Lexicographic Unranking Algorithms for the Twelvefold Way

Amaury Curiel
Sorbonne Université, CNRS, LIP6 - UMR 7606, Paris, France

Antoine Genitrini
Sorbonne Université, CNRS, LIP6 - UMR 7606, Paris, France

Abstract

The Twelvefold Way represents Rota's classification, addressing the most fundamental enumeration problems and their associated combinatorial counting formulas. These distinct problems are connected to enumerating functions defined from a set of elements denoted by $\mathcal{N}$ into another one $\mathcal{K}$. The counting solutions for the twelve problems are well known. We are interested in unranking algorithms. Such an algorithm is based on an underlying total order on the set of structures we aim at constructing. By taking the rank of an object, i.e. its number according to the total order, the algorithm outputs the structure itself after having built it. One famous total order is the lexicographic order: it is probably the one that is the most used by people when one wants to order things. While the counting solutions for Rota's classification have been known for years it is interesting to note that three among the problems have yet no lexicographic unranking algorithm. In this paper we aim at providing algorithms for the last three cases that remain without such algorithms. After presenting in detail the solution for set partitions associated with the famous Stirling numbers of the second kind, we explicitly explain how to adapt the algorithm for the two remaining cases. Additionally, we propose a detailed and fine-grained complexity analysis based on the number of bitwise arithmetic operations.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Generating random combinatorial structures

Keywords and phrases Twelvefold Way, Set partitions, Unranking, Lexicographic order

Digital Object Identifier 10.4230/LIPIcs.AofA.2024.17

Related Version *Full Version*: `https://hal.science/hal-04411470`

Supplementary Material *Software (Source Code)*: `https://github.com/AMAURYCU/setpartition_unrank` [3], archived at `swh:1:dir:b01a69e78b0fd972fdafc0c080421688cd9c9be6`

Funding ANR-FWF project PAnDAG ANR-23-CE48-0014-01.

Acknowledgements The authors thank the anonymous referees for their comments and suggested improvements. All these remarks have increased the quality of the paper.

1 Introduction

The Twelvefold Way, a classification from the 1960s by Rota, was introduced to address the most fundamental enumeration problems associated with their combinatorial counting formulas. It has been extensively discussed in Stanley's book [23, Section 1.9]. The distinct problems are related to the enumeration of functions defined from a set of elements denoted by $\mathcal{N}$ into another set denoted by $\mathcal{K}$. The respective cardinalities of these sets are denoted as n and k. Each set may consist of either distinguishable or indistinguishable elements, resulting in consideration of four pairs of sets. Additional constraints pertain to the properties of the functions, whether they are injective, surjective, or arbitrary. Consequently, we encounter twelve cases when enumerating these functions. The counting solutions are well-known, as presented in Stanley's book [23, Section 1.9]. In Table 1, we illustrate the classical combinatorial object enumerating each set of functions, in contrast to Stanley, who directly presents the counting solution.

© Amaury Curiel and Antoine Genitrini;
licensed under Creative Commons License CC-BY 4.0
35th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA 2024).
Editors: Cécile Mailler and Sebastian Wild; Article No. 17; pp. 17:1–17:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 The Twelvefold Way*

*The notation $k^{\underline{n}}$ corresponds to the product $k \cdot (k-1) \cdots (k-n+1)$; $[n \leq k]$ is the Iverson bracket returning 1 when $n \leq k$ and 0 otherwise; $\left\{ \begin{smallmatrix} \cdot \\ \cdot \end{smallmatrix} \right\}$ and $\binom{\cdot}{\cdot}$ stand respectively for the Stirling numbers of the second kind and the binomial coefficients; and $p_k(n)$ is the number of integer partitions of n into k positive integers.

elts of $\mathcal{N}$	elts of $\mathcal{K}$	f is arbitrary	f is injective	f is surjective
dist.	dist.	**1.** n-sequence in $\mathcal{K}$	**2.** n-permutation of $\mathcal{K}$	**3.** composition of $\mathcal{N}$ with k subsets
enumeration		k^n	$k^{\underline{n}}$	$k! \cdot \left\{ {n \atop k} \right\}$
lex. unranking		easy	[7, Section 5]	Section 4.1
indist.	dist.	**4.** n-multisubset of $\mathcal{K}$	**5.** n-subset of $\mathcal{K}$	**6.** composition of n with k terms
enumeration		$\binom{k+n-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
lex. unranking		see survey [7] and references therein		
dist.	indist.	**7.** partition of $\mathcal{N}$ into $\leq k$ subsets	**8.** partition of $\mathcal{N}$ into $\leq k$ elements	**9.** partition of $\mathcal{N}$ into k subsets
enumeration		$\sum_{i=0}^{k} \left\{ {n \atop i} \right\}$	$[n \leq k]$	$\left\{ {n \atop k} \right\}$
lex. unranking		Section 4.2	easy	Theorem 11
indist.	indist.	**10.** partition of n into $\leq k$ parts	**11.** partition of n into $\leq k$ parts $\{1\}$	**12.** partition of n into k parts
enumeration		$p_k(n+k)$	$[n \leq k]$	$p_k(n)$
lex. unranking		[20, Section 4.8]	easy	[20, Section 4.8]

In this paper, our focus lies in the generation of these classical combinatorial objects. To initiate our exploration, we arrange each object within a given class in lexicographic order. Subsequently, given the rank of an object, our goal is to construct it directly. This process is referred to as a *lexicographic unranking algorithm*. For instance, among the six permutations of $\{1, 2, 3\}$, the first one (with rank 0 in lexicographic order) is $[1, 2, 3]$, followed by the second one (with rank 1), which is $[1, 3, 2]$, and so forth, culminating with the last one (rank 5) being $[3, 2, 1]$. Consequently, the lexicographic unranking algorithm for the function with rank 4 returns $[3, 1, 2]$. In Table 1, we provide references to such algorithms for 9 out of the 12 cases. However, for cases 3, 7, and 9, no knowledge about lexicographic unranking algorithms seems available in the literature. This paper introduces an approach to unranking in lexicographic order for the *set partitions of an n-set into k blocks* (case 9). Furthermore, we present extensions of this approach to address cases 3 and 7.

