# Skyline Groups Are Ideals. An Efficient Algorithm for Enumerating Skyline Groups

Simon Coumes[1], Tassadit Bouadi[2](✉), Lhouari Nourine[3], and Alexandre Termier[2]

[1] ENS Rennes, 35170 Bruz, France
simon.coumes@ens-rennes.fr
[2] Univ Rennes, Inria, CNRS, IRISA, 35000 Rennes, France
{tassadit.bouadi,alexandre.termier}@irisa.fr
[3] Univ Clermont Auvergne, 63000 Clermont-Ferrand, France
lhouari.nourine@uca.fr

**Abstract.** Skyline queries are multicriteria queries that are of great interest for decision applications. Skyline Groups extend the idea of skyline to groups of objects. In the recent years, several algorithms have been proposed to extract, in an efficient way, the complete set of skyline groups. Due to the novelty of the skyline group concept, these algorithms use custom enumeration strategies. The first contribution of this paper is the observation that a skyline group corresponds to the notion of *ideal* of a partially ordered set. From this observation, our second contribution consists in proposing a novel and efficient algorithm for the enumeration of all ideals of a given size $k$ (*i.e.* all skyline groups of size $k$) of a poset. This algorithm, called GenIdeals, has a time delay complexity of $O(w^2)$, where $w$ is the width of the poset, which improves the best known time output complexity for this problem: $O(n^3)$ where $n$ is the number of elements in the poset. This work present new theoretical results and applications on skyline queries.

**Keywords:** Skyline queries · Ideal enumeration · Time delay complexity

## 1 Introduction

In decision making, one often wants to optimize simultaneously several characteristics. For example, consider a soccer coach who wants to recruit, into her team, a player who has both a low miss rate (corresponding to a high accuracy) and doesn't take long to cross the field. These characteristics (*i.e.* dimensions) are often multiple and conflicting: there is rarely a single solution optimizing all the characteristics at the same time. *Skyline queries* [3] solve this problem by considering the "best compromises" between the different dimensions. More formally, in a multidimensional space where the dimension domains are ordered (totally or partially), *skyline queries* return the objects that are not dominated by any other object. An object dominates another object, *if it is as good or better in all dimensions and strictly better in at least one dimension.* This notion of dominance is also called *Pareto dominance.*

In the soccer example, results of skyline queries may be: (i) the best goal-scorer (whatever her running speed), (ii) the best runner (whatever her miss rate), or (iii) a player being average on both criteria, with no other player both missing less and taking less time to cross the field.

An interesting and challenging problem arises when, in some applications, users are interested in capturing skyline *groups* instead of points. This is for example the case when looking for the best soccer *team* and not the best soccer player. The Pareto dominance concept of one object over another is not directly applicable to the notion of groups of objects. For example, it is obvious that the best soccer team may not correspond to the group of the best individual players, nor necessarily to the group of the most average players. Recent works [6, 8,15,18] have considered the issue of *skyline group computation* by extending the dominance relation between points to groups of points: this is the *group dominance relation*, also called *g-dominance*, which allows comparison between groups of the same size. The *g-dominance* relation allows comparison between groups of the same size, and is defined by [8] as: "*Given two different groups $G$ and $G'$ with $k$ points, we say that $G$ g-dominates $G'$, if we can find a permutation of the $k$ points for $G$ and $G'$, such that either $p_i$ dominates $p'_i$* or $p_i$=$p'_i$ *for all* $1 \leq i \leq k$*, and for at least one $i$, $p_i$ dominates $p'_i$*".

Enumerating skyline groups defined by such group dominance relation is challenging given the huge size of the search space considered: in a set of $n$ points, there are $\binom{n}{k}$ possible group skylines of size $k$. This prompted the need to design efficient enumeration algorithms dedicated to the discovery of group skylines. In the papers proposing g-dominance [8,9], the authors proposed to exploit a novel structure based on skyline layers (*i.e.* models the dominance links between the points), and two efficient search strategies to enumerate the skyline groups. Other works studied the g-dominance relation and proposed algorithmic improvements [7,15,17,19]. The state of the art approach is G-MDS [15], which is based on a structure called *minimum dominance graph (MDG)*, a directed acyclic graph representing the dominance relation among relevant points.

In this work, our objective is to get a finer understanding of the space of skyline groups as defined by the g-dominance relation, in order to propose an algorithm exploring only the space of solutions. By going back to enumeration theory concepts, we could show that the skyline group notion corresponds to the well known concept of *ideal.* This allows us to propose an elegant and efficient algorithm to compute skyline groups, that does not visit any unnecessary group. Furthermore, the algorithm that we propose to enumerate ideals of size $k$ improves the state of the art for ideal enumeration, having a time delay complexity of $O(w^2)$ (with $w$ the posets' width), where the best know time output complexity for this problem is $O(n^3)$ per ideal [16] (with $n$ the number of elements in the poset).

We first recall that an ideal (or downset) of a poset is a subset that is closed by the order relation $\leq$, *i.e.* an ideal containing an element $x$, contains all elements inferior or equal to $x$. The notion of ideals of partially ordered sets has been found many times in several applications as (scheduling [13], verification of distributed systems [2]). Listing or enumerating all ideals of a poset simply means outputing them one after the other. Many algorithms have been proposed in the literature [1,4,10,12,13]. But all these algorithms cannot be used for our purposes, since they list all ideals. In addition their adaptation does not allow to have a good complexity.

