COMBINATORIAL GENERATION VIA PERMUTATION LANGUAGES

ELIZABETH HARTUNG, HUNG P. HOANG, TORSTEN MÜTZE, AND AARON WILLIAMS

Abstract. In this work we present a general and versatile algorithmic framework for exhaustively generating a large variety of different combinatorial objects, based on encoding them as permutations. This approach provides a unified view on many known results and allows us to prove many new ones. In particular, we obtain the following four classical Gray codes as special cases: the Steinhaus-Johnson-Trotter algorithm to generate all permutations of an n-element set by adjacent transpositions; the binary reflected Gray code to generate all n-bit strings by flipping a single bit in each step; the Gray code for generating all n-vertex binary trees by rotations due to Lucas, van Baronaigien, and Ruskey; the Gray code for generating all partitions of an n-element ground set by element exchanges due to Kaye.

We present two distinct applications for our new framework: The first main application is the generation of pattern-avoiding permutations, yielding new Gray codes for different families of permutations that are characterized by the avoidance of certain classical patterns, (bi)vincular patterns, barred patterns, Bruhat-restricted patterns, mesh patterns, monotone and geometric grid classes, and many others. We thus also obtain new Gray code algorithms for the combinatorial objects that are in bijection to these permutations, in particular for five different types of geometric rectangulations, also known as floorplans, which are divisions of a square into n rectangles subject to certain restrictions.

The second main application of our framework are lattice congruences of the weak order on the symmetric group S_n. Recently, Pilaud and Santos realized all those lattice congruences as $(n-1)$-dimensional polytopes, called quotientopes, which generalize hypercubes, associahedra, permutahedra etc. Our algorithm generates the equivalence classes of each of those lattice congruences, by producing a Hamilton path on the skeleton of the corresponding quotientope, yielding a constructive proof that each of these highly symmetric graphs is Hamiltonian. We thus also obtain a provable notion of optimality for the Gray codes obtained from our framework: They translate into walks along the edges of a polytope.

(Elizabeth Hartung) Massachusetts College of Liberal Arts, United States

(Hung P. Hoang) Department of Computer Science, ETH Zürich, Switzerland

(Torsten Mütze) Department of Computer Science, University of Warwick, United Kingdom

(Aaron Williams) Bard College at Simon's Rock, United States

E-mail addresses: e.hartung@mcla.edu, hung.hoang@inf.ethz.ch, torsten.mutze@warwick.ac.uk, awilliams@simons-rock.edu.

Key words and phrases. Exhaustive generation algorithm, Gray code, pattern-avoiding permutation, weak order, lattice congruence, quotientope, Hamilton path, rectangulation.

Torsten Mütze is also affiliated with Charles University, Faculty of Mathematics and Physics, and was supported by GACR grant GA 19-08554S, and by DFG grant 413902284.

1. Introduction

In computer science we frequently encounter different kinds of combinatorial objects, such as permutations, binary strings, binary trees, set partitions, spanning trees of a graph, and so forth. There are three recurring fundamental algorithmic tasks that we want to perform with such objects: counting, random generation, and exhaustive generation. For the first two tasks, there are powerful general methods available, such as generating functions [FS09] and Markov chains [Jer03], solving both problems for a large variety of different objects. For the third task, namely exhaustive generation, however, we are lacking such a powerful and unifying theory, even though some first steps in this direction have been made (see Section 1.2 below). Nonetheless, the literature contains a vast number of algorithms that solve the exhaustive generation problem for specific classes of objects, and many of these algorithms are covered in depth in the most recent volume of Knuth's seminal series 'The Art of Computer Programming' [Knu11].

1.1. **Overview of our results.** The main contribution of this paper is a general and versatile algorithmic framework for exhaustively generating a large variety of different combinatorial objects, which provides a unified view on many known results and allows us to prove many new ones. The basic idea is to encode a particular set of objects as a set of permutations $L_n \subseteq S_n$, where S_n denotes the set of all permutations of $[n] := \{1, 2, \ldots, n\}$, and to use a simple greedy algorithm to generate those permutations by cyclic rotations of substrings, an operation we call a *jump*. This works under very mild assumptions on the set L_n, and allows us to generate more than double-exponentially (in n) many distinct sets L_n. Moreover, the jump orderings obtained from our algorithm translate into listings of combinatorial objects where consecutive objects differ by small changes, i.e., we obtain *Gray code algorithms* [Sav97], and those changes are smallest possible in a provable sense. The main tools of our framework are Algorithm J and Theorem 1 in Section 2. In particular, we obtain the following four classical Gray codes as special cases: (1) the Steinhaus-Johnson-Trotter algorithm to generate all permutations of $[n]$ by adjacent transpositions, also known as plain change order [Tro62, Joh63]; (2) the binary reflected Gray code (BRGC) to generate all binary strings of length n by flipping a single bit in each step [Gra53]; (3) the Gray code for generating all n-vertex binary trees by rotations due to Lucas, van Baronaigien, and Ruskey [LvBR93]; (4) the Gray code for generating all set partitions of $[n]$ by exchanging an element in each step due to Kaye [Kay76].

We present two distinct applications for our new framework: The first main application is the generation of pattern-avoiding permutations, yielding new Gray codes for different families of permutations characterized by the avoidance of certain classical patterns, (bi)vincular patterns [BS00, BMCDK10], barred patterns [Wes90], Bruhat-restricted patterns [WY06], mesh patterns [BC11], monotone and geometric grid classes [HV06, AAB$^+$13], and many others. We thus also obtain new Gray code algorithms for the combinatorial objects that are in bijection to these permutations, in particular for five different types of geometric rectangulations/floorplans [ABP06, Rea12b, ABBM$^+$13, CSS18], which are divisions of a square into n rectangles subject to different restrictions; see Figure 2. Our results in this area are summarized in Theorem 4, Remark 5, and Table 1 in Section 3.

The second main application of our framework are lattice congruences of the weak order on the symmetric group S_n. This area has beautiful ramifications into groups, posets, polytopes, geometry, and combinatorics [Rea12a, Rea16a, Rea16b]. There are double-exponentially many distinct such

lattice congruences, and they generalize many known lattices such as the Boolean lattice, the Tamari lattice [Tam62], and certain Cambrian lattices [Rea06, CP17]. Recently, Pilaud and Santos [PS19] realized all those lattice congruences as $(n-1)$-dimensional polytopes, called quotientopes, which generalize hypercubes, associahedra, permutahedra etc. Our algorithm generates the equivalence classes of each of those lattice congruences, by producing a Hamilton path on the skeleton of the corresponding quotientope, yielding a constructive proof that each of these highly symmetric graphs is Hamiltonian; see Figure 6. We thus also obtain a provable notion of optimality for the Gray codes obtained from Algorithm J: *Jump operations translate into walks along the edges of a polytope.* Our results in this area are summarized in Theorem 6 and Corollary 7 in Section 4. In this extended abstract, many formal proofs are omitted due to length constraints; for details see [HHMW19] and [HM19].

1.2. **Related work.** Avis and Fukuda's [AF96] *reverse-search* is a general technique for exhaustive generation. They consider the objects to be generated as nodes of a graph, and connect them by edges that model local modifications. This *flip graph* is equipped with an objective function and traversed backwards by a local search algorithm. Reverse-search is complementary to our permutation based approach, as it uses a fundamentally different encoding of objects. The permutation encoding seems to allow for more fine-grained control (optimal Gray codes) and even faster algorithms.

