COMBINATORIAL GENERATION VIA PERMUTATION LANGUAGES. III. RECTANGULATIONS

ARTURO MERINO AND TORSTEN MÜTZE

Abstract. A generic rectangulation is a partition of a rectangle into finitely many interior-disjoint rectangles, such that no four rectangles meet in a point. In this work we present a versatile algorithmic framework for exhaustively generating a large variety of different classes of generic rectangulations. Our algorithms work under very mild assumptions, and apply to a large number of rectangulation classes known from the literature, such as generic rectangulations, diagonal rectangulations, 1-sided/area-universal, block-aligned rectangulations, and their guillotine variants, including aspect-ratio-universal rectangulations. They also apply to classes of rectangulations that are characterized by avoiding certain patterns, and in this work we initiate a systematic investigation of pattern avoidance in rectangulations. Our generation algorithms are efficient, in some cases even loopless or constant amortized time, i.e., each new rectangulation is generated in constant time in the worst case or on average, respectively. Moreover, the Gray codes we obtain are cyclic, and sometimes provably optimal, in the sense that they correspond to a Hamilton cycle on the skeleton of an underlying polytope. These results are obtained by encoding rectangulations as permutations, and by applying our recently developed permutation language framework.

1. Introduction

Partitioning a geometric shape into smaller shapes is a fundamental theme in discrete and combinatorial geometry. In this paper we consider *rectangulations*, i.e., partitions of a rectangle into finitely many interior-disjoint rectangles. Such partitions have an abundance of practical applications, which motivates their combinatorial and algorithmic study. For example, rectangulations are an appealing way to represent geographic information as a cartogram. This is a map where each country is represented as a rectangle, the adjacencies between rectangles correspond to those between countries, and the areas of the rectangles are determined by some geographic variable, such as population size [vKS07]. If the rectangulation is *area-universal* [EMSV12] or *aspect-ratio-universal* [FNT21], respectively, then such an adjacency-preserving cartogram can be drawn for any assignment of area values or aspect ratios to the rectangles. Another important use of rectangulations is as floorplans in VLSI design and architectural design. These problems often involve additional constraints on top of adjacency, such as extra space for wires [Ott82] or proportion limits for the rooms [MSL76]. An important notion in this context are *slicing* floorplans [Ott82], also known as *guillotine* floorplans, i.e.,

(Arturo Merino) Department of Mathematics, TU Berlin, Germany

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

E-mail addresses: merino@math.tu-berlin.de, torsten.mutze@warwick.ac.uk.

Key words and phrases. Exhaustive generation, Gray code, flip graph, polytope, generic rectangulation, diagonal rectangulation, cartogram, floorplan, permutation pattern.

An extended abstract of this paper appeared in the Proceedings of SoCG 2021. Arturo Merino was supported by ANID Becas Chile 2019-72200522. Torsten Mütze is also affiliated with the Faculty of Mathematics and Physics, Charles University Prague, Czech Republic, and he was supported by Czech Science Foundation grant GA 19-08554S. Both authors were supported by German Science Foundation grant 413902284.

floorplans that can be subdivided into their constituent rectangles by a sequence of straight vertical or horizontal cuts.

Rectangulations have rich combinatorial properties, and a task that has received a lot of attention is counting, i.e., determining the number of rectangulations of a particular type with n rectangles, either exactly as a function of n [YCCG03] or asymptotically as n grows [SC03]. This led to several beautiful bijections of rectangulations with pattern-avoiding permutations [ABP06a, Rea12, ABBM$^+$13] or with twin binary trees [YCCG03]. The focus of this paper is on another fundamental algorithmic task, which is more fine-grained than counting, namely exhaustive generation, meaning that every rectangulation from a given class must be produced exactly once. While such generation algorithms are known for many other discrete objects such as permutations, combinations, subsets, trees etc. and covered in standard textbooks such as Knuth's [Knu11], much less is known about the generation of geometric objects such as rectangulations.

The ultimate goal for a generation algorithm is to produce each new object in time $\mathcal{O}(1)$, which requires that consecutively generated objects differ only by a 'small local change'. Such a minimum change listing of combinatorial objects is often called a *Gray code* [Sav97]. If the time bound $\mathcal{O}(1)$ for producing the next object holds in every step, then the algorithm is called *loopless* [Ehr73], and if it holds on average it is called *constant amortized time* (CAT) [Rus16]. The Gray code problem entails the definition of a *flip graph*, which has as nodes all the combinatorial objects to be generated, and an edge between any two objects that differ in the specified small way. Clearly, computing a Gray code ordering of the objects is equivalent to traversing a Hamilton path or cycle in the corresponding flip graph. It turns out that some interesting flip graphs arising from rectangulations can be equipped with a natural lattice structure [Mee19], analogous to the Tamari lattice on triangulations, and realized as polytopes in high-dimensional space [LR12], analogous to the associahedron (see [PPR21] for generalizations). This ties in the Gray code problem with deep methods and results from lattice and polytope theory.