In this work, we show that given the classical dominance relation for points of the data, group skylines of size $k$ correspond exactly to ideals of size $k$ for this relation (Proposition 1). We are then interested in the enumeration of ideals of a given size $k$. To the best of our knowledge the only works for enumerating ideals of a given size have been considered in [5] for particular cases of posets, and Wild's algorithm in [16] for the general case. We identify a directed graph whose vertices are ideals of size $k$ and such that there is an edge between two ideals if there is a small transformation to obtain one from the others. We then propose an efficient algorithm to enumerate all ideals of a given size $k$ using polynomial space.

The rest of the paper is organized as follows. Section 2 introduces the basic concepts related to skyline queries and the concept of skyline group. We present and detail, in Sect. 3, the established correspondence between key concepts from the field of skyline queries and the field of partial order theory. In Sect. 4, we develop the formal aspects, highlight new and useful properties, and present a way to organize the space of ideals (*i.e.* skyline groups) as a Depth First Search (DFS) tree. Our algorithm GenIdeals for enumerating $k$-ideals based on this tree-shaped structuration of the space is then discussed in Sect. 5. In this same section, we present the detailed complexity analysis of GenIdeals, showing its $O(w^2)$ complexity for $k$-ideals enumeration. Section 6 concludes the paper.

## 2 Preliminaries

In this section, we introduce the concept of skylines and extend it to the concept of group skylines. We then review a few useful results from the literature.

The various definitions are illustrated using the example of Table 1 which describes proposals for hotels according to the dimensions Price, Distance from the beach, and Distance from transportation. Without any loss of generality, we assume that it is always better if the values of the attributes are low.

**Definition 1 (Pointset).** *A pointset is a set of same size tuples of real numbers. We usually write the pointset using the letter $D$ and assume said numbers to be positives. The elements of the pointset are called points, the elements of the points are its attributes.*

*Example 1.* In Table 1, $D = \{(10,1,4),(10,2,4),(10,4,1),(20,1,4), (40,1,1), (40,5,1), (50,4,1)\}$ is a pointset defined in a 3-dimensional space $\mathrm{F} =$ (Price, Distance, Transportation). The domain of each attribute is totally ordered.

**Table 1.** A set of hotels

| Hotel ID | Price | Distance | Transportation |
|---|---|---|---|
| a | 10 | 1 | 4 |
| b | 10 | 4 | 4 |
| c | 10 | 1 | 5 |
| d | 20 | 2 | 1 |
| e | 40 | 1 | 2 |
| f | 40 | 3 | 2 |
| g | 50 | 2 | 3 |

**Definition 2 (Point domination).** *A point $p$ dominates a different point $p'$, denoted by $p \prec p'$, if $p$ is lower or equal to $p'$ on any attribute and $p$ is strictly lower than $p'$ on at least one attribute. We write $p \preceq p'$ to say "either $p$ dominates $p'$ or $p = p'$".*

*Example 2.* In Table 1, we have $(20, 2, 1) \preceq (50, 2, 3)$, since the price value 20 is lower than 50 and the other attribute values are equal. We also note that $(10, 1, 4) \npreceq (20, 2, 1)$, since the transportation value 4 is higher than 1.

**Definition 3 (Skyline).** *The Skyline of a pointset $D$ is the set of all points in $D$ that are not dominated by any other point.*

## 2.1 Skyline Groups

In the hotel example (Table 1), one may consider the case of a travel agency, that wants to pre-book exactly $k$ rooms (supposed in $k$ different hotels for sake of simplicity). Each room has the same price and distances characteristics as before: among the many rooms available ($k << |D|$), the travel agency wants to identify the *best groups*, able to satisfy the needs of many potential customers.

The definition of Group domination and skyline group used in this paper was first introduced in [8]. It is the notion that, in our opinion, is the best adaptation of the idea of skylines to groups of points.

**Definition 4 (Group domination).** *A group (set) $G$ containing $k$ points dominates another group $G'$ of size $k$, denoted by $G \prec_g G'$, if and only if there is a bijection $f$ from $G$ to $G'$ such that: $\forall p \in G$, $p \preceq f(p) \wedge \exists p \in G$ s.t. $p \prec f(p)$.*

*Example 3.* For the pointset in Table 1, we have $\{a, e, g\} \prec_g \{b, f, g\}$, because $a \prec b$, $e \prec f$, and $g \preceq g$.

**Definition 5 (Skyline group).** *We say that a group $G$ of size $k$ is a skyline group if and only if it is dominated by no other group.*

*Example 4.* In Table 1, $\{a, c, e\}$ is a skyline group of size 3. Since the three points are skyline points, it is easy to verify that this group cannot be dominated by any other group of size 3.