Another method for combinatorial counting and exhaustive generation is the *ECO framework* by Barcucci, Del Lungo, Pergola, and Pinzani [BDLPP99]. It uses an infinite tree with integer node labels, and a set of production rules for creating the children of a node based on its label. Bacchelli, Barcucci, Grazzini, and Pergola [BBGP04] also used ECO for exhaustive generation. Dukes, Flanagan, Mansour, and Vajnovszki [DFMV08], Baril [Bar09], and Do, Tran and Vajnovszki [DTV19] used ECO for deriving Gray codes for different classes of pattern-avoiding permutations, which works under certain regularity assumptions on the production rules. The main difference between ECO and our framework is that the change operations on the label sequences of the ECO tree do not necessarily correspond to Gray-code like changes on the corresponding combinatorial objects. Minimal jumps in a permutation, on the other hand, correspond to minimal changes on the combinatorial objects in a provable sense, even though they may involve several entries of the permutation.

Li and Sawada [LS09] considered another tree-based approach for generating so-called *reflectable languages*, yielding Gray codes for k-ary strings and trees, restricted growth strings, and open meandric systems (see also [XCU10]). Ruskey, Sawada, and Williams [RSW12, SW12] proposed a generation framework based on binary strings with a fixed numbers of 1s, called *bubble languages*, which allows to generate e.g. combinations, necklaces, Dyck words, and Lyndon words. In the resulting cool-lex Gray codes, any two consecutive words differ by a cyclic prefix rotation.

Pattern avoidance is central in combinatorics, as illustrated by the books [Kit11, Bón12], and by the conference 'Permutation Patterns', held annually since 2003. Many fundamental classes of combinatorial objects are in bijection with pattern-avoiding permutations (see Table 1 and [Ten18]). For instance, Knuth [Knu98] proved that all 123-avoiding or 132-avoiding permutations are counted by the Catalan numbers. Concerning counting and generation, a few tree-based algorithms for pattern-avoiding permutations have been proposed [Eli07, DFMV08, Bar08, Bar09]. The problem has also been studied extensively from an algorithmic point of view. E.g., testing whether a permutation π contains a pattern τ is NP-complete in general [BBL98]. Jelínek and Kynčl [JK17] proved that the problem remains hard even if π and τ have no decreasing subsequence of length 4

and 3, respectively, which is best possible. On the other hand, Guillemot and Marx [GM14] showed that the problem can be solved in time $2^{O(k^2 \log k)} n \log n$, where n is the length of π and k is the length of τ, a considerable improvement over the obvious $O(n^k)$ algorithm (see also [Koz19]).

2. Generating permutations by jumps

In this section we present a simple greedy algorithm, Algorithm J, for exhaustively generating a given set $L_n \subseteq S_n$ of permutations, and we show that the algorithm works successfully under very mild assumptions on the set L_n (Theorem 1).

2.1. **Preliminaries.** We use S_n to denote the set of all permutations of $[n] := \{1, \dots, n\}$, and we write $\pi \in S_n$ in one-line notation as $\pi = \pi(1)\pi(2)\dots\pi(n) = a_1 a_2 \dots a_n$. We use $\mathrm{id}_n = 12\dots n$ to denote the identity permutation, and $\varepsilon \in S_0$ to denote the empty permutation. For any $\pi \in S_{n-1}$ and any $1 \le i \le n$, we write $c_i(\pi) \in S_n$ for the permutation obtained from π by inserting the new largest value n at position i of π, i.e., if $\pi = a_1 \dots a_{n-1}$ then $c_i(\pi) = a_1 \dots a_{i-1}\, n\, a_i \dots a_{n-1}$. Moreover, for $\pi \in S_n$, we write $p(\pi) \in S_{n-1}$ for the permutation obtained from π by removing the largest entry n. Here, c_i and p stand for the child and parent of a node in the tree of permutations discussed shortly. Given a permutation $\pi = a_1 \dots a_n$ with a substring $a_i \dots a_j$ with $a_i > a_{i+1}, \dots, a_j$, a *right jump of a_i by $j - i$ steps* is a cyclic left rotation of this substring by one position to $a_{i+1} \dots a_j a_i$. Similarly, given a substring $a_i \dots a_j$ with $a_j > a_i, \dots, a_{j-1}$, a *left jump of a_j by $j - i$ steps* is a cyclic right rotation of this substring to $a_j a_i \dots a_{j-1}$. For example, a right jump of 5 in the permutation 265134 by 2 steps yields 261354.

2.2. **The basic algorithm.** Our approach starts with the following simple greedy algorithm to generate a set of permutations $L_n \subseteq S_n$. We say that a jump is *minimal* (w.r.t. L_n), if a jump of the same value in the same direction by fewer steps creates a permutation that is not in L_n.

Algorithm J *(Greedy minimal jumps).* This algorithm attempts to greedily generate a set of permutations $L_n \subseteq S_n$ using minimal jumps starting from an initial permutation $\pi_0 \in L_n$.

J1. [Initialize] Visit the initial permutation π_0.

J2. [Jump] Generate an unvisited permutation from L_n by performing a minimal jump of the largest possible value in the most recently visited permutation. If no such jump exists, or the jump direction is ambiguous, then terminate. Otherwise visit this permutation and repeat J2.

Put differently, in step J2 we consider the entries $n, n-1, \dots, 2$ of the current permutation in decreasing order, and for each of them we check whether it allows a minimal left or right jump that creates a previously unvisited permutation, and we perform the first such jump we find, unless the same entry also allows a jump in the opposite direction, in which case we terminate. If no minimal jump creates an unvisited permutation, we also terminate the algorithm prematurely. For example, consider $L_4 = \{1243, 1423, 4123, 4213, 2134\}$. Starting with $\pi_0 = 1243$, the algorithm generates $\pi_1 = 1423$ (obtained from π_0 by a left jump of 4 by 1 step), then $\pi_2 = 4123$, then $\pi_3 = 4213$ (in π_2, 4 cannot jump, as π_0 and π_1 have been visited before; 3 cannot jump either to create any permutation from L_4, so 2 jumps left by 1 step), then $\pi_4 = 2134$, successfully generating L_4. If instead we initialize with $\pi_0 = 4213$, then the algorithm generates $\pi_1 = 2134$, and then stops, as no further jump is possible. If we choose $\pi_0 = 1423$, then we may jump 4 to the left or right (by 1 step), but as the direction is ambiguous, the algorithm stops immediately. As mentioned before, the

algorithm may stop prematurely only either because no minimal jump leading to a new permutation from L_n is possible, or because the direction of jump is ambiguous in some step. By the definition of step J2, the algorithm will never visit any permutation twice.

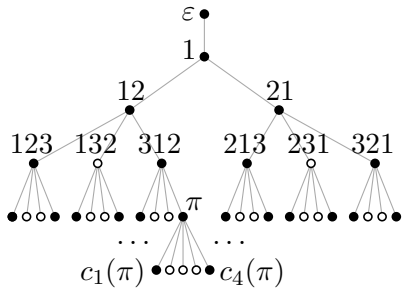
The following main result of our paper provides a sufficient condition on the set L_n to guarantee that Algorithm J is successful. This condition is captured by the following closure property of the set L_n. A set of permutations $L_n \subseteq S_n$ is called a *zigzag language*, if either $n = 0$ and $L_0 = \{\varepsilon\}$, or if $n \geq 1$ and $L_{n-1} := \{p(\pi) \mid \pi \in L_n\}$ is a zigzag language satisfying the following condition:

(z1) For every $\pi \in L_{n-1}$ we have $c_1(\pi) \in L_n$ and $c_n(\pi) \in L_n$.

Theorem 1. *Given any zigzag language of permutations L_n and initial permutation $\pi_0 = \mathrm{id}_n$, Algorithm J visits every permutation from L_n exactly once.*

Remark 2. The number of zigzag languages is at least $2^{(n-1)!(n-2)} = 2^{2^{\Theta(n \log n)}}$, i.e., more than double-exponential in n. We will see that many of these languages do in fact encode interesting combinatorial objects. Moreover, minimal jumps as performed by Algorithm J always translate to small changes on those objects in a provable sense, i.e., our algorithm defines Gray codes for a large variety of combinatorial objects, and Hamilton paths/cycles on the corresponding flip graphs and polytopes.