1.1. **Our results.** The main contribution of this paper is a versatile algorithmic framework for generating a large variety of different classes of generic rectangulations, i.e., rectangulations with the property that no four rectangles meet in a point. In particular, we obtain efficient generation algorithms for several interesting classes known from the literature, in some cases loopless or CAT algorithms; see Table 1.

The initialization time and memory requirement for all these algorithms is linear in the number of rectangles. The classes of rectangulations shown in the table arise from generic rectangulations by imposing structural constraints, such as the guillotine property or forbidden configurations, or by equivalence relations, and they will be defined in Section 2.2. We implemented the algorithms generating the classes of rectangulations from the table in C++, and we made the code available for download and experimentation on the Combinatorial Object Server [cos].

The classes of rectangulations that our algorithms can generate are not limited to the examples shown in Table 1, but can be described by the following *closure property*; see Figure 1. Given an infinite class of rectangulations $\mathcal{C}$, we require that if a rectangulation R is contained in $\mathcal{C}$, then the rectangulation obtained from R by deleting the bottom-right rectangle is also in $\mathcal{C}$, and the two rectangulations obtained from R by inserting a new rectangle at the bottom or right, respectively, are also in $\mathcal{C}$ (formal definitions of deletion and insertion are given in Section 2). If $\mathcal{C}$ satisfies this property, then our algorithms allow generating the set $\mathcal{C}_n \subseteq \mathcal{C}$ of all rectangulations from $\mathcal{C}$ with exactly n rectangles, for every $n \geq 1$, by so-called *jumps*, a minimum change operation that generalizes simple flips, T-flips, and wall slides studied in [Rea12, CSS18] (the formal definition of jumps is in Section 3.1). Moreover, if the class $\mathcal{C}$ is symmetric, i.e., if R

TABLE 1. Classes of rectangulations that can be generated by our algorithms. The second column gives a description of the class in terms of forbidden rectangulation patterns (n/a means not applicable), and one or more bijectively equivalent classes of pattern-avoiding permutations. Underlined permutation patterns are so-called vincular patterns; see [HHMW22] and the papers referenced in the table. The last column specifies the obtained runtime bound for generating each rectangulation, where n is the number of rectangles. These are all worst case bounds that apply in every step (in particular, LL=loopless), with the exception of the $\mathcal{O}(1)$ bound for generic rectangulations, which holds on average (CAT=constant amortized time). For more extensive counting results on pattern-avoiding rectangulations, see Section 10.

Class	**Forbidden patterns**	**Counts/OEIS** [oei20]	**Refs.**	**Runtime**
generic	$\emptyset$ $\{3\underline{51}24, 3\underline{51}42, 24\underline{51}3, 42\underline{51}3\}$: 2-clumped permutations	$1, 2, 6, 24, 116, 642, 3938, 26194, \ldots$ A342141	[Rea12, Mee19]	$\mathcal{O}(1)$ CAT Thm. 12
diagonal =mosaic floorpl. /R-equivalence	$\{$  $\}$ $\{2\underline{41}3, 3\underline{14}2\}$: Baxter $\{2\underline{41}3, 3\underline{41}2\}$: twisted Baxter $\{2\underline{14}3, 3\underline{14}2\}$	$1, 2, 6, 22, 92, 422, 2074, 10754, \ldots$ A001181 (Baxter numbers)	[YCCG03, ABP06a, LR12, CSS18]	$\mathcal{O}(1)$ LL Thm. 15
1-sided =area-universal	$\{$  $\}$ $\{2\underline{41}3, 3\underline{14}2, 2\underline{14}3, 3\underline{41}2\}$	$1, 2, 6, 20, 72, 274, 1088, 4470, \ldots$	[EMSV12, Lei21]	$\mathcal{O}(n)$ Thm. 19
block-aligned /S-equivalence	n/a $\{2\underline{14}3, 3\underline{41}2\}$	$1, 1, 2, 6, 22, 88, 374, 1668, 7744, \ldots$ A214358	[ABBM$^+$13]	$\mathcal{O}(1)$ LL Thm. 37
guillotine: generic	$\{$$\}$	$1, 2, 6, 24, 114, 606, 3494, 21434, \ldots$		$\mathcal{O}(n)$ Thm. 19
guillotine: diagonal =slicing fl.pl. /R-equiv.	$\{$  $\}$ $\{2413, 3142\}$: separable	$1, 2, 6, 22, 90, 394, 1806, 8558, \ldots$ A006318 (Schröder numbers)	[YCCG03, ABP06a, AM10, ABBM$^+$13]	$\mathcal{O}(n)$ Thm. 19
guillotine: 1-sided =aspect-ratio- universal	$\{$  $\}$ $\{2413, 3142, 2\underline{14}3, 3\underline{41}2\}$	$1, 2, 6, 20, 70, 254, 948, 3618, \ldots$ A078482	[AM10, FNT21, Lei21]	$\mathcal{O}(n)$ Thm. 19
guillotine:	$\{$$\}$ $\{2413, 3142, 2143, 3412\}$	$1, 2, 6, 20, 68, 232, 792, 2704, \ldots$ A006012	[AM10]	$\mathcal{O}(n^2)$ Thm. 19
guillotine: block-aligned /S-equiv.	n/a $\{2413, 3142, 2\underline{14}3, 3\underline{41}2\}$	$1, 1, 2, 6, 20, 70, 254, 948, 3618, \ldots$ A078482	[ABBM$^+$13]	$\mathcal{O}(n)$ Thm. 38