**Definition 6 (g-skyline).** *The g-skyline of size $k$ of a pointset $D$, denoted $S_k$, is the set of all groups of $k$ elements of $D$ that are not dominated by any other group, i.e. the set of all the skyline groups of size $k$ of $D$.*

*Example 5.* In Table 1, $S_4 = \{\{a,b,c,d\},\{a,b,c,e\},\{a,b,d,e\},\{a,c,d,e\},\{a,c,e,g\},\{a,c,e,f\},\{c,e,f,g\}\}$

We also recall two useful notations for discussing partially ordered sets (*i.e.* posets) which we will need later. Those are the notions of *ideal* and that of the *width of a poset.*

**Definition 7 (Ideal).** *Given $(E,\leq)$ a partially ordered set, a subset $I$ of $E$ is an ideal if and only if: $\forall(x,y)\in E\times I$, if $x\leq y$, then $x\in I$.*
*In other words, $I$ is closed by the order relation $\leq$.*

**Definition 8 (Poset width).** *Given $(E,\leq)$ a poset, the width of that poset is the size of the maximal subset $I$ of $E$ such that: $\forall(x,y)\in I^2$, $x\not\leq y$. $I$ is called a maximal antichain of $(E,\leq)$.*

## 3 Skyline Groups Are Ideals

In this section, we present our first contribution: a new correspondence between concepts from the field of skyline queries and concepts from the field of partial order theory. Namely, we see that a skyline group is an ideal. This correspondence is in our eyes the most important in this paper. In [15], the authors introduced the concept of *Unit group* as follow:

**Definition 9 (Unit group).** *The unit group of point $p$ is the set of all points that dominate it plus $p$. It is written $u(p)$. Formally : $u(p)=\{p'\in D\mid p'\preceq p\}$.*

The concept of *unit group* is equivalent to the concept of *principal ideal* in a partially ordered set (poset). For example, in Table 1, $u(b)=\{a,b\}$.

**Proposition 1.** *Consider a group $G$ of size $k$. The following properties are equivalent: (1) $G$ is a skyline group, (2) $\cup_{p\in G}\, u(p)=G$, (3) $\forall p\in G$, $u(p)\subseteq G$, (4) and $G$ is an Ideal of $(D,\preceq)$.*

Because of the third property of Proposition 1, we know that a point with a unit group of size strictly superior to $k$ cannot belong to a skyline group. Because domination is transitive, unit group size increases with domination (*i.e.*, $p\preceq p' \implies u(p)\subseteq u(p')$). Therefore, when searching for skyline groups, all points with an unit group of size strictly superior to $k$ can be entirely removed without affecting in any way the information at hand on the other points.

The fourth property has important implications: it means that any algorithm suitable for the enumeration of ideals of size $k$ of a poset is also suitable to be used for the enumeration of skyline groups of size $k$ of a pointset, using $\preceq$ as an order relation.

In the rest of the paper, we draw on Proposition 1 and consider skyline groups on a given pointset $D$ to be defined as *the ideals of the poset* $(D, \preceq)$. Thus, our problem becomes the enumeration of all ideals of a given size $k$ of a poset.

We consider the particular case of skyline queries for our explanations and will make references to the canonical lexicographic order on points. However, all that follows is compatible with any poset with a topological order. Because calculating a topological order for a poset can be done in $O(n^2)$ with $n$ the size of the poset, everything bellow this point and especially the algorithm we present can be adapted to list the ideals of size $k$ of any given poset.

In the next section we will describe new properties to explore the graph of skyline groups. These properties allowed us to propose a simple and efficient $k$-ideal (*i.e.* skyline groups of size $k$) enumeration algorithm.

## 4 The Tree of Skyline Groups

Given a pointset $D$ of dimension $d$ and an integer $k$, we want to enumerate the elements of the set $S_k$ of its skyline groups of size $k$, *i.e.* its g-skyline of parameter $k$. We identify a rooted covering tree $T = (S_k, \mathcal{P})$ which leads us to an efficient algorithm to search this tree in polynomial time and space. But first, we must introduce a few notions. We begin by some order relations on points and groups.

We define the usual lexicographical order relation $\leq$ on points of $D$ as follows:

**Definition 10.** *$p = (x_1, x_2, ..., x_d) \leq q = (y_1, y_2, ..., y_d)$ if and only if $p = q$ or there is $i \in \{1, 2, ..., d\}$ such that for all $1 \leq j < i$, $x_j = y_j$ and $x_i < y_i$.*

It has the following interesting property.

*Property 1.* $\leq$ is a topological ordering for the dominance relation.

Because the notion of skyline group only depends on the dominance relation between points and not on their coordinates, we can rename points according to the lexicographic order and only remember which points dominate which ones.

*Example 6.* In Table 1, we have the following points order (according to $\leq$): $p_1 = (10, 1, 4)$, $p_2 = (10, 4, 4)$, $p_3 = (10, 1, 5)$, $p_4 = (20, 2, 1)$, $p_5 = (40, 1, 2)$, $p_6 = (40, 3, 2)$, $p_7 = (50, 2, 3)$

Fig. 1 shows the point domination relation between the points defined above. Such graphical representation helps to quickly identify skyline groups. Then, we can easily notice that $(p_1, p_2, p_3, p_6)$ is not a skyline group because $p_4$ is not in it and $p_4 \preceq p_6$. Also, $(p_1, p_2, p_3, p_4)$ is a skyline group since no point could be replaced by another that dominates it.

