The BNL^{++} Algorithm for Evaluating Pareto Preference Queries

Timotheus Preisinger and **Werner Kießling** and **Markus Endres** [1]

Abstract. Deeply personalized database applications require intuitive and powerful preference query languages like Preference SQL, employing preference constructors that are closed under strict partial order semantics. However, sophisticated preference query optimization and efficient evaluation techniques are essential for a large-scale and successful practical use. In this paper we focus on the evaluation of an important class of Pareto preference queries that frequently occur in practice, a subset of which are the well-known skyline queries. Our new algorithm, called BNL^{++}, succeeds in considerably speeding up the usual block-nested loop (BNL) algorithm. In fact, a careful analysis of the underlying 'better-than' graph enables us to identify new and effective pruning conditions. The applicability of BNL^{++} also covers complex situations, where existing index-based evaluation algorithms cannot be used. At this stage BNL^{++} is preliminary work. The next step will be to evaluate the performance of BNL^{++} with a large practical e-commerce use case.

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. [13, 11, 16, 7]).

Preferences are dealt with in different reseach fields, e.g. constraint programming ([8]), AI ([2]), or database technology.

There is a number of different approaches to add preference handling to standard database systems to enable better and deeper personalization ([4]). In this paper, we will use Kießling's approach of modelling preferences as *strict partial orders* ([9, 13, 10]). It contains a variety of intuitive preference constructors for both finite categorical domains and infinite numerical data types. Simple preferences can be combined in different (also recursive) ways suitable for expressing complex user preferences.

A sample preference query in Preference SQL could look like this:

```
SELECT *
FROM used_cars c, features f
WHERE c.id = f.car
PREFERRING (f.color IN ('red', 'black') AND
            c.hp BETWEEN (120, 150, 20) AND
            (f.price LOWEST PRIOR TO
             c.date_made HIGHEST)
```

The query expresses a Pareto preference (`AND`, $\otimes$) containing two base preferences (preferences on simple attributes) and one prioritization (`PRIOR TO`, &) of two other base preferences. `IN` denotes a preference for members of a given set. The whole preference is evaluated on the result of a join. The corresponding operator tree of the preference can be seen in figure 1.

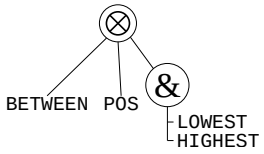

Figure 1. Operator tree of the car preference

Algorithms for evaluating Pareto preferences have been examined in the context of skyline algorithms. They divide in general algorithms suitable for use with arbitrary data ([1, 4]) and index-based algorithms ([15, 19, 18]). Our goal is to find an efficient algorithm for evaluating complex expressions as seen in the code sample. The starting point and the measure of comparison of our considerations is Börzsönyi's algorithm BNL ([1]). Index algorithms apparently cannot be applied in cases of complex queries like the one given here.

The rest of this paper is organized as follows: In the next section we give an overview over our preference model and the evaluation of preferences in a database context. In section 3, we analyze the better-than-graph, a visual representation of preference relations, and establish the basis of our algorithm. In section 4, we present our new algorithm BNL^{++}. Finally, we conclude and give an outline of our next research steps in section 5.

2 BACKGROUND

In this section we will discuss essential backgrounds of preferences in databases. First, we will have a short look at the basic constructors of the preference model we are using. We will then focus on better-than graphs as visual representation of the domination-relations defined by preferences. After this we will address the combination of base preferences to Pareto preferences and their evaluation in databases.

2.1 Preference foundations revisited

Let's start with a look at the preference model we are using. More details and theoretical background information can be found in [9] and [10]. Generally speaking, this preference model is based on strict partial orders. A preference is denoted as $P = (A, <_P)$, where A is a set of attributes and $<_P$ is a strict partial order on the domain of A. Some values are considered to be *better than* some others. Two

[1] University of Augsburg, Germany, email: {preisinger, kiessling, endres}@informatik.uni-augsburg.de

values not ordered by the strict partial order $<_P$ are regarded as *indifferent*. As strict partial orders are transitive, *better-than* relations in this preference model are, too. (Most preference models known in literature use transitive relations, but there are others ([20]).)

A number of base and complex preference constructors makes this approach easily utilizable by average users. We will now see some of them.

2.1.1 Base preference constructors

To express simple preferences like minimum or maximum, we define diverse base preference constructors. Each of these base preference constructors will target only one attribute. There are two distinct types of base preferences, namely *numerical* and *categorical*, depending on the data type of the domain of A. For the scope of this paper we concentrate on base preference constructors that specify *weak orders*, i.e. strict partial orders for which negative transitivity holds. For weak order preferences the better-than test can be specified efficiently by a numerical score function. As a unique feature of our approach, we will study weak preferences that can be characterized by a *level* function which maps a domain value to an *integer* number:

$$level : dom(A) \to \mathbb{N}_0$$
$$x <_P y \iff level_P(x) > level_P(y) \qquad (*)$$

Note that for weak order preferences two domain values x and y with the same level are either equal or *indifferent*, i.e. $\neg(x <_P y) \wedge \neg(y <_P x)$. Two values with the same level in a weak order preference belong to the same *equivalence class*.