2.3. **The tree of permutations.** There is an intuitive characterization of zigzag languages via the *tree of permutations*. This is an infinite (unordered) rooted tree which has as nodes all permutations from S_n at distance n from the root; see the figure on the right. Specifically, the empty permutation ε is at the root, and the children of any node $\pi \in S_{n-1}$ are exactly the permutations $c_i(\pi)$, $1 \leq i \leq n$, i.e., the permutations obtained by inserting the new largest value n in all possible positions. Consequently, the parent of any node $\pi' \in S_n$ is exactly the permutation $p(\pi')$ obtained by removing the largest value n.



In the figure, for any node $\pi \in S_{n-1}$, the nodes representing the children $c_1(\pi)$ and $c_n(\pi)$ are drawn black, whereas the other children are drawn white. Any zigzag language of permutations can be obtained from this full tree by pruning subtrees, where by condition (z1) a subtree may be pruned only if its root $\pi' \in S_n$ is neither the child $c_1(\pi)$ nor the child $c_n(\pi)$ of its parent $\pi = p(\pi') \in S_{n-1}$, i.e., only subtrees rooted at white nodes may be pruned. For any subtree obtained by pruning according to this rule and for any $n \geq 1$, the remaining permutations of length n form a zigzag language L_n; see Figure 1.

Consider all nodes in the tree for which the entire path to the root consists only of black nodes. Those nodes never get pruned and are therefore contained in any zigzag language. These are exactly all permutations without peaks. A *peak* in a permutation $a_1 \dots a_n$ is a triple $a_{i-1}a_ia_{i+1}$ with $a_{i-1} < a_i > a_{i+1}$, and the language of permutations without peaks is generated by the recurrence $P_0 := \{\varepsilon\}$ and $P_n := \{c_1(\pi), c_n(\pi) \mid \pi \in P_{n-1}\}$ for $n \geq 1$. It follows that we have $|P_n| = 2^{n-1}$ and $P_n \subseteq L_n \subseteq S_n$ for any zigzag language L_n, i.e., L_n is sandwiched between the language of permutations without peaks and between the language of all permutations.

2.4. **Proof idea of Theorem 1.** Given a zigzag language L_n, we define a sequence $J(L_n)$ of all permutations from L_n and prove that Algorithm J generates the permutations of L_n exactly in

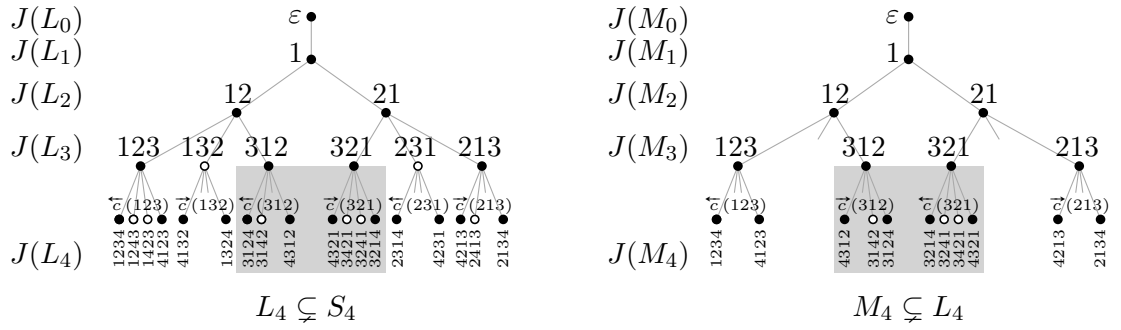


FIGURE 1. Ordered tree representation of two zigzag languages L_4 and M_4 with $M_4 \subsetneq L_4$. Both trees contain the same permutations in the highlighted subtrees, but in different order due to the node 132, which was pruned from the right tree.

this order. For any $\pi \in L_{n-1}$ we let $\overrightarrow{c}(\pi)$ be the sequence of all $c_i(\pi) \in L_n$ for $i = 1, 2, \ldots, n$, starting with $c_1(\pi)$ and ending with $c_n(\pi)$, and we let $\overleftarrow{c}(\pi)$ denote the reverse sequence, i.e., it starts with $c_n(\pi)$ and ends with $c_1(\pi)$. In words, those sequences are obtained by inserting into π the new largest value n in all possible positions from left to right, or from right to left, respectively. The sequence $J(L_n)$ is defined recursively as follows: If $n = 0$ then $J(L_0) := \varepsilon$, and if $n \geq 1$ then we consider the finite sequence $J(L_{n-1}) =: \pi_1, \pi_2, \ldots$ and define $J(L_n) := \overleftarrow{c}(\pi_1), \overrightarrow{c}(\pi_2), \overleftarrow{c}(\pi_3), \overrightarrow{c}(\pi_4), \ldots$, i.e., this sequence is obtained from the previous sequence by inserting the new largest value n in all possible positions alternatingly from right to left, or from left to right, in a zigzag fashion; see Figure 1. The proof of Theorem 1 proceeds by induction on n; see [HHMW19].

Algorithm J thus defines a left-to-right ordering of the nodes at distance n of the root in the tree representation of the zigzag language L_n described before, and this ordering is captured by the sequence $J(L_n)$; see Figure 1. Clearly, the same is true for all the zigzag languages $L_0, L_1, \ldots, L_{n-1}$ that are induced by L_n through the rule $L_{k-1} := \{p(\pi) \mid \pi \in L_k\}$ for $k = n, n-1, \ldots, 1$. The unordered tree of permutations is thus turned into an ordered tree, and it is important to realize that pruning operations change the ordering. Specifically, given two zigzag languages L_n and M_n with $M_n \subseteq L_n$, the tree for M_n is obtained from the tree for L_n by pruning, but in general $J(M_n)$ is *not* a subsequence of $J(L_n)$, as shown by the example in Figure 1. This shows that our approach is quite different from the one presented by Vajnovszki and Vernay [VV11], which considers only subsequences of the Steinhaus-Johnson-Trotter order $J(S_n)$.

2.5. **Further properties of Algorithm J.** From our proof of Theorem 1 we can easily deduce under which conditions the algorithm generates a *cyclic* listing of permutations:

Lemma 3. *In the ordering of permutations $J(L_n)$ generated by Algorithm J, the first and last permutation are related by a minimal jump if and only if $|L_k|$ is even for all $2 \leq k \leq n-1$.*

It also follows from the proof of Theorem 1 that instead of initializing the algorithm with the identity permutation $\pi_0 = \mathrm{id}_n$, we may use any permutation without peaks as a seed π_0.

2.6. **Efficiency considerations.** Let us make it very clear that in its stated form, Algorithm J is not an efficient algorithm to actually generate a particular zigzag language of permutations. The reason is that it requires storing the list of all previously visited permutations in order to

decide which one to generate next. However, by introducing a few additional arrays, the algorithm can be made memoryless, so that such lookup operations are not needed anymore, and hence no permutations need to be stored at all. The efficiency of the resulting algorithm is then only determined by the efficiency with which we are able to compute minimal jumps with respect to the input zigzag language L_n for a given entry of the permutation. This leads to an algorithm that computes the next permutation to be visited in polynomial time. In many cases, this can be improved to a loopless algorithm that generates each new permutation in constant worst-case time. The key insight here is that any jump changes the inversion table of a permutation only in a single entry. By maintaining only the inversion table, jumps can thus be performed efficiently, even if the number of steps is big. This extensive and important discussion, however, is not the focus of this paper, and is hence deferred to future parts of the series [HHMW19, HM19].

need to
te Arxiv
r second
rs

2.7. **A general recipe.** Here is a step-by-step approach to apply our framework to the generation of a given family X_n of combinatorial objects. The first step is to establish a bijection f that encodes the objects from X_n as permutations $L_n \subseteq S_n$. If L_n is a zigzag language, which can be checked by verifying the closure property, then we may run Algorithm J with input L_n, and interpret the resulting ordering $J(L_n)$ in terms of the combinatorial objects, by applying f^{-1} to each permutations in $J(L_n)$, yielding an ordering on X_n. We may also apply f^{-1} to Algorithm J directly, which will yield a simple greedy algorithm for generating X_n. The final step is to make these algorithms efficient, by introducing additional data structures that allow the change operations on X_n (which are the preimages of minimal jumps under f) as efficiently as possible.

