied depends on the current database contents. Thus a match-making between wishes and data has to be made.

For this purpose the **Best-Matches-Only (BMO)** query model has been proposed by [13].

DEFINITION 5. ***BMO-Set***
The ***Best-Matches-Only*** *result set contains only the best matches w.r.t the strict partial order of a preference* P*. It is a selection of unordered result tuples where all tuples in the BMO-set are undominated by others regarding the preference* P*.*

In principle, efficient BMO query evaluation requires two new relational operators. We define

$$\sigma[P](R) := \{t \in R \mid \neg\exists t' \in R : t[A] <_P t'[A]\}$$

as **preference selection**. It finds all best matching tuples t for a preference $P = (A, <_P)$ with $A \subseteq \text{attr}(R)$[2]. If none exists, it delivers *best-matching alternatives*, but *nothing worse*.

A preference can also be evaluated in grouped mode, given some $B \subseteq \text{attr}(R)$. This can be expressed as the **grouped preference selection**

$$\sigma[P \text{ groupby } B](R) :=$$
$$\{t \in R \mid \neg\exists t' \in R : t[A] <_P t'[A] \wedge t[B] = t'[B]\}\,.$$

$\sigma[P](R)$ and $\sigma[P \text{ groupby } B](R)$ can perform the match-making process as required by BMO semantics.

[2]We use attr(R) to denote all attributes of a relation R

4. OPTIMIZATION TECHNIQUES

[11, 15] and [3] laid the foundations of a framework for preference query optimization that extends established query optimization techniques from relational databases. They presented a variety of new laws for preference relational algebra to optimize preference queries.

The first key to our algebraic optimization is a **dominance criterion**, which allows us to neglect dominated tuples which do not satisfy the user's preferences, before evaluating the cartesian product. The second optimization are additional **selection operators** which can be retrieved from the hard constraints and even applied before building the cartesian product. This speeds-up evaluation and reduces memory and computation costs.

4.1 Dominance Criterion

In [5] we introduced a dominance criterion to optimize *Preference SUM-constraint queries*, but this criterion is only valid for a single sum constraint in the query. Now we consider more complex hard constraints (e.g. calories, vitamin C and fat in example 1) and therefore we have to modify this dominance criterion given in [5]. But the dominance criterion is not restricted to *sum* constraints, it is also valid for multiplication, or, generally for each **monotone function**. Before we give the extended definition of the dominance criterion, we define the class of preference queries our approach can handle.

DEFINITION 6. ***Preference Query with Multiple Constraints***
We define a ***Preference Query with Multiple Constraints*** *as* [3]

$$\mathbf{Q} := \sigma[P_1\ \Phi_1 ...\ \Phi_r\ P_r]\ \sigma_F\ (R_1 \times ... \times R_r)$$

where

$$F := F_1 \wedge ... \wedge F_r$$

are multiple constraints with

$$F_k = \sum_{i=1}^{r} \rho_i \cdot A_{i_j}\ \Theta_k\ c_k, \quad j = 1, ..., n_i\ ,$$

a ***SUM-constraint*** *or*

$$F_k = \prod_{i=1}^{r} \rho_i \cdot A_{i_j}\ \Theta_k\ c_k, \quad j = 1, ..., n_i\ ,$$

a ***Multiplication-constraint*** *with*

- $P_i = (B_i, <_{P_i})$ *arbitrary preferences*
- $\Phi_i \in \{\otimes, \&\},\ i = 1, ..., r$ *a* ***Pareto*** *or* ***Prioritization*** *operator (cp. section 3)*
- $R_i(A_{i_1}, ..., A_{i_{n_i}}, B_i),\ i = 1, ..., r$ *database relations, where* n_i *is the number of numerical attributes in relation* R_i
- A_{i_j} *positive numerical attributes*
- $\rho_i \in \mathbb{R}_0^+,\ i = 1, ..., r$ *a numerical multiplicator*
- $\Theta_k \in \{<, \leq, >, \geq, =, \neq\}$,
- $c_k \in \mathbb{R}_0^+$ ***a constant***

[3]$\sigma[P]$ means preference selection, i.e. a soft constraint, whereas σ_F denotes a classical relational algebra selection, i.e. a hard constraint.

Note: The definition can also be extended to queries with join conditions, see section 4.3.

Now we provide the first optimization technique for preference queries with multiple constraints.