Since all elements of $S_k$ have the same size, a group $G \in S_k$ is represented by a tuple $(p_1, p_2, ..., p_k)$ such that $p_1 \leq p_2 \leq ... \leq p_k$. In the same way as for $D$, we define a lexicographical ordering $\trianglelefteq$ on the set $S_k$.

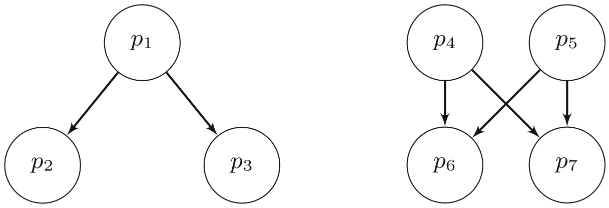

**Fig. 1.** Domination graph between points of Table 1

*Example 7.* In Table 1, we have the following skyline groups of size 4 (ordered according to $\leq$): $G_0 = (p_1, p_2, p_3, p_4)$, $G_1 = (p_1, p_2, p_3, p_5)$, $G_2 = (p_1, p_2, p_4, p_5)$, $G_3 = (p_1, p_3, p_4, p_5)$, $G_4 = (p_1, p_4, p_5, p_6)$, $G_5 = (p_1, p_4, p_5, p_7)$, $G_6 = (p_4, p_5, p_6, p_7)$

In the following examples, we fix the size of skyline groups to 4.

We denote by $G_0$ the smallest group according to the lexicographical ordering $\trianglelefteq$. We then define the parent relation.

**Definition 11 (Parent).** *Let $G$ be a group in $S_k$ with $G \neq G_0$. The parent of $G$, denoted by $Parent(G)$, is obtained from $G$ by deleting the largest element $b$ in $G$ (w.r.t $\leq$) and adding the smallest element $a \in D \backslash G$.*

**Proposition 2.** *The parent of a skyline group is a skyline group of the same size.*

We say that $G'$ is a *child* of $G$ if $Parent(G') = G$, and denote by $Children(G)$ the set of all the children of $G$. Let $\mathcal{T} = (S_k, Parent)$ be the directed graph whose vertices are elements of $S_k$ and edges correspond to the parent relation.

**Proposition 3.** *The directed graph $\mathcal{T} = (S_k, Parent)$ is a tree rooted at $G_0$*

*Proof.* Let $G \neq G_0$ be a group in $S_k$. We show that there is a unique path $G_0 \leq G_1 \leq G_2 \leq ... \leq G_m = G$ such that $Parent(G_i) = G_{i-1}$ for $0 < i \leq m$. Let $G' = Parent(G)$ with $G' = (G \setminus \{b\}) \cup \{a\}$. Since $G \neq G_0$, the smallest point not in $G$ is smaller than the highest point in $G$. Hence, $a < b$ and $G' < G$. So for every group different from $G_0$, its parent is smaller for $\leq$ than it. By repetitive application of the relation parent we inevitably reach $G_0$ since $S_k$ is finite. $\square$

Our algorithm will simply run a tree exploration. However, we need to be able to enumerate the children of a given group.

We assume that the points in $D$ are numbered according to the lexicographic order $\leq$. We have $p_1 \leq p_2 \leq ... \leq p_n$ such that $G_0 = \{p_1, p_2, ..., p_k\}$.

**Definition 12 (Starting prefix).** *Let $G$ be a group in $S_k$. The starting prefix of $G$, denoted by $SPrefix(G)$ is the longest common prefix of $G$ with $G_0$.*

*Example 8.* Starting prefixes are shown below in Fig. 2 in bold.

In the following we show that the children of a given group $G$ are exactly the groups that can be obtained by removing one point of its starting prefix and adding another point greater than any point of $G$ such that the new group is an ideal.

**Proposition 4.** *Let $G_1$ and $G_2$ be two skyline groups. $G_2$ is a child of $G_1$ iff $G_2$ is obtained by removing one element of the starting prefix of $G_1$ from $G_1$ and adding another element higher than any element of $G_1$.*

*Proof.* Let $G_1$ and $G_2$ be two groups in $S_k$.
Suppose that $G_2 = (G_1 \backslash \{a\}) \cup \{b\}$ where $a$ is a point of the starting prefix of $G_1$ and $b \notin G_1$ is greater than every elements of $G_1$. We show that $G_1 = Parent(G_2)$. Clearly $b$ is the largest element in $G_2$. Moreover by definition of the starting prefix, $a$ is the smallest point (for $\leq$) that is not in $G_1 \backslash \{a\} = G_2 \backslash \{b\}$. Hence $a$ is the smallest element that can be added to $G_2 \backslash \{b\}$, and thus $G_1 = Parent(G_2)$. Now, suppose that $G_2$ is child of $G_1$, i.e. $G_1 = Parent(G_2)$. By definition of the relation Parent, $G_1$ is obtained from $G_2$ by deleting the largest$_\leq$ element $b$ in $G_2$ and adding the smallest$_\leq$ element $a$ not in $G_2$. We show that $a$ belongs to the starting prefix of $G_1$. For contradiction, suppose there is $a'$ not in $G_1$, $a' < a$ and $a'$ belongs to $G_0$. Then $a' \notin G_2 \backslash \{b\}$ and $b \nleq a'$. Thus $a'$ can be added to $G_2 \backslash \{b\}$ which contradict that $a$ is the smallest element that can be added. □

Moreover the depth of a group $G$ in the tree $\mathcal{T} = (S_k, Parent)$ is equal to the size of $\mid G_0 \backslash G \mid$. So the depth of a leaf in the tree is bounded by $k$. Figure 2 shows the tree $\mathcal{T}$ for our example set of points.