Let us illustrate these steps for the set X_n of binary strings of length $n-1$. We map any binary string $x = x_2 \dots x_n$ to a permutation $f(x) \in S_n$ by setting $f(\varepsilon) := 1$ and $f(x_2 \dots x_n) := c_n(f(x_2 \dots x_{n-1}))$ if $x_n = 0$, and $f(x_2 \dots x_n) := c_1(f(x_2 \dots x_{n-1}))$ if $x_n = 1$, i.e., we build the permutation $f(x)$ by inserting the values $i = 2, \dots, n$ one by one, either at the leftmost or rightmost position, depending on the bit x_i. Observe that $f(X_n)$ is exactly the set of permutations without peaks $P_n \subseteq S_n$ discussed in Section 2.3 before, and a jump of the entry i in the permutation translates to flipping the bit x_i. Moreover, $f^{-1}(J(P_n))$ is exactly the well-known reflected Gray code BRGC for binary strings of length $n-1$ [Gra53], which can be implemented efficiently [BER76]. Applying f^{-1} to Algorithm J yields the following simple greedy algorithm for generating the BRGC (see [Wil13]): **J1.** Visit the initial all-zero string. **J2.** Repeatedly flip the rightmost bit that yields a previously unvisited string.

3. Pattern-avoiding permutations

The first main application of our framework is the generation of pattern-avoiding permutations. As mentioned before, many combinatorial objects are in bijection to pattern-avoiding permutations, and we will see that generating the permutations with Algorithm J yields Gray codes for those objects. Our main results in this section are summarized in Theorem 4, Remark 5, and Table 1.

3.1. **Preliminaries.** Given two permutations $\pi \in S_n$ and $\tau \in S_k$, we say that π *contains the pattern* τ, if and only if $\pi = a_1 \dots a_n$ contains a subpermutation $a_{i_1} \dots a_{i_k}$, $i_1 < \dots < i_k$, whose elements appear in the same relative order as in τ. If π does not contain the pattern τ, then we say that π *avoids* τ. For example, $\pi = 6\,35\,41\,2$ contains the pattern $\tau = 231$ at the highlighted positions. On the other hand, $\pi = 654123$ avoids $\tau = 231$. We refer to this notion of pattern-containment as

TABLE 1. Tame permutation patterns and corresponding combinatorial objects and orderings generated by Algorithm J. See also the extended table in [HHMW19].

Tame patterns	**Combinatorial objects and ordering**	**References/OEIS** [oei19]
none	permutations by adjacent transpositions → plain change order	[Joh63, Tro62], A000142
$231 = 2\underline{31}$	*Catalan families* • binary trees by rotations → Lucas--van Baronaigien-Ruskey order • triangulations by edge flips • Dyck paths by hill flips	A000108 [LvBR93]
$\underline{23}1$	*Bell families* • set partitions by element exchanges → Kaye's order	A000110 [Kay76, Wil13]
$132 \wedge 231 = \underline{13}2 \wedge 2\underline{31}$: permutations without peaks	binary strings by bitflips → reflected Gray code order (BRGC)	[Gra53], A011782
2143: vexillary permutations		[LS85], A005802
conjunction of v_k tame patterns with $v_2 = 35, v_3 = 91, v_4 = 2346$ (see [Bil13]): k-vexillary permutations ($k \geq 1$)		[BP14], A224318, A223034, A223905
$2143 \wedge 3412$: skew-merged permutations		[Sta94, Atk98], A029759
$2143 \wedge 2413 \wedge 3142$		[DMR10, SV14], A033321
$2143 \wedge 2413 \wedge 3142 \wedge 3412$: X-shaped permutations		[Wat07, Eli11], A006012
$2413 \wedge 3142$: separable permutations	*Schröder families* • slicing floorplans • topological drawings of $K_{2,n}$	A006318 [AN81, BBL98, ABP06] [CF18]
$2\underline{41}3 \wedge 3\underline{14}2$: Baxter $2\underline{41}3 \wedge 3\underline{41}2$: twisted Baxter $2\underline{14}3 \wedge 3\underline{14}2$	mosaic floorplans (=diagonal rectangulations=R-equivalent rectangulations)	[YCCG03, ABP06] [LR12, CSS18] A001181
$2\underline{14}3 \wedge 3\underline{41}2$	S-equivalent rectangulations	[ABBM+13], A214358
$2\underline{14}3 \wedge 3\underline{41}2 \wedge 2413 \wedge 3142$	S-equivalent guillotine rectangulations	[ABBM+13], A078482
$3\underline{51}24 \wedge 3\underline{51}42 \wedge 24\underline{51}3 \wedge 42\underline{51}3$: 2-clumped permutations	generic rectangulations (=rectangular drawings)	[Rea12b]
conjunction of c_k tame patterns with $c_k = 2(k/2)!(k/2+1)!$ for k even and $c_k = 2((k+1)/2)!^2$ for k odd: k-clumped permutations		[Rea12b]
conjunction of 12 tame patterns: perm. with 0-1 Schubert polynomial		[FMSD19]
$2143 \wedge 2413 \wedge 3412 \wedge 314562 \wedge 412563 \wedge 415632 \wedge 431562 \wedge 512364$ $\wedge\, 512643 \wedge 516432 \wedge 541263 \wedge 541632 \wedge 543162$: widdershins permut.		[BEV18]

a *classical pattern*. This notion can been generalized in many ways, allowing us to encode even more families of combinatorial objects. Some of these more general types of patterns are listed in Remark 5 below. In this extended abstract we formally define only one additional type, namely *vincular patterns*, introduced by Babson and Steingrímsson [BS00]. In a *vincular* pattern τ, there is exactly one underlined pair of consecutive entries, with the interpretation that an occurence of τ in π requires that the underlined entries appear at adjacent positions in π. For instance, the permutation $\pi = 3\,1\,4\,2$ contains the pattern $\tau = 231$, but it avoids the vincular pattern $\tau = \underline{23}1$.

For any permutation τ, we let $S_n(\tau)$ denote all permutations from S_n avoiding the (classical or vincular) pattern τ. For propositional formulas F and G consisting of logical ANDs $\wedge$, ORs $\vee$, and patterns as variables, we define $S_n(F \wedge G) := S_n(F) \cap S_n(G)$ and $S_n(F \vee G) := S_n(F) \cup S_n(G)$. For