THEOREM 1. ***Extended Dominance-Criterion***
Consider a query

$$\mathbf{Q} := \sigma[P_1 \ \Phi_1 ... \ \Phi_r \ P_r] \ \sigma_F \ (R_1 \times ... \times R_r)$$

as described in definition 6.
Let tuples $t, t' \in R_i$ such that

$$t[B_i] <_{P_i} t'[B_i] \quad \wedge \qquad t[A_{i_1}] \ \hat{\Theta}_1 \ t'[A_{i_1}] \ \wedge \ ... \ \wedge \ t[A_{i_r}] \ \hat{\Theta}_r \ t'[A_{i_r}] \qquad (*)$$

and $\hat{\Theta}_j, \ \ j = 1, ..., r$ defined as

$$\hat{\Theta}_j := \begin{cases} \geq & \textit{iff} \quad \Theta_j \in \{\leq, <\} \\ \leq & \textit{iff} \quad \Theta_j \in \{\geq, >\} \\ = & \textit{iff} \quad \Theta_j \in \{=, \neq\} \end{cases}$$

*Then an optimal solution exists for our Preference Query with Multiple Constraints Q **without** the tuple $t \in R_i$, i.e. using the dominance criterion $(*)$ for each relation, Q leads to a correct and complete solution.*

PROOF. Let $w := (t_1, ..., t, ..., t_r)$ and $v := (t_1, ..., t', ..., t_r)$ two tuples in $R := R_1 \times ... \times R_r$ which only differ in t and t' with $t, t' \in R_i$ and

$$t[B_i] <_{P_i} t'[B_i] \quad \wedge \qquad t[A_{i_1}] \ \hat{\Theta}_1 \ t'[A_{i_1}] \ \wedge \ ... \ \wedge \ t[A_{i_r}] \ \hat{\Theta}_r \ t'[A_{i_r}]$$

Then, it is evident that:

1) if one of the hard constraints F_k in F fails for v, also the hard constraint fails for w, since $t[A_i] \ \hat{\Theta}_j \ t'[A_i]$ (**we only consider monotone functions**). Therefore w is not an element of the solution.
2) if v fulfills all hard constraints, then
   a) if w fails one hard constraint in F, then w is not an element of the solution.
   b) if w also fulfills all hard constraints in F, then we know $t[B_i] <_{P_i} t'[B_i]$, i.e. tuple t' is preferred to t.
   Now, it follows from the preference $P := P_1 \ \Phi_1 ... \ \Phi_r \ P_r$ that v is preferred over w since t' is preferred w.r.t P_i and all others are equal.

□

Remarks: With the SUM-constraints respectively the Multiplication-constraints F_k in definition 6 we specified a wide range of monotone arithmetic functions possible in a *SELECTION* clause of a Preference SQL query. However, the dominance criterion is valid for **all monotone functions**, since the proof only refers to monotone functions and the given preferences.

We will give a short example for the extended dominance criterion.

EXAMPLE 3. *Revisit example 1 with Mrs. Diet's hard constraints. For her query we have three SUM-constraints with maximal 1100 kcal (Θ_1 is $\leq$), at least 38g of Vitamin C (Θ_2 is $\geq$) and maximal 9g fat (Θ_3 is $\leq$). Further on, she prefers chicken soup. Table 1 represents a simple soup relation.*

Table 1: Example for the dominance criterion

Soups	*ID*	*Name*	*Cal*	*Vc*	*Fat*
	S1	Vegetable	59	12	1
	S2	Chicken	110	2	4
	S3	Chicken	198	9	2
	~~S4~~	~~Noodle~~	~~453~~	~~1~~	~~17~~

Since 'S4' has more calories than 'S3' ($\hat{\Theta}_1$ is $\geq$), less Vitamin C than 'S3' ($\hat{\Theta}_2$ is $\leq$), more fat than 'S3' ($\hat{\Theta}_3$ is $\geq$) and 'S4' is worse than 'S3' concerning the soup preference, tuple 'S4' is dominated, i.e. we have not to consider 'S4' in the cartesian product and the hard selection. In the preference evaluation tuple 'S4' is unimportant, since 'S2' and 'S3' are preferred, even if the combination with 'S4' would fulill all hard constraints. This results in the undominated tuples of {'S1','S2','S3'}. Tuple 'S3' is not worse than 'S2' with regard to the preference. Tuple 'S1' is worse than 'S2' and 'S3' concerning the same preference, but maybe 'S2' and 'S3' do not fulfill the hard constraints ('S1' has less calories, higher vitamin C and less fat). So we must take into account tuple 'S1'.