The problem of unranking objects emerges as one of the most fundamental challenge in combinatorial generation, as seen in [21], and is applicable in various domains such as software testing [17], optimization [9], or scheduling [24]. In different contexts, it serves as the core problem for generating complex structures, as observed in phylogenetics [2] and bioinformatics [1]. As mentioned earlier, to unrank, one must first establish a total order over the objects in question. The often-utilized order is the *lexicographic* order due to its ease of handling, leading to extensive study in the literature. However, Ruskey notes in [20, p. 59] that lexicographic generation is typically not the most efficient, thus requiring particular care in lexicographic unranking. Knuth dedicates a section to the lexicographic generation of combinatorial objects in [11], relating it to the special case of Gray codes. Other combinatorial objects are also explored in Ruskey's and Kreher and Stinson's books on combinatorial generation [20, 12]. Skiena focuses on the practical implementation of such algorithms [22].

Usually, the approach for constructing structures using a recursive decomposition schema involves leveraging this decomposition to build a larger object from smaller ones. This method is extensively detailed in the well-known book by Nijenhuis and Wilf [18]. The approach has been systematically applied to decomposable objects in the context of analytic combinatorics, initially for recursive generation [6], and later for unranking methods [16].

Related work. Let us first quickly detail the classical unranking methods for the Twelvefold Way. As indicated in Table 1, cases 1, 8, and 11 are straightforward. In fact, an n-sequence in $\mathcal{K}$ consists of a word of length n over the finite alphabet $\mathcal{K}$, making lexicographic unranking direct. Cases 8 and 11 are extreme situations, both corresponding to the Iverson bracket $[n \leq k]$. As a result, the enumeration problems contain either one function (only when $n \leq k$) or none. The unranking method is trivial.

Cases 4, 5, and 6 are all associated with the enumeration of subsets and are directly related to combination enumerations. Various algorithms to solve such lexicographic unranking problems are relevant in the literature. In [7], we present a survey of the most efficient methods with a modern algorithm complexity analysis. Moreover, we introduce a new algorithm based on the factoradics number system, which is at least as efficient as the others.

Case 12 is associated with integer partition enumerations, and [12] presents an efficient recursive algorithm. This algorithm follows lexicographic order but for the reverse standard form of printing a partition. In standard form, partitions print the components from the largest to the smallest, whereas this algorithm is based on the reverse printing (from the smallest component to the largest one). It appears that, currently, there is no existing lexicographic unranking method specifically designed for the standard form of printing. case 10 can be considered an extension of case 12, much like case 7 is an extension of case 9.

The last three cases pertain to set partition problems. Various combinatorial objects, such as permutations with a specific pattern [4], graph coloring [10], walks in graphs [5], or trees for phylogenetics [2], are enumerated by set partitions. In a recent paper [14] the uniform random generating for set partitions for given n and k is studied, in the context of clustering algorithms. However, as far as we know, there is no lexicographic algorithm that takes arguments n, k and the rank r, returning the r-th partition in lexicographic order. Instead, there exists another classical object called a *restricted growth sequence* that is in bijection with set partitions (see [15, 20]). The unranking approaches presented in these works return such restricted growth sequences in lexicographic order. However, the natural bijection from restricted growth sequences to partitions does not preserve the lexicographic order.

Main results. To develop an efficient unranking generator for set partitions, we first introduce the lexicographic order over set partitions. Some care must be taken since we are dealing with sets of integers. Therefore, we use a standard printing of a set partition to obtain a canonical representation. We then introduce an *ad hoc* combinatorial algorithm to unrank set partitions in lexicographic order. Due to the very large integers manipulated in the algorithms, of order of $n \ln n$ bits, our algorithm computes the necessary ones on-the-fly in a lazy paradigm. The correctness and complexity of the algorithm are managed based on specific combinatorial properties derived throughout the paper. Finally, we present some experiments using a Go[1] implementation for our algorithm. We leverage the simple and efficient parallelism mechanism provided by this language to significantly reduce space consumption without degrading time consumption for large values of n.

Organization of the paper. Following the introduction of the paper, Section 2 highlights the combinatorial aspects of set partitions and presents some preliminary properties. In Section 3, we introduce our method for unranking set partitions, providing key insights into proving the correctness and complexity of our approach. Additionally, we present ideas for running calculations in parallel and share experiments that validate our parallel approach. Finally, Section 4 extends our algorithm to address cases 3 and 7 from the Twelvefold Way.

2 Preliminaries

2.1 Context of set partitions

▶ **Definition 1.** *Let $\mathcal{N}$ be a set of n distinguishable elements. A partition π of $\mathcal{N}$ in k blocks is a collection $B_1, B_2, \ldots, B_k$ of disjoint non-empty subsets of $\mathcal{N}$ such that every element from $\mathcal{N}$ belongs to exactly one B_i, for $i \in \{1, \ldots, k\}$.*

As an example let $\mathcal{N}$ be $\{1, \alpha, 2, 3, 4, \beta, 6, 12\}$. The collection $\{2, 3\}, \{4, 6, 12\}, \{\beta, 1, \alpha\}$ is a partition of $\mathcal{N}$ in 3 blocks. In the rest of for paper, the set of positive integers from 1 to n is denoted by $[\![n]\!]$. We can identify a set $\mathcal{N}$ of n elements with $[\![n]\!]$, thus from now we will only be interested in partitions for $[\![n]\!]$.