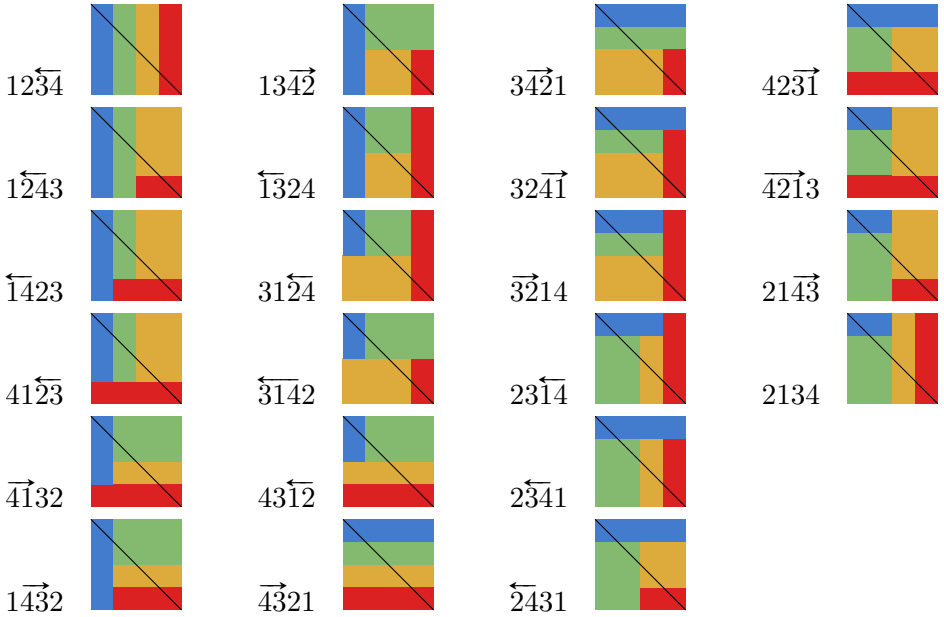


FIGURE 2. Twisted Baxter permutations ($\underline{24}13 \wedge 3\underline{41}2$-avoiding) for $n = 4$ generated by Algorithm J, with jumps highlighted by arrows, and the resulting Gray code for diagonal rectangulations. Read the figure column by column, from left to right.

instance, $S_n(\tau_1 \wedge \cdots \wedge \tau_\ell)$ is the set of permutations avoiding each of the patterns $\tau_1, \ldots, \tau_\ell$, and $S_n(\tau_1 \vee \cdots \vee \tau_\ell)$ is the set of permutations avoiding at least one of the patterns $\tau_1, \ldots, \tau_\ell$.

3.2. **Tame patterns.** The following theorem describes some mild conditions on the patterns in the formula F which ensure that we can generate $S_n(F)$ with Algorithm J.

Theorem 4. *Let F be a propositional formula consisting of logical ANDs $\wedge$, ORs $\vee$, and patterns that satisfy one of the following conditions:*

(i) classical patterns that do not have the largest value at the leftmost or rightmost position,

(ii) vincular patterns that do not have the largest value at the leftmost of rightmost position, and the largest value is part of the vincular pair.

Then $S_n(F)$, $n \geq 0$, is a zigzag language of permutations, and hence can be generated by Algorithm J.

We say that a pattern satisfying (i) or (ii) is *tame*. Table 1 lists several tame classical and vincular patterns and the combinatorial objects encoded by the corresponding zigzag languages. The bijections between those permutations and the combinatorial objects are well-known and are described in the listed papers (recall also Section 2.7). Some patterns are interesting in their own right and have no 'natural' associated combinatorial objects (empty second column). The resulting ordering for twisted Baxter permutations of length $n = 4$, and the resulting Gray code for diagonal rectangulations, is shown in Figure 2. See also Figures 3 and 4 for the resulting Gray codes for three different Catalan objects (231-avoiding permutations) and for set partitions ($\underline{23}1$-avoiding permutations), respectively.

Remark 5. In addition to classical and vincular patterns, the literature knows a large number of additional types of permutation patterns. Algorithm J works successfully for almost all of them, i.e.,

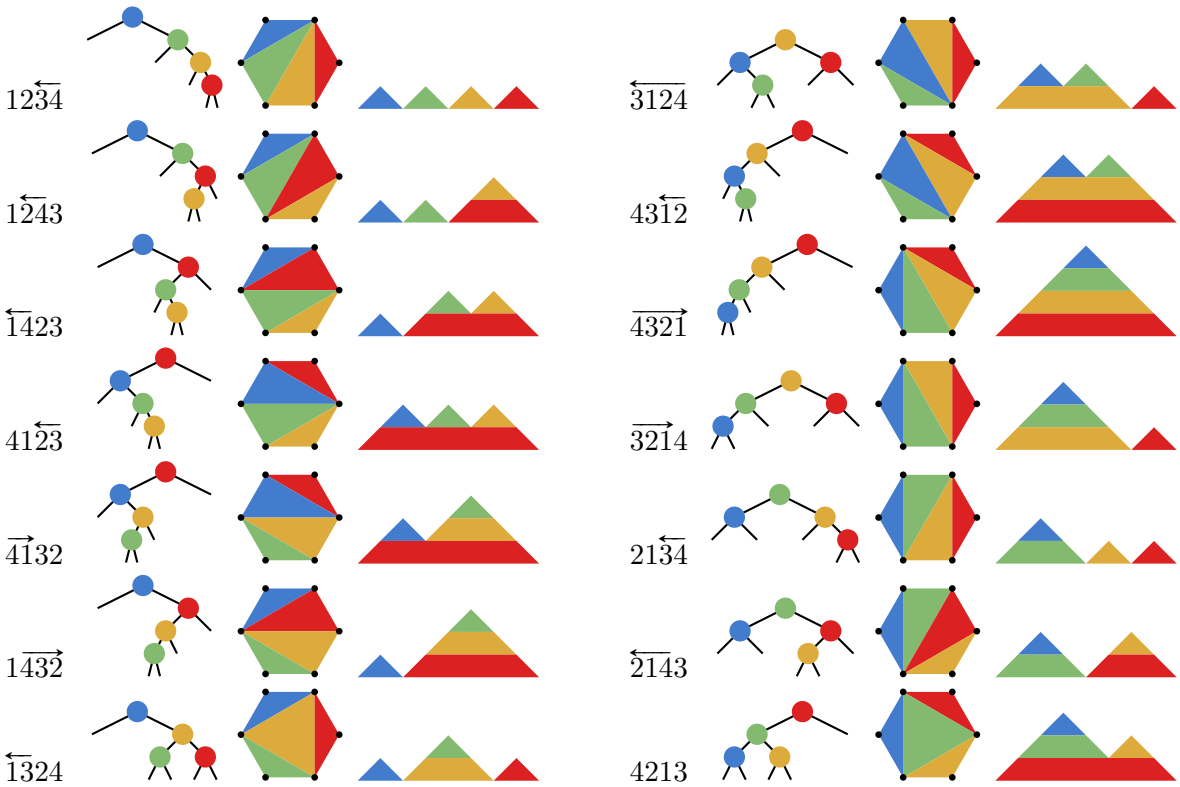


FIGURE 3. 231-avoiding permutations of length $n = 4$ generated by Algorithm J and resulting Gray codes for Catalan families (binary trees, triangulations, Dyck paths).

$12\overleftarrow{34}$	1|2|3|4	$\overleftarrow{3142}$	13|24
$1\overleftarrow{243}$	1|2|34	$4\overleftarrow{312}$	134|2
$\overleftarrow{1423}$	1|24|3	$\overrightarrow{4321}$	1234
$41\overleftarrow{23}$	14|2|3	$\overrightarrow{3214}$	123|4
$\overrightarrow{4132}$	14|23	$2\overleftarrow{134}$	12|3|4
$1\overrightarrow{432}$	1|234	$\overleftarrow{2143}$	12|34
$\overleftarrow{1324}$	1|23|4	4213	124|3
$31\overleftarrow{24}$	13|2|4		

FIGURE 4. $\underline{231}$-avoiding permutations of length $n = 4$ generated by Algorithm J and resulting Gray code for set partitions.

the list of pattern types started in Theorem 4 can be extended considerably, by formulating suitable (still very mild) conditions for each of the following pattern types: (iii) bivincular patterns, introduced by Bousquet-Mélou, Claesson, Dukes, and Kitaev [BMCDK10]; (iv) barred patterns, introduced by West [Wes90]; (v) patterns with Bruhat restrictions, introduced by Woo and Yong [WY06]; (vi) mesh patterns, introduced by Brändén and Claesson [BC11]; (vii) partially ordered patterns, introduced by Kitaev [Kit05]; (viii) monotone grid classes, introduced by Huczynska and Vatter [HV06];

(ix) geometric grid classes, introduced by Albert, Atkinson, Bouvel, Ruškuc, and Vatter [AAB+13]. See [HHMW19] for more details.