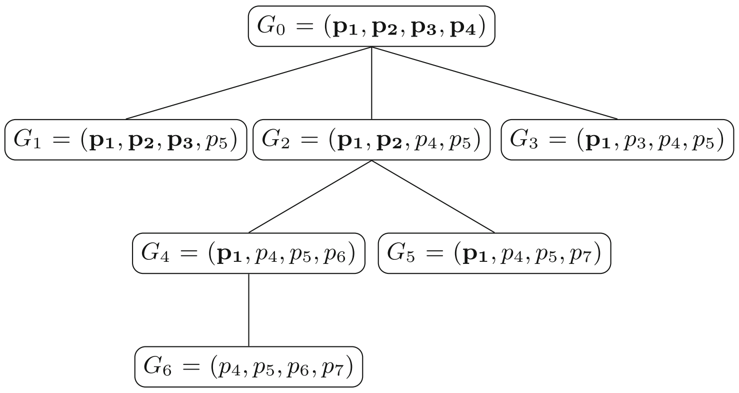
$G_0 = (\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4})$
$G_1 = (\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, p_5)$
$G_2 = (\mathbf{p_1}, \mathbf{p_2}, p_4, p_5)$
$G_3 = (\mathbf{p_1}, p_3, p_4, p_5)$
$G_4 = (\mathbf{p_1}, p_4, p_5, p_6)$
$G_5 = (\mathbf{p_1}, p_4, p_5, p_7)$
$G_6 = (p_4, p_5, p_6, p_7)$

**Fig. 2.** Example tree of skyline groups

## 5 Algorithm

### 5.1 First Look at Our Algorithm

Our algorithm will be a DFS (Depth First Search) on the tree $\mathcal{T}$. First we show how to generate all the children of a given group $G$. Recall that a child is obtained by exchanging two elements (deleting one and adding another one).

Let $p_1, p_2, ..., p_n$ be a topological order of $(D, \leq)$ and $G$ a group in $S_k$. We use $max(G)$ to denote the largest point in $G$ w.r.t $\leq$. We denote by $Pred(p)$ the points that dominate a given point $p$ and by $Succ(p)$ the points it dominates.

Let $SPrefix(G)$ be the set of elements in the starting prefix in $G$ and $SPrefixToDel(G)$ the set of points in $SPrefix(G)$ that are maximal w.r.t. $\leq$ in $G$, *i.e* all points $a \in G$ such that $G \backslash \{a\}$ is still a skyline group (of size $k-1$). No other elements of $G$ can be removed to create a child of $G$.

We also consider the set of all potential candidates that could be added to $G$ after deleting one element from $SPrefixToDel(G)$. Let $CandToAdd(G)$ be the set of points greater than $max(G)$ that are minimal in $D \backslash G$ w.r.t $\preceq$. The set of candidates when deleting an element $a \in SPrefixToDel(G)$ from $G$ is obtained from $CandToAdd(G)$ by removing from it the successors of $a$ in $D$, *i.e.* $CandToAdd(G, a) = CandToAdd(G) \backslash Succ(a)$.

**Theorem 1.** *Let $G$ be a group in $S_k$. Then $Children(G) = \{(G \backslash \{a\}) \cup \{b\} \mid a \in SPrefixToDel(G), b \in CandToAdd(G, a)\}$.*

*Proof.* Let $G$ be a group in $S_k$ and let $G' \in \{(G \backslash \{a\}) \cup \{b\} \mid a \in SPrefix ToDel(G), b \in CandToAdd(G, a)\}$. We show that $Parent(G') = G$. We fix $G' = (G \backslash \{a\}) \cup \{b\}$ with $a \in SPrefixToDel(G)$ and $b \in CandToAdd(G, a)$. Since $a$ belongs to the starting prefix of $G$ then for all $a' \leq a$ we have $a'$ in $G$. Thus $a$ is the smallest point not in $G'$. Furthermore, by definition of $CandToAdd(G, a)$ we have $b \geq max(G)$ which implies that $b$ is the largest point in $G'$. Hence $G$ is the parent of $G'$. Conversely let $G$ and $G'$ be two groups ($G' \neq G_0$) such that $G = Parent(G')$, we show that $G' \in \{(G \backslash \{a\}) \cup \{b\} \mid a \in SPrefixToDel(G), b \in CandToAdd(G, a)\}$. Again, we fix $G = (G' \backslash \{b\}) \cup \{a\}$ with $b$ the largest point in $G'$ and $a$ the smallest one not in $G'$. We show that $a \in SPrefixToDel(G)$. $a$ is the smallest point not in $G'$ and therefore $a$ belongs to the starting prefix of $G$. Since $G \cap G'$ is also an ideal, we conclude that $a$ is maximal in $G$ for $\preceq$ and thus $a \in SPrefixToDel(G)$. Since $b$ is maximal in $G'$ it is higher than any point in $G$. Because $b \notin G$ and because it can be added to $G$, we have $b \in min(D \backslash G)$. Hence $b \in CandToAdd(G)$. Moreover because $b$ can be added to $G \backslash \{a\}$, we know $a \npreceq b$ (*i.e.* $b \notin Succ(a)$) and thus $b \in CandToAdd(G, a)$. □