▶ **Definition 2.** *Let $1 \le k \le n$ be two positive integers and $\mathcal{N}$ be $[\![n]\!]$. The set of k-partitions of $\mathcal{N}$ is denoted by $\mathcal{P}_k^n$. The* sequential form *of a partition of $\mathcal{P}_k^n$ (i.e. a k-partitions of $\mathcal{N}$) is such that for all $i \in [\![k]\!]$, the block B_i contains the smallest integer from $[\![n]\!]$ not present in $\cup_{j<i} B_j$. Furthermore for each block, it is represented in the increasing order of its elements.*

For example $\{1\}, \{2, 3, 5\}, \{4, 6\}$ is a 3-partition of $[\![6]\!]$ represented in its sequential form. The sequential form is a canonical representation of the partition. As a shortcut, we will from now represent a partition simply as 1/235/46. In the paper we have chosen to use the terminology and the notations from Mansour [15].

▶ **Fact.** *Let $1 \le k \le n$ be two positive integers. The number of partitions in $\mathcal{P}_k^n$ is the Stirling number of the second kind denoted by $\left\{ {n \atop k} \right\}$. It satisfies the following recurrence:*

$$\left\{ {n \atop k} \right\} = \begin{cases} \left\{ {n-1 \atop k-1} \right\} + k \cdot \left\{ {n-1 \atop k} \right\} & \text{if } 1 < k < n; \\ 1 & \text{otherwise, i.e. when } k = 1 \text{ or } k = n. \end{cases} \tag{1}$$

This sequence is stored in `OEIS A008277`[2]. We now introduce a natural order over k-partitions.

▶ **Fact.** *Let A and B two subsets of positive integers. We say that $A \le B$ iff either*
- $A = B$, *or*
- $A \subset B$ *and* $\max(A) < \min(B \setminus A)$, *or*
- $B \subset A$ *and* $\min(A \setminus B) < max(B)$, *or*
- $\min(A \setminus B) < min(B \setminus A)$.

The relation $\le$ is a total order over subsets of $[\![n]\!]$.

[1] The Go language offers routines to manage `concurrency`.
[2] OEIS stands for the On-line Encyclopedia of Integer Sequences.

For example $\{1,3\} \leq \{1,3,4\}$ and $\{1,3\} \leq \{1,4\}$. But we also have $\{1,3,4\} \leq \{1,4\}$.

▶ **Fact.** *Let $1 \leq k \leq n$ be two positive integers. The lexicographic order*[3] *over partitions from $\mathcal{P}_k^n$, in sequential form, is well defined using the latter order to compare two blocks: in fact a partition in k blocks is a Cartesian product of k subsets of positive integers.*

There is another classical representation for partitions called *canonical form* in [15]. A partition in k blocks is represented as a word over a k-letters alphabet. For example the partition 1/235/46 is represented by the word 122323. The ith letter is the index of the block containing the integer i. Using this representation we can also define a lexicographic order over partitions, but here we compare partitions that do not necessarily contain the same numbers of blocks. The lexicographic order over the sequential form is not compatible with the lexicographic order used for the sequentical form we are interested in. This can be noted in the Table 2.

▶ **Definition 3.** *Using the lexicographic order over the sequential form for partitions in $\mathcal{P}_k^n$, we define a ranking function assigning to each partition its rank corresponding to the number of k-partitions smaller than it in the lexicographic order.*

Table 2 Ranking of the 3-partitions of $[\![5]\!]$.

Rank	Partition	Canonical form [15]	Rank	Partition	Canonical form [15]
0	1/2/345	12333	13	13/2/45	12133
1	1/23/45	12233	14	13/24/5	12123
2	1/234/5	12223	15	13/25/4	12132
3	1/235/4	12232	16	134/2/5	12113
4	1/24/35	12323	17	135/2/4	12131
5	1/245/3	12322	18	14/2/35	12313
6	1/25/34	12332	19	14/23/5	12213
7	12/3/45	11233	20	14/25/3	12312
8	12/34/5	11223	21	145/2/3	12311
9	12/35/4	11232	22	15/2/34	12331
10	123/4/5	11123	23	15/23/4	12231
11	124/3/5	11213	24	15/24/3	12321
12	125/3/4	11231			

▶ **Definition 4.** *Let $1 \leq k \leq n$ be two positive integers. Let P be a partition from $\mathcal{P}_k^n$, represented in the sequential form as $B_1/B_2/\dots/B_k$. An integer subset p is called* prefix *of P if $p \subset B_1$ and $p \leq B_1$.*

For the partition 12/35/4, there are three possible prefixes $\emptyset, 1$ and 12. We can further extend the definition of prefixes of a partition by letting $\mathcal{N}$ being any subset of $[\![n]\!]$. Thus removing the first block of the latter partition gives 35/4, we define prefixes of the 2-partition (of $\{3,4,5\}$) to be $\emptyset, 3$ and 35. Here we formalize this extension.

▶ **Definition 5.** *The definition of a prefix p of a partition is extended to any set $\mathcal{N}$ partitioned in a sequence of blocks (with the first one being denoted by B_1) such that $p \leq B_1$.*

[3] The lexicographic order of partitions from $\mathcal{P}_k^n$ in sequential form is a total order.

2.2 Combinatorial properties

We are now interested in counting results for partitions sharing the same prefix. These are the core results for our unranking algorithm.