3.3. **Patterns with multiplicities.** All the aforementioned results generalize to bounding the number of appearances of any tame pattern. Formally, a *counted pattern* is a pair $\sigma = (\tau, c)$, where τ is a pattern of any of the types discussed before, and c is a non-negative integer. Moreover, $S_n(\sigma)$ denotes the set of all permutations from S_n that contain at most c occurences of the pattern τ, where the special case $c = 0$ is pattern-avoidance. We can now form propositional formulas F consisting of logical ANDs $\wedge$, ORs $\vee$, and counted patterns (τ_i, c_i) with tame τ_i as variables, yielding a zigzag language $S_n(F)$ that can be generated by Algorithm J. A somewhat contrived example for such a language would be $F = \big((231, 3) \wedge (2143, 5)\big) \vee (3\underline{14}2, 2)$, the language of permutations that contain at most 3 occurences of the pattern 231 AND at most 5 occurences of the pattern 2143, OR at most 2 occurences of the vincular pattern $3\underline{14}2$.

4. Lattice congruences of the weak order

The second main application of our framework are lattice congruences of the weak order on the symmetric group S_n. With lattice congruences, we obtain a provable notion of optimality for the Gray codes obtained from Algorithm J: *Jump operations translate into walks along the edges of a polytope*; see Figure 6. The main results in this section are summarized in Theorems 6 and Corollary 7.

4.1. **Preliminaries.** We begin recalling a few basic notion from poset theory. The *weak order* on the symmetric group S_n is the poset obtained by considering the *inversion set* of a permutation, defined as $\mathrm{inv}(\pi) := \{(\pi(i), \pi(j)) \mid 1 \le i < j \le n \text{ and } \pi(i) > \pi(j)\}$, and by defining $\pi < \rho$ if and only if $\mathrm{inv}(\pi) \subseteq \mathrm{inv}(\rho)$; see the left hand side of Figure 5. The cover relations in this poset, drawn as edges in the cover graph shown in the figure, are exactly adjacent transpositions between the two involved permutations. Moreover, the weak order is a lattice, i.e., for any two $\pi, \rho \in S_n$ there is a unique smallest element σ, called the *join* $\pi \vee \rho$ *of* π *and* ρ, such that $\sigma > \pi$ and $\sigma > \rho$, and there is unique largest element σ, called the *meet* $\pi \wedge \rho$ *of* π *and* ρ, satisfying $\sigma < \pi$ and $\sigma < \rho$. A *lattice congruence* of the weak order is an equivalence relation $\equiv$ on S_n that is compatible with taking joins and meets, i.e., $\pi \equiv \pi'$ and $\rho \equiv \rho'$ implies that $\pi \vee \rho \equiv \pi' \vee \rho'$ and $\pi \wedge \rho \equiv \pi' \wedge \rho'$. Given any lattice congruence $\equiv$, we obtain the *lattice quotient* $S_n/\equiv$ by taking the equivalence classes as elements, and ordering them by $X < Y$ if and only if there is a $\pi \in X$ and a $\rho \in Y$ such that $\pi < \rho$ in S_n. See Figure 5 for an example of a lattice congruence (left) and the resulting lattice quotient (right).

It turns out that there are double-exponentially many distinct lattice congruences of the weak order on S_n, and they generalize many known lattices, such as the Boolean lattice, the Tamari lattice [Tam62] (shown on the right hand side of Figure 5), and certain Cambrian lattices [Rea06, CP17]. This area of study has beautiful ramifications into groups, posets, polytopes, geometry, and combinatorics, and has been developed considerably in recent years, in particular thanks to Nathan Reading's works, summarized in [Rea12a, Rea16a, Rea16b].

4.2. **Jumping through lattice congruences.** For any lattice congruence $\equiv$ of the weak order on S_n, a *set of representatives* for the equivalence classes $S_n/\equiv$ is a subset $R_n \subseteq S_n$ such that for every equivalence class $X \in S_n/\equiv$, exactly one permutation is contained in R_n, i.e., $|X \cap R_n| = 1$. We let $X(\pi)$, $\pi \in S_n$, denote the equivalence class from $S_n/\equiv$ containing π. A meaningful definition of 'generating the lattice congruence' is to generate a set of representatives for its equivalence

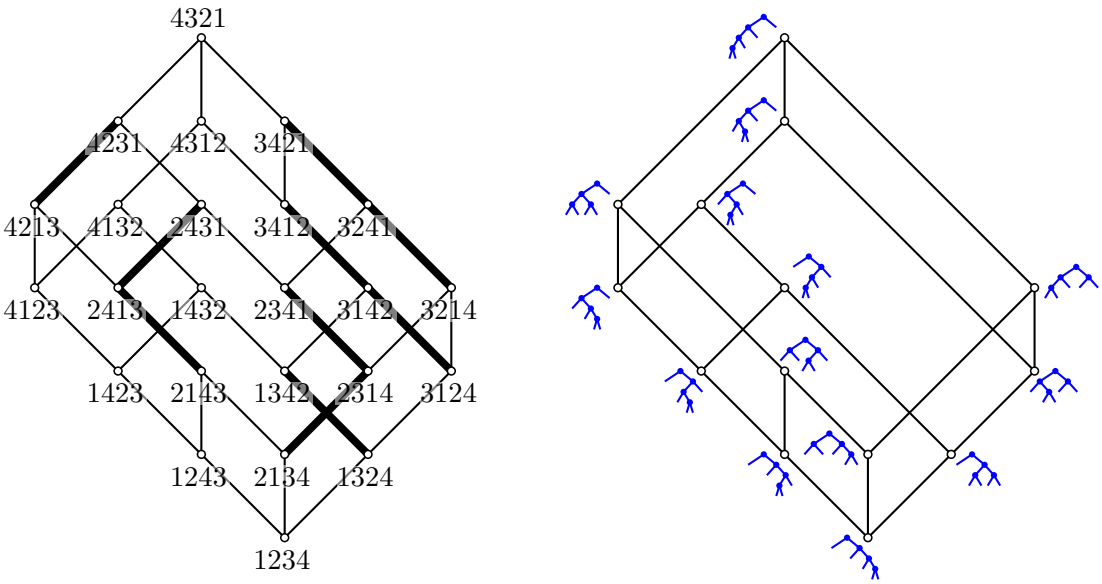


FIGURE 5. Hasse diagrams of the weak order on S_4 (left), with the lattice congruence for 231-avoiding permutations (bold edges), and of the resulting lattice quotient $S_n/\equiv$ (right), which is the well-known Tamari lattice (with corresponding binary trees).

classes. We also require that any two successive representatives form a cover relation in the lattice quotient $S_n/\equiv$. This is what we achieve with the help of Algorithm J.

Theorem 6. *For every lattice congruence $\equiv$ of the weak order on S_n, there is a set of representatives $R_n \subseteq S_n$, such that Algorithm J generates a sequence $J(R_n) = \pi_1, \pi_2, \ldots$ of all permutations from R_n for which the equivalence classes $X(\pi_1), X(\pi_2), \ldots$ form a Hamilton path in the cover graph of the lattice quotient $S_n/\equiv$.*

For every lattice congruence $\equiv$, Pilaud and Santos [PS19] defined a polytope, called the *quotientope* for $\equiv$, whose skeleton is exactly the cover graph of the lattice quotient $S_n/\equiv$. These polytopes generalize many known polytopes, such as hypercubes, associahedra, permutahedra etc. The following result is an immediate corollary of Theorem 6, and it is illustrated in Figure 6.