As shown in the example, the dominance criterion from theorem 1 allows us to eliminate tuples from the relations before building the cartesian product. This reduces relation sizes and therfore speeds-up the computation.

For evaluation of the dominance criterion we use the BMO query model and apply the so called CUTOFF preference constructor, also see [5]. Since we extended the definition of the dominance criterion, we also have to change the CUTOFF constructor for a valid BMO evaluation.

DEFINITION 7. ***CUTOFF Preference Constructor***
Given a preference $P = (B, <_P)$ on a relation $R(A_1, ..., A_n, B)$ with tuples $x = (x_1, ..., x_n, x_B), \ y = (y_1, ..., y_n, y_B)$ in a Preference Query with Multiple Constraints (cp. definition 6).
Then $P_c := CUTOFF(P)$, if:

$$x <_{P_c} y \ \ \textit{iff} \ \ x_B <_P y_B \ \wedge \ x_1 \ \hat{\Theta}_1 \ y_1 \ \wedge \ ... \ \wedge \ x_n \ \hat{\Theta}_n \ y_n$$

where

$$\hat{\Theta}_j := \begin{cases} \geq & \textit{iff} \quad \Theta_j \in \{\leq, <\} \ \ j = 1, ..., n \\ \leq & \textit{iff} \quad \Theta_j \in \{\geq, >\} \ \ j = 1, ..., n \\ = & \textit{iff} \quad \Theta_j \in \{=, \neq\} \ \ j = 1, ..., n \end{cases}$$

*Tuple x is worse than y concerning the CUTOFF-preference P_c, if y_B is better than x_B regarding the given preference P (the preference specified by the user) **and** all comparison operators $\hat{\Theta}_j$ are evaluated to true.*

The evaluation of the dominance criterion by the CUTOFF constructor makes the computation of Preference Queries with Multiple Constraints evident: Our Preference SQL engine [16] applies the CUTOFF preference independently on each stream of tuples. Afterwards it performs the cartesian product (checking the hard sum constraint) and finally evaluates the preference selection by a skyline algorithm [2, 21, 20], also cp. section 4.3.

4.2 Selection Operators

In this section we add hard selection constraints to the original Preference SQL query to filter tuples before building the cartesian product. The basic idea was developed by [19] and is now adapted to preference queries with multiple constraints. For this, each constraint in the query condition is rewritten as a set of simpler constraints and an additional hard **selection operator** is applied on each relation.

4.2.1 Selection Operator for SUM-Constraints

Consider a sum constraint like

$$\rho_1 A_1 + ... + \rho_r A_r \; \Theta \; c \qquad (**)$$

with c a constant, $\Theta \in \{<, >, \leq, \geq\}$ and $\rho_k \in \mathbb{R}^+$. Obviously

$$\rho_i A_i \;\; \Theta \;\; c - \sum_{k=1, k\neq i}^{r} \rho_k A_k \quad \text{for } i = 1, ..., r$$

From this expression we can calculate the lower (lb_i) and upper (ub_i) bounds of it as follows:

$$lb_i = c - \sum_{k=1, k\neq i}^{r} max(\rho_k A_k) \quad \text{for } i = 1, ..., r$$

$$ub_i = c - \sum_{k=1, k\neq i}^{r} min(\rho_k A_k) \quad \text{for } i = 1, ..., r$$

Using lb_i and ub_i it is possible to create additional constraints for attribute A_i depending on $\Theta \in \{<, >, \leq, \geq\}$. If $\Theta \in \{<, \leq\}$ we call such constraints **maximum sum constraints** and derive

$$A_i \; \Theta_{<,\leq} \; \frac{ub_i}{\rho_i} \; .$$

For $\Theta \in \{>, \geq\}$ we call such constraints **minimum sum constraints** and derive the following additional constraint for A_i:

$$A_i \; \Theta_{>,\geq} \; \frac{lb_i}{\rho_i} \; .$$

Any value which does not fulfill the additional constraint can not satisfy the sum constraint shown in ($**$). Through these additional hard constraints we can filter out tuples from the relations that will not lead to any query result due to the hard constraints before building the cartesian product.