▶ **Proposition 6.** *Let $1 \leq k \leq n$ be two positive integers. Let ℓ and d be two integers such that either $\ell = d = 1$ or $1 < \ell \leq d$. For a given prefix $1\ \alpha_2\ \alpha_3 \ldots \alpha_{\ell-1}\ d$, we define $S_k^n(\ell, d)$ to be the number of partitions in $\mathcal{P}_k^n$ accepting this prefix of length ℓ: We have*

$$S_k^n(\ell, d) = \sum_{u=0}^{\min(n-k-\ell+1, n-d)} \left\{ {n-\ell-u \atop k-1} \right\} \binom{n-d}{u}.$$

Let us remark that the notation $S_k^n(\ell, d)$ is a bit confusing in the sense that it is relative to the whole prefix $1\ \alpha_2\ \alpha_3 \ldots \alpha_{\ell-1}\ d$. However the specific values of α_2 up to $\alpha_{\ell-1}$ are not modifying the values of $S_k^n(\ell, d)$.

Proof. First if $\ell = d = 1$, then in the sequential form the first block necessarily contains 1. Thus $S_k^n(1,1) = |\mathcal{P}_k^n| = \left\{ {n \atop k} \right\} = \sum_{u=0}^{n-k} \left\{ {n-1-u \atop k-1} \right\} \binom{n-1}{u}$. The latter equality is given e.g. in [8, p. 251, Table 251].

In the second case when $1 < \ell \leq d$, we aim at counting the number of partitions in $\mathcal{P}_k^n$ accepting $1\ \alpha_2\ \alpha_3 \ldots \alpha_{\ell-1}\ d$ as a prefix. In order to exhibit a combinatorial interpretation, we rewrite $S_k^n(\ell, d)$ as

$$S_k^n(\ell, d) = \sum_{u=0}^{\min((n-\ell)-(k-1), n-d)} \left\{ {n-(\ell+u) \atop k-1} \right\} \binom{n-d}{u}.$$

Once the prefix is given, it remains to complete the first block B_1 from the partition, and then to calculate how we can further partition the other elements in the next blocks. The variable u in the sum corresponds to the number of elements that are appended to the prefix to complete B_1. Its value ranges from 0 up to the maximal number of elements that we can append i.e. $(n - \ell) - (k - 1)$ because at least $k - 1$ among the remaining $n - \ell$ elements must be assigned to the other $k - 1$ blocks. Obviously the number of possible elements u is also upper bounded by the number of remaining elements, i.e. $n - d$. Once the number u of elements for the completion of B_1 is given, we choose u elements greater than d: the number of possibilities is given by the binomial coefficient. Finally it remains to build the other blocks of the partition: we partition $n - (\ell + u)$ elements into $k - 1$ blocks. Hence the formula is proved. ◀

We introduce an example using Table 2 for $\mathcal{P}_3^5$. If we are interested in the prefix 13, then there are 3 partitions without completing block B_1, in the sum, when $u = 0$ we get $\left\{ {3 \atop 2} \right\} = 3$. The other possible value is $u = 1$ with the general term being $\left\{ {2 \atop 2} \right\} \binom{2}{1} = 2$ as it appears in the table.

In order to get a formula that is more efficient to calculate, we observe that the latter numbers $S_k^n(\ell, d)$ depend essentially in three variables instead of four. The proof is direct with some variable renaming.

▶ **Proposition 7.** *Let n, k, d be integers with $0 \leq k \leq n$ and $0 \leq d \leq n$. By defining*

$$\tilde{S}_k^n(d) = \sum_{u=0}^{\min(n-k, n-d)} \left\{ {n-u \atop k} \right\} \binom{n-d}{u}, \text{ we get } S_k^n(\ell, d) = \tilde{S}_{k-1}^{n-\ell}(d - \ell).$$

We note that $S_k^n(1,1) = \tilde{S}_{k-1}^{n-1}(0) = \left\{ {n \atop k} \right\}$. Note that the 3-dimension sequence $\tilde{S}$ seems not to be stored in OEIS. There exit several generalizations of Stirling numbers, but none of them apparently corresponds to our sequence $\tilde{S}$.

▶ **Corollary 8.** *The numbers $\tilde{S}_k^n(d)$ satisfy the following recurrence:*

$$\tilde{S}_k^n(d) = \begin{cases} \tilde{S}_{k-1}^{n-1}(d-1) + k \cdot \tilde{S}_k^{n-1}(d-1) & \text{if } 1 \le k \le n \text{ and } 1 \le d \le n; \\ \left\{ {n+1 \atop k+1} \right\} & \text{if } d = 0 \text{ and } 0 \le k \le n; \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note the later recurrence is similar to the one satisfied by Stirling numbers of the second kind (but with here a third variable d giving some kind of level of numbers).

▶ **Proposition 9.** *Let n, k, d be integers with $0 \le k \le n$ and $0 \le d \le n$. The function $\tilde{S}_k^n(d)$ can be represented as a binomial transform:*

$$\tilde{S}_k^n(d) = \sum_{u=0}^{\min(n-k,d)} (-1)^u \left\{ {n+1-u \atop k+1} \right\} \binom{d}{u}.$$

The main idea of the proof consists in proving that the two expressions given in Propositions 7 and 9 are satisfying the same recurrence and thus are equal.

Proof. In order to prove this new expression for $\tilde{S}$, we just have to prove that this expression satisfy the recurrence stated in Corollary 8. Substituting d by 0 we get the base case. We now consider the case where the three integers n, k, d satisfy $0 \le k \le n$ and $1 \le d \le n$. Using Proposition 9 in the case where $0 < k < n$ (the cases $k = 0$ or $k = n$ are obvious) we have

$$\begin{aligned} \tilde{S}_{k-1}^{n-1}(d-1) + k \cdot \tilde{S}_k^{n-1}(d-1) = & \sum_{u=0}^{\min(n-k,d-1)} (-1)^u \left\{ {n-u \atop k} \right\} \binom{d-1}{u} \\ & + k \cdot \sum_{u=0}^{\min(n-1-k,d-1)} (-1)^u \left\{ {n-u \atop k+1} \right\} \binom{d-1}{u} \end{aligned}$$