is in $\mathcal{C}$ then the rectangulation obtained from R by reflection at the diagonal from top-left to bottom-right is also in $\mathcal{C}$, then the jump Gray code for $\mathcal{C}_n$ is cyclic, i.e., the last rectangulation differs from the first one only by a jump. In other words, we not only obtain a Hamilton path in the corresponding flip graph, but a Hamilton cycle. In fact, all the classes of rectangulations listed in Table 1 satisfy the aforementioned closure and symmetry properties, so in all those cases we obtain cyclic jump Gray codes.

Generic rectangulations and diagonal rectangulations, shown in the first two rows of Table 1, have an underlying lattice and polytope structure [LR12, Mee19], and in those two cases our Gray codes form a Hamilton cycle on the skeleton of this polytope, i.e., jumps are provably

optimal minimum change operations. The Gray codes for these two rectangulation classes with $n = 1, \ldots, 5$ rectangles are shown in the appendix.

It turns out that many interesting classes of rectangulations can be characterized by pattern avoidance; see the second column in Table 1. Under very mild conditions on the patterns, these classes satisfy the aforementioned closure property, and can hence be generated by our framework. In this work we initiate a systematic investigation of pattern avoidance in rectangulations, and we obtain the first counting results for many known and new classes; see the third column in Table 1 and the more extensive tables in Section 10.

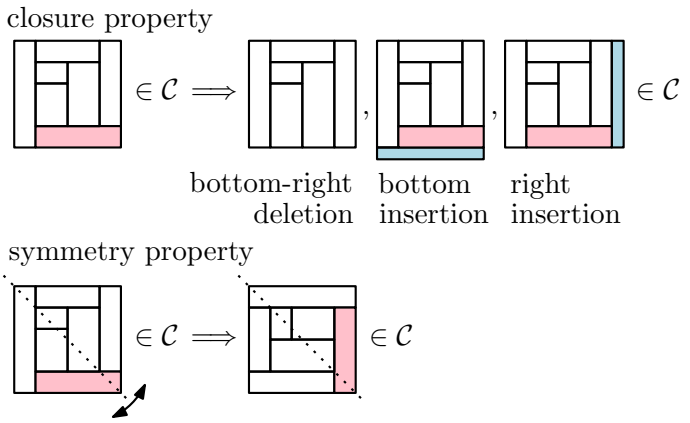


FIGURE 1. Closure property and symmetry property.

Our generation framework for rectangulations consists of two main algorithms. The first is a simple greedy algorithm that generates a jump Gray code ordering for any set of rectangulations $\mathcal{C}_n \subseteq \mathcal{C}$ for which $\mathcal{C}$ satisfies the aforementioned closure property; see Algorithm $\mathrm{J}^{\square}$ and Theorem 5 in Section 3. The second is a memoryless version of the first algorithm, which computes the same ordering of rectangulations; see Algorithm $\mathrm{M}^{\square}$ and Theorem 8 in Section 5. This algorithm can be fine-tuned to derive efficient algorithms for several known rectangulation classes such as the ones listed in Table 1, by providing corresponding jump oracles for the class $\mathcal{C}$.