EXAMPLE 4. *Consider the example database in table 2 and Mrs. Diet's preference query*

```
SELECT S.name, M.name, B.name
FROM   Soups S, Meats M, Beverages B
WHERE  S.cal + M.cal + B.cal ≤ 1100
  AND  S.Vc + M.Vc + B.Vc ≥ 38
  AND  S.fat + M.fat + B.fat ≤ 9
PREFERRING
  S.name IN ('Chicken soup')          AND
 (M.name IN ('Beef')
          AND M.Cholesterol LOWEST) AND
  B.name IN ('Red wine')
```

For the hard constraint of 1100 kcal we can express the following upper bounds and therefore maximum constraints:

- $min(M.Cal) + min(B.Cal) = 903 \Rightarrow S.Cal \leq 197$
- $min(S.Cal) + min(B.Cal) = 144 \Rightarrow M.Cal \leq 956$
- $min(S.Cal) + min(M.Cal) = 877 \Rightarrow B.Cal \leq 223$

Table 2: Example database

	Soups	*ID*	*Name*	*Cal*	*Vc*	*Fat*
		S1	Vegetable	59	12	1
$S.Vc < 3$		~~S2~~	~~Chicken~~	~~140~~	~~2~~	~~4~~
$S.Cal > 197$		~~S3~~	~~Chicken~~	~~198~~	~~9~~	~~2~~
$DC, S.Cal > 197$		~~S4~~	~~Noodle~~	~~453~~	~~1~~	~~17~~

	Meats	*ID*	*Name*	*Cal*	*Vc*	*Fat*	*Cholesterol*
		M1	Turkey	818	13	8	6
		M2	Beef	857	14	6	4
DC		~~M3~~	~~Pork~~	~~911~~	~~12~~	~~12~~	~~15~~

	Beverages	*ID*	*Name*	*Cal*	*Vc*	*Fat*
$B.Vc < 12$		~~B1~~	~~Red Wine~~	~~85~~	~~1~~	~~0~~
		B2	Red Wine	181	14	0
		B3	Coke	220	21	2
$B.Cal > 223$		~~B4~~	~~Lemonade~~	~~281~~	~~17~~	~~2~~
$B.Cal > 223$		~~B5~~	~~Red Wine~~	~~190~~	~~1~~	~~0~~

This means, there only exists a solution for the hard sum constraint with tuples in the soup relation with $S.Cal \leq 197$. Tuples with $S.Cal > 197$ ($\{$S3, S4$\}$) can be eliminated from the relation (take not part in the cartesian product) since there is no combination with less than 1100 kcal possible. The same with beverages. In the meats relation all tuples have $M.Cal \leq 956$ and therefore none can be eliminated by additional selection operators. But tuple 'M3' is dominated due to the dominance criterion (theorem 1), see table 2. For the hard constraint of 38 g of Vitamin C it follows: $S.Vc \geq 3$, $M.Vc \geq 5$ and $B.Vc \geq 12$. We can dominate tuple 'S2' ($S.Vc = 2$ and it should be $S.Vc \geq 3$), and tuple 'B1'. For the fat constraint there are no valid additional selection operators.

Considering this simple database we reduced the cartesian product from 60 possible combinations to 4 combinations. Afterwards it is possible to check all hard constraints and thereafter apply a skyline algorithm to evalute the user preference on the remaining combinations which leads to the result $\{$S1, M2, B2$\}$.

4.2.2 Selection Operator for Multiplication Constraints

We can also introduce selection operators for multiplication constraints. This is analogous to the SUM constraints. Consider

$$\rho_1 A_1 \cdot ... \cdot \rho_r A_r \; \Theta \; c \qquad (***)$$

with c a constant, $\Theta \in \{<, >, \leq, \geq\}$ and $\rho_k \in \mathbb{R}^+$. We again calculate the lower (lb_i) and upper (ub_i) bounds of it as follows:

$$lb_i = c \cdot \left(\prod_{k=1, k\neq i}^{r} max(\rho_k A_k) \right)^{-1} \quad \text{for } i = 1, ..., r$$

$$ub_i = c \cdot \left(\prod_{k=1, k\neq i}^{r} min(\rho_k A_k) \right)^{-1} \quad \text{for } i = 1, ..., r$$

We get the **maximum multiplication constraint** for $\Theta \in \{<, \leq\}$

$$A_i \; \Theta_{<,\leq} \; \frac{ub_i}{\rho_i}$$

and the **minimum multiplication constraint** for $\Theta \in \{>, \geq\}$

$$A_i \; \Theta_{>,\geq} \; \frac{lb_i}{\rho_i} \; .$$

Depending on the constraints we can eliminate tuples from the relations to speed-up evaluation.