Corollary 7. *For every lattice congruence $\equiv$ of the weak order on S_n, Algorithm J generates a Hamilton path on the skeleton of the corresponding quotientope.*

5. ACKNOWLEDGMENTS

We thank Michael Albert, Mathilde Bouvel, Sergi Elizalde, Vít Jelínek, Sergey Kitaev, Vincent Vajnovszki, and Vincent Vatter for very insightful feedback on this work, for pointing out relevant references, and for sharing their knowledge about pattern-avoiding permutations. We also thank Jean Cardinal and Vincent Pilaud for several stimulating discussions about lattice congruences of the weak order on the symmetric group. Figure 6 in this paper was obtained by modifying and augmenting Figure 9 from [PS19], and the original source code for this figure was provided to us by Vincent Pilaud.

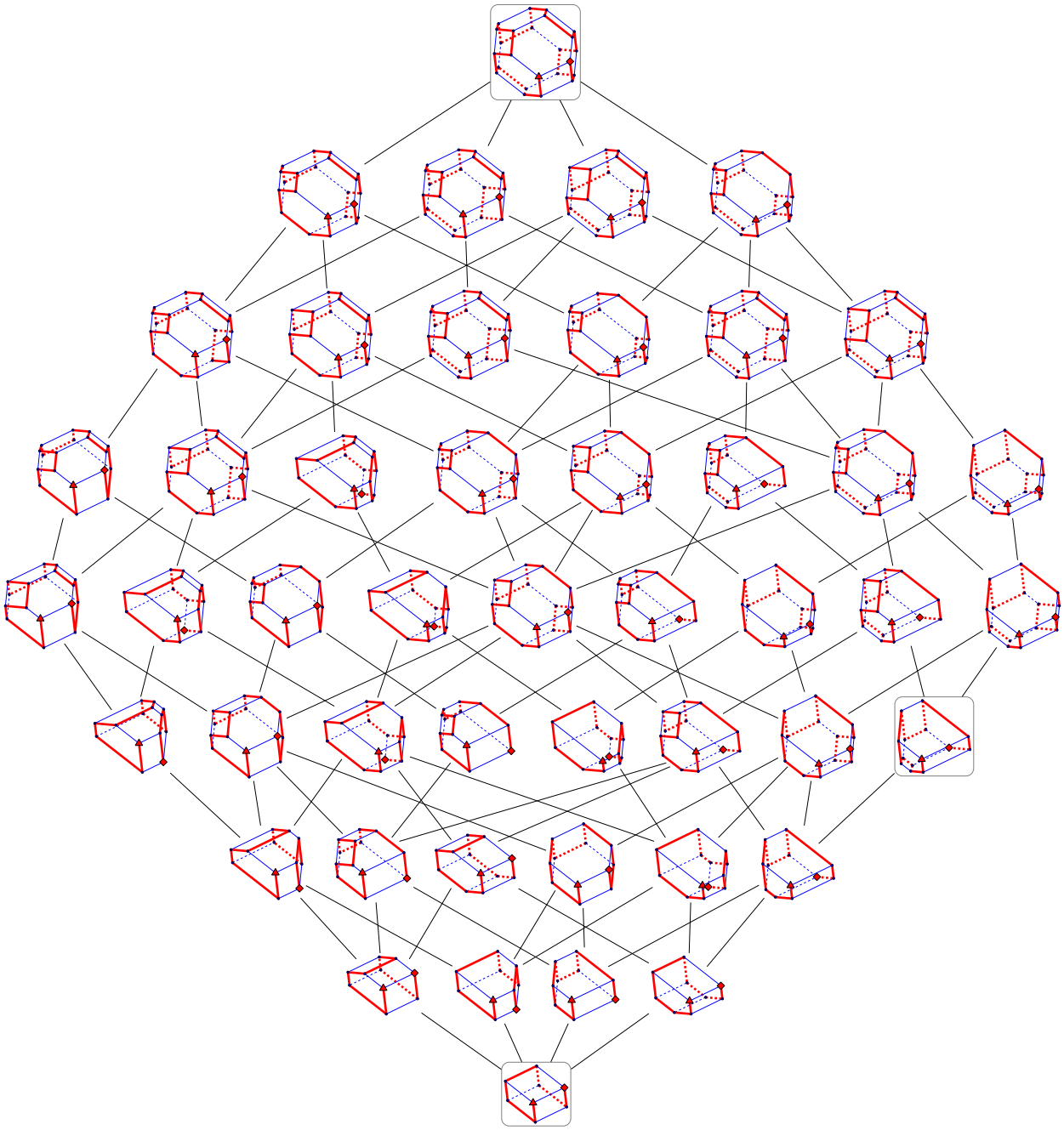


FIGURE 6. Lattice congruences of the weak order on S_4, ordered by refinement and realized as polytopes (only full-dimensional polytopes shown). Hamilton paths generated by Algorithm J are highlighted, with start vertex (triangle) and end vertex (diamond). Permutahedron, associahedron (one of four isomorphic variants) and hypercube are framed.

References

[AAB+13] M. H. Albert, M. D. Atkinson, M. Bouvel, N. Ruškuc, and V. Vatter. Geometric grid classes of permutations. *Trans. Amer. Math. Soc.*, 365(11):5859–5881, 2013.

[ABBM+13] A. Asinowski, G. Barequet, M. Bousquet-Mélou, T. Mansour, and R. Y. Pinter. Orders induced by segments in floorplans and (2-14-3, 3-41-2)-avoiding permutations. *Electron. J. Combin.*, 20(2):Paper 35, 43, 2013.

[ABP06] E. Ackerman, G. Barequet, and R. Y. Pinter. A bijection between permutations and floorplans, and its applications. *Discrete Appl. Math.*, 154(12):1674–1684, 2006.

[AF96] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65(1-3):21–46, 1996. First International Colloquium on Graphs and Optimization (GOI), 1992 (Grimentz).

[AN81] D. Avis and M. Newborn. On pop-stacks in series. *Utilitas Math.*, 19:129–140, 1981.

[Atk98] M. D. Atkinson. Permutations which are the union of an increasing and a decreasing subsequence. *Electron. J. Combin.*, 5:Research paper 6, 13, 1998.

[Bar08] J.-L. Baril. Efficient generating algorithm for permutations with a fixed number of excedances. *Pure Math. Appl. (PU.M.A.)*, 19(2-3):61–69, 2008.

[Bar09] J.-L. Baril. More restrictive Gray codes for some classes of pattern avoiding permutations. *Inform. Process. Lett.*, 109(14):799–804, 2009.

[BBGP04] S. Bacchelli, E. Barcucci, E. Grazzini, and E. Pergola. Exhaustive generation of combinatorial objects by ECO. *Acta Inform.*, 40(8):585–602, 2004.

[BBL98] P. Bose, J. F. Buss, and A. Lubiw. Pattern matching for permutations. *Inform. Process. Lett.*, 65(5):277–283, 1998.

[BC11] P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *Electron. J. Combin.*, 18(2):Paper 5, 14, 2011.

[BDLPP99] E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani. ECO: a methodology for the enumeration of combinatorial objects. *J. Differ. Equations Appl.*, 5(4-5):435–490, 1999.

[BER76] J. R. Bitner, G. Ehrlich, and E. M. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Comm. ACM*, 19(9):517–521, 1976.

[BEV18] R. Brignall, M. Engen, and V. Vatter. A counterexample regarding labelled well-quasi-ordering. *Graphs Combin.*, 34(6):1395–1409, 2018.

[Bil13] S. Billey. Permutation patterns for k-vexillary permutations, 2013. `https://sites.math.washington.edu/~billey/papers/k.vex.html`.

[BMCDK10] M. Bousquet-Mélou, A. Claesson, M. Dukes, and S. Kitaev. $(2+2)$-free posets, ascent sequences and pattern avoiding permutations. *J. Combin. Theory Ser. A*, 117(7):884–909, 2010.

[Bón12] M. Bóna. *Combinatorics of permutations*. Discrete Mathematics and its Applications (Boca Raton). CRC Press, Boca Raton, FL, second edition, 2012. With a foreword by Richard Stanley.