By using factorization and Stirling numbers of the second kind recurrence, we obtain:

$$\tilde{S}_{k-1}^{n-1}(d-1) + k \cdot \tilde{S}_k^{n-1}(d-1) = \sum_{u=0}^{\min(n-k,d-1)} (-1)^u \left(\left\{ {n+1-u \atop k+1} \right\} - \left\{ {n-u \atop k+1} \right\} \right) \binom{d-1}{u}.$$

After having telescoped the two sums we get the stated result. ◀

Finally, given two prefixes, one being smaller than the second one, the next proposition allows to compute how many partitions are in-between the two prefixes. More formally:

▶ **Proposition 10.** *Let $1 \le k \le n$ be two positive integers. Let $d_1 \in [\![n]\!] \setminus \{1\}$, $d_0 \in [\![d_1 - 1]\!]$ and $\ell > 1$ be integers. For a given prefix $1\ \alpha_2\ \alpha_3 \dots \alpha_{\ell-2}\ d_0$, the number of elements of $\mathcal{P}_k^n$ that admit a length-ℓ prefix satisfying $1\ \alpha_2 \dots \alpha_{\ell-2}\ d_0\ \tilde{d}_1$ (for all $\tilde{d}_1$ ranging from $d_0 + 1$ to d_1) is given by*

$$R_k^n(\ell, d_0, d_1) = \tilde{S}_{k-1}^{n-\ell}(d_0 - \ell) - \tilde{S}_{k-1}^{n-\ell}(d_1 + 1 - \ell).$$

3 Methods for unranking set partitions

Merging the combinatorial properties stated in the previous section, we are now ready to design algorithms to unrank set partitions in the lexicographic order.

3.1 Unranking algorithm design

Our aim consists in constructing the r-th partition related to a pair n, k in sequential form for the lexicographic order. The constructions follow the next main lines. The global idea consists in building the normalization of the partition. So we build together its block pattern and its reversed factoradics (seeing the partition as a size-n permutation).

- The building of the blocks is going from left to right;
- The construction of a block is also from left to right, component by component using a binary search approach;
- Finally once the block pattern and the reversed factoradics are set, a slight adaptation of the lexicographic permutation unranking algorithm gives the result.

The details for the correctness of our approach lies on the ranking function associated to the set partitions.

We first present in detail the main function Unranking of Algorithm 1. Using a loop, at each turn it defines the next block of the partition and then refine the value of the rank related to the remaining part of the partition. The result B returned by Next_block contains the indices of the components of the block that has been calculated and *acc* allows to update the rank so that it is related to the remaining part of the partition that must still be computed. With our previous definition, B is the normalization of the corresponding partition block. At the end of the function a dynamic extraction is executed in an array containing elements from 1 to n according to the indices in *Res*.

Algorithm 1 Lexicographic unranking of the partition with rank r in $\mathcal{P}_k^n$.

```
function Unranking(n, k, r)
    n' := n
    Res := [ ]
    while k > 1 do
        (B, acc) := Next_block(n, k, r)
        Append(Res, B)
        r := r − acc
        n := n − len(B)
        k := k − 1
    Append(Res, [0, 0, ..., 0])
    Res := Extract(n', Res)
    return Res

function Extract(n, R)
    L := [1, 2, ..., n]
    P := [ ]
    for r in R do
        p := [ ]
        for i in r do
            Append(p, L[i])
            Remove(L, i)
        Append(P, p)
    return P
```

Remove(L, i) removes element with index i in L.

```
function Next_block(n, k, r)
    Block := [0]; acc := {n−1 \atop k−1}
    if r < acc then
        return (Block, 0)
    d0 := 1; index := 2; inf := 2; sup := n
    complete := False
    while not complete do
        while inf < sup do
            mid := ⌊(inf + sup)/2⌋
            if r >= acc+R_k^n(index−1, d0, mid−1) then
                inf := mid + 1
            else
                sup := mid
        mid := inf; threshhold := {n−index \atop k−1}
        acc := acc + R_k^n(index − 1, d0, mid − 2)
        Append(Block, mid − index)
        if r < threshhold + acc then
            complete := True
        else
            index := index + 1
            d0 := mid; inf := d0 + 1; sup := n
            acc := acc + threshhold
    return (Block, acc)
```

The function NEXT_BLOCK takes parameters n, k, r and returns essentially the first block of the r-th partition in $\mathcal{P}_k^n$. In fact, using Table 2 the call NEXT_BLOCK$(5, 3, 16)$ returns 0 1 1 (instead of 1 3 4), the latter block being obtained through a dynamic extraction of the element 0 in $[1, 2, 3, 4, 5]$ then the element 1 is extracted in the remaining part $[2, 3, 4, 5]$ and finally the element 1 in $[2, 4, 5]$. Constructing the blocks of indices instead of the blocks of values allows to neglect about the remaining elements for the further blocks construction. Note that obviously the last block of the partition contains only the indices 0 (Line 10 from UNRANKING Algorithm) and the first element of a block is always index 0, both due to the sequential form. Finally while calling UNRANKING$(5, 3, 16)$, at the end of Line 10, *Res* contains [[0 1 1][0][0]]. Reading the components from right to left we get the factoradics 0 0 1 1 0 of the number 8 corresponding to the lexicographic rank of the permutation $[1, 3, 4, 2, 5]$ (*cf.* [7] for details).

