Universität Augsburg

Preference Structures and their Lattice Representations

M. Endres **T. Preisinger**

Report 2016-02 March 2016

Institut für Informatik
D-86135 Augsburg

Preference Structures and their Lattice Representations

Markus Endres[a] and Timotheus Preisinger[b]

[a]University of Augsburg, Germany, endres@informatik.uni-augsburg.de
[b]DEVnet Holding GmbH, Germany, t.preisinger@devnet.de

Abstract

Preferences are an important natural concept in real life and are well-known in the database and artificial intelligence community. The integration of preference queries in database systems enables satisfying search results by delivering best matches, even when no object in a dataset fulfills all preferences perfectly. Skyline queries are the most prominent representatives of preferences queries. The target is to select those tuples from a dataset that are optimal with respect to a set of designated preference attributes. But users do not only think of finding the Pareto frontier, they often want to find the best objects concerning an *explicit specified preference order*. While preferences themselves often are defined as *general strict partial orders*, almost all algorithms are designed to evaluate Pareto preferences combining weak orders, i.e., Skylines. In this paper, we consider general strict partial orders and we present a method to evaluate such explicit preferences by embedding any strict partial order into a lattice. This enables preference evaluation with specialized lattice based algorithms.

1 Introduction

The Skyline operator [2] has emerged as an important and popular technique for searching the best objects in multi-dimensional datasets. A Skyline query selects those objects from a dataset D that are not dominated by any others. An object p having d attributes (dimensions) dominates an object q, if p is strictly better than q in at least one dimension and not worse than q in all other dimensions, for a defined comparison function.

An example for a Skyline query is the search for a car that is *cheap* and has *high horse power* (hp). Unfortunately, these two goals are complementary as cars with high power tend to be more expensive, cp. Table 1. The *Skyline* consists of all cars that are not worse than any other car in both dimensions. In our example these are the cars with id $\in \{3, 4, 7\}$.

Table 1: Sample dataset of cars.

car	**id**	**make**	**color**	**price**	**hp**
	1	Ford	black	70K	180
	2	Mercedes	purple	75K	200
	3	BMW	red	50K	230
	4	Audi	blue	45K	170
	5	Mercedes	cyan	55K	190
	6	GMC	yellow	70K	150
	7	BMW	green	48K	220

Skyline queries and the more general concept of *preference* database queries have been subject to research for more than one decade [38]. Preferences enable users to specify a pattern that describes the type of information he is searching for. Since preferences express *soft constraints*, the most similar data will be returned when no data exactly matches that pattern. From this point of view, preference database queries are an effective method to reduce very large datasets to a small set of highly relevant tuples that are optimal compromises for the user.

In many approaches, preferences are modeled as *strict partial orders* (SPO), and therefore *transitivity* holds, cf. e.g., [22, 5]. When evaluating a preference P on a dataset D, the tuples in D that are not dominated by any other tuple in D w.r.t. P are called the *maximal values* or *the Skyline* in the case of a Pareto preference query. The objective of a preference query is to find the tuple(s) in a dataset that are maximal with respect to a given set of preferences.

Example 1. *Figure 1 expresses a simple user preference on the domain of colors* $\mathrm{dom}(color) = \{$*red, blue, green, yellow, purple, black, cyan*$\}$ *when searching for a car in Table 1. The colors* red*,* blue*, and* green *are preferred over* yellow *and* purple*, which are better than* black *and* cyan*. Thereby, all colors in the same set should be considered as equally good and as* substitutable *(we refer to this as* regular Substitutable Values (SV) semantics *later on in this paper [23]).*

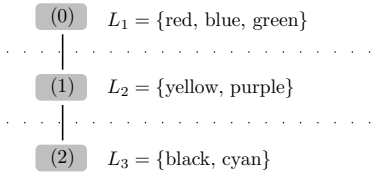


Figure 1: Preference on colors.

A *Better-Than Graph* (BTG, also known as Hasse diagram) is a visual representation of the domination of domain elements for a preference as can be seen in Figure 1. The *nodes* in this BTG represent *equivalence classes*. Each equivalence class contains objects which are mapped to the same level by a *utility function*. The *edges* in the BTG state dominance.

The preference order in Figure 1 forms a weak order, because all values in the same equivalence class are considered as substitutable and for each equivalence class we can specify a numerical value which represents a node in the BTG. In real life a user does not have necessarily such simple preferences, but specifies his wishes in a more explicit way.

Example 2. *A general strict partial order which does* not *form a weak order is given in Figure 2. In fact,* red*,* blue*, and* green *are the best values, but they are not considered as* substitutable *(*trivial SV-semantics*). In addition,* green *is not better than* yellow *or* purple*, i.e., the 'best values' cannot lie in the same equivalence class, we cannot represent* red*,* blue*, and* green *with one single numerical value as in Figure 1.*

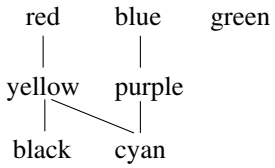


Figure 2: Hasse diagram for a general SPO.

Based on such simple preferences on single domains, we can build complex preferences on multiple attributes of a database relation. For example, the *Pareto preference* renders all included preferences as *equally important*. In the literature this kind of data selection is also called the *Skyline operator* [2]. Typically such techniques are used to find optimal tuples w.r.t. potentially conflicting goals.

There are many algorithms for the computation of preference queries (see [6] for an overview). Many of them rely on a nested-loop and tuple-to-tuple comparison approach (e.g., Block-Nested-Loop, BNL [2]). The major advantage of the BNL algorithm is its simplicity and suitability for computing the maxima of *general strict partial orders*. However, most of the tuple-comparison algorithms have a *quadratic* worst-case time complexity, $\mathcal{O}(n^2 d)$ – where n is the size of the d-dimensional input data.

On the other hand there are many algorithms which exploit the *lattice structure* induced by a Pareto preference for efficient preference evaluation, cp. [30, 33, 12, 11, 26, 21, 14]. In a lattice, two arbitrary elements have an infimum and a supremum. Most of these algorithms offer excellent performance for domains with low cardinalities. The lattice based algorithms are the only ones with a *linear runtime complexity*, $\mathcal{O}(dV + dn)$, where V is the product of the cardinalities of the d low-cardinality domains. Apart from the domain

size restriction, lattice based algorithms can only deal with *Pareto preferences consisting of preferences that form weak orders* on their domains. But this is not always suitable for real life applications as in Figure 2.

In this paper, we present a method to embed any strict partial order into a lattice. That means, we overcome the weak order restriction and thus *make lattice-based algorithms capable of dealing with general strict partial orders*. For this, we show 1.) how to transform general strict partial orders into lattice structures, 2.) how to construct smaller lattices in the case of special base preferences, and 3.) how to combine weak orders and general strict partial orders into lattices.

The rest of this paper is organized as follows: Section 2 contains the formal background. Section 3 presents the embedding of general database preferences into lattices, and in Section 4 we present a method to construct smaller lattices for base preferences. In Section 5 we show how to combine weak order preferences and general strict partial orders. In Section 6 we discuss some implementation details. Section 7 provides some related work and Section 8 contains our concluding remarks.

2 Background

Preference queries in database systems have been in focus for some time, leading to diverse approaches, e.g., [5, 22]. We review [22, 23], where preferences are modeled as strict partial orders.

2.1 Preference Modeling

Following [22] a database preference P is defined as $P := (A, <_P)$, where A is a set of attributes and $<_P$ is a *strict partial order* (SPO) on the domain of A. The term $x <_P y$ for $x, y \in \text{dom}(A)$ is interpreted as "*I like y more than x*", "*is preferred to*", or "*is more probable than*", and so forth. As strict partial orders are *transitive*, *better-than* relations in this model are, too. The *maximal values* of $P = (A, <_P)$ are defined as

$$M(P) := \{v \in \text{dom}(A) | \nexists w \in \text{dom}(A) : v <_P w\} \tag{1}$$

If of two different objects none is better w.r.t. a preference, we call them *indifferent*, cp. [22, 10, 15]. The *indifference* relation $\|_P$ is defined as:

$$x \|_P y \Leftrightarrow \neg(x <_P y \vee y <_P x)$$

Keep in mind that, in general, $\|_P$ is reflexive, symmetric, but *not transitive* [15].