*Remark 1.* It is worth noticing that if $G' \in Children(G)$ with $G' = (G \backslash \{a\}) \cup \{b\}$, then $a$ cannot be added and $b$ cannot be deleted from any group in the subtree rooted at $G'$. In other words, in any path of the execution tree, any point can be deleted or added at most once. This is because all deleted elements belong to the starting prefix (*i.e.* elements ranging from $p_1$ to $p_k$), and all added ones are greater than $p_k$ in the topological ordering $\leq$.

We now describe an algorithm called GenIdeals (Algorithm 2) that takes the starting prefix of a group $G$ in $S_k$, the lists $SPrefixToDel(G)$ and $CandToAdd(G)$ and outputs all groups in the subtree of $\mathcal{T}$ rooted at $G$. The first call is GenIdeals($SPrefix(G_0), SPrefixToDel(G_0)$, $CandToAdd$ $(G_0)$ , $G_0$) which lists all groups in $S_k$.

## 5.2 Detailed Explanation of GenIdeals: Data structures and algorithms

The GenIdeals algorithm uses the following global data structures:

- $Succ[1..n]$ an array where $Succ[i]$ is a list of successors of $p_i$ in $(D, \leq)$.
- $Pred[1..n]$ an array where $Pred[i]$ is a list of predecessors of $p_i$ in $(D, \leq)$.
- $Tlook[1..n]$ an array where $Tlook[i] = 1$ if the point $p_i$ is present and $Tlook[i] = 0$ otherwise.
- $PredCount[1..n]$ an array where $PredCount[i]$ is the number of predecessors of $p_i$ not in the current group $G$. This allows us to check in $O(1)$ if a point $p_i$ is ready to be added, *i.e.* minimal in $(D \backslash G)$.
- $SuccCount[1..n]$ an array where $SuccCount[i]$ is the number of successors of $p_i$ in the current group. This counter let us check in $O(1)$ if a point $p_i$ is maximal in the current group.

*Remark 2.* The sizes of $Succ[i]$, $Pred[i]$, $SPrefixToDel(G)$, $CandToAdd(G)$ and $Children(G)$ are bounded in $w$ the width of $(D, \leq)$.

The algorithm GenIdeals_Main is split into two phases. We first call the *initialization* process to compute $G_0$ and then lists $SPrefixToDel(G_0)$ and $CandToAdd(G_0)$. Then we start the recursive process GenIdeals that corresponds to the core of the algorithm.

We describe the function Initialisation (Algorithm 1) briefly here. The first **for** loop initializes the arrays $Tlook$ and $PredCount[1..n]$ and adds those points that are minimal in $(D, \leq)$ and greater than $p_k$ to the list $Min$. This loop can be achieved in $O(n + m)$ time complexity where $n$ is the number of points and $m$ the size of the lists $Pred$.

The second **for** loop computes $G_0$ and updates the counter $PredCount$ and the list $Min$. At the end of this loop, $Min$ contains the minimal points in $D \backslash G_0$ and $SuccCount[i]$ the number of successor points of $P_i$ in $G_0$. The time spent by this loop is bounded by $O(kw)$.

The third **for** loop computes the maximal points in $G_0$ in $O(k)$. Thus the total complexity of the algorithm initialization is bounded by $O(n + m + kw)$ which is less than $O(n^2)$. Please mind that initialization isn't purely described by its output, it also initializes global variables in the form of the arrays $PredCount$ and $SuccCount$.

We now describe in details the recursive algorithm GenIdeals (Algorithm 2). It takes a group $G$ in $S_k$, the lists $SPrefixToDel(G)$ and $CandToAdd(G)$ and outputs all groups in the subtree of $\mathcal{T}$ rooted in $G$.