▶ **Theorem 11.** *UNRANKING$(n, k, \cdot)$ is a lexicographic unranking algorithm for set partitions from $\mathcal{P}_k^n$.*

Proof key-ideas. The core of the function NEXT_BLOCK relies on the `while` loop (line 7). When it is entered (let say for the i-th time), the variable `Block` contains the length-i prefix of the normalized (final) block. Thus at this evaluation the loop determines (with a binary search) the $i + 1$th element that is appended to `Block`. Then, we calculate if the block is finished (line 17) or if we continue (line 19). ◀

3.2 Complexity analysis and experiments for unranking

In our implementation in Go[4], we offer two approaches for the necessary Stirling numbers calculations: either a precomputation of them or a computation on the fly of those that are needed at each step. We never precompute the 3 dimension table $\tilde{S}_k^n(d)$. In fact, in many bad cases these numbers are of order of $n!$, thus precomputing would be too expensive while only few of the numbers are needed. We compute the necessary numbers $\tilde{S}_k^n(d)$ on the fly.

First let us recall the behavior of the sequence of Stirling numbers of the second kind when k is ranging from 1 to n.

▶ **Fact.** *Let $1 \le k \le n$ be two positive integers. The sequence $\left\{ {n \atop k} \right\}$ is unimodal and its maximum is reached when $k_n \sim n/\ln n$. Around this value, we have $\log \left\{ {n \atop k_n} \right\} = \Theta(n \log n)$[5]. Furthermore, we have an upper bound valid for all $1 \le k \le n$:*

$$\log \left\{ {n \atop k} \right\} \le (n-k)\log k + \log \binom{n}{k} \le (n-k)\log k + k \log\left(\frac{n \cdot e}{k}\right).$$

See the fundamental paper of Rennie and Dobson [19] to get a proof for these results.

In the following we propose six distinct implementations of the function $\tilde{S}$ presented in Proposition 7 and underlying the function R from Proposition 10.

- **S_v1** direct implementation of the formula stated in Proposition 7;
- **S_v2** implementation of the formula from Proposition 7 taking into account the symmetry of binomial coefficients, thus the sum contains half of the terms in comparison to S_v1 (and thus half of the multiplications);
- **S_v3** direct implementation of the formula stated in Proposition 9;

[4] Go implementation and the material used for repeating the experiments are all available `here`.
[5] In this paper we use the notation log for the logarithm in basis 2.

S_v4 implementation of the formula from Proposition 9 taking into account the symmetry of binomial coefficients, thus the sum contains half of the terms in comparison to S_v3;

S_v5 is our most efficient algorithm without precomputations. The calculation way consists in deciding according whether a call to S_v2 or to S_v4 should be the most efficient, according to the number of terms in the sums interacting with Propositions 7 and 9;

S_v6 same algorithm than S_v5 but with all necessary Stirling numbers of the second kind precomputed.

The integers computed during the unranking algorithm are very large, thus a classical complexity in the number of arithmetical operations is not precise. We hence are interested in the bit-complexity, corresponding the the number of atomic operations on digits.

▶ **Theorem 12.** *For the time complexity, the algorithm* Unranking *based on each of the function S_v· has a bit-complexity bounded by*

$$O\left(\frac{(n-k)^3 M(n)}{n} \ln n \log k + \frac{k(n-k)^2 M(n)}{n} \ln n \log\left(\frac{n \cdot e}{k}\right)\right),$$

where $M(n)$ is the bit complexity for the multiplication of two numbers, each one containing n bits.

The naïve multiplication algorithm satisfies $M(n) = \Theta(n^2)$. But using, for example, Karatsuba algorithm, we obtain $M(n) = \Theta(n^{\log 3})$ for the time complexity. In Go[6], as soon as the integers are greater than 2^{40}, Karatsuba multiplication algorithm is used. In our context, almost all cases are thus based on the latter algorithm.

Proof. We are interested in a worst case complexity analysis when n is large and for k ranging in $[\![n]\!]$. We are using the same kind of analysis in bit complexity as the one presented in [7, Section 4.3]. We compute an upper bound of the complexity in the central range of the Stirling numbers of the second kind. The central range, when n tends to infinity, is observed when $k = \Theta(n/\ln n)$. A detailed similar analysis is presented in the paper [13]. In our context each Stirling number necessitates $\log \left\{{n \atop k}\right\}$ bits to be stored. They are multiplied by binomial coefficients containing at most n bits. Thus Stirling numbers are separated in blocks of n bits in order to use a multiplication of similar sizes numbers, inducing a time complexity bounded by $\log \left\{{n \atop k}\right\}/n \cdot M(n)$. Furthermore the number of calls the the function $\tilde{S}$ is $O((n-k)\ln n)$ induced by the repetitive calls to the binary search algorithm. Compiling all these upper bounds gives the stated bit-complexity. ◀

For approach S_v6, the following result establishes that the precomputation is negligible in terms of time complexity compared to the unranking itself. However note that the memory complexity is quadratic instead of linear (in n) by using the precomputation step.

▶ **Proposition 13.** *The bit-complexity for the Stirling numbers precomputation is bounded by*

$$O\left(k(n-k)^2 M(\log k) + k^2(n-k)\frac{\log n}{\log k} M(\log k)\right).$$

In order to get the Stirling numbers on the fly, we use parallel compuatations. In fact, for each block determination, we observe that only two neighbors columns from the triangle of numbers are needed. Thus during the determination of a block, we compute in parallel the

[6] Go documentation for `big integers manipulations`.