To prove Theorems 5 and 8, we encode rectangulations by permutations as described by Reading [Rea12], and we then apply our framework for exhaustively generating permutation languages presented in [HHMW22, HM21]. The minimum change operations on permutations used in that framework translate to jumps on rectangulations. Generating different classes of rectangulations efficiently is thus another major new application of our permutation language framework, and in this paper we flesh out the details of this application.

1.2. **Related work.** There has been some prior work on generating a few special classes of rectangulations, all based on Avis and Fukuda's reverse search method [AF96]. Specifically, Nakano [Nak01] described a CAT generation algorithm for generic rectangulations, which does not produce a Gray code, however. This algorithm has been adapted by Takagi and Nakano [TN04] to generate generic rectangulations with bounds on the number of rectangles that do not touch the outer face. Yoshii, Chigira, Yamanaka and Nakano [YCYN06] gave a Gray code for generic rectangulations based on a generating tree that is different from ours, resulting in a loopless algorithm. Their Gray code changes at most 3 edges of the rectangulation in each step, whereas our algorithm changes only 1 edge in each step for generic and for diagonal rectangulations. Consequently, none of the listings produced by these earlier algorithms corresponds to a walk along the skeleton of the underlying polytope.

There has been a lot of work on combinatorial properties of rectangulations. Yao, Chen, Cheng and Graham [YCCG03] showed that diagonal rectangulations are counted by the Baxter numbers and that guillotine diagonal rectangulations are counted by the Schröder numbers, using a bijection between diagonal rectangulations and twin binary trees. Ackerman, Barequet

and Pinter [ABP06a] presented another bijection between diagonal rectangulations and Baxter permutations, which also yields a bijection between guillotine diagonal rectangulations and separable permutations. Leifheit [Lei21] showed that this bijection can be restricted to the 1-sided variants of these two rectangulation classes by adding two permutation patterns; see Table 1. Shen and Chu [SC03] provided asymptotic estimates for diagonal rectangulations and their guillotine variant. Moreover, He [He14] presented an optimal encoding of diagonal rectangulations with n rectangles using only $3n - 3$ bits, which is optimal.

The term 'generic rectangulation' was coined by Reading [Rea12], who established a bijection between generic rectangulations and 2-clumped permutations, proving that these permutations are representatives of equivalence classes of a lattice congruence of the weak order on the symmetric group. Earlier, generic rectangulations had been studied under the name 'rectangular drawings' by Amano, Nakano and Yamanaka [ANY07] and by Inoue, Takahashi and Fujimaki [FIT09, ITF09], who established recursion formulas and asymptotic bounds for their number. More general classes of rectangular partitions were analyzed by Conant and Michaels [CM14].

Ackerman, Barequet and Pinter [ABP06b] considered the setting where we are given a set of n points in general position in a rectangle, and the goal is to partition the rectangle into smaller rectangles by n walls, such that each point from the set lies on a distinct wall. They showed that for every set of points that forms a separable permutation in the plane, the number of possible rectangulations is the $(n + 1)$st Baxter number, and for every point set the number of possible guillotine rectangulations is the nth Schröder number. They also presented a counting and generation procedure based on simple flips and T-flips using reverse search, which was later improved by Yamanaka, Rahman and Nakano [YRN18].

1.3. **Outline of this paper.** In Section 2 we provide basic definitions and concepts that will be used throughout the paper. In Section 3 we present a greedy algorithm for generating a set of rectangulations by jumps, and we provide a sufficient condition for the algorithm to succeed. In Section 4 we show that the algorithm applies to a large number of rectangulation classes that are characterized by pattern avoidance. In Section 5 we demonstrate how to make our generation algorithm memoryless and efficient. The data structures and basic functions used by our algorithms are provided in Sections 6 and 7. The proofs of Theorems 5 and 8 are presented in Section 8, by establishing a connection between rectangulations and permutations and by applying our permutation language framework. The results for one special class of rectangulations mentioned in Table 1 are deferred to Section 9. In Section 10 we report on our computer experiments about counting pattern-avoiding rectangulations. We conclude the paper with some interesting open questions in Section 11. Several visualizations of Gray codes produced by our algorithms are shown in the appendix.

2. Preliminaries

2.1. **Generic rectangulations.** A *generic rectangulation*, or rectangulation for short, is a partition of a rectangle into finitely many interior-disjoint axis-aligned rectangles, such that no four rectangles of the partition have a point in common; see Figure 2. In other words, every point where three rectangles meet, or where two rectangles meet the outer face forms a T-joint with the incident rectangle boundaries. Given rectangles r and s, we say that r is *left* of s, and s is *right* of r, if the right side of r intersects the left side of s (necessarily in a line segment, rather than a single point). Similarly, we say that r is *below* s, and s is *above* r, if the top side of r intersects the bottom side of s. We consider generic rectangulations up to equivalence that preserves the left/right and below/above relations between rectangles, and we use $\mathcal{R}_n$, $n \geq 1$, to

denote the set of all rectangulations with n rectangles. We write $\square$ for the unique rectangulation in $\mathcal{R}_1$, i.e., the rectangulation consisting of a single rectangle.