2.2 Weak Order Preferences

An important subclass of strict partial orders are *weak order preferences* (WOP). Following [15, 16], a weak order preference is a strict partial order in which *indifference is transitive*. For any WOP we can define a *utility function* [15] mapping each attribute value to a number to determine dominance between two values. The utility function depends on the type of preference as can be seen in the next section.

Lemma 1 (Utility Function for WOPs [15])**.** *Each weak order preference* $P = (A, <_P)$ *has a* utility function *to determine the dominance between values:*

$$u_P : \mathrm{dom}(A) \to \mathbb{R}_0^+$$
$$x <_P y \iff u_P(x) > u_P(y)$$

For weak order preferences indifferent values belong to the same *equivalence class*. The equivalence class of an attribute value x can be identified by $u_P(x)$. Note that if P is a weak order then $\|_P$ is an equivalence relation (reflexive, symmetric, transitive).

Definition 1 ($\max(P)$)**.** $\max(P) \in \mathbb{R}_0$ *is the* maximum u_P value *for a weak order preference* P*.*

$$\max(P) := \max\{u_P(v) \mid \forall v \in \mathrm{dom}(A)\}$$

2.3 Base Preferences

Preferences on single attributes like *discrete* (*categorical*) or *continuous* (*numerical*) domains are called *base preferences*, cp. [24]. Usually they can be defined as WOPs having a utility function, such that

$$u_P(v) := \begin{cases} f(v) & \text{if } \ d = 0 \\ \left\lceil \frac{f(v)}{d} \right\rceil & \text{if } \ d > 0 \end{cases} \tag{2}$$

where $f : \mathrm{dom}(A) \to \mathbb{R}_0^+$ is a score function and $d \in \mathbb{R}_0^+$. In the case of $d = 0$ the function $f(v)$ models the *distance* to the best value. A d-parameter $d > 0$ represents a discretization, which is used to group ranges of scores together. The d-parameter maps different function values to a single integer number. Choosing $d > 0$ effects that attribute values with identical $u_P(v)$ value become *indifferent* and stay in the same *equivalence class*.

The definition of the function f depends on the type of preference. The $\mathrm{BETWEEN}_d(A, [low, up])$ preference for example expresses the wish for a value between a *lower* and an *upper* bound. If this is infeasible, values having the smallest distance to $[low, up]$ are preferred, where the distance is discretized by the parameter d. The scoring function is $f(v) = \max\{low - v, 0, v - up\}$. The $\mathrm{AROUND}_d(A, v)$ is a special case of the former, where $low = up =: v$. The preferences $\mathrm{LOWEST}_d(A, \inf_A)$ and $\mathrm{HIGHEST}_d(A, \sup_A)$ express the wish

for the minimum and maximum, where $\inf_A$ and $\sup_A$ are the infimum and supremum of $\text{dom}(A)$.

In a categorical domain $\text{LAYERED}_m(A, \{L_1, \ldots, L_m\})$ expresses that a user has a set of preferred values given by the disjoint sets L_i, which form a partition of $\text{dom}(A)$. Thereby the values in L_1 are the most preferred values. The scoring function equals $f(v) = i - 1 \Leftrightarrow x \in L_i$. Figure 1 shows an example for such a preference.

2.4 Complex Preferences

Complex preferences combine other preferences and determine the relative importance of these. Intuitively, people speak of "*this preference is more important to me than that one*" or "*these preferences are all equally important*" (Pareto) [22, 5]. Hence, we need a notion of equality w.r.t. a preference.

A simple approach for the notion of equality w.r.t. a preference is to use strict equality of the domain values. But *equivalence classes* can be applied here as well. For example, if for two values x, x', $u_P(x) = u_P(x')$, i.e., the tuples have the same utility value; they belong to the same *equivalence class* and hence $x <_P y \Leftrightarrow x' <_P y$ for all y.

The *substitutable value semantics* has been introduced in [23] to have *indifference as a transitive relation* and every preference P is associated with an SV-relation $\cong_P$ on $\text{dom}(A)$, where the equivalence classes contain "equally good" objects.

Definition 2 (Substitutable Values (SV))**.**
Let $P = (A, <_P)$. $\cong_P$ is P's substitutable values relation *(SV) iff $\forall x, y, z \in dom(A)$*

a) $x \cong_P y \;\Rightarrow\; x \parallel_P y$

b) $(x <_P y \;\wedge\; y \cong_P z) \vee (x \cong_P y \;\wedge\; y <_P z) \Rightarrow x <_P z$

c) $\cong_P$ is reflexive, symmetric, and transitive

The identity relation on attribute set A is called trivial SV-relation $(=_P)$*. The* regular SV-relation $\sim_P$ *is the equivalence relation induced by the equivalence classes computed by u_P.*

Note that this definition of SV-semantics is needed to preserve the strict order property of complex preferences, cp. [23, 15].

The intuition behind SV-relations is that a tuple x can be substituted by x', if $x \cong_P x'$ holds. For *base preferences* regular SV-semantics $\sim_P$ does not affect $<_P$ itself, but expresses that it is admissible to substitute values for each other. The difference occurs when such preferences are combined to *complex preferences*, e.g., a Pareto preference, where $\sim_P$ does affect $<_P$. Then, the value of the utility function u_P alone is not sufficient to determine domination.

Definition 3 (Pareto Preference)**.** *Given m preferences $P_i = (A_i, <_{P_i})$ and objects $x = (x_1, \ldots, x_m)$, $y = (y_1, \ldots, y_m) \in \mathrm{dom}(A_1 \times \cdots \times A_m)$. A Pareto preference $P := P_1 \otimes \ldots \otimes P_m$ is defined as:*

$$x <_P y \iff \exists i : x_i <_{P_i} y_i \ \wedge \\ \big(\forall i, j \in \{1, \ldots, m\}, j \neq i : (x_j <_{P_j} y_j \ \vee \ x_j \cong_{P_j} y_j)\big)$$

Note that although we have only WOPs as input preferences for a Pareto preference P, P itself forms a strict partial order, but not a WOP anymore [5]. All input values leading to the same utility value combination $(u_{P_1}(v), \ldots, u_{P_m}(v))$ for the WOPs P_i in P belong to the same equivalence class.

Example 3. *Reconsider Example 1. The preference P_1 on colors (Figure 1) now should be equally important to $P_2 :=$ AROUND$_5(price, 50K)$ (Pareto preference query). Let $x = (red, 50K)$ and $y = (blue, 45K)$ be two tuples as in Table 1. Using* trivial *SV-semantics the tuple $(red, 50K)$ is not better than $(blue, 45K)$, although a price of \$50K ($u_{P_1}(50K) = 0$) is better than \$45K ($u_{P_1}(45K) = 1$). Due to the trivial SV-semantics 'red' and 'blue' are* not *substitutable. Note that the same holds for $(green, 48K)$, i.e., the result are the cars with id $\in \{3, 4, 7\}$.*

Having regular *SV-semantics, 'red' and 'blue' (and 'green') become substitutable in the preference on the colors. Hence, $(red, 50K)$ is equally good as $(blue, 45K)$ concerning the color, but a price of \$50K is better than \$45K concerning the preference* AROUND$_5$*. This means, $(red, 50K)$ is the only tuple in the result set which corresponds to the car with id $= 3$. It also dominates $(green, 48K)$.*

2.5 Better-Than-Graph and Lattices

A *Better-Than-Graph* (BTG) is a visualization of the partial order induced by a preference. A graph like this is equivalent to a *Hasse diagram* (directed and acyclic) [10]. An edge between two nodes in a BTG denotes dominance of the upper node over the lower.

Definition 4 (Better-Than Graph (BTG))**.** *The* better-than graph *of a preference $P = (A, <_P)$ is the Hasse diagram of $<_P$ with the additional characteristics:*

a) A node in the BTG corresponds to a value or a set of substitutable values (equivalence class) in dom(A)*.*

b) The utility function value *of a node is the length of a longest path leading from the best node to it.*

The BTG for weak order preferences forms a *total order* where each node represents one equivalence class u_P, cp. Figure 1. A Pareto preference P has one node for each possible *utility value combination* of the P_is of P and constitutes a lattice [10, 30], where each node in the BTG stands for one equivalence class of the preference.

Definition 5 (Lattice [10]). *A partially ordered set D with operator $<_P$ is a* lattice, *if $\forall a, b \in D$, the set $\{a, b\}$ has a* least upper bound *(supremum) and a* greatest lower bound *(infimum) in D. If a least upper bound and a greatest lower bound is defined for all subsets of D, we have a* complete lattice.