For the purpose of this paper we consider the following base preference constructors (more can be found in [9, 10]).

a) *Categorical base preference constructors*

- POS-preference: POS$(A, POS\text{-}set)$
  The user specifies a set of values he favors over all other values. The preferred values are called *POS-set*. The level function expresses this as follows:

$$level_{\text{POS}}(x) := \begin{cases} 0 & \text{iff } x \in POS\text{-}set \\ 1 & \text{iff } x \notin POS\text{-}set \end{cases}$$

- POS/POS-preference: POS/POS$(A, POS_1\text{-}set, POS_2\text{-}set)$
  The user specifies two sets of values. The first one, the *POS*$_1$-set, contains the most preferred values. If no such values can be found, elements of *POS*$_2$ will be chosen. Only if no member of these two sets occurs, all other domain values are selected.

$$level_{\text{POS/POS}}(x) := \begin{cases} 0 & \text{iff } x \in POS_1\text{-}set \\ 1 & \text{iff } x \in POS_2\text{-}set \\ 2 & \text{iff } x \notin (POS_1\text{-}set \cup POS_2\text{-}set) \end{cases}$$

Example 1 *Homer wants to buy a new car. He has a distinct preference for the color. He tells the vendor: "The color should be black or silver. If none of these is available, red is also okay." This can easily be rendered into a POS/POS-preference:*

$$P_1 = \textit{POS/POS}(color, \{\text{'black'}, \text{'silver'}\}, \{\text{'red'}\})$$

So 'black' and 'silver' are better than all other values and indifferent w.r.t. each other. 'Red' is better than any other domain value (except for 'black' and 'silver'). □

b) *Numerical base preference constructors*
These preferences are used to handle numerical attributes. Their original versions have been introduced in [9]. We will use the advanced versions of [10], allowing the partitioning of the range of domain values. For this purpose the so-called *d-parameter* ($d > 0$) is introduced. We will see it in action in the level functions:

$$level_P(x) := \lceil dist(x)/d \rceil$$

The function *dist* has to be defined individually for every type of numerical base preference. It is interpreted as the numerical distance from a perfect value.

- HIGHEST$_d$-preference: HIGHEST$_d(A)$
  The distance function has to map higher inputs to lower function values. The best possible value of course is the maximum value of the domain of A, max.

$$dist_{\text{HIGHEST}}(x) := max - x$$

- LOWEST$_d$-preference: LOWEST$_d(A)$
  LOWEST$_d$ is the dual preference to HIGHEST$_d$. The best possible value is the minimum value of the domain of A, min.

$$dist_{\text{LOWEST}}(x) := x - min$$

- BETWEEN$_d$-preference: BETWEEN$_d(A, [low, up])$
  A user declares an interval in which desired values lie. The best values are in the interval. The greater the distance from the interval of a value is, the worse it is.
  The distance function for a BETWEEN$_d$-preference for the closed interval $[low, up]$ can be defined as follows:

$$dist_{\text{BETWEEN}}(x) := \begin{cases} low - x & \text{iff } x < low \\ 0 & \text{iff } low \le x \le up \\ x - up & \text{iff } x > up \end{cases}$$

Example 2 *Homer would like to buy a car with an engine output between 120 and 150 hp. He does not mind power differences of 20 hp.*

$$P_2 = \textit{BETWEEN}_{20}(hp, [120, 150])$$

A typical car vendor database with cars from 60 to 200 hp would lead to a maximum level value of $\lceil (120 - 60)/20 \rceil = 3$. □

2.1.2 "Better-Than" graph

It is common to visualize strict partial orders by a directed acyclic graph called Hasse diagram ([6]). We introduce its use for our preferences as well, calling it the *"better-than" graph* (BTG) (see also [9]).

Definition 1 *The "better-than" graph (BTG) for a preference* $P = (A, <_P)$ *is defined as follows:*

a) Each equivalence class in the domain of A is represented by one node.
b) A directed edge from a node v_1 to a node v_2 means that v_2 is dominated by (i.e. is worse than) v_1. As domination is transitive, v_1 also dominates all values dominated by v_2.
c) For every node a level value can be defined: $level_P : A \to \mathbb{N}_0$ It is the length of the longest path leading to it.

Note that all nodes that are not dominated by any other node have a level of zero. They are the *top* nodes. Two nodes are indifferent iff there is no directed path between them.

Given a preference $P = (A, <_P)$ and the level function of its associated BTG, it generally holds that:

$$x <_P y \Rightarrow level_P(x) > level_P(y)$$

Remark:

- The definition of a level function for each of the stated base preference constructors and its corresponding level in the associated BTG coincide.
- Since each such constructor specifies a weak order preference, the *reverse implication* of the above statement also holds (see formula (*) in the previous section).

Referring back to example 1, the regular BTG of this POS/POS preference is given in figure 2, where values belonging to one equivalence class are enclosed by '{ ... }'. For base preferences as we have seen, for each node there is exactly one path to the only existing top node. The length of this path defines the level value according to definition 1(c).

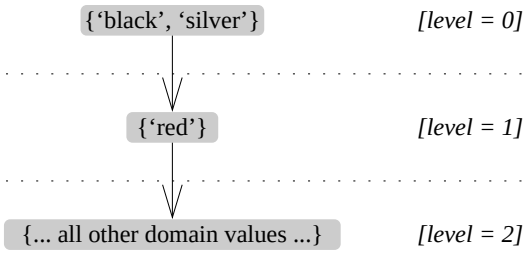

Figure 2. Sample regular BTG for a POS/POS preference

2.1.3 Pareto preference constructor

Base preferences like the ones seen before can be combined to form complex preferences. In this paper our focus will lie on *Pareto preferences*, the combination of equally important preferences $P_1, \ldots, P_m$ ([9, 10]). Importantly, we will restrict our attention to $P_1, \ldots, P_m$ being weak order preferences.

Definition 2 *Let $P_1 = (A_1, <_{P_1}), \ldots, P_m = (A_m, <_{P_m})$ be weak order preferences. A* Pareto preference

$$P = \otimes(P_1, \ldots, P_m) = (A_1 \times \ldots \times A_m, <_P)$$

is defined as:

$$(x_1, \ldots, x_m) <_P (y_1, \ldots, y_m) \text{ iff}$$
$$\exists i : x_i <_{P_i} y_i \wedge (\forall\, i, j \in \{1, \ldots, m\} \wedge j \neq i : (x_j <_{P_j} y_j \vee x_j \cong_{P_j} y_j))$$

In case of $m = 2$ we will adopt infix notation, i.e. $P = P_1 \otimes P_2$.