We refer to every rectangle corner in a rectangulation as a *vertex*, to every minimal line segment between two vertices as an *edge*, and to every maximal line segment between two vertices that are not corners of the rectangulation as a *wall*. The *type* of a vertex that is not a corner of the rectangulation describes the shape of the T-joint at this vertex, and it is one of $\top$, $\vdash$, $\bot$, or $\dashv$.

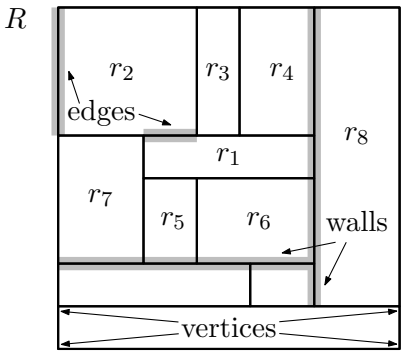


FIGURE 2. Generic rectangulation R with 11 rectangles. The rectangle r_1 is below r_2, r_3 and r_4, above r_5 and r_6, right of r_7 and left of r_8.

2.2. **Flip operations and classes of rectangulations.** Our Gray codes use three types of local change operations on rectangulations; see Figure 3.

A *wall slide* swaps the order of two neighboring vertices of types $\vdash$ and $\dashv$ along a vertical wall, or of types $\top$ and $\bot$ along a horizontal wall. A *simple flip* swaps the orientation of a wall that separates two rectangles. Given a vertex v that belongs to three rectangles, we consider the wall w that goes through v and the wall t that ends at v, and we let w' and w'' be the two halves of w meeting in v. If w' or w'' is an edge, respectively, then a *T-flip* swaps the orientation of this edge so that it merges with t.

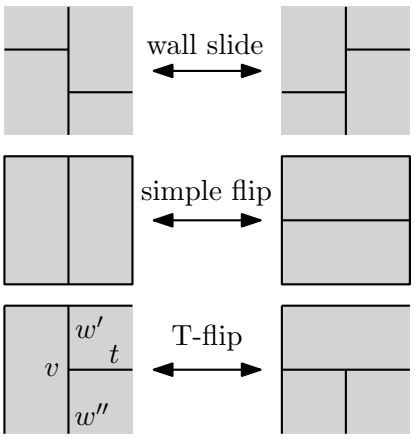
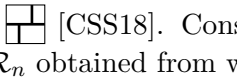


FIGURE 3. Local change operations on rectangulations.

We now define various interesting subclasses of generic rectangulations that have been studied in the literature and that appear in Table 1. Examples illustrating these classes are in Figure 4. A *diagonal* rectangulation is one in which every rectangle intersects the *main diagonal* that goes from the top-left to the bottom-right corner of the rectangulation. We write $\mathcal{D}_n \subseteq \mathcal{R}_n$ for the set of all diagonal rectangulations with n rectangles. Diagonal rectangulations are characterized by avoiding the wall patterns  and  [CSS18]. Consider the equivalence relation on $\mathcal{R}_n$ obtained from wall slides, sometimes referred to as *R-equivalence* [ABBM$^+$13]. The equivalence classes are referred to as *mosaic floorplans*, and every equivalence class contains exactly one diagonal rectangulation, obtained by repeatedly destroying occurrences of or by wall slides [CSS18]. Consequently, in a diagonal rectangulation, along every vertical wall, all $\vdash$-vertices are below all $\dashv$-vertices, and along every horizontal wall, all $\bot$-vertices are to the left of all $\top$-vertices.

In a *1-sided* rectangulation, every wall is the side of at least one rectangle, i.e., these rectangulations are characterized by avoiding the four patterns  ,  ,  and  . The notion of 1-sidedness was introduced by Eppstein, Mumford, Speckmann, and Verbeek [EMSV12] to characterize *area-universal* rectangulations, i.e., for any assignment of areas to the rectangles, the rectangulation can be drawn so that each rectangle has the prescribed area.