**Algorithm 1.** Initialization

```
 1: $Min = \emptyset$
 2: **for** $i = 1$ to $n$ **do**
 3:     $Tlook[i] = 0$
 4:     $PredCount[i] = sizeof(Pred[i])$
 5:     $SuccCount[i] = 0$
 6:     **if** $PredCount[i] = 0$ and $i > k$ **then**
 7:         Add $i$ to $Min$
 8: { Now we initialize $G_0$}
 9: $G_0 = \emptyset$
10: **for** $i = 1$ to $k$ **do**
11:     Add $p_i$ to $G_0$
12:     **for** $v \in Succ[i]$ **do**
13:         $PredCount[v] = PredCount[v] - 1$
14:         **if** $v \leq k$ **then**
15:             $SuccCount[i] = SuccCount[i] + 1$
16:         **if** $PredCount[v] = 0$ and $v > k$ **then**
17:             Add $v$ to $Min$
18: $CandToAdd = Min$
19: $SPrefixToDel = \emptyset$
20: **for** $i = 1$ to $k$ **do**
21:     **if** $SuccCount[i] = 0$ **then**
22:         Add $i$ to $SPrefixToDel$
23: **return** $(SPrefixToDel, CandToAdd, G_0)$
```

The outer loop **while** considers all points in the starting prefix of $G$ that might be removed to create children of $G$. For each such $a \in SPrefixToDel(G)$, we delete $a$ and compute the list $L = CandToAdd(G, a)$. This is done using the algorithm UpdateCandToAdd1 (Algorithm 3). First we insert points in $Succ(a)$ into $Tlook$, and for each $v \in CandToAdd(G)$: we check if $Tlook[v] = 0$, then we add $a$ to the output, else we delete it from $Tlook$ in order to keep the array $Tlook$ empty when entering and exiting the algorithm UpdateCandToAdd1. The time complexity of algorithm UpdateCandToAdd1 is bounded by $O(w)$. Note that the list $L$ may be empty for all $a \in SPrefixToDel(G)$. So the worst case is when $G$ is a leaf and in this case the total cost is bounded by $O(w^2)$.

Then, the inner loop **while** takes any point $b$ in the list $L$, calls the function $Print$ and prepare the parameters for the new group $G' = (G \backslash \{a\}) \cup \{b\}$. For the former step, we use two update functions UpdateCandToAdd2 (Algorithm 4) and UpdateSPrefixToDel$(a, b, SPrefixtoDel)$ (Algorithm 5). The first function UpdateCandToAdd2 adds the new candidates to be added when deleting the point $b$. The second one computes the starting prefix of $G'$ and $SPrefixtoDel$, *i.e.* maximal elements in $G'$ that can be deleted from $G'$.

The time complexity needed by the algorithms UpdateCandToAdd2 and UpdateSPrefixToDel is bounded by $O(w)$.

**Algorithm 2.** GenIdeals(SPrefixToDel, CandToAdd, G)

```
1: while $SPrefixToDel$ is not empty do
2:     $a$ = Delete and return an element in $SPrefixToDel$
3:     $L$ = UpdateCandToAdd1$(a, CandToAdd)$
4:     while $L$ is not empty do
5:         $b$ = Delete and return an element in $L$
6:         $G' = (G \backslash \{a\}) \cup \{b\}$
7:         Print $(G')$
8:         $SprefD$ =UpdateSPrefixToDel$(a, b, SPrefixetoDel)$
9:         $Cand$ =UpdateCandToAdd2$(b, L)$
10:        GenIdeals$(SprefD, Cand, G')$
11:        Cancel changes to $PredCount$ and $SuccCount$ done to add $b$
12:    Cancel changes to $PredCount$ and $SuccCount$ done to remove $a$
```

**Algorithm 3.** UpdateCandToAdd1$(a, CandToAdd)$

```
1: $L = \emptyset$
2: for $v \in Succ[a]$ do
3:     $Tlook[v] = 1$
4: for $v \in CandToAdd$ do
5:     if $Tlook[v] = 0$ then
6:         Add $v$ to $L$
7:     else
8:         $Tlook[v] = 0$
9: return $L$
```

**Algorithm 4.** UpdateCandToAdd2$(b, CandToAdd)$

```
1: for $v \in Succ[b]$ do
2:     $PredCount[v] = PredCount[v] - 1$
3:     if $PredCount[v] = 0$ and $v > b$ then
4:         Add $v$ to $CandToAdd$
5: return $CandToAdd$
```

**Algorithm 5.** UpdateSPrefixToDel $(a, b, SPrefixetoDel)$

```
1: for $v \in Pred[a]$ do
2:     $SuccCount[v] = SuccCount[v] - 1$
3:     if $SuccCount[v] = 0$ then
4:         Add $v$ to $SPrefixetoDel$
5: for $v \in Pred[b]$ do
6:     if $v \leq k$ then
7:         $SuccCount[v] = SuccCount[v] + 1$
8:         Delete $v$ from $SPrefixetoDel$
9: return $SPrefixetoDel$
```

Finally, the main algorithm GenIdeals_main is presented in Algorithm 6. It simply consists of a call to Initialization followed by the first call to the core function GenIdeals.

**Algorithm 6.** GenIdeals_main

```
1: $(SPrefixToDel, CandToAdd, G_0)$ =Initialization()
2: GenIdeals$(SPrefixToDel, CandToAdd, G_0)$
```

**Theorem 2.** *The algorithm* GenIdeals_main *lists all groups of size* $k$ *in* $O(w^2)$ *delay and polynomial space.*

*Proof.* The correctness of the algorithm GenIdeals_main comes from Theorem 1. The complexity of each call to GenIdeals is dominated by the running time of the algorithm UpdateCandToAdd1 which is bounded by $O(w^2)$ as discussed before. Thus the algorithm takes $O(w^2)$ time per ideal, but the delay between two outputs can be greater. This is the case when we output a group in depth $k$ and the next outputted one is in depth 1, so the delay between them is $O(kw^2)$ since the depth is $O(k)$. To show $O(w^2)$ delay we use the idea in [11,14]. Indeed, the algorithm GenIdeals_main is internal output, that is it outputs an ideal at each node of the tree rather than outputing only for leaves. So we alternatively output depending on the parity of the depth of a node. As suggested in [14], we do the following two changes in algorithm GenIdeals:

– We change the Line 7 by: *If the depth of the call is odd then output* $G'$
– We add after line 10 the instruction: *If the depth of the call is even then output* $G'$.