Lemma 2 (#Nodes of a BTG [33]). *Let $P := P_1 \otimes \ldots \otimes P_m$ be a Pareto preference on the domain $\mathrm{dom}(A) := \mathrm{dom}(A_1) \times \ldots \times \mathrm{dom}(A_m)$ and P_i, $i = 1, \ldots, m$ WOPs with $\max(P_i)$ as in Definition 1. Then*

$$\#nodes(BTG_P) = \prod_{i=1}^{m} (\max(P_i) + 1) \tag{3}$$

More properties of BTGs (#edges, height, width, etc.) can be found in [33].

Example 4. *Consider our car sample in Table 1. Let P_1 be a* $\mathrm{LAYERED}_3$ *preference as in Figure 1, i.e., the u_P mapping is as follows:*

red, blue, green	$\rightarrow$	*0*
yellow, purple	$\rightarrow$	*1*
black, cyan	$\rightarrow$	*2*

Let P_2 be a $\mathrm{LAYERED}_5$ *preference on the* make *in Table 1, where each make forms its own layer in the order GMC $\rightarrow$ 0, BMW $\rightarrow$ 1, Ford $\rightarrow$ 2, Mercedes $\rightarrow$ 3, Audi $\rightarrow$ 4.*

Figure 3 shows the BTG for the Pareto preference $P_1 \otimes P_2$ with the maximum values $\max(P_1) = 2$ and $\max(P_2) = 4$. The node $(0, 0)$ presents the best node*, i.e., the supremum, whereas $(2, 4)$ is the* worst node *(infimum). The bold numbers next to each node are* unique identifiers *(ID) for each node in the lattice, cp. [33].*

*A dataset D does not necessarily contain tuples for each lattice node. In Figure 3, the gray nodes are occupied (*non-empty*) with elements from the dataset in Table 1 whereas the white nodes have no element (*empty*). Each node contains the objects mapped by u_P to the same feature vector of the preference query. For example, the tuples (red, BMW) and $(green, BMW)$ both correspond to the node $(0, 1)$. All values in the same node / equivalence class are indifferent and considered substitutable. The number of nodes is $(2 + 1) \cdot (4 + 1) = 15$.*

2.6 Lattice Skyline Revisited

Lattice based algorithms like *LS-B* [30] and *Hexagon* [33] exploit the observations from the last section to find the maximal values of a dataset w.r.t. some preferences. The elements of the dataset D that compose the *maximal values* is formed by those nodes in the BTG that have *no path leading to them from another non-empty node*. All other nodes have direct or transitive edges from the maximal nodes, and therefore are *dominated*.

For the implementation of such algorithms the lattice is usually represented by an *array*, where each position stands for one node in the lattice [30] (according to the *ID* for each node as in Example 4). The array stores the

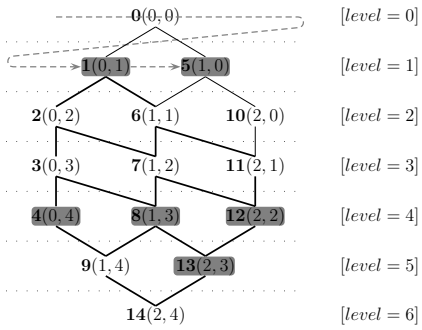


Figure 3: BTG for a Pareto preference.

empty, *non-empty*, and *dominated* state of a node. For each element $t \in D$ the algorithms compute the unique position in the array and mark this position as *non-empty*. Next, the nodes are visited in a breadth-first order (BFT, dashed line in Figure 3). Non-empty nodes cause a depth-first traversal (DFT, thick black edges in Figure 3) where the dominance flags are set. Finally those nodes represent the maximal values which are both *non-empty and non-dominated*. In Example 4, only the nodes $(0, 1) \simeq \{(red, \text{BMW}), (green, \text{BMW})\}$ and $(1, 0) \simeq \{(yellow, \text{GMC})\}$ are not dominated. All other nodes have direct or transitive edges from these two nodes, and therefore are *dominated*. Note that in general strict partial orders do not form lattices (e.g., Figure 2) and therefore the above approach cannot be applied.

The original lattice based algorithms have linear runtime complexity w.r.t. the number of input objects and the size of the BTG. More precisely, the complexity is $\mathcal{O}(dV + dn)$, where d is the dimensionality, n is the number of input tuples, and V is the product of the cardinalities of the d domains from which the attributes are drawn.

3 General Strict Partial Orders

In Section 2 we have seen that the BTG of a Pareto preference is a lattice. Now, we will use such lattices as an abstraction from the underlying preference to integrate *general strict partial orders*. This will enable us to handle arbitrary preferences in the same way as Pareto preferences.

3.1 Embedding General SPOs into Lattices

To define general strict partial orders the preference constructor EXPLICIT(A, E) was introduced in [22]. It constructs a preference from a given set of edges E. Unmentioned values are considered worse than any value in some element of E. The transitive hull of E is the Better-Than-Graph of the strict partial order expressed by EXPLICIT. In contrast to other base preferences it does *not* construct a weak order.

Example 5. *A typical simple general strict partial order expressed as* EXPLICIT *is*

I like `red` *more than* `black`*. And I like* `blue`*.*

Figure 4 shows the Hasse diagram of this preference, which does not construct a weak order.

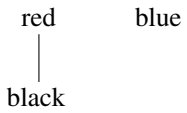


Figure 4: Hasse diagram for a general SPO.

Unfortunately there is no efficient algorithm to evaluate arbitrary database preferences as above but BNL, because BNL compares each tuple to all other tuples in the dataset (worst-case complexity $\mathcal{O}(dn^2)$, cp. [2]). To be able to apply efficient algorithms like [30, 33, 12, 11, 14] on EXPLICIT, we have to embed the strict partial order defined by it into a distributive lattice.

For this embedding, we start with a Hasse diagram representing a general strict partial order. Since lattices need a least upper bound and a greatest lower bound (cp. Definition 5) we just add virtual nodes to the existing Hasse diagram. Then, we will assign a so-called *signature* to each node. This signature is a combination of integers that is *unique for each node* and hence can be used to identify it.

For nodes in the same level (which are indifferent) we construct Pareto incomparable signatures. A node n which is directly dominated by a node m needs a "worse" signature than the dominator. For this we just increase one position in the signature of m and assign it to n. Which position we use is determined by a depth-first traversal. When the construction of the distributed lattice structure is complete, the signature of a node is identical to the *integer combination* of the BTG node it is mapped to. Algorithm 1 describes our approach in detail.

Algorithm 1. (Embedding SPOs into Lattices).

The mapping is done in the following steps:

1) Identify non-dominated nodes and generate an unlabeled virtual top node $\triangle$ *for them. Add edges from* $\triangle$ *to the non-dominated nodes. Also add a virtual bottom node* $\bigtriangledown$ *that is dominated by all the nodes not dominating other values in the graph.*

2) Do a depth-first search beginning at the top node. The algorithm used for the depth-first search is irrelevant, but the following issues have to be kept in mind:

a) Keep a counter. Each time the search finds a dead end, increase the counter by one.

b) Annotate each edge during the search with the counter value.

c) Do not follow annotated edges.

3) Do a breadth-first search on the graph, starting at the top node again. There are two possibilities in each node n:

a) n is directly dominated by exactly one node and reached by an edge with an annotated value of v. The signature value of n is the signature value of its dominating node increased by one at position v.

b) n is directly dominated by a number of nodes d_1, d_2, ..., d_x. The signature of n at each position i is given by the maximum value of the d_i at the same position.

If two nodes n and m (or more) are dominated directly by exactly the same set of nodes $d_1, \ldots, d_x$, this yields the same signature. In this case, increase the value of n (resp. m) at the position of the edge on which it was reached first by the depth-first search.

4) Check the maximum values in use at each position of the node signature and remove those positions with a maximum value of zero.

Please note that in step 3b also signature values at other positions could be increased. This can lead to smaller BTGs. It is only necessary that n's and m's signatures are increased at different positions, preserving their indifference. Step 4 is unnecessary for the correctness of the algorithm. It is simply removing elements not containing any information for any node to reduce the signature length. For an EXPLICIT preference, all domain values not mentioned in its constructor are mapped to the virtual bottom node.

We now prove that Algorithm 1 preserves the original strict partial order.