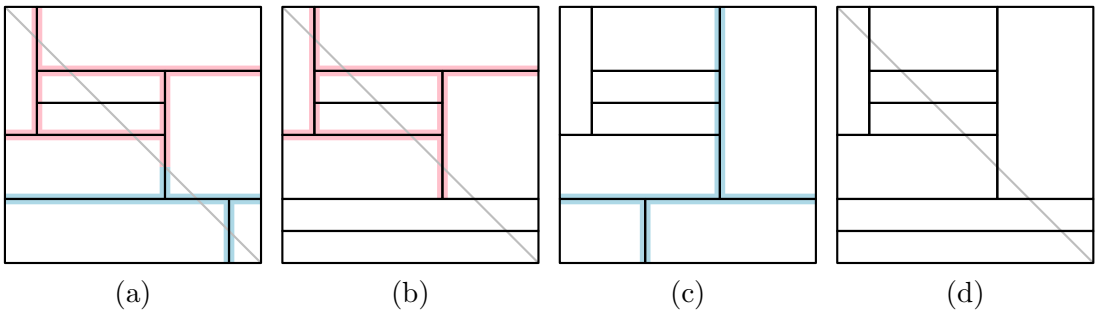


FIGURE 4. Examples of different classes of rectangulations: (a) diagonal, but neither 1-sided nor guillotine (b) 1-sided, but not guillotine (c) guillotine, but not diagonal (d) guillotine and 1-sided. Occurrences of the corresponding forbidden patterns are highlighted.

Asinowski et al. [ABBM+13] also considered the equivalence relation on $\mathcal{R}_n$ obtained from wall slides and simple flips, and they called it *S-equivalence*. By definition, S-equivalence is a coarser relation than R-equivalence, i.e., the equivalence classes are obtained by identifying mosaic floorplans that differ in simple flips. In Section 9 we introduce *block-aligned* rectangulations, which are a subset of diagonal rectangulations with the property that every equivalence class of S-equivalence contains exactly one block-aligned rectangulation.

A rectangulation is *guillotine*, if each of its rectangles can be cut out from the entire rectangulation by a sequence of straight vertical or horizontal cuts. Guillotine rectangulations are characterized by avoiding the windmill patterns  and  , which is a folklore result. Various special classes of guillotine diagonal rectangulations, characterized by the avoidance of certain wall configurations, were introduced by Asinowski and Mansour [AM10] (see Section 4 for precise definitions of these configurations). Mosaic floorplans that are guillotine are also known as *slicing* floorplans.

Felsner, Nathenson, and Tóth [FNT21] showed that 1-sided guillotine rectangulations are precisely the *aspect-ratio-universal* rectangulations, i.e., for any assignment of aspect ratios to the rectangles, the rectangulation can be drawn so that each rectangle has the prescribed aspect ratio.

2.3. **Deletion of rectangles.** We now describe two operations on a generic rectangulation R, namely deleting a rectangle and inserting a rectangle. The resulting rectangulations will be denoted by $p(R)$ and $c_i(R)$, notations that refer to the parent and children of R, in a tree structure that will be discussed shortly. The deletion and insertion operations were introduced in [HHC+00] and heavily used e.g. in [ABP06a] and [Nak01].

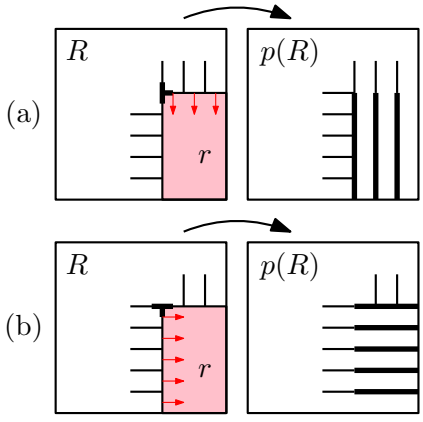


FIGURE 5. Deletion operation.

The idea of deletion is to contract the rectangle in the bottom-right corner of the rectangulation. Formally, given a rectangulation $R \in \mathcal{R}_n$, $n \geq 2$, we consider the rectangle r in the bottom-right corner, and we consider the top-left vertex of r. If this vertex has type $\vdash$, then we collapse r by sliding its top side, which forms a wall, downwards until it merges with the bottom side of r; see Figure 5 (a). Similarly, if this vertex has type $\top$, then we collapse r by sliding its left side, which forms a wall, to the