4.3 Rewriting Technique

Hafenrichter and Kießling [11, 15] constructed a preference query optimizer as an extension of an existing SQL optimizer, adding new heuristics like 'push preference'. This query optimizer is based on a classical Hill-Climbing algorithm given by Ullmann [23]. With the preparatory work from section 4 we can develop transformation laws for preference relational algebra that allow us to eliminate dominated tuples before building the cartesian product by *inserting the CUTOFF preference* and the *additional selection operators* into the query.

We intergrated our novel rewriting techniques - the *dominance criterion* (section 4.1) and the additional *selection operators* (4.2) - into this preference query optimizer.

For this, we refer to the notation introduced in section 4, this means, we skip some formal definitions and keep it short and clearly arranged. We abbreviate the use of the CUTOFF(P) constructor for a preference P with $\mathbf{P_c}$ and the selection operators with $\mathbf{SO}$. Since figures are more meaningful than sentences, we demonstrate the rewriting technique using an operator tree.

With the help of P_c and the selection operators SO we can transform a Preference Query with Multiple Constraints into an optimized query as follows:

i) Determine the lower respectively upper bounds (abbr. b_i) for each constraint. Insert the corresponding selection operators $\sigma_{A_i\ \Theta_i\ b_i}$ into the query tree.

ii) Apply the CUTOFF preference constructor P_c to the result of the selection operator.

iii) Build cartesian product and check the constraints $F_1 \wedge ... \wedge F_r$.

iv) Evaluate the preference selection $P_1\ \Phi_1 ...\ \Phi_r\ P_r$ by a skyline algorithm [4].

Formal:

COROLLARY 1. ***Insert $\mathbf{P_c}$ and $\mathbf{SO}$ into Cartesian Product*** *Consider a preference query*

$$Q := \sigma[P_1\ \Phi_1 ...\ \Phi_r\ P_r]\ \sigma_F\ (R_1 \times ... \times R_r)$$

where $F := F_1\ \wedge\ ...\ \wedge\ F_r$ *and*

- $P_i = (B_i, <_{P_i})$ *arbitrary preferences*
- $\Phi_i \in \{\otimes,\ \&\},\ \ i = 1, ..., r$ *a* ***Pareto*** *or* ***Prioritization*** *operator (cp. section 3)*
- $R_i(A_{i_1}, ..., A_{i_{n_i}}, B_i),\ i = 1, ..., r$ *database relations, where* n_i *is the number of numerical attributes in relation* R_i
- A_{i_j} *positive numerical attributes*
- $F_k = \sum_{i=1}^{r} \rho_i \cdot A_{i_j}\ \Theta_k\ \ c_k$ ***SUM-constraints*** *or*
- $F_k = \prod_{i=1}^{r} \rho_i \cdot A_{i_j}\ \Theta_k\ \ c_k$ ***Multiplication-constraints***
- $\rho_i \in \mathbb{R}_0^+,\ i = 1, ..., r$ *a numerical multiplicator*
- $\Theta_k \in \{<, \leq, >, \geq, =, \neq\}$, *and* $c_k \in \mathbb{R}_0^+$ *a* ***constant***
- b_i *the* ***lower*** *respectively* ***upper bounds*** *(cp. section 4.2) for the selection operators (only valid, if* $\Theta_k \in \{<, \leq, >, \geq\}$*)*

Than we can transform the query Q into the following optimized operator tree, see figure 1:

[4]Since a skyline algorithm [2, 21, 20] only can handle Pareto preferences, for Prioritization an algorithm developed by [11] can be applied.

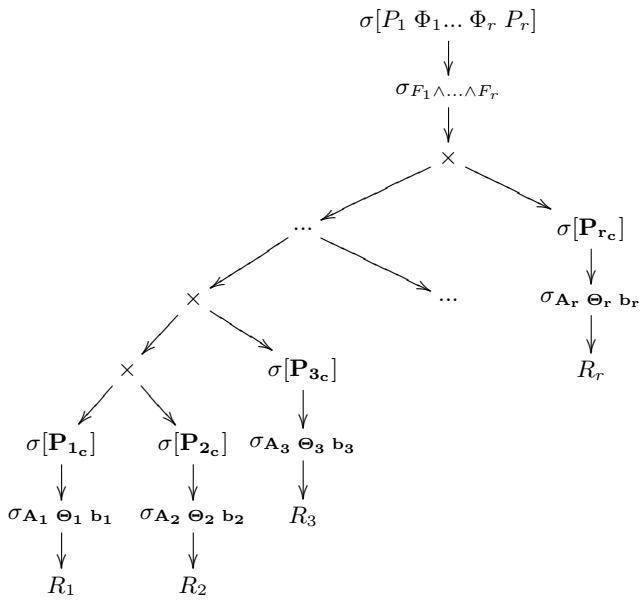

Figure 1: Optimized Preference Query

PROOF. The proof is given by theorem 1. □

Remarks:

- It is allowed to change CUTOFF and the additional hard selection operator in the Hill-Climbing algorithm. However, applying hard selection first reduces computation costs for evaluating CUTOFF.
- In the case of joins like $R_1 \bowtie_{R_1.X=R_2.X} R_2$ we have to ensure that we do not eliminate join partners, i.e. for each tuple in the first relation there must exist a join partner in the second relation. To get rid of this problem we have to evaluate the CUTOFF preference as a **grouped preference selection**, see section 3.2, [5] and [11]. Therefore, the CUTOFF preference is only evaluated for tuples in the same equivalence class, i.e. grouped by X.

5. EXPERIMENTAL RESULTS

In order to evaluate our rewriting techniques, we performed several experiments. For this we intergrated our rewriting techniques into the preference query optimizer developed by Hafenrichter [11], also cp. section 4.3. We used a real-world food database published by the USDA [24]. This database contains nutritional facts for more than 7000 types of food. From this database we created three relations as in example 1: *Soups*, *Meats* and *Beverages* containing information about their eponymous types of food. The sizes of these relations are as follows: There are 500 soups, 700 meats and 250 beverages available, i.e. about 87.500.000 possible combinations.

We run all test queries on an Oracle 10.0 database system in combination with the preference query optimizer described in [11] (Java implementation). The system is running on a Linux machine (Intel Dual Core CPU 1.6 GHz, 2GB main memory).

We evaluated the efficiency of our rewriting techniques introduced in section 4 by comparing the response times of several *Preference Queries with Multiple Constraints*. **Due to limited space** we only report a few results, with representative performance, shown below. In our result figures, we abbreviate the standard approach (full cartesian product) with **no rewriting**. Using the dominance

criterion from section 4.1 we write **DC**, using the selection operators from section 4.2 we write **SO** and for the evaluation of the dominance criterion in combination with the selection operators we write **DC+SO**.

The test query is based on example 1 and contains three constraints. For representation we only varied the amount of calories *cal*, which must be less or equal than a value called **max_cal**. The amount of vitamin C (*Vc*) and the fat value (*fat*) we fixed to the values given in example 1. This leads to the query

```
SELECT S.name, M.name, B.name
FROM   Soups S, Meats M, Beverages B
WHERE  S.cal + M.cal + B.cal ≤ max_cal
  AND  S.Vc + M.Vc + B.Vc ≥ 38
  AND  S.fat + M.fat + B.fat ≤ 9
PREFERRING
  S.name IN ('Chicken soup')        AND
 (M.name IN ('Beef')
          AND M.Cholesterol LOWEST) AND
  B.name IN ('Red wine')
```

Notice, varying the parameter **max_cal** varies the selectivity of the query, while varying the size of the relations changes the size of the problem to be solved. Therefore, we varied the required sum constraints in order to change the selectivity and we varied the size of the relations.

In our first test series we only varied the **max_cal** value in a range from 600 to 1700 calories, see figure 2.

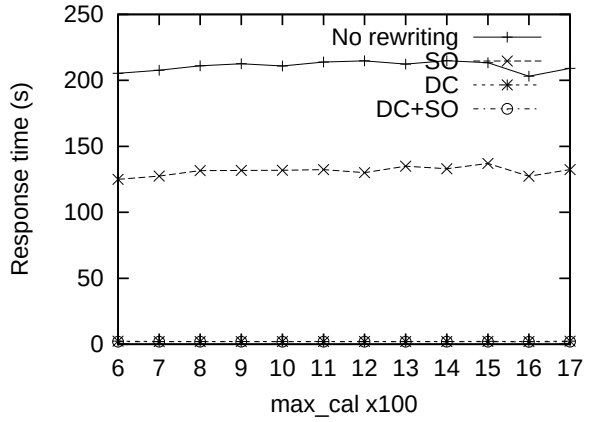

Figure 2: Performance results for different max_cal.