In general, the relation "$\cong$" is interpreted as "*is substitutable to*" or as "*is equally good as*", leading to the notion of *SV-semantics* ([10]). Given this SV-semantics for Pareto preferences, there is a degree of freedom on how to choose a valid SV-relation "$\cong$". As also shown in [10], for weak orders "$\cong$" can be chosen as the *indifference* relation. For the scope of this paper we will adopt this choice of "$\cong$". Then domain values are substitutable iff their level functions have the same values. Same values belong to the same *equivalence class*.

Note that not only the base preferences presented in section 2.1.1 are weak orders. There are complex preference constructors that preserve weak orders like *prioritization*s over weak order preferences ([4]) or $rank_F$ (see [9]). The example presented in the introduction included such a prioritization.

Now let's look at some Pareto preference example.

Example 3 *We have already seen Homer's preferences for cars in examples 1 and 2. If both preferences are equally important to him, we can combine them to a Pareto preference:*

$$P := \textit{POS/POS}(\textit{color}, \{\text{'black'}, \text{'silver'}\}, \{\text{'red'}\}) \otimes \textit{BETWEEN}_{20}(\textit{hp}, [120, 150])$$

For the following database excerpt, we can easily determine which tuples dominate others.

ID	*color*	*hp*	$level_{P_1}$	$level_{P_2}$
t_1	*yellow*	*120*	*2*	*0*
t_2	*red*	*95*	*1*	*2*
t_3	*silver*	*150*	*0*	*0*
t_4	*blue*	*60*	*2*	*3*
t_5	*black*	*130*	*0*	*0*
t_6	*silver*	*70*	*0*	*3*

- *t_1, t_2 and t_6 are indifferent.*
- *t_3 and t_5 show top values. They show perfect values for both base preference constructors. All other tuples are worse than these two. As both of their attributes are substitutable, they belong to one equivalence class.*
- *t_4 is worse than all other tuples. It shows higher level values than all others for both base preference constructors.*
- *t_6 is worse than t_5 as their colors are substitutable and t_6's hp value is worse.* □

2.2 Pareto preference queries revisited

Having revisited our preference model, now let's have a look at the basic concepts of declarative database preference queries. Thereafter the focus will lie on efficient evaluation algorithms for such queries.

2.2.1 BMO query model

In standard database query languages like SQL or XPath, a user states *hard constraints* that have to be satisfied. If no database entry fulfills all hard constraints, the result set is empty.

When using preferences in a database query, a user indicates which attribute values he prefers. The BMO query model then guarantees *best-matches only* (BMO) to be in the result set of a preference query P on a given database relation R. If all preferences can be matched exactly by some tuples, these form the result set. Using only hard constraints would lead to an identical result. If not all preferences can be matched exactly, all tuples not worse than other tuples are returned. Hard constraints would produce an empty result in this case.

Definition 3 *The* best-matches only *(BMO) result set for a preference $P = (A, <_P)$ on a database relation R is defined as:*

$$\sigma_P(R) := \{t \in R \mid \nexists\, t' \in R : t[A] <_P t'[A]\}$$

This BMO semantics is also used by Chomicki's *winnow* operator ([4]). Skyline queries described in [1] are a special case of this BMO approach: Basically, they only allow HIGHEST and LOWEST base preference constructors to participate in a Pareto preference ([9]). Therefore all results of this paper apply to skyline queries as well.

The BMO query model has been implemented so far in two query languages, Preference SQL ([13]) and Preference XPath ([12]).

Example 4 *The Pareto preference seen in example 3 can be turned right away into a Preference SQL query:*

```
SELECT *
FROM used_cars
PREFERRING color IN ('silver', 'black')
                ELSE ('red') AND
           hp BETWEEN (120, 150, 20);
```

Note that '`AND`' is the key word for Pareto combination, '`IN`' ... '`ELSE`' encodes the POS/POS preference. □

2.2.2 Evaluation of Pareto preference queries

The evaluation of Pareto preference queries has been brought into database context by Börzsönyi et al. when introducing the *skyline* operator in [1]. Basically, they proposed two different ways of evaluating such queries: index-based and nested-loop algorithms. We will have a look at both types now.

Börzsönyi already dealt with index use and addressed the major problem of index-based algorithms: the lack of applicability for computations with joined relations. Despite this lack of generality, index-based algorithms have been a center of interest in the last few years. Algorithms developed deal with different kinds of indices, may it be e.g. multiple B-trees in the *Index* algorithm ([19]) or a single R-tree for *NN* or *BBS* ([15, 18]).

Nested-loop algorithms are the well-known trivial way of computing a skyline ([17]). Despite the lower performance compared to index algorithms, they are capable of processing arbitrary data without any preparations to be done. Roughly speaking, every tuple is compared to each other, always discarding tuples that are dominated by another tuple. After processing all tuples, the remaining ones form the skyline. The most common nested-loop algorithm is *block-nested-loop* (BNL), introduced in [1]. It allows to use a nested loop with memory too small to store the whole list of skyline candidates (during the computation) respectively skyline members (after the computation).

Though BNL was originally designed to evaluate Pareto preferences over total orders (represented by the operations MIN and MAX on numerical data), it can be applied to evaluate more general Pareto preferences over weak orders, too.

As far as we know, the last major improvement to BNL has been made by Chomicki et al. in *Sort-Filter-Skyline* (SFS, [5]). This algorithm is based on a preliminary sorting of all input tuples. A scoring function $\sum_{i=1}^{k} ln(t[a_i]+1)$ defines a topological order over all input tuples. It sums up values of a monotone function for all attributes a_i of a tuple t used in the skyline computation. It is shown that a tuple can only be dominated by tuples with a lower scoring function value.