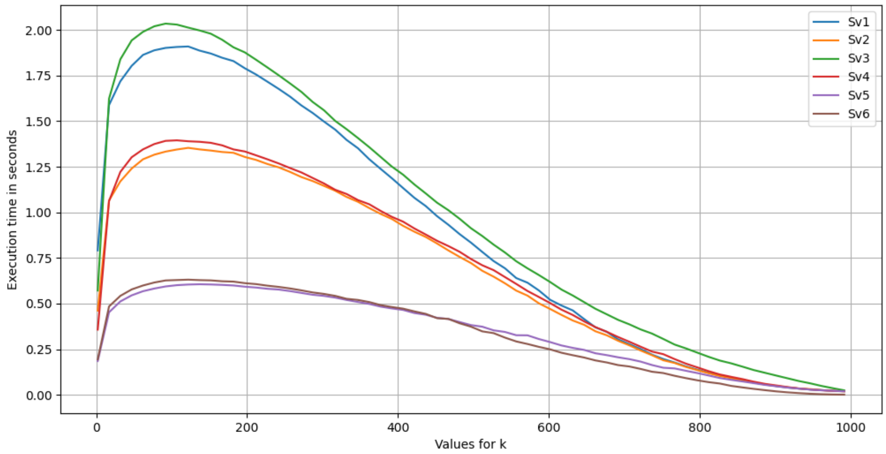

Figure 1 Time (in seconds) for unranking a partition in $\mathcal{P}_k^{1000}$ when k is ranging in $[\![1000]\!]$.

next two columns that will be necessary for the next block. Computing column $n-1$ from column n costs less time than the unranking algorithm. Thus, with parallel computation of the Stirling numbers, we achieve the same time complexity as the algorithm where Stirling numbers are precomputed while consuming a $O(n)$ amount of memory thus needing $O(n^2)$ memory size.

In Figure 1, we run experiments[7] by fixing $n = 1000$ and k ranging from 2 to 992 with steps of 15 units. For each value of k, 500 uniform samples are computed and the average time for the building of the partition is drawn for each Algorithm S_v1 up to S_v6. Obviously Algorithms S_v2 and S_v4 are better than their naïve versions respectively S_v1 and S_v2. It is interesting to note that the optimization S_v5, obtained by computing the most efficient formula between Propositions 7 and 9. Finally we remark that the Algorithm S_v5 is almost as efficient as S_v6 where all precomputation of Stirling numbers have been stored before the computation of the partition. Strangely, for the smallest values of k, we note that S_v5 is even faster than S_v6. This is probably due to the RAM accesses: in fact in some preliminary experiments with computers equipped with DDR5 RAM Algorithm S_v6 is always faster than S_v5, and this is what is expected.

4 Extension and conclusion

As we observe in Table 1, both enumeration cases 3 and 7 are some extended version of the enumeration case 9. An adaptation for the Ranking function allows to rank the families counted by cases 3 and 7; then adapting the unranking algorithm solves these cases.

4.1 Ordered set partitions

Recall Stirling numbers of the second kind are counting the numbers of surjective functions from set $\mathcal{N}$ to set $\mathcal{K}$, where the elements of $\mathcal{N}$ are distinguishable and those of $\mathcal{K}$ are indistinguishable. We can represent these functions as set partitions. Now, what happens

[7] The experiments provided in this paper are driven by using a PC equipped with an Intel Xeon X5677 processor, 32GB of DDR4-SDRAM and running Debian GNU/Linux 12.

when elements of $\mathcal{K}$ are distinguished? These functions are counted by *ordered Stirling numbers of the second kind*. In addition, they can be represented as *ordered set partitions*, which are similar to set partitions except that the order of the subsets matters. For instance, while in the world of unordered set partitions, elements $14/25/3; 14/3/25; 25/14/3; 25/3/14;$ $3/14/25$ and $3/25/14$ are equivalent and represented by the partition $14/25/3$ in sequential form, in the world of ordered set partition, the 6 elements are all different.

▶ **Proposition 14.** *Let $1 \leq k \leq n$, be two integers with n being the cardinality of set $\mathcal{N}$. The number of ordered set partitions of $\mathcal{N}$ in k (non empty disjoint) subsets is $k! \cdot \left\{ {n \atop k} \right\}$. The family of these partitions is denoted by $\mathcal{O}_k^n$.*

The proof is direct: the blocks in the sequential form of a set partition are distinguishable, thus permuting them gives the associated ordered set partitions.

▶ **Fact.** *Let $1 \leq k \leq n$, be two integers, the lexicographic order on set partitions $\mathcal{P}_k^n$ is easily extended to get the lexicographic order for the ordered set partitions from $\mathcal{O}_k^n$.*

We can now derive the enumeration core result in this new context.

▶ **Proposition 15.** *Let $1 \leq k \leq n$ be two integers. Let ℓ and d be two integers such that either $\ell = d = 1$ or $1 < \ell \leq d$. For a given prefix $1\ \alpha_2\ \alpha_3 \dots \alpha_{\ell-1}\ d$, we define $T_k^n(\ell, d)$ to be the number of ordered set partitions in $\mathcal{O}_k^n$ accepting this prefix.*

$$T_k^n(\ell, d) = \sum_{u=0}^{\min(n-k-\ell+1, n-d)} k! \left\{ {n - \ell - u \atop k - 1} \right\} \binom{n-d}{u}.$$

This formula is the analogous to $S_k^n(\ell, d)$. Using the same variable changes, we also get a three variable function, like $\tilde{S}_k^n(d)$. Then we can deduce an adaptation of our first algorithm by replacing Stirling numbers of the second kind by ordered Stirling numbers of the second kind and using the latter formula.

4.2 Bell's set partitions

We denote by $\mathcal{F}$ the family of these functions. Such functions can be represented as unordered set partitions with at most k blocks where k is the numbers of elements in $\mathcal{K}$.
Let $\mathcal{K}_i \subset \mathcal{K}$ be a subset of i distinguishable elements and $\mathcal{B}_i$ the functions that are surjective from $\mathcal{N}$ into $\mathcal{K}_i$. We have $\mathcal{B}_k^n = \bigcup_{i=1}^k \mathcal{B}_i$ and for a given $i \in [\![k]\!]$, $|\mathcal{B}_i| = \left\{ {n \atop i} \right\}$. Obviously $|\mathcal{B}_k^n| = \sum_{i=0}^k |\mathcal{B}_i| = \sum_{i=0}^k \left\{ {n \atop i} \right\}$. The cardinality of $\mathcal{B}_k^n$ is counted by the *k-restricted Bell numbers* and finally, when $k = n$, we get the Bell numbers.