Since the dominance criterion only depends on the preferences and not on the hard constraints, the response time for the preference query with different **max_cal** is nearly constant for each approach. Using the selection operators (*SO*), the query response time is better than the standard approach (*no rewriting*), since some tuples can be eliminated before building the cartesian product. Using the dominance criterion (*DC*) or the combination (*DC+SO*), the response time of the query is about 2 seconds, as can be seen in figure 3, which is a zoomed figure of figure 2.

In all our tests with varied *cal*, *Vc* and *fat* it performed out, that *DC+SO* performs best for all, but is only a little bit better than *DC* alone. The reason is the worse evaluation of the additional selection operators for our real-world data from the USDA, since there are many tuples with a *NULL* or 0 (zero) entry in the corresponding attribute. For data with no or less *NULLs* and 0 entries the additional selection operators should perform much better.

Next, we run the query above with different relation sizes (but

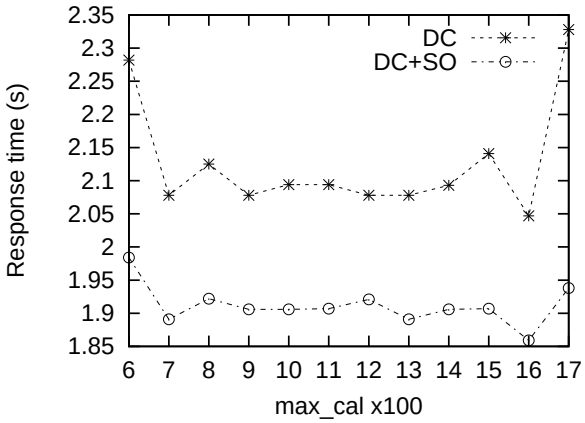

Figure 3: Comparision of DC and DC+SO.

fixed max_cal = 1100) and demonstrate the performance results in figure 4.

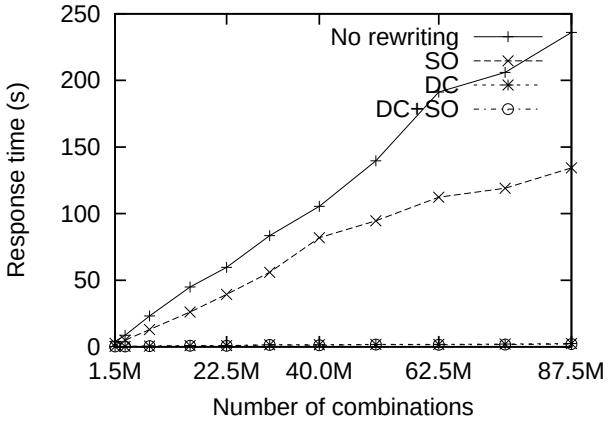

Figure 4: Performance results for different relations sizes.

As above, *DC* and *DC+SO* performs extremly good and for a comparison of these two, we show again a zoomed figure from the one above, see figure 5. The difference between *DC* and *DC+SO* is marginal, because of the reason given above (*NULL* entries).

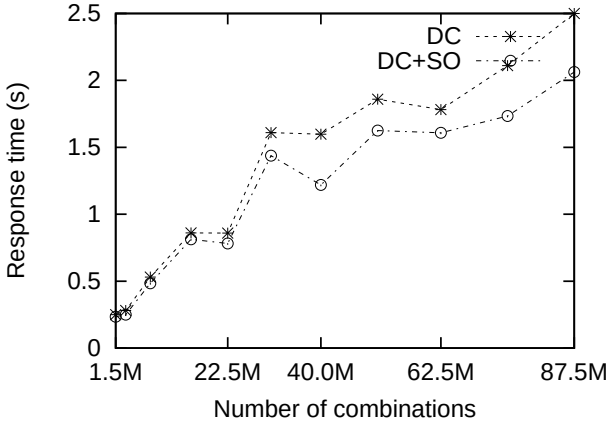

Figure 5: Comparision of DC and DC+SO.

Obviously our rewriting techniques speeds-up the evaluation because of writing in the *CUTOFF* preference and the additional selection operators into the query. The combination *DC+SO* is better than all other approaches, but the difference between *DC* and *DC+SO* is not very large.

From the experimental results of our benchmark queries, we can see that our proposed rewriting techniques improve the query performance consistently for different types of sum constraint queries.