[BP14] S. Billey and B. Pawlowski. Permutation patterns, Stanley symmetric functions, and generalized Specht modules. *J. Combin. Theory Ser. A*, 127:85–120, 2014.

[BS00] E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the Mahonian statistics. *Sém. Lothar. Combin.*, 44:Art. B44b, 18, 2000.

[CF18] J. Cardinal and S. Felsner. Topological drawings of complete bipartite graphs. *J. Comput. Geom.*, 9(1):213–246, 2018.

[CP17] G. Chatel and V. Pilaud. Cambrian Hopf algebras. *Adv. Math.*, 311:598–633, 2017.

[CSS18] J. Cardinal, V. Sacristán, and R. I. Silveira. A note on flips in diagonal rectangulations. *Discrete Math. Theor. Comput. Sci.*, 20(2):Paper No. 14, 22, 2018.

[DFMV08] W. M. B. Dukes, M. F. Flanagan, T. Mansour, and V. Vajnovszki. Combinatorial Gray codes for classes of pattern avoiding permutations. *Theoret. Comput. Sci.*, 396(1-3):35–49, 2008.

[DMR10] E. Deutsch, E. Munarini, and S. Rinaldi. Skew Dyck paths. *J. Statist. Plann. Inference*, 140(8):2191–2203, 2010.

[DTV19] P. T. Do, T. T. H. Tran, and V. Vajnovszki. Exhaustive generation for permutations avoiding (colored) regular sets of patterns. *Discrete Applied Mathematics*, 2019.

[Eli07] S. Elizalde. Generating trees for permutations avoiding generalized patterns. *Ann. Comb.*, 11(3-4):435–458, 2007.

[Eli11] S. Elizalde. The X-class and almost-increasing permutations. *Ann. Comb.*, 15(1):51–68, 2011.

[FMSD19] A. Fink, K. Mészáros, and A. St. Dizier. Zero-one Schubert polynomials. `https://arxiv.org/abs/1903.10332`, 2019.

[FS09] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, Cambridge, 2009.

[GM14] S. Guillemot and D. Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–101. ACM, New York, 2014.

[Gra53] F. Gray. Pulse code communication, 1953. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.

[HHMW19] E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. `https://arxiv.org/abs/1906.06069`, 2019.

[HM19] H. P. Hoang and T. Mütze. Combinatorial generation via permutation languages. II. Lattice congruences. `https://arxiv.org/abs/xxxx.xxxxx`, 2019.

[HV06] S. Huczynska and V. Vatter. Grid classes and the Fibonacci dichotomy for restricted permutations. *Electron. J. Combin.*, 13(1):Research Paper 54, 14, 2006.

[Jer03] M. Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003.

[JK17] V. Jelínek and J. Kynčl. Hardness of permutation pattern matching. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 378–396. SIAM, Philadelphia, PA, 2017.

[Joh63] S. Johnson. Generation of permutations by adjacent transposition. *Math. Comp.*, 17:282–285, 1963.

[Kay76] R. Kaye. A Gray code for set partitions. *Information Processing Lett.*, 5(6):171–173, 1976.

[Kit05] S. Kitaev. Partially ordered generalized patterns. *Discrete Math.*, 298(1-3):212–229, 2005.

[Kit11] S. Kitaev. *Patterns in permutations and words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg, 2011. With a foreword by Jeffrey B. Remmel.

[Knu98] D. E. Knuth. *The art of computer programming. Vol. 3*. Addison-Wesley, Reading, MA, 1998. Sorting and searching, Second edition [of MR0445948].

[Knu11] D. E. Knuth. *The art of computer programming. Vol. 4A. Combinatorial algorithms. Part 1*. Addison-Wesley, Upper Saddle River, NJ, 2011.

[Koz19] L. Kozma. Faster and simpler algorithms for finding large patterns in permutations. `https://arxiv.org/abs/1902.08809`, 2019.

[LR12] S. Law and N. Reading. The Hopf algebra of diagonal rectangulations. *J. Combin. Theory Ser. A*, 119(3):788–824, 2012.

[LS85] A. Lascoux and M.-P. Schützenberger. Schubert polynomials and the Littlewood-Richardson rule. *Lett. Math. Phys.*, 10(2-3):111–124, 1985.

[LS09] Y. Li and J. Sawada. Gray codes for reflectable languages. *Inform. Process. Lett.*, 109(5):296–300, 2009.

[LvBR93] J. M. Lucas, D. R. van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366, 1993.

[oei19] OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2019. `http://oeis.org`.

[PS19] V. Pilaud and F. Santos. Quotientopes. *Bull. Lond. Math. Soc.*, 51:406–420, 2019.

[Rea06] N. Reading. Cambrian lattices. *Adv. Math.*, 205(2):313–353, 2006.

[Rea12a] N. Reading. From the Tamari lattice to Cambrian lattices and beyond. In *Associahedra, Tamari lattices and related structures*, volume 299 of *Prog. Math. Phys.*, pages 293–322. Birkhäuser/Springer, Basel, 2012.

[Rea12b] N. Reading. Generic rectangulations. *European J. Combin.*, 33(4):610–623, 2012.

[Rea16a] N. Reading. Finite Coxeter groups and the weak order. In *Lattice theory: special topics and applications. Vol. 2*, pages 489–561. Birkhäuser/Springer, Cham, 2016.

[Rea16b] N. Reading. Lattice theory of the poset of regions. In *Lattice theory: special topics and applications. Vol. 2*, pages 399–487. Birkhäuser/Springer, Cham, 2016.

[RSW12] F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *J. Combin. Theory Ser. A*, 119(1):155–169, 2012.

[Sav97] C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997.

[Sta94] Z. E. Stankova. Forbidden subsequences. *Discrete Math.*, 132(1-3):291–316, 1994.

[SV14] R. Smith and V. Vatter. A stack and a pop stack in series. *Australas. J. Combin.*, 58:157–171, 2014.

[SW12] J. Sawada and A. Williams. Efficient oracles for generating binary bubble languages. *Electron. J. Combin.*, 19(1):Paper 42, 20, 2012.

[Tam62] D. Tamari. The algebra of bracketings and their enumeration. *Nieuw Arch. Wisk. (3)*, 10:131–146, 1962.

[Ten18] B. Tenner. Database of permutation pattern avoidance, 2018. `https://math.depaul.edu/bridget/patterns.html`.

[Tro62] H. F. Trotter. Algorithm 115: Perm. *Commun. ACM*, 5(8):434–435, 1962.

[VV11] V. Vajnovszki and R. Vernay. Restricted compositions and permutations: from old to new Gray codes. *Inform. Process. Lett.*, 111(13):650–655, 2011.

[Wat07] S. D. Waton. *On permutation classes defined by token passing networks, gridding matrices and pictures: three flavours of involvement.* PhD thesis, University of St Andrews, 2007.

[Wes90] J. West. *Permutations with forbidden subsequences and stack-sortable permutations*. ProQuest LLC, Ann Arbor, MI, 1990. Thesis (Ph.D.)–Massachusetts Institute of Technology.

[Wil13] A. Williams. The greedy Gray code algorithm. In *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 525–536, 2013.

[WY06] A. Woo and A. Yong. When is a Schubert variety Gorenstein? *Adv. Math.*, 207(1):205–220, 2006.

[XCU10] L. Xiang, K. Cheng, and K. Ushijima. Efficient generation of Gray codes for reflectable languages. In *Computational Science and Its Applications - ICCSA 2010, International Conference, Fukuoka, Japan, March 23-26, 2010, Proceedings, Part IV*, pages 418–426, 2010.

[YCCG03] B. Yao, H. Chen, C.-K. Cheng, and R. L. Graham. Floorplan representations: Complexity and connections. *ACM Trans. Design Autom. Electr. Syst.*, 8(1):55–80, 2003.