Proof. We have to show that 1.) the signatures can be used to determine a supremum and an infimum for each pair of nodes, which is the basic characteristic of a lattice, cp. Definition 5. Apart from that, 2.) the relation between any two nodes of the SPO has to be preserved.

1.) Let's assume for a SPO our algorithm yields node signatures with n positions. Using the signatures, the supremum for any two nodes a and b with signatures $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ is defined as

$$\sup(a, b) := (\min(a_1, b_1), \ldots, \min(a_n, b_n)) \ ,$$

their infimum as

$$\inf(a, b) := (\max(a_1, b_1), \ldots, \max(a_n, b_n)) \ .$$

Both infimum and supremum of two nodes might not be elements of the original SPO (like $\triangle$ and $\triangledown$), cp. [10]. The top node $\triangle$ by definition has the signature $(t_1, \ldots, t_n) = (0, \ldots, 0)$. The supremum of any node x and $\triangle$ is $\triangle$, while their infimum is x. In this sense Algorithm 1 constructs a lattice from the node signatures.

2.) Given two nodes a and b in the SPO with $b <_P a$. Then there is a path from a to b.

- For any node $g = (g_1, \ldots, g_n)$ on this path that is dominated by exactly *one* node, the signature is increased at the position given by the edge's annotation. Assume this position is i. So for all such nodes g on that path it holds that $g_i > a_i$. This holds for b as well.

  A node h on the path dominated by *two* nodes f' and f'' will have signature values defined by the maximum values of the dominating nodes (a generalization to more than two nodes is obvious). As $f' \parallel_P f''$, it holds that

  $$\begin{aligned} &\exists i, j : f'_i > f''_i \ \wedge \ f'_j < f''_j \text{ , and} \\ &\forall k : h_k \geq f'_k \ \wedge \ h_k \geq f''_k \\ &\qquad\qquad \wedge \ (\exists i, j : h_i > f''_i \ \wedge \ h_j > f'_j). \end{aligned}$$

  In summary, nodes which are dominated by one or more nodes in the original SPO are modeled as dominated in the lattice structure, too.

- We have to consider indifferent nodes in the original SPO. For this consider two nodes $s \parallel_P t$. We can find a supremum as

  $$x := \sup(s, t) := (\min(s_1, t_1), \ldots, \min(s_n, t_n)) \ .$$

  We assume a path from x to s starting with an edge annotated with i, reaching the node x' with

  $$\begin{aligned} &x' := \\ &(\min(s_1, t_1), ..., \min(s_i, t_i) + 1, ..., \min(s_n, t_n)) \end{aligned}$$

  A path from x to t starts with an edge annotated with j, leading to a node

  $$\begin{aligned} &x'' := \\ &(\min(s_1, t_1), ..., \min(s_j, t_j) + 1, ..., \min(s_n, t_n)) \end{aligned}$$

  Note that $i \neq j$, as otherwise x' and x'' would be the equal node and the supremum of s and t. So $x' \parallel_P x''$ in the lattice, since x' and x'' cannot dominate a node that dominates both s and t as this would be their supremum. Hence s and t must be indifferent.

We see that domination between nodes is preserved.

□

Based on our algorithm we are now able to embed any strict partial order into a lattice. After constructing the lattice and mapping the nodes to an array as described in Section 2.6, we apply a lattice based algorithm like LS-B or Hexagon. The remaining nodes contain the maximal values. Note that our approach still has a linear runtime complexity. The construction of the lattice in Algorithm 1 only relies on a DFT and a BFT, both are linear in the number of nodes and edges.

The exponential lattice size is not a major limitation, because in most cases preferences are specified only over a few items, all others can be considered as worse then the mentioned objects and hence can reside in the virtual bottom node of the lattice. That means, it is not necessary to construct the lattice on the complete domain, but only on the objects specified by the user. Therefore we create small lattices which can be handled efficiently.

3.2 Examples

We now present some examples on how to embed general strict partial orders into BTGs.

Example 6. *Reconsider the preference given in Figure 4. Following Algorithm 1, we add a virtual top and bottom node. Then, the depth-first traversal annotates all edges as shown in Figure 5a. The highest annotation is 2 and hence the virtual top node corresponds to* $\triangle \rightarrow (0,0)$*. Then the mapping into a lattice is "red"* $\rightarrow (0,1)$*, "blue"* $\rightarrow (1,0)$*, and "black"* $\rightarrow (0,2)$*. In this case the signature for the virtual bottom node is* $\triangledown \rightarrow (1,2)$ *which characterizes the complete structure of the lattice. Figure 5b shows the lattice with the order embedded into it. The nodes with frames represent the original Hasse diagram's nodes.*

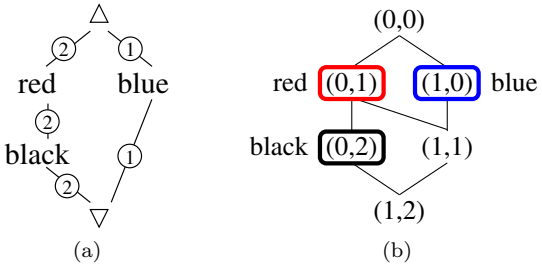


Figure 5: Lattice for the SPO in Figure 4.

Example 7. *Another Hasse diagram for a general partial order is known from the introductory Example 2 and presented again in Figure 6a, where we use the abbreviations red (r), blue (b), green (g), black (k), etc.*

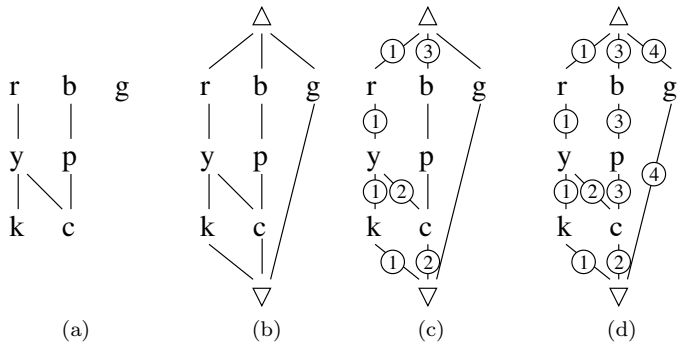


Figure 6: Embedding a SPO into a BTG (1).

Following Algorithm 1, we add top and bottom nodes in Figure 6b. Then, the depth-first marking of the edges begins. Figure 6c shows the moment when the counter value is "3" and the first edge has been marked with it. In Figure 6d, all edges are marked. The depth-first orders leads to annotations on the edges which make each path from the top node to any other node in the Hasse diagram unique.

Then, we determine the node signatures, starting at $\triangle$*. As the highest number assigned to an edge is 4, the node signatures consists of four integer values. The top node has a signature of* $(0, 0, 0, 0)$*. For* r*, we get the signature* $(1, 0, 0, 0)$*, as it is dominated by the top node by an edge marked with* 1 *and so the signature value of* $\triangle$ *is increased by one at position* 1*. Table 2 shows signatures for all nodes after step 3 of the algorithm.*

Table 2: Signatures for all nodes after step 3.

node	r	y	k
signature	$(1, 0, 0, 0)$	$(2, 0, 0, 0)$	$(3, 0, 0, 0)$

b	p	c	g
$(0, 0, 1, 0)$	$(0, 0, 2, 0)$	$(2, 0, 2, 0)$	$(0, 0, 0, 1)$

We see that the maximum value at position 2 is 0. Hence it is removed. So we keep the maximum signature values 3, 2, and 1. The resulting BTG can be seen in Figure 7 showing all signature values and the framed nodes connected to a value of the original order. Note that the $\max(P)$ *values are 3, 2, and 1, hence the BTG has* $4 \cdot 3 \cdot 2 = 24$ *nodes.*

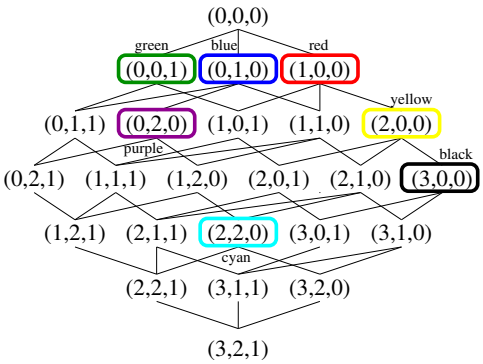


Figure 7: Lattice structure for the preference expressed in Figure 6a.