6. SUMMARY AND OUTLOOK

Finding efficient query optimization techniques for *Preference Queries with Multiple Constraints* is beneficial for a variety of practical database applications, e.g. in planning tasks or tourism. In this paper we extended our dominance criterion to work with multiple hard constraints like sum or multiplication constraints. The dominance criterion as well as the insertion of additional selection operators based on mathematical observations eliminate tuples from relations before building the cartesian product and therefore reduces memory and computation costs, i.e. speeds-up the evaluation of *Preference Queries with Multiple Constraints*. The performance speed-ups observed so far give already strong evidence that a tightly coupled implementation inside an existing SQL query engine can achieve excellent performance.

For future work we want to develop a cost-based optimization for Pareto preference and Prioritization queries with multiple constraints. We want to develop algorithms which completly avoid the cartesian product. For this we developed a first approach which seems to be good. However, this is much more complex and needs more attention and deeper research.

7. REFERENCES

[1] P. K. Agarwal, L. Arge, J. Erickson, P. G. Franciosa, and J. S. Vitter. Efficient Searching with Linear Constraints. In *PODS*, pages 169–178, 1998.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pages 421–430. IEEE Computer Society, 2001.

[3] J. Chomicki. Preference Formulas in Relational Queries. In *ACM Transactions on Database Systems (TODS)*, volume 28, pages 427–466. ACM Press, 2003.

[4] S. Döring, T. Preisinger, and M. Endres. Advanced Preference Query Processing for E-Commerce. In *Proceedings of 23rd Annual ACM Symposium on Applied Computing (SAC)*, pages 1457–1462. ACM, 2008.

[5] M. Endres and W. Kießling. Optimization of Preference Queries under Hard Sum Constraints. In *Proceedings of the 4th Multidisciplinary Workshop on Advances in Preference Handling (AAAI). Chicago, Illinois (USA)*, 2008.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 102–113. ACM, 2001.

[7] B. Faltings, P. Pu, and J. Zhang. Agile Preference Models based on Soft Constraints. In *Challenges to Decision Support in a Changing World*, 2005.

[8] P. Georgiadis, I. Kapantaidakis, V. Christophides, E. M. Nguer, and N. Spyratos. Efficient rewriting algorithms for preference queries. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*.

[9] J. Goldstein, R. Ramakrishnan, U. Shaft, and J.-B. Yu. Processing Queries By Linear Constraints. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona*, pages 257–267. ACM Press, 1997.

[10] S. Guha, D. Gunopoulos, N. Koudas, D. Srivastava, and M. Vlachos. Efficient approximation of optimization queries under parametric aggregation constraints. In *Proceedings of the 29th international conference on Very large data bases (VLDB)*, pages 778–789. VLDB Endowment, 2003.

[11] B. Hafenrichter and W. Kießling. Optimization of Relational Preference Queries. In *The 16th Australasian Database Conference (ADC)*, pages 175–184, 2005.

[12] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting Top-k Join Queries in Relational Databases. In *Proceedings of the 29th international conference on Very large data bases (VLDB)*, pages 754–765. VLDB Endowment, 2003.

[13] W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

[14] W. Kießling. Preference Queries with SV-Semantics. In *Proceedings of the 11th International Conference on Management of Data (COMAD)*, pages 15–25, 2005.

[15] W. Kießling and B. Hafenrichter. Optimizing Preference Queries for Personalized Web Services. In M. H. Hamza, editor, *Communications, Internet, and Information Technology*, pages 461–466. IASTED/ACTA Press, 2002.

[16] W. Kießling and G. Köstler. Preference SQL - Design, Implementation, Experiences. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 990–1001, 2002.

[17] C. Liu, L. Yang, and I. Foster. Efficient Relational Joins with Arithmetic Constraints on Multiple Attributes. In *Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS)*, pages 210–220. IEEE Computer Society, 2005.

[18] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications, 2002.

[19] S. Nestorov, C. Liu, and I. T. Foster. Efficient Processing of Relational Queries with Sum Constraints. In *APWeb/WAIM*, pages 440–451, 2007.

[20] T. Preisinger and W. Kießling. The Hexagon Algorithm for Evaluating Pareto Preference Queries. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling (VLDB)*, 2007.

[21] T. Preisinger, W. Kießling, and M. Endres. The BNL++ Algorithm for Evaluating Pareto Preference Queries. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling (ECAI)*, 2006.

[22] M. Stonebraker and J. M. Hellerstein. Content integration for e-business. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 552–560. ACM, 2001.

[23] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

[24] USDA. USDA national nutrient database for standard reference. http://www.nal.usda.gov/fnic/, 2007.