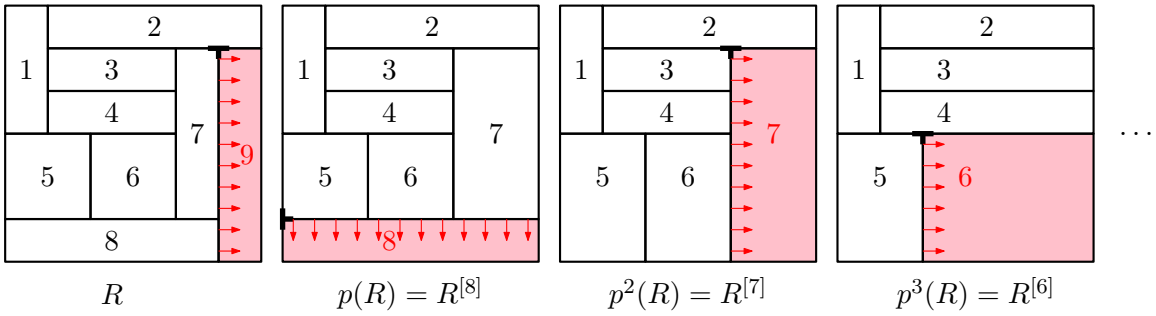


FIGURE 6. A rectangulation and the indexing of its rectangles given by repeated deletion.

right until it merges with the right side of r; see Figure 5 (b). We denote the resulting rectangulation with $n-1$ rectangles by $p(R) \in \mathcal{R}_{n-1}$, and we say that $p(R)$ is obtained from R by *deletion*.

Moreover, we denote the n rectangles of R by $r_n, r_{n-1}, \ldots, r_1$ in the order in which they are deleted when applying the deletion operation exhaustively; see Figure 6. Clearly, if r_i is deleted and its top-left vertex has type $\vdash$, then the rightmost rectangle above r_i is r_{i-1}. Similarly, if the top-left vertex has type $\top$, then the lowest rectangle to the left of r_i is r_{i-1}.

For any $R \in \mathcal{R}_n$ and $i = 1, \ldots, n$ we define $R^{[i]} := p^{n-i}(R)$, i.e., this is the sub-rectangulation of R formed by the first i rectangles; see Figure 6.

2.4. **Insertion of rectangles.** The idea of insertion is to add a new rectangle into the bottom-right corner of the rectangulation. Given a rectangulation $R \in \mathcal{R}_{n-1}$, we first define a set of points in R that can become the top-left corner of the newly added rectangle; see Figure 7.

For any rectangle r in $R \in \mathcal{R}_{n-1}$, $n \geq 2$, that touches the bottom boundary of R, we consider all edges forming the left side of r, and from every such edge we select one interior point, and we refer to it as a *vertical insertion point*.

Similarly, for any rectangle r in R that touches the right boundary of R, we consider the set of all edges forming the top side of r, and from every such edge we select one interior point, and we refer to it as a *horizontal insertion point*. Combinatorially it does not make a difference which interior point of each edge is selected.

We order the insertion points linearly, by sorting all vertical insertion points lexicographically by their (x, y)-coordinates, followed by all horizontal insertion points sorted lexicographically by their (y, x)-coordinates; see Figure 7. We write $I(R) = (q_1, q_2, \ldots, q_\nu)$ for the sequence of all insertion points ordered in this linear order. In particular, $\nu = \nu(R)$ denotes the number of insertion points.

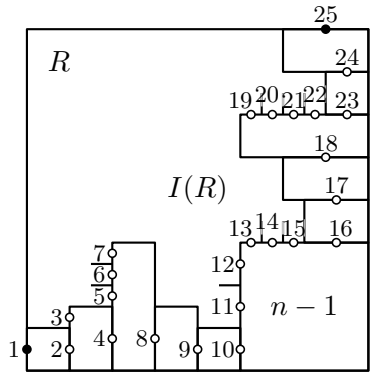


FIGURE 7. Linear ordering of insertion points. First and last insertion point are filled.

Lemma 1. *For any rectangulation $R \in \mathcal{R}_{n-1}$ we have $\nu(R) \leq n$.*

Proof. Each rectangle in R has at most one vertical insertion point on its right side, and at most one horizontal insertion point on its bottom side. Moreover, no rectangle has both, the bottom-right rectangle r_{n-1} has neither of the two, and exactly 2 insertion points lie on the boundary of R. Combining these observations shows that $\nu(R) \leq ((n-1)-1)+2 = n$. □