Our algorithm has no problem in integrating any kind of strict partial order. Isolated nodes in a Hasse diagram belong to the set of top nodes that is linked to the virtual single top node created in step one of the algorithm. Please note that our algorithm for embedding a strict partial order into a distributive lattice like a BTG is not necessarily producing minimal BTGs.

Example 8. *We will have another look at the strict partial order in Figure 6a. Some depth-first search algorithm yields the edge annotation of Figure 8. Then, the maximum integer values for the embedding are 2, 1, 2, 1. With those maximum values, the BTG that is constructed has 36 nodes; it is 50% bigger than the one shown in Example 7.*

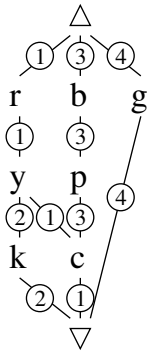


Figure 8: Embedding a SPO in a BTG (2).

3.3 Remarks

Another mapping of partial orders to distributive lattices has been presented in [36]. Using only *two integer* numbers to represent each value in a partial order, a BTG constructed according to the maximum signature levels tends to be smaller than when using Algorithm 1. But not all partial orders can be expressed when such a mapping is used, as we will see in Lemma 3.

Lemma 3. *We have some values a, b, ..., f and a partial order on them defined by $d < a$, $d < b$, $e < a$, $e < c$, $f < b$, and $f < c$. The Hasse diagram for this order is given in Figure 9. Then a mapping of each node to a BTG node defined by two integer values is not able to preserve the original partial order.*

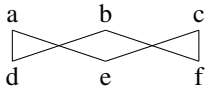


Figure 9: SPO needing more than two integer values.

Proof. Let $a = (a_1, a_2)$, $b = (b_1, b_2)$, and $c = (c_1, c_2)$ the indifferent two integer representations of the corresponding nodes. We will assume $a_1 < b_1 < c_1$ and $a_2 > b_2 > c_2$. From the strict partial order we know:

$$e < a \wedge e < c \Rightarrow a_1 \leq e_1 \wedge a_2 \leq e_2 \wedge c_1 \leq e_1 \wedge c_2 \leq e_2$$

Hence, the combination of smallest possible values for e is $(e_1, e_2) = (c_1, a_2)$. So e always is dominated by b, too, which is a contradiction to the original partial ordered set. □

Using the Hasse diagram structure of Figure 9 as a pattern, similar proofs for more than two values in a signature can be found. The number of integers needed to construct a BTG to embed a partial order depends on the partial order and does not have an upper border. In Example 9, a possible embedding of the given partial order into a BTG using three integer values can be seen.

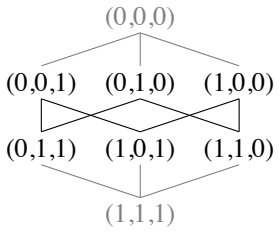
Example 9. *Consider the partial order in Figure 9. A possible embedding to a distributed lattice is shown in Figure 10. The maximum values are* $(1, 1, 1)$.

4 WOPs with Trivial SV-Semantics

Our approach of embedding general SPOs does not necessarily construct minimal lattices. We assume that finding a minimal lattice is an NP-hard problem. Therefore, we leave this for further investigations and future work. Nevertheless, in this section we show a method which in general constructs smaller lattices than Algorithm 1. However, this only applies for base preferences like LAYERED_m and BETWEEN_d and their sub-constructors.

node	signature
a	(0,0,1)
b	(0,1,0)
c	(1,0,0)
d	(0,1,1)
e	(1,0,1)
f	(1,1,0)

(a)



(b)

Figure 10: Mappings (a) and lattice structure (b) for the SPO in Figure 9.

We have seen that graphs of WOPs (with regular SV-semantics ($\cong_P$)) build a total order and therefore always construct a minimal lattice (cp. Section 2.5). A Pareto preference consisting of WOPs also constitutes a complete lattice, which is minimal w.r.t. the base preferences. Therefore, we now consider base preferences with trivial SV-semantics ($=_P$). Keep in mind that for regular SV-semantics all objects in an equivalence class are substitutable, but in the case of trivial SV-semantics they are not. That means each object in a dataset forms its own (single valued) class.

4.1 Categorical Base Preferences with Trivial SV-Semantics

Modeling incomparability of values in categorical base preferences is straightforward. Using trivial SV-semantics in $\text{LAYERED}_m(A, \{L_1, \ldots, L_m\})$, all values in one of the L_i are incomparable.

Theorem 1. *Let* $P := \text{LAYERED}_m(A, \{L_1, \ldots, L_m\})$ *and* P' *derived from* P *by replacing regular with trivial SV-semantics. Each value in* $\text{dom}(A)$ *is mapped to a pair of integer values.*

The elements of the L_i *are labeled with indexes:* $L_i := \{l_{i,1}, l_{i,2}, \ldots, l_{i,|L_i|}\}$. *Every element of* L_i *has to get a unique index value. Then, the integer combination for each* $l_{i,j}$ *can be found as follows:*

$$l_{i,j} \to (l_l, l_r)$$

where

$$\begin{aligned} l_l &:= \left|\bigcup_{x=1}^{i-1} L_x\right| - i + j \\ l_r &:= \left|\bigcup_{x=1}^{i} L_x\right| + 1 - (i + j) + \\ &\qquad |\{x \mid x \leq i \wedge |L_x| = 1 \wedge |L_{x-1}| = 1\}| \end{aligned}$$

Proof. Consider three categorical values $l_{i,j}, l_{i,k}, l_{i+1,q} \in \mathrm{dom}(A)$ with $j < k$. A value $l_{x,y}$ is mapped to $(l_{x,y}[0], l_{x,y}[1])$. We have to prove that P' constructs the same order as P on elements of different layers and renders elements of the same layer indifferent. For readability, we will abbreviate $\left|\bigcup_{x=1}^{i-1} L_x\right|$ with s and $|\{x \mid x \leq i \wedge |L_x| = 1 \wedge |L_{x-1}| = 1\}|$ with *t(i)*.

- $l_{i,j} \parallel_{P'} l_{i,k}$:
  - $l_{i,j}[0] - l_{i,k}[0] = (s - i + j) - (s - i + k) = j - k \Rightarrow l_{i,j}[0] < l_{i,k}[0]$
  - $l_{i,j}[1] - l_{i,k}[1] = (s + |L_i| + 1 - (i+j) + t(i)) - (s + |L_i| + 1 - (i+k) + t(i)) = -j + k \Rightarrow l_{i,j}[0] > l_{i,k}[0]$

  With $l_{i,j}[0] < l_{i,k}[0] \wedge l_{i,j}[0] > l_{i,k}[0]$ it follows that $l_{i,j} \parallel_{P'} l_{i,k}$.

- $l_{i+1,q} <_{P'} l_{i,j}$:
  - $l_{i,j}[0] \leq l_{i+1,q}[0] \Leftrightarrow j \leq |L_i| - 1 + q$
    This always holds as $j \leq |L_i| \wedge (-1 + q) \geq 0$
    $\Rightarrow j = |L_i| - 1 - q \Leftrightarrow j = |L_i| \wedge q = 1$ and $j < |L_i| - 1 - q \Leftrightarrow j < |L_i| \vee q > 1$

- $l_{i,j}[1] \leq l_{i+1,q}[1] \Leftrightarrow -j + t(i) \leq |L_{i+1}| - 1 - q + t(i+1)$
  - case 1: $|L_i| = 1 \wedge |L_{i+1}| = 1 \Leftrightarrow t(i+1) = t(i) + 1 \Rightarrow j = 1 \wedge q = 1$
    $\Rightarrow -j + t(i) \leq |L_{i+1}| - 1 - q + t(i) + 1$
  - case 2: $|L_i| > 1 \vee |L_{i+1}| > 1 \Leftrightarrow t(i+1) = t(i)$
    $\Rightarrow -j + t(i) \leq |L_{i+1}| - 1 - q + t(i)$
    For $j = 1 \wedge q = |L_{i+1}|$, both sides are equal. As $(j \geq 1) \wedge (q - |L_{i+1}| \leq 0)$, the inequation holds in all other cases, too.