Since the BNL algorithm basically addresses the problem of Pareto preferences over weak orders, it was generalized by [3] to an algorithm called *BNL*$^+$, dealing with Pareto preferences over general strict partial orders. In this paper we shall address the same issue as BNL, focussing on Pareto preferences over weak partial orders, which cover a wide range of practical use cases.

3 ANALYSIS OF THE REGULAR BTG

We have already seen a sample regular BTG as visualization for a base preference in figure 2. Regular BTGs show some interesting properties which we will look at now. Thereafter we identify new pruning conditions that will form the basis of our new Pareto evaluation algorithm.

3.1 Properties of regular BTGs

Let's first figure out the maximal height of a regular BTG for a Pareto preference P.

Definition 4 *The maximum level value for a base preference P_i is called* $\max(P_i)$.

The computation of the maximum level is easy for categorical base preferences. As there is a fix number of different sets all domain values belong to, the number of possible different level values is defined by the preference itself: one set makes up one equivalence class. All equivalence classes have different level values and all values from 0 to the maximum exist.

For a numerical preference P_i, level values depend on the distance function. So we have to look at $dist$ to find a maximum level value. As we are trying to find a distance from a given value or interval, the value clearly is highest for one of the extrema of the domain:

$$\max(P_i) = \max(level_d(min), level_d(max))$$

Example 5 *Once again, we will look at Homer's car preferences. We can now compute the maximum level value for his two base preferences $P_1 = POS/POS(color, \{\text{'black'}, \text{'silver'}\}, \{\text{'red'}\}))$ and $P_2 = BETWEEN_{20}(hp, [120, 150]))$.*

$$\begin{aligned} \max(P_1) &= 2 \\ \max(P_2) &= \max(level_{P_2}(60), level_{P_2}(200)) \\ &= \max(\lceil (120-60)/20 \rceil, \lceil (200-150)/20 \rceil) \\ &= 3 \end{aligned}$$ □

The next step is to compute a level value for a Pareto preference P. Of course, the properties stated in definition 1 must hold for this level value.

Theorem 1 *The level function for the BTG of a Pareto preference $P := \otimes(P_1, \ldots, P_m)$ can be computed as follows:*

$$level_P(a_1, \ldots, a_m) = \sum_{i=1}^{m} level_{P_i}(a_i)$$

Proof:
For the purpose of this proof, we will abbreviate $level_{P_i}(a_i)$ by $l_{P_i}(a_i)$. Since each P_i is weak order, better-than tests can be decided by comparing level function values. Doing so, instead of $(a_1, ..., a_m)$ we consider $(l_{P_1}(a_1), \ldots, l_{P_m}(a_m))$. Then obviously $v_0 = (0, \ldots, 0)$ characterizes the only top node of the BTG of P at level 0. Flipping exactly one of the 0's into a 1, let's consider $v_1 = (0, \ldots, 0, 1, 0, \ldots, 0)$. According to the Pareto definition of P we have $v_1 <_P v_0$. Since $level$ is an integer-valued function, v_1 characterizes a direct successor of v_0 in the BTG of P.

Thus if $(\overline{a}_1, \ldots, \overline{a}_m)$ represents a value with level vector v_1 and $(a_1, \ldots, a_m)$ represents a value with level vector v_0, then
$l_P(\overline{a}_1, \ldots, \overline{a}_m) = l_P(a_1, \ldots, a_m) + 1 = 0 + 1 = 1.$

In general, consider
$v_n = (l_{P_1}(a_1), \ldots, l_{P_m}(a_m))$ and
$v_{n+1} = (l_{P_1}(a_1), \ldots, l_{P_{i-1}}(a_{i-1}), l_{P_i}(a_i) + 1,$
$l_{P_{i+1}}(a_{i+1}), \ldots, l_{P_m}(a_m))$,
differing from v_n by increasing exactly one level from $l_{P_i}(a_i)$ to $l_{P_i}(a_i) + 1$. Assuming that v_n represents a node at level
$n = \sum_{i=1}^{m} l_{P_i}(a_i)$
in the BTG of P, then v_{n+1} represents a direct successor of v_n at level
$n + 1 = \sum_{i=1}^{m} l_{P_i}(a_i) + 1$.
Thus by induction over the height of the BTG of P the proof is achieved. □

Note that since a Pareto preference P itself is not a weak order ([4, 10]), it only holds that: $x <_P y \Rightarrow level_P(x) > level_P(y)$, but not vice versa. Surprisingly, as we will show subsequently, the knowledge of the level function for P will allow some conclusion about the preference order, i.e., in certain situations also the reverse implication holds.

We will first have a look at some interesting characteristics of the BTG of P.

Theorem 2 *Given a Pareto preference $P = \otimes(P_1, \ldots, P_m)$, its regular BTG shows the following characteristics.*

a) Height of the BTG:

$$height(BTG) = 1 + \sum_{i=1}^{m} \max(P_i)$$

b) Number of different nodes of the BTG:

$$nodes(BTG) = \prod_{i=1}^{m} (\max(P_i) + 1)$$

c) Width of the BTG at a specific level k, $width(BTG, k)$:

$$width(BTG, k) \quad = \quad w(k, \{P_1, \ldots, P_m\}), \text{ where}$$

$$w(k, \{P_1, \ldots, P_m\}) = \sum_{i=0}^{\min(k, \max(P_1))} w(k - i, \{P_2, \ldots, P_m\}), \text{ if } m > 1$$

$$w(k, \{P_m\}) \quad = \quad \begin{cases} 0, & \text{if } \max(P_m) < k \\ 1, & \text{if } \max(P_m) \geq k \end{cases}$$

Proof:

a) The lowest possible level of a node is 0. The highest level as sum of maximum level values of the $P_1, \ldots, P_m$ follows from theorem 1. All levels between 0 and the maximum must exist.
b) Every combination of different level values of the weak order preferences is possible. Note that each of them can take values from 0 to $\max(P_i)$, leading to $(\max(P_i) + 1)$ different values.
c) We want to find the width of a level k in the regular BTG, which is identical to the number of equivalence classes of the underlying preference $P = \otimes(P_1, \ldots, P_m)$ belonging to level k. P's equivalence classes are characterized by the level values of the weak order preferences $P_1, \ldots, P_m$. For an equivalence class in a level k, the sum of all contained level values is k (see theorem 1).
Of this sum k, a value of 0 to $\max(P_1)$ can be contributed by the level value for P_1.The rest has to be contributed by the other weak order preferences' level values.