▶ **Fact.** *Let $1 \leq k \leq n$, be two integers, the lexicographic order on set partitions $\mathcal{P}_k^n$ is also lexicographic for $\mathcal{B}_k^n$.*

▶ **Proposition 16.** *Let $1 \leq k \leq n$ be two integers. Let ℓ and d be two integers such that either $\ell = d = 1$ or $1 < \ell \leq d$. For a given prefix $1\ \alpha_2\ \alpha_3 \dots \alpha_{\ell-1}\ d$, we define $U_k^n(\ell, d)$ to be the number of Bell's set partitions in $\mathcal{B}_k^n$ accepting of this prefix.*

$$U_k^n(\ell, d) = \sum_{u=0}^{\min(n-k-\ell+1, n-d)} \sum_{i=1}^{k} \left\{ {n - \ell - u \atop i - 1} \right\} \binom{n-d}{u}.$$

As a final remark, the correctness of both previous algorithms is directly hanging to the one for the set partition algorithm. What is remaining is their complexity analysis: it is not difficult, and it will be written in a long version of this paper.

References

1 N. Ali, A. Shamoon, N. Yadav, and T. Sharma. Peptide Combination Generator: a Tool for Generating Peptide Combinations. *ACS Omega*, 5(11):5781–5783, 2020. doi:10.1021/acsomega.9b03848.

2 O. Bodini, A. Genitrini, C. Mailler, and M. Naima. Strict monotonic trees arising from evolutionary processes: Combinatorial and probabilistic study. *Advances in Applied Math.*, 133:102284, 2022. doi:10.1016/J.AAM.2021.102284.

3 Amaury Curiel and Antoine Genitrini. Set Partition Unranking. Software, swhId: swh:1:dir:b01a69e78b0fd972fdafc0c080421688cd9c9be6 (visited on 2024-07-09). URL: https://github.com/AMAURYCU/setpartition_unrank.

4 C. Defant. Highly sorted permutations and Bell numbers. *Enumerative Combinatoricas and Applications*, 1(1), 2021. doi:10.48550/arXiv.2012.03869.

5 A. Dzhumadil'daev and D. Yeliussizov. Walks, partitions, and normal ordering. *Electronic Journal of Combinatorics*, 22(4):4, 2015. doi:10.37236/5181.

6 P. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994. doi:10.1016/0304-3975(94)90226-7.

7 A. Genitrini and M. Pépin. Lexicographic Unranking of Combinations Revisited. *Algorithms*, 14(3), 2021. doi:10.3390/A14030097.

8 R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, MA, second edition, 1989. URL: https://www-cs-faculty.stanford.edu/%7Eknuth/gkp.html.

9 I. Grebennik and O. Lytvynenko. Developing software for solving some combinatorial generation and optimization problems. In *7th Int. Conf. on Application of Information and Communication Technology and Statistics in Economy and Education*, pages 135–143, 2017. URL: http://openarchive.nure.ua/handle/document/5498.

10 A. Hertz and H. Mélot. Counting the number of non-equivalent vertex colorings of a graph. *Discrete Applied Mathematics*, 203:62–71, 2016. doi:10.1016/J.DAM.2015.09.015.

11 D. E. Knuth. *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms*. Addison-Wesley Professional, 2011. URL: https://www-cs-faculty.stanford.edu/~knuth/taocp.html.

12 D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms: generation, enumeration, and search*. CRC Press, 1999. doi:10.1145/309739.309744.

13 G. Louchard. Asymptotics of the Stirling numbers of the second kind revisited. *Applicable Analysis and Discrete Mathematics*, 7(2):193–210, 2013. doi:10.46298/dmtcs.501.

14 Q. Lutz, E. de Panafieu, M. Stein, and A. Scott. Active clustering for labeling training data. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, pages 8469–8480, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/47841cc9e552bd5c40164db7073b817b-Abstract.html.

15 T. Mansour. *Combinatorics of Set Partitions*. Discrete Mathematics and Its Applications. Taylor & Francis, 2012. doi:10.1201/b12691.

16 C. Martínez and X. Molinero. A generic approach for the unranking of labeled combinatorial classes. *Random Structures & Algorithms*, 19(3-4):472–497, 2001. doi:10.1002/RSA.10025.

17 A. F. Myers. k-out-of-n:G System Reliability With Imperfect Fault Coverage. *IEEE Transactions on Reliability*, 56(3):464–473, 2007. doi:10.1109/TR.2007.903229.

18 A. Nijenhuis and H. S. Wilf. *Combinatorial algorithms*. Computer science and applied mathematics. Academic Press, New York, NY, 1975. URL: https://www2.math.upenn.edu/~wilf/website/CombAlgDownld.html.

19 B.C. Rennie and A.J. Dobson. On stirling numbers of the second kind. *Journal of Combinatorial Theory*, 7(2):116–121, 1969. doi:10.1016/S0021-9800(69)80045-1.

20 F. Ruskey. Combinatorial Generation, 2003.

21 Y. Shablya, D. Kruchinin, and V. Kruchinin. Method for Developing Combinatorial Generation Algorithms Based on AND/OR Trees and Its Application. *Mathematics*, 8(6):962, 2020. doi:10.3390/math8060962.

22 S. Skiena. *The Algorithm Design Manual, Third Edition.* Texts in Computer Science. Springer, 2020.

23 R. P. Stanley. *Enumerative Combinatorics: Volume 1.* Cambridge Univ. Press, 2011. doi:10.1017/CBO9781139058520.

24 Y. Tamada, S. Imoto, and S. Miyano. Parallel Algorithm for Learning Optimal Bayesian Network Structure. *J. Mach. Learn. Res.*, 12:2437–2459, 2011. doi:10.5555/1953048.2021080.