To sum up the preceding points, we showed that
$l_{i+1,q} <_{P'} l_{i,j}$ always holds:

- $j = 1 \wedge q = |L_{i+1}|$
  $\Longrightarrow l_{i,j}[0] < l_{i+1,q}[0] \wedge l_{i,j}[1] = l_{i+1,q}[1]$
- $1 < j < |L_i| \wedge 1 < q < |L_{i+1}|$
  $\Longrightarrow l_{i,j}[0] < l_{i+1,q}[0] \wedge l_{i,j}[1] < l_{i+1,q}[1]$
- $j = |L_i| \wedge q = 1$
  $\Longrightarrow l_{i,j}[0] = l_{i+1,q}[0] \wedge l_{i,j}[1] < l_{i+1,q}[1]$

As we can see, all elements of the same layer are indifferent and better than all elements of (w.r.t. their indexes) higher layers. □

Example 10. *Consider the color preference in Figure 1. We derive a preference P' with the same sets but trivial instead of regular SV-semantics. This means for example that 'red', 'blue', and 'green' are* not *substitutable. Table 3 shows the integer combinations (l_l, l_r) for each color. For example, for 'red' we compute $l_l = 0 - 1 + 1 = 0$ and $l_r = 3 + 1 - (1 + 1) + 0 = 2$, i.e., 'red' $\rightarrow (0, 2)$.*

Table 3: Mappings for Example 10.

color	L_i	u_P	$l_{i,j}$	(l_l, l_r)
red	L_1	0	$l_{1,1}$	$(0, 2)$
blue	L_1	0	$l_{1,2}$	$(1, 1)$
green	L_1	0	$l_{1,3}$	$(2, 0)$
yellow	L_2	1	$l_{2,1}$	$(2, 3)$
purple	L_2	1	$l_{2,2}$	$(3, 2)$
black	L_3	2	$l_{3,1}$	$(3, 4)$
cyan	L_3	2	$l_{3,2}$	$(4, 3)$

Figure 11 shows the lattice for P'. Nodes with invalid level combinations are white, nodes with other colors filled and labeled with the color and its assigned integer combination. Following Eq. 3, the size of the BTG is $5 \cdot 5 = 25$. Note that the number of occupied nodes is the size of the categorical domain, $|dom(A)| = 7$.

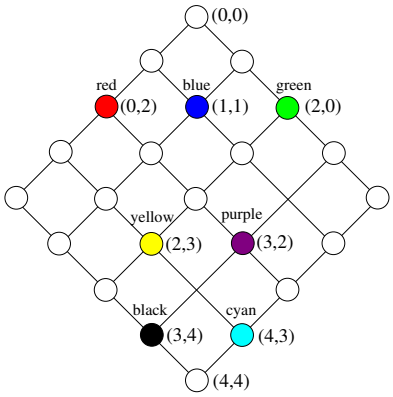


Figure 11: BTG for LAYERED$_m$ with trivial SV-semantics.

4.2 Numerical Base Preferences with Trivial SV-Semantics

In the case of numerical preferences it is sufficient to analyse the problem only for BETWEEN$_d$, because all other numerical base preferences are special cases of the former. However, embedding BETWEEN$_d$ into a lattice is not a trivial task, because if two values exist in the same equivalence class, they are not substitutable when using trivial SV-semantics and must be modeled incomparable in the lattice structure.

Example 11. *Let* $P := \text{AROUND}_5(price, 50K)$ *be a preference with* $\text{dom}(price) = \{45K, 48K, 50K\}$. *The value* $50K$ *is the* maximal value*, whereas* $45K, 48K$ *lie in a distance of* d *from the best value. Since* $45K, 48K \in [45K, 50K[$*, they share the same* u_P *value* 1*, but are not substitutable and must be modeled as incomparable in the lattice which they should be embedded in.*

Since the domain of an attribute could be infinite, this makes the embedding of numerical base preferences with trivial SV-semantics difficult. However, in database relations we generally assume a *closed-world*, hence we have a finite number of objects in an equivalence class. In this case, BETWEEN_d could be modeled in the same way as LAYERED_m. That means, construct a preference LAYERED_m based on the numerical values preferred by BETWEEN_d and compute $l_{i,j}$ as in Theorem 1.

Under some assumptions we can produce smaller lattices, e.g., when we have single occupied equivalence classes, or if a user specifies an "interval as the preferred value". In these cases we avoid the problem of several indifferent objects in the same equivalence class; objects having the same u_P function value are either identical or lie "left and right" of the maximal value.

Theorem 2. *Given* $P := \text{BETWEEN}_d(A, [low, up])$ *and let* P' *be derived from* P *by replacing regular by trivial SV-semantics. Assume each equivalence class contains at most one element. We map* $x \in \text{dom}(A)$ *to the integer combination* (l_1, l_2) *in* P' *as follows:*

$$x \to (l_1, l_2) = \begin{cases} (u_P(x) \quad\;\; , \; u_P(x) - 1) & \text{if} \quad x > up \\ (u_P(x) - 1 \;\; , \; u_P(x) \quad\;\;) & \text{if} \quad x < low \end{cases}$$

For $x \in [low, up]$ *we set* $x \to (0, 0)$*. Then,* P' *models the same order w.r.t.* $\text{dom}(A)$ *as* P*, but distinguishes between values lower and values higher than the interval borders.*

This can be interpreted as two WOPs being connected and used to model a strict partial order. A "virtual" Pareto preference is constructed by the numerical base preference.

Proof. Consider a value v mapped to (v_1, v_2), and a value w mapped to (w_1, w_2). The following cases may occur:

- $u_P(v) = u_P(w) + 1$:
  - $v < low \wedge w < low \Rightarrow (w_1 = v_1 + 1) \wedge (w_2 = v_2 + 1)$
  - $v < low \wedge up < w \Rightarrow (w_1 = v_1 + 2) \wedge (w_2 = v_2)$
- $u_P(v) = u_P(w)$:
  - $v < low \wedge w < low \Rightarrow (v_1, v_2) = (w_1, w_2)$
    $\Rightarrow v \sim_{P'} w$

– $v < low \wedge up < w$
  $\Rightarrow$ $(v_1, v_2) = (u_P(v), u_P(v) + 1)$
  $(w_1, w_2) = (u_P(v) + 1, u_P(v))$
  $\Rightarrow v \parallel_{P'} w$

All other possible cases can be derived from those above. So the integer combination assigned to domain values fulfills the specification of the preference. □

Lemma 4 (#Nodes)**.** *The number of nodes in the BTG P' defined by a preference $P :=$ BETWEEN$_d$ by replacing regular by trivial SV-semantics is given by:*

$$\#nodes(BTG_P) = (\max(P) + 1)^2 \tag{4}$$

Proof. Looking at the computation of integer combinations for values to be rated, the BTG that is constructed is identical to one for a Pareto preference containing two WOPs with maximum values of $\max(P) + 1$, cp. Eq. 3. □

Lemma 5 (#Used Nodes)**.** *Consider a preference P' which is defined as a* BETWEEN$_d$ *preference P with trivial instead of regular SV-semantics. The number of* used nodes *(i.e. the number of nodes that can be matched by values evaluated by P') in the $BTG_{P'}$ is given by*

$$\#used\_nodes(BTG_{P'}) = 2 \cdot \max(P) + 1 \tag{5}$$

Proof. The node $(0, 0)$ is used for perfect matches. Other nodes used have combinations of $(x, x + 1)$ or $(x + 1, x)$. The minimum value for x is 1, the maximum is $\max(P)$, leading to $2 \cdot \max(P) + 1$ values in use. □

Note that Lemma 4 and 5 only hold when considering equivalence classes which contain at most one object.

Example 12. *We search for a car which should cost around \$50K, i.e., $P :=$* AROUND$_5(price, 50K)$ *in the domain* $\mathrm{dom}(price) = \{45K, 50K, 55K, 70K, 75K\}$. *Then* $\max(P) = 5$.

No two domain values lie in the same equivalence class, we derive P' with trivial SV-semantics and create pair mappings: A perfect value of 50K is mapped to $(0, 0)$, 45K and 55K (with $u_P(45K) = u_P(55K) = 1$) are mapped to incomparable value combinations $(0, 1)$ and $(1, 0)$, respectively. All combinations can be found in Table 4.