So we sum up all possible cases:

- 0 from P_1, k from $P_2, \ldots, P_m$,
- 1 from P_1, $k - 1$ from the rest,
- ...
- $\max(P_1)$ from P_1, $k - \max(P_1)$ from the rest.

Of course, if $k < \max(P_1)$, we only have to distribute 0 to k to P_1, so the minimum of k and $\max(P_1)$ forms the upper border of the sum's index in the formula.
The distribution of the remaining value $(k - \ldots)$ of course is done recursively. The recursion terminates when there is only one weak order preference left. If the value to be contributed by this P_i is higher than $\max(P_i)$, this cannot be a possible distribution. Otherwise, there is exactly one way.
Instead of P_1, every other P_i could be chosen as well.
Note that the number of nodes with the same overall level is equal to the extension of the binominal distribution known as *Vandermonde's identity* from 2 to m distinguishable sets ([14]). [2] □

Having obtained a precise idea about the shape of the regular BTG of a Pareto preference P, from now on we will study a variation of it, called regular *BTG**. The difference of a BTG* to our original BTG is that nodes are now represented by level vectors instead of attributes.

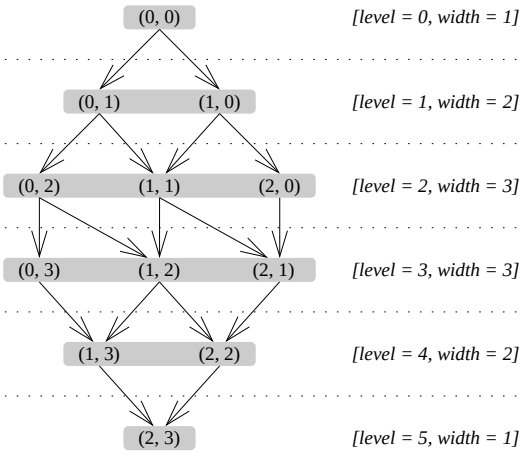

Figure 3. Sample regular BTG* for a Pareto preference

Now let's reconsider the Pareto preference from example 3. Its associated regular BTG* is depicted in figure 3.

The Pareto preference in figure 3 is $P = POS/POS \otimes BETWEEN_d$. Applying the results of example 5 to theorem 2 yields:

$$height(BTG^*) \quad = \quad 1 + (2 + 3) = 6$$
$$nodes(BTG^*) \quad = \quad (2 + 1) * (3 + 1) = 12$$

Widths of this BTG* are given in figure 3 itself. Also, let's have a look at e.g. node $(1, 2)$ at level 3: This level vector represents the equivalence class $\{\{'red'\}, [80; 100[\cup]170; 190]\}$ for P.

[2] We were not able to find a closed formula for the BTG width yet. Also, the types of reducing recursion depth we have tried, e.g. a divide-and-conquer approach of splitting the set of preferences in two or more sets with more than one element, have failed up to now.

The *hexagonal diamond-shape* of the BTG* in figure 3 is typical: At the top, there is only one node. Going down, there is an increasing number of ways to distribute the level value on the contained preferences until a level of satiation is reached. Starting at the bottom, there also is one node. Going up, there is an increasing number of ways to distribute the "missing" amount of the current level to the maximum level. There are equally many ways to distribute a level value x or the missing value x, i.e. $width(BTG^*, k) = width(BTG^*, height(BTG^*) - k)$. In the middle, there may be more levels with the same width.

3.2 Pruning conditions

Knowing the exact shape of the BTG* will give us new possibilities of enhancing query performance by improved result set pruning. In fact, for each node in the BTG* we will be able to identify a so-called *pruning level*.

Definition 5 *Given a Pareto preference $P = \otimes(P_1, \ldots, P_m)$, the* pruning level pl_P *of a value $a = (a_1, \ldots, a_m)$ is defined as follows:*

$$\begin{aligned} &pl_P : \mathbb{N}_0 \times \ldots \mathbb{N}_0 \to \mathbb{N}_0 \\ &\forall\, \overline{a} = (\overline{a}_1, \ldots, \overline{a}_m) : \\ &\qquad level_P(\overline{a}) \geq pl_P(level_{P_1}(a_1), \ldots, level_{P_m}(a_m)), \\ &\qquad \Longrightarrow \overline{a} <_P a \end{aligned}$$

Thus $pl_P(level_{P_1}(a_1), \ldots, level_{P_m}(a_m))$ is the minimum level for which all equivalence classes belonging to this level are dominated by a.