The space used by the algorithm is linear for each node of the searching tree. Moreover the depth of the tree is bounded by $k$. □

## 6 Conclusion

This work is the first to show that skyline groups are objects set by partial order theory, called *ideals*. This allowed us to bring out interesting properties to ease the exploration of the graph of k-ideals (*i.e.* skyline groups of size $k$). Indeed, we presented a novel way to organize the space of skyline groups as a DFS tree. This helped us to propose a simple and efficient $k$-ideal enumeration algorithm: GenIdeals_main. The time delay complexity analysis that was performed highlights the relevance of our approach and shows that it outperforms the current state-of-the-art algorithm. Moreover, there are two directions to improve the result of this work. First one can improve the complexity of the algorithm UpdateCandToAdd1 to $O(1)$ when its output is empty. Second, to improve the space complexity, one may find a lexicographic order on the groups to avoid the re-enumeration in the reverse search technique, but the time complexity may increase.

**Acknowledgments.** The third author is supported by the French government IDEX ISITE initiative 16-IDEX-0001 (CAP 20–25).

## References

1. Abdo, M.: Efficient generation of the ideals of a poset in gray code order, part ii. Theor. Comput. Sci. **502**, 30–45 (2013), generation of Combinatorial Structures
2. Chang, Y., Garg, V.K.: Quicklex: a fast algorithm for consistent global states enumeration of distributed computations. In: Anceaume, E., Cachin, C., Potop-Butucaru, M.G. (eds.) 19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14–17, 2015, Rennes, France. LIPIcs, vol. 46, pp. 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)
3. Chomicki, J., Ciaccia, P., Meneghetti, N.: Skyline queries, front and back. ACM SIGMOD Rec. **42**(3), 6–18 (2013)
4. Habib, M., Medina, R., Nourine, L., Steiner, G.: Efficient algorithms on distributive lattices. Discrete Appl. Math. **110**(2), 169–187 (2001)
5. Habib, M., Nourine, L., Steiner, G.: Gray codes for the ideals of interval orders. J. Algorithms **25**, 52–66 (1997)
6. Im, H., Park, S.: Group skyline computation. Inf. Sci. **188**, 151–169 (2012)
7. Yang, Z., Xiao, G., Li, K., Li, K., et al.: Progressive approaches for pareto optimal groups computation. IEEE Trans. Knowl. Data Eng. **31**(3), 521–534 (2018)
8. Liu, J., Xiong, L., Pei, J., Luo, J., Zhang, H.: Finding pareto optimal groups: group-based skyline. Proc. VLDB Endowment **8**(13), 2086–2097 (2015)
9. Liu, J., Xiong, L., Pei, J., Luo, J., Zhang, H., Yu, W.: Group-based skyline for pareto optimal groups. IEEE Trans. Knowl. Data Eng. (2019)
10. Medina, R., Nourine, L.: Algorithme efficace de génération des ideaux d'un ensemble ordonné. C.R. Acad. Sci. Paris Sér. I Math. **319**, 1115–1120 (1994)
11. Nakano, S., Uno, T.: Constant time generation of trees with specified diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30559-0_3
12. Squire, M.: Enumerating the ideals of a poset. Preprint available electronically at (1995). http://citeseer.ist.psu.edu/465417.html
13. Steiner, G.: An algorithm for generating the ideals of a partial order. Oper. Res. Lett. **5**, 317–320 (1986)
14. Uno, T.: Two general methods to reduce delay and change of enumeration algorithms. NII Technical report (2003)
15. Wang, C., Wang, C., Guo, G., Ye, X., Philip, S.Y.: Efficient computation of g-skyline groups. IEEE Trans. Knowl. Data Eng. **30**(4), 674–688 (2017)
16. Wild, M.: Output-polynomial enumeration of all fixed-cardinality ideals of a poset, respectively all fixed-cardinality subtrees of a tree. Order **31**(1), 121–135 (2014)
17. Yang, Z., Zhou, X., Li, K., Xiao, G., Gao, Y., Li, K.: Efficient processing of top k group skyline queries. Knowl.-Based Syst. **182**, 104795 (2019)
18. Zhang, N., Li, C., Hassan, N., Rajasekaran, S., Das, G.: On skyline groups. IEEE Trans. Knowl. Data Eng. **26**(4), 942–956 (2013)
19. Zhou, X., Li, K., Yang, Z., Gao, Y., Li, K.: Efficient approaches to k representative g-skyline queries. ACM Trans. Knowl. Discov. Data (TKDD) **14**(5), 1–27 (2020)