Table 4: Value combinations for P'.

price	45K	50K	55K	70K	75K
u_P	1	0	1	4	5
(l_1, l_2)	(0,1)	(0,0)	(1,0)	(4,3)	(5,4)

Figure 12a shows the $BTG_{P'}$ of P' with trivial SV-semantics. *The black nodes have tuples belonging to them, the gray nodes represent valid integers for l_1 and l_2, while the white nodes are unused dummy nodes given by the graph structure. The number of nodes is $(5+1)^2 = 36$ from which $2 \cdot 5 + 1 = 11$ might be used.*

In Figure 12b we present BTG_P for P with regular SV-semantics *for comparison only. Values with the same u_P value are substitutable and reside in the same equivalence class. The BTG forms a chain.*

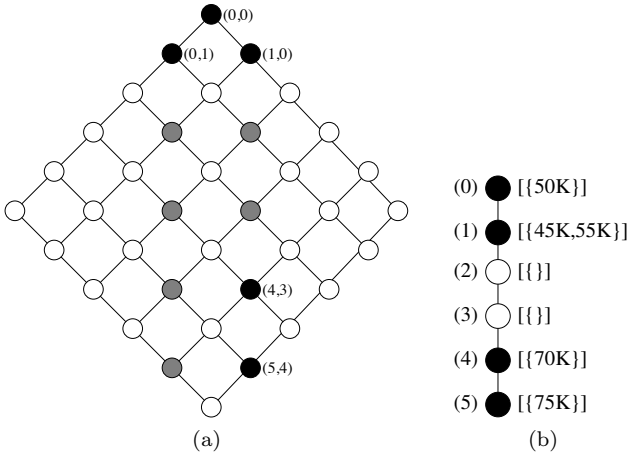


Figure 12: BTG for AROUND_d with (a) trivial SV-semantics and (b) regular SV-semantics.

5 Combining WOPs and SPOs

As we have seen, a single integer value is not enough to express the semantics of strict partial orders in general. We have overcome this limitation by using two or more integer values. Now we have to integrate these preferences in the standard Pareto preference introduced in Definition 3.

Theorem 3. *Consider a strict partial order S embedded into a BTG G_S, a Pareto preference P and the corresponding BTG G_P. The order constructed by the combination of P and S is visualized by the product of G_S and G_P.*

Proof. Both G_S and G_P are lattices. Following [10], the product of them yields a lattice with a combined order of both input lattices. □

The BTG for such a combination surely holds unused nodes (as the BTG for the strict partial order does already). Nevertheless the embedding can be very useful as it enables us to evaluate base preferences that are strict partial orders just like Pareto preferences and Pareto preferences consisting of strict partial

orders just as if they only used standard WOPs as input preferences. Example 13 defines a BTG that is the result of the combination of a WOP and a general strict partial order.

Example 13. *Remember the strict partial order on colors of Example 6. A possible lattice this order can be embedded in is shown in Figure 13a, where the nodes representing nodes of the original order are framed and labeled accordingly. The construction steps are shown in Figure 13b.*

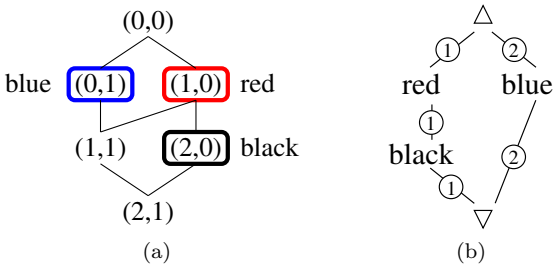


Figure 13: Lattice for the color preference.

Now we combine this order with a WOP with a maximum value $\max(P) = 3$*, e.g.,* $\text{AROUND}_{10}(price, 50K)$ *as in Example 12, but with* $d = 10$ *and regular SV-semantics. The BTG for* AROUND *forms a chain with* $u_P \in \{0, 1, 2, 3\}$.

The lattice for the combined (Pareto) order can be seen in Figure 14. As the resulting BTG is the product of the BTGs of the two underlying preferences, each of the original nodes in Figure 13a is multiplied. For example, the multiplication of $(0, 1)$ *by* $\{0, 1, 2, 3\}$ *results in* $(0, 0, 1)$*,* $(1, 0, 1)$*,* $(2, 0, 1)$*,* $(3, 0, 1)$*. Nodes representing no reachable integer combination (due to the strict partial order) are printed in gray.*

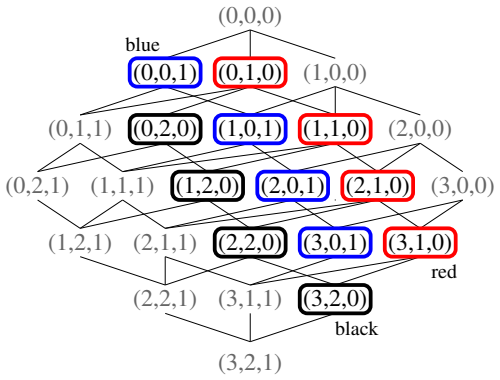


Figure 14: BTG for the combination of a WOP and a strict partial order.

6 Optimizations

As we have seen our algorithms do not produce minimal lattices. However, we propose some improvements for the lattice-based algorithms, which can be applied if the lattice is "nondense".

6.1 Reduction on Existing Values

In many cases, only a small number of domain values actually appear in a relation. For weak orders on numerical domains, this can have the effect that only some of the possible u_P values are met.

Example 14. *Consider* $P = \text{AROUND}_1(A, 4)$ *with trivial SV-semantics and let* $\text{dom}(A) = \{5, 10, 15, 20\}$ *be a numerical domain. Using Theorem 2 would lead to* $\max(P) = 16$ *levels, i.e., a BTG of size* 289*, even though there are only* four *different values in the domain with* $u_P(5) = 1$*,* $u_P(10) = 6$*,* $u_P(15) = 11$*, and* $u_P(20) = 16$*. A mapping of these values to* $0, 1, 2$*, and* 3 *would reduce the size of the lattice to* 16*.*

One way to gain information of unused domain values could be the use of a histogram or a simple B-tree index. With this we can find existing u_P values for the corresponding WOPs. Then we can efficiently reduce BTG sizes by "removing" unused nodes, leading to less memory needed and less effort for preference evaluation using lattice-based algorithms. The mapping of domain values to the few required u_P values could be done using minimal perfect hashing functions, leading to constant access times and minimum memory requirements [8, 9, 18]. Note, that this method is also valid for BETWEEN_d without the assumption of *single occupied equivalence classes*, cp. Section 4.2.

6.2 Data Structure

In the implementation of the original lattice Skyline algorithms *Hexagon* [33] and *LS-B* [30], the lattice is represented by an *array*, where each position stands for one node in the lattice, cp. Section 2.6. Following [33], the array based implementation needs at least a memory of $\left\lceil \frac{1}{4} \prod_{i=1}^{m} (\max(P_i) + 1) \right\rceil$ bytes, where the P_i's are the participating preferences.

Since our approach produces "nondense lattices", i.e., there are many empty nodes (cp. for example Figure 7 and 12a), using an array as data structure makes no sense. Therefore we propose the approach of level-based storage. An array models the *levels* (computed by u_P, Eq. 2) of the BTG. Then the *nodes* are stored in a HashMap or SkipList [34], cp. Figure 15.

Adding an element to the BTG means computing the level it belongs to and marking the node at the right position as *non-empty* or *dominated*. The advantage of the level-based storage using SkipLists in contrast to HashMaps lies in the reduced memory requirements, because we do not have to initialize the whole data structure in main memory. A node is initialized on-the-fly if it is marked as *non-empty* or *dominated*. Additionally, if each node in a level is

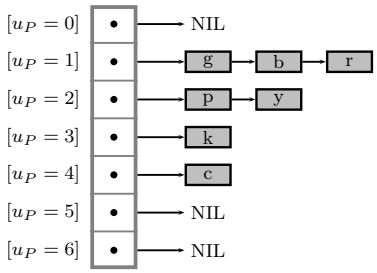


Figure 15: Level-based storage of the BTG in Figure 7 using SkipLists.

dominated, we can remove all nodes from the corresponding SkipList, mark the level-entry in the array as *dominated* and free memory.

Using a level-based representation of the BTG with a HashMap for each level, we have a constant access for each level and $\mathcal{O}(1)$ for the look-up in the HashMap, since we can use a perfect hash function due the known width of the BTG in each level, cp. [19]. In summary this leads to a runtime complexity of $\mathcal{O}(dV + dn)$, too. For the SkipList based BTG implementation we have $\mathcal{O}(dV + dn \log w)$, since operations on SkipLists are $\mathcal{O}(\log w)$ [34], where w is the number of elements in the SkipList, i.e., the width of the BTG in the worst case.