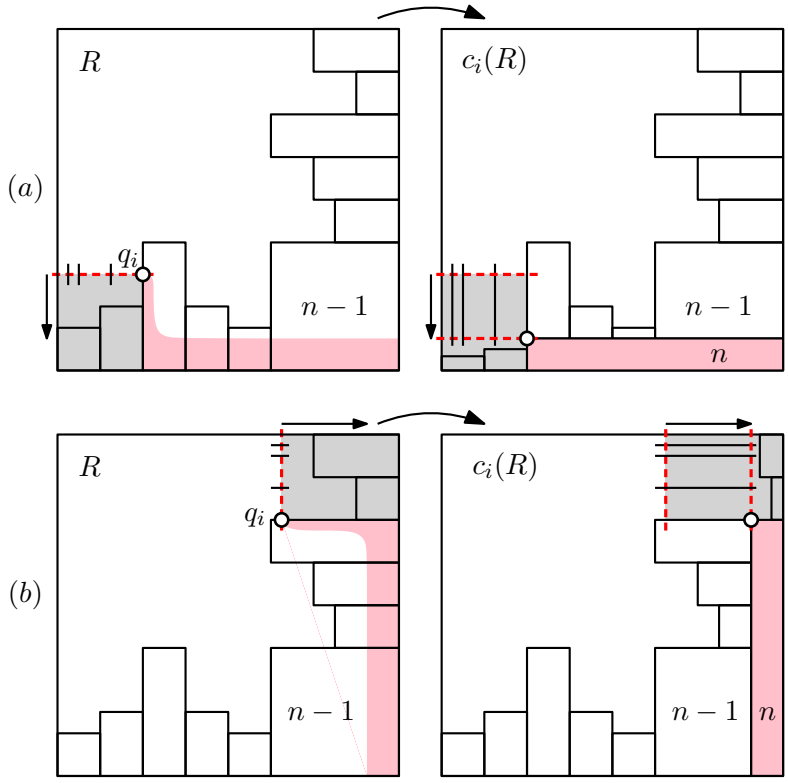


FIGURE 8. Insertion operation.

Clearly, the upper bound in Lemma 1 is attained if every rectangle touches the bottom or right boundary of R.

Given $R \in \mathcal{R}_n$ and the sequence of insertion points $I(R) = (q_1, \ldots, q_\nu)$, for each $i = 1, \ldots, \nu$ we define a rectangulation $c_i(R) \in \mathcal{R}_n$ as follows: If q_i is a vertical insertion point, then $c_i(R)$ is obtained from R by inserting a new rectangle r_n in the bottom-right corner such that r_n has above it exactly all rectangles which in R lie to the right of q_i and touch the bottom boundary of R, and such that r_n has to its left exactly all rectangles which in R touch the vertical wall through q_i below q_i; see Figure 8 (a). Similarly, if q_i is a horizontal insertion point, then $c_i(R)$ is obtained from R by inserting a new rectangle r_n in the bottom-right corner such that r_n has to its left exactly all rectangles which in R lie below q_i and touch the right boundary of R, and such that r_n has above it exactly all rectangles which in R touch the horizontal wall through q_i to the right of q_i; see Figure 8 (b). We say that $c_i(R)$ is obtained from R by *insertion*.

By these definitions, the operations of deletion and insertion are inverse to each other, which we record in the following lemma.

Lemma 2. *For any rectangulation $R \in \mathcal{R}_{n-1}$ and any two distinct insertion points q_i and q_j from $I(R)$, the rectangulations $c_i(R) \in \mathcal{R}_n$ and $c_j(R) \in \mathcal{R}_n$ are distinct, and we have $R = p(c_i(R)) = p(c_j(R))$. Moreover, for any $R' \in \mathcal{R}_n$ with $p(R') = R$ there is an insertion point q_i in $I(R)$ such that $c_i(R) = R'$.*

The first and last insertion point play a special role in our arguments, which is why they are highlighted in Figure 8. We say that R is *bottom-based* if R has a rectangle whose bottom side is the entire bottom boundary of R, and R is *right-based* if R has a rectangle whose right side is the entire right boundary of R. Note that the rectangulation $\square \in \mathcal{R}_1$ is both bottom-based

and right-based, and if $n \geq 2$, then $R \in \mathcal{R}_n$ is bottom-based if and only if $R = c_1(p(R))$ and right-based if and only if $R = c_{\nu(p(R))}(p(R))$.

3. The basic algorithm

In this section we present the basic algorithm that we use to generate a set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$.

3.1. **Jumps in rectangulations.** To state the algorithm, we first introduce a local change operation that generalizes the three kinds of flips introduced in Section 2.2 (recall Figure 3) and that will be applied when moving from one rectangulation in $\mathcal{C}_n$ to the next in the algorithm. A *jump* changes the insertion point for exactly one rectangle of the rectangulation. Formally, for a rectangulation $R \in \mathcal{R}_n$, we say that $R' \in \mathcal{R}_n$ differs from R by a