Theorem 3 *Consider a Pareto preference $P = \otimes(P_1, \ldots, P_m)$. Then a value $a = (a_1, \ldots, a_m)$ has the following pruning level:*

a) $pl_P(level_{P_1}(a_1), \ldots, level_{P_m}(a_m)) = 1$
iff a is the top node.

b) $pl_P(level_{P_1}(a_1), \ldots, level_{P_m}(a_m)) = height(BTG)$
iff a is the worst node.

c) otherwise:

$$\begin{aligned} &pl_P(level_{P_1}(a_1), \ldots, level_{P_m}(a_m)) = \\ &\quad \textstyle\sum_{j=1}^{m} \max(P_j) - \min(\{\max(P_i) - level_{P_i}(a_i) | \\ &\qquad\qquad 1 \leq i \leq m \wedge level_{P_i}(a_i) > 0\}) \end{aligned}$$

Proof:

a) All level values of the top node are zero. According to the Pareto preference, it dominates all other nodes in the BTG.

b) It is clear that a worst node cannot prune nodes from the BTG as it does not dominate any.

c) Once again, we will abbreviate $level_{P_i}(a_i)$ by $l_{P_i}(a_i)$. We have a value a with level values $(l_{P_1}(a_1), \ldots, l_{P_m}(a_m))$ for a preference $P = \otimes(P_1, \ldots, P_m)$. We select a value $\overline{a} = (\overline{a}_1, \ldots, \overline{a}_m)$ not dominated by a and with a level as high as possible. So there must be one attribute of $\overline{a}$ with $l_{P_i}(\overline{a}_i) < l_{P_i}(a_i)$. The maximum value for this $l_{P_i}(\overline{a}_i)$ surely is $l_{P_i}(a_i) - 1$. The other attributes' level values should be as high as possible to maximize the overall level. That means, the vector $[\,\overline{a}\,]$ of $\overline{a}$'s level values is one of those:

$$\begin{aligned} [\,\overline{a}^{(1)}\,] &:= (l_{P_1}(a_1) - 1, \max(P_2), \max(P_3), \ldots, \max(P_m)) \\ [\,\overline{a}^{(2)}\,] &:= (\max(P_1), l_{P_2}(a_2) - 1, \max(P_3), \ldots, \max(P_m)) \\ &\vdots \qquad\quad \vdots \qquad\qquad\qquad\qquad \vdots \\ [\,\overline{a}^{(m)}\,] &:= (\max(P_1), \max(P_2), \max(P_3), \ldots, l_{P_m}(a_m) - 1) \end{aligned}$$

For any $l_{P_i}(a_i) = 0$, the corresponding $[\overline{a}^{(i)}]$ clearly is not valid as level values smaller than 0 cannot occur. So no related $\overline{a}_i$ can exist. We will not regard them furthermore.
The overall levels of the different $\overline{a}_i$ are:

$$level_P(\overline{a}^{(i)}) = \sum_{j=1}^{m} \max(P_j) - \max(P_i) + l_{P_i}(a_i) - 1$$

The node representing $\overline{a}^{(i)}$ itself only dominates one node in the next higher level: the one with a value of $l_{P_i}(a_i)$ for preference P_i and maximum values for all other preferences. So any value dominated by one of the $\overline{a}^{(i)}$ is also dominated by a. That means, when we find the highest level of a $\overline{a}^{(i)}$, we will find the highest overall level a value can have without being dominated by a.
In the algorithm, the sum of maximum level values is fix, so the result is maximized by minimizing the subtrahend, $\max(P_i) - l_{P_i}(a_i)$.
The level computed is the highest overall level for which not all values belonging to it are dominated by a. By adding 1, we receive the formula introduced in theorem 3. In this level, all values must be dominated by a. If all values having one specific overall level are dominated, of course all values with higher overall levels are dominated, too.
There are two cases when no $\overline{a}^{(i)}$ indifferent to a can be found:
$l_{P_i}(a_i) = 0$ (which we have already excluded) and
$\forall\, i \in \{1, \ldots, m\} : l_{P_i}(a_i) = \max(P_i)$.
If all level values of a are maximal, a is a worst node, so part (b) of the theorem holds. There cannot be any worse nodes which could be pruned. Note if there is only one $l_{P_i}(a_i) < \max(P_i)$, the corresponding $\overline{a}^{(i)}$ is better than a as it has level values in the form of $(l_{P_1}(a_1), ..., l_{P_i}(a_i) - 1, ..., l_{P_m}(a_m))$. This case is covered by part (c) as there is at least one $l_{P_j}(a_j) = \max(P_j)$. This $l_{P_j}(a_j)$ leads to a subtrahend of zero, which is the minimal possible value. So $l_{P_i}(a_i)$ will not be regarded. □

Example 6 *Revisiting our regular BTG* from figure 3, for example we can state the following pruning levels:*

$$\begin{aligned} pl_P(0,1) &= 5 - min(\{3-1\}) = 3 \\ pl_P(1,1) &= 5 - min(\{2-1, 3-1\}) = 5 - 1 = 4 \end{aligned}$$

□

Note that nodes at the same level can have different pruning levels. This is demonstrated by the next example, which also illustrates that BTGs with very large number of levels can possess effect pruning possibilities.

Example 7 *Let's assume a Pareto preference P over two numerical base preferences with a maximum level value of 10000 each and a POS preference, hence $\max(P) = 20001$. Let's e.g. consider the following level vectors at level 10000 and their pruning levels:*

$$\begin{aligned} pl_P(5000, 5000, 0) &= 20001 - 5000 = 15001 \\ pl_P(5000, 4999, 1) &= 20001 - 0 = 20001 \\ pl_P(4000, 6000, 0) &= 20001 - 4000 = 16001 \end{aligned}$$

Although all tuples have the same level, their pruning levels are quite different. □

4 THE NEW BNL^{++} ALGORITHM

We will now use the results made in the preliminary sections to enhance BNL computation of Pareto preference queries. The algorithm uses two different methods to identify and omit useless comparisons:

- optimization of better-than-tests by pruning the BTG
- omission of indifference tests for nodes with the same overall level

In the algorithm, $pruning\_lvl$ is the current pruning level, and $cand\_list(x)$ means the set containing all nodes with overall level x.