7 Related Work

Skyline queries and the more general concept of preferences are well-known in the database community since more than a decade. There are many algorithms to compute the *Skyline set*, cp. [6] for an overview. The most prominent algorithms are based on a block-nested-loop style tuple-to-tuple comparison (e.g., BNL [2], or [25, 7, 20]). Based on this several algorithms have been published for parallel Skyline computation [37, 29, 4] or utilizing an index structure [31, 28]. The BNL algorithm is the only one who can evaluate general strict partial orders due to its tuple-to-tuple comparison approach [5]. Most of the other algorithms are restricted to Pareto preferences, because they rely on the u_P value of an object.

Other algorithms exploit the lattice structure induced by a Skyline query, cp. e.g. [33, 30]. Instead of direct comparisons of tuples, a lattice structure represents the better-than relationships. There is also work on parallel preference computation exploiting the lattice [12] and the authors of [14] present how to handle high-cardinality domains. Also the work of [39, 27, 13] exploit the lattice to compute top-k (subspace) Skylines.

The most similar work to ours is [1], [3], and [36]. In [1, 17, 32] the authors create a spanning tree on a direct acyclic graph where each node is associated with an interval. The authors of [3] map all data points in a new space, where each partially ordered value is substituted by associated coordinates. Their aim

was Skyline processing on partially ordered domains. Also [36] handle partially ordered domains using topological sorting. In [35] the author presents a method for the decomposition of strict partial orders into fundamental preference constructs. For this the author studies which preference operators and operands are necessary to express any strict partial order. Non of these papers produce lattice structures, but 'arbitrary' graphs.

8 Conclusion and Future Work

In this paper we presented a method to embed all kinds of strict partial orders into lattices. As a consequence, existing lattice based algorithms can be applied to general strict partial orders and prior restrictions on weak order preferences and their combinations in Pareto preferences do not apply anymore.

As we have mentioned, the lattices we construct with our algorithm are not minimal. Nevertheless a reduction of the lattice size is wise as the size of a lattice for a Pareto preference grows exponentially with the lattice sizes of the underlying preferences. Hence our next step will be to address this and improve our algorithm so that it produces minimal lattices embedding strict partial orders. However, this could be a challenging task.

References

[1] R. Agrawal, A. Borgida, and H. Jagadish. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In *Proceedings of SIGMOD '89*, pages 253–262, New York, NY, USA, 1989. ACM.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of ICDE '01*, pages 421–430, Washington, DC, USA, 2001. IEEE.

[3] C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified Computation of Skylines with Partially-ordered Domains. In *SIGMOD '05*, pages 203–214, New York, NY, USA, 2005. ACM.

[4] S. Chester, D. Sidlauskas, I. Assent, and K. S. Bøgh. Scalable Parallelization of Skyline Computation for Multi-Core Processors. In *Proceedings of ICDE '15*, pages 1083–1094, 2015.

[5] J. Chomicki. Preference Formulas in Relational Queries. In *TODS '03: ACM Transactions on Database Systems*, volume 28, pages 427–466, New York, NY, USA, 2003. ACM Press.

[6] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline Queries, Front and Back. *SIGMOD*, 42(3):6–18, 2013.

[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *Proceedings of ICDE '03*, pages 717–816, 2003.

[8] R. J. Cichelli. Minimal Perfect Hash Functions Made Simple. *Commun. ACM*, 23(1):17–19, 1980.

[9] Z. J. Czech, G. Havas, and B. S. Majewski. An Optimal Algorithm for Generating Minimal Perfect Hash Functions. *Information Processing Letters*, 43(5):257–264, 1992.

[10] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK, 2nd edition, 2002.

[11] M. Endres and W. Kießling. Semi-Skyline Optimization of Constrained Skyline Queries . In *Proceedings of ADC '11*. ACS, 2011.

[12] M. Endres and W. Kießling. High Parallel Skyline Computation over Low-Cardinality Domains. In *Proceedings of ADBIS '14*, pages 97–111. Springer, 2014.

[13] M. Endres and T. Preisinger. Behind the Skyline. In *Proceedings of DBKDA '15*. IARIA, 2015.

[14] M. Endres, P. Roocks, and W. Kießling. Scalagon: An Efficient Skyline Algorithm for all Seasons. In *Proceedings of DASFAA '15*, 2015.

[15] P. Fishburn. Preference Structures and their Numerical Representation. *Theor. Comput. Sci.*, 217(2):359–383, 1999.

[16] P. C. Fishburn. Intransitive Indifference in Preference Theory: A Survey. *Operations Research*, 18(2):207–228, 1970.

[17] P. C. Fishburn. Interval graphs and interval orders. *Discrete Mathematics*, 55(2):135 – 149, 1985.

[18] E. A. Fox, L. S. Heath, Q. F. Chen, and A. M. Daoud. Practical Minimal Perfect Hash Functions for Large Databases. *Commun. ACM*, 35(1):105–121, 1992.

[19] R. Glück, D. Köppl, and G. Wirsching. Computational Aspects of Ordered Integer Partition with Upper Bounds. In *SEA '13: 12th International Symposium on Experimental Algorithms*, pages 79–90, 2013.

[20] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and Analyses for Maximal Vector Computation. *The VLDB Journal*, 16(1):5–28, 2007.

[21] H. Han, H. Jung, H. Eom, and H. Y. Yeom. An Efficient Skyline Framework for Matchmaking Applications. *J. Netw. Comput. Appl.*, 34(1):102–115, Jan. 2011.

[22] W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of VLDB '02*, pages 311–322, Hong Kong, China, 2002. VLDB.

[23] W. Kießling. Preference Queries with SV-Semantics. In *Proceedings of COMAD '05*, pages 15–26, Goa, India, 2005. Computer Society of India.

[24] W. Kießling, M. Endres, and F. Wenzel. The Preference SQL System - An Overview. *Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society*, 34(2):11–18, 2011.

[25] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proceedings of VLDB '02*, pages 275–286.

[26] J. Lee and S. w. Hwang. BSkyTree: Scalable Skyline Computation Using a Balanced Pivot Selection. In *Proceedings of EDBT '10*, pages 195–206, NY, USA, 2010. ACM.

[27] J. Lee, G. w. You, and S. w. Hwang. Personalized Top-k Skyline Queries in High-Dimensional Space. *Information Systems*, 34(1):45–61, Mar. 2009.

[28] K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the Skyline in Z Order. In *Proceedings of VLDB '07*, pages 279–290. VLDB Endowment, 2007.

[29] S. Liknes, A. Vlachou, C. Doulkeridis, and K. Nørvåg. APSkyline: Improved Skyline Computation for Multicore Architectures. In *Proc. of DASFAA '14*.

[30] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *Proceedings of VLDB '07*, pages 267–278, 2007.

[31] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *Proceedings of SIGMOD '03*, pages 467–478. ACM, 2003.

[32] M. Pirlot and P. Vincke. *Semi Orders*. Kluwer Academic, Dordrecht, 1997.

[33] T. Preisinger and W. Kießling. The Hexagon Algorithm for Evaluating Pareto Preference Queries. In *Proceedings of MPref '07*, 2007.

[34] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Commun. ACM*, 33(6):668–676, 1990.

[35] P. Roocks. Preference Decomposition and the Expressiveness of Preference Query Languages. In *Proceedings of MPC '15*, volume 9129 of *LNCS*, pages 71–92. Springer, 2015.

[36] D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically Sorted Skylines for Partially Ordered Domains. In *Proceedings of ICDE '09*, pages 1072–1083, Washington, DC, USA, 2009. IEEE.

[37] J. Selke, C. Lofi, and W.-T. Balke. Highly Scalable Multiprocessing Algorithms for Preference-Based Database Retrieval. In *Proceedings of DASFAA '10*, volume 5982 of *LNCS*, pages 246–260. Springer, 2010.

[38] K. Stefanidis, G. Koutrika, and E. Pitoura. A Survey on Representation, Composition and Application of Preferences in Database Systems. *ACM TODS*, 36(4), 2011.

[39] Y. Tao, X. Xiao, and J. Pei. Efficient Skyline and Top-k Retrieval in Subspaces. *IEEE TKDE*, 19(8):1072–1088, 2007.