All equivalence classes with the same overall level are indifferent w.r.t. each other (this follows from the proof of theorem 1). We do not need to compare them with each other. It is also possible to integrate a *killer tuple heuristic* as indicated in [1]. Starting with an unsorted list of nodes for one overall level, a node is moved to the beginning of the list when it dominates an input tuple. As a consequence, killer tuples will gather at the beginning of the list.

If the node of a tuple already exists, no comparisons have to be made. Tuples belonging to it only have to be added. Using a sophisticated storage strategy, a tuple can be allocated to its equivalence class in memory in linear time. Although not directly mentioned in the algorithm, this can also be used for discarded nodes: A note of them being dominated has to be stored. So all other tuples belonging to them can be discarded at once.

More has to be done for preliminary unused nodes. If they can be pruned away, once more no comparisons are needed (see algorithm line 11). Otherwise, we have to look if the new node changes the pruning level. If so, all BMO candidates belonging to now pruned levels have to be removed.

Now possibly existing better nodes have to be found. As BTG nodes with lower overall levels more likely dominate other nodes, we start comparisons with the equivalence classes with lowest overall level. Then we go up step-by-step until we reach the next-lower level of the new node.

As soon as the new node's level is reached it is clear that it belongs to the candidate list (at least for now), as no node can be dominated by one with a higher overall level. Now only worse nodes have to be found and removed from the candidate list.

We will now see the algorithm in action in an example.

Example 8 *We want to evaluate Homer's preference for cars known from the previous examples. It is a Pareto preference P consisting of a POS/POS preference for the color (P_1) and a BETWEEN preference for the engine's hp (P_2). The maximum values of P_1 and P_2 are 2 and 3, so $height(BTG) = 6$.*

We will use the following sample database for our test. The level values and the pruning level of course will not be stored in a database but be computed on the fly when reading a tuple.

ID	color	hp	$level_{P_1}$	$level_{P_2}$	$level_P$	pl_P
t_1	green	110	2	1	3	5
t_2	white	160	2	1	3	5
t_3	yellow	140	2	0	2	5
t_4	blue	60	2	3	5	5
t_5	red	200	1	3	4	5
t_6	black	70	0	3	3	5
t_7	silver	100	0	1	1	3
t_8	red	180	1	2	3	4

We start with an empty candidate list. The tuples will be added in order of their ID.

Algorithm 1 BNL^{++}

Data structures:
- a BTG node in the BMO candidate set holds a list of tuples belonging to it.
- BTG nodes in level i are kept in the list $cand\_list(i)$
- $cand$ is a set of all $cand\_list(i)$ belonging to the BMO candidates

Input: Relation R, Pareto preference P
Output: BMO set

1. $pruning\_lvl \leftarrow$ height(BTG)
2. $cand \leftarrow \emptyset$

3. **for all** $t \in R$ **do**
4. $node \leftarrow$ compute the BTG node that represents t
5. $lvl \leftarrow level(t)$

// check for existing BTG node
6. **if** $cand\_list(lvl) \in cand \wedge node \in cand\_list(lvl)$ **then**
7. add t to $node$
8. goto line 3
9. **end if**

// check if t's level is pruned
10. **if** $lvl \geq pruning\_lvl$ **then**
11. goto line 3 // discard t and start next loop cycle
12. **end if**

// check if better pruning level is found and prune $cand$
13. **if** $pl(node) < pruning\_lvl$ **then**
14. **for** $i = pl(node)$ to $pruning\_lvl$ **do**
15. $cand \leftarrow cand \setminus cand\_list(i)$
16. **end for**
17. pruning_lvl $\leftarrow$ pl(node)
18. **end if**

// check for better nodes (which always have lower levels)
19. **for all** $e \in cand\_list(i), 0 \leq i \leq lvl - 1$ **do**
20. **if** $node <_P e$ **then**
21. goto line 3 // discard t and start next loop cycle
22. **end if**
23. **end for**

// add t to candidate list
24. **if** $cand\_list(lvl) \notin cand$ **then**
25. $cand\_list(lvl) \leftarrow \emptyset$ // initialize new level list
26. $cand \leftarrow cand \cup cand\_list$
27. **end if**
28. $cand\_list(lvl) \leftarrow cand\_list(lvl) \cup node$

// check for worse nodes
29. **for all** $e \in cand\_list(i), lvl + 1 \leq i \leq pruning\_lvl$ **do**
30. **if** $e <_P node$ **then**
31. $cand\_list(lvl) \leftarrow cand\_list(lvl) \setminus e$
32. **end if**
33. **end for**

34. **end for**

t_1 *is added to the candidate list. The pruning level is set to 5.*
t_2 *is simply added to its already existing node.*
t_3 *is – for lack of existance – not compared to tuples with better levels. (lines 19-23). After being added to the candidate list (line 28), worse nodes are searched – and found:* $(2, 1)$ *(and so* t_1 *and* t_2*) is removed.*
t_4 *is dismissed as its level value is equal to the pruning level.*
t_5 *is added to the list of nodes with level 4. Before it was checked that it is indifferent to the only node in the candidate list:* $(2, 0)$.
t_6 *is added to the list of nodes with level 3 after being compared with* $(2, 0)$*. Then, it dominates and dismisses* $(3, 1)$.
t_7 *lowers the pruning level to 3. All nodes with this level or higher ones are removed from the candidate list (lines 14-16). It is added to level 1's nodes.* t_7 *does not dominate* $(2, 0)$.
t_8 *is discarded as its level is equal to the pruning level.*

In this scenario, only 5 comparisons have to be made. The simple BNL from [1] would have needed 11. □

A combination with presorting in any topological order of course is also possible. The only condition is that a tuple once in the candidate list cannot be dominated by a subsequent tuple. The algorithm could be changed so that the loop needs less computational effort. The removal of dominated nodes (lines 29 to 33) would be obsolete. The algorithm would terminate as soon as a tuple with an overall level equal to the pruning level is found.

5 CONCLUSION & OUTLOOK

We have presented a new algorithm called BNL^{++} for evaluating Pareto preference queries, which frequently occur in personalized database applications. BNL^{++} evaluates a Pareto preference query, constructed from arbitrary weak order preferences. The key factor leading to BNL^{++} has been a careful analysis of the underlying 'better-than' graph, yielding novel pruning conditions. It has to be pointed out that the range of applicability is not merely restricted the Pareto preferences constructed over special base preferences like e.g. POS, BETWEEN, HIGHEST, etc. as demonstrated in this paper. In fact, it covers also more complex cases like e.g. weak orders that are constructed by prioritization over weak orders ([4]) or weak orders generated by multidimensional numerical ranking (cmp. the $rank_F$ constructor, [9]). In such situations index-based evaluation algorithms cannot be used, hence BNL^{++} is the best choice available.

The key for BNL^{++} to succeed in finding effective pruning conditions is the existence of an *integer-valued* level function. This requirement is achieved for numerical base preference constructors by so-called d-parameters. By choosing d sufficiently small one can simulate situations where no d-parameters are wanted. Thus considering integer-valued level functions for BNL^{++} does not impose practical restriction (see also [5]).

By first evidence of a preliminary performance benchmark, taken from a complex practical B2B use case ([7]), our BNL^{++} can significantly improve the runtime efficiency compared to existing BNL-type algorithms that originated in the context of skyline queries. As next steps we plan to integrate a highly tuned implementation of BNL^{++} into our Preference XPath query engine and to systematically evaluate the performance of BNL^{++}.

REFERENCES

[1] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker, 'The Skyline Operator.', in *ICDE*, pp. 421–430. IEEE Computer Society, (2001).
[2] Ronen I. Brafman and Doron A. Friedman, 'Adaptive rich media presentations via preference-based constrained optimization.', in *Preferences*, eds., Gianni Bosi, Ronen I. Brafman, Jan Chomicki, and Werner Kießling, volume 04271 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, (2004).
[3] Chee Yong Chan, Pin-Kwang Eng, and Kian-Lee Tan, 'Efficient Processing of Skyline Queries with Partially-ordered Domains.', in *ICDE*, pp. 190–191. IEEE Computer Society, (2005).
[4] Jan Chomicki, 'Preference formulas in relational queries.', *ACM Trans. Database Syst.*, **28**(4), 427–466, (2003).
[5] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang, 'Skyline with Presorting.', in *ICDE*, eds., Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, pp. 717–816. IEEE Computer Society, (2003).
[6] Brian A. Davey and Hilary A. Priestley, *Introduction to lattices and order*, Cambridge University Press, Cambridge, UK, 2nd edn., 2002.
[7] Sven Döring, Stefan Fischer, Werner Kießling, and Timotheus Preisinger, 'Evaluation and Optimization of the Catalog Search Process of E-Procurement Platforms', *Journal of Electronic Commerce Research and Applications*, **5**(1), 44–56, (2006).
[8] Ulrich Junker, 'Preference-based inconsistency proving: When the failure of the best is sufficient', in *Multidisciplinary IJCAI-05 Workshop on Advances in Preference Handling*, pp. 106–111, (2005).
[9] Werner Kießling, 'Foundations of Preferences in Database Systems.', in *VLDB*, pp. 311–322, (2002).
[10] Werner Kießling, 'Preference Queries with SV-Semantics.', in *COMAD*, eds., Jayant R. Haritsa and T. M. Vijayaraman, pp. 15–26. Computer Society of India, (2005).
[11] Werner Kießling, Stefan Fischer 0003, and Sven Döring, 'COSIMA B2B - Sales Automation for E-Procurement.', in *CEC*, pp. 59–68. IEEE Computer Society, (2004).
[12] Werner Kießling, Bernd Hafenrichter, Stefan Fischer, and Stefan Holland, 'Preference XPath: A Query Language for E-Commerce', in *5th Int. Conf. on Wirtschaftsinformatik – Information Age Economy, Augsburg, Germany*, eds., Hans U. Buhl, Andreas Huther, and Bernd Reitweisner, pp. 427–440, Heidelberg, (2001). Physica-Verlag.
[13] Werner Kießling and Gerhard Köstler, 'Preference SQL - Design, Implementation, Experiences.', in *VLDB*, pp. 990–1001, (2002).
[14] Wolfram Koepf, *Hypergeometric Summation: An Algorithmic Approach to Hypergeometric Summation and Special Function Identities*, Advanced Lectures in Mathematics, Vieweg, Braunschweig, Germany, 1st edn., 1998.
[15] Donald Kossmann, Frank Ramsak, and Steffen Rost, 'Shooting Stars in the Sky: An Online Algorithm for Skyline Queries.', in *VLDB*, pp. 275–286, (2002).
[16] Georgia Koutrika and Yannis E. Ioannidis, 'Constrained Optimalities in Query Personalization.', in *SIGMOD Conference*, ed., Fatma Ozcan, pp. 73–84. ACM, (2005).
[17] H. T. Kung, Fabrizio Luccio, and Franco P. Preparata, 'On Finding the Maxima of a Set of Vectors.', *J. ACM*, **22**(4), 469–476, (1975).
[18] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger, 'An Optimal and Progressive Algorithm for Skyline Queries.', in *SIGMOD Conference*, eds., Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, pp. 467–478. ACM, (2003).
[19] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi, 'Efficient Progressive Skyline Computation.', in *VLDB*, eds., Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, pp. 301–310. Morgan Kaufmann, (2001).
[20] Riccardo Torlone and Paolo Ciaccia, 'Finding the Best when it's a Matter of Preference.', in *SEBD*, pp. 347–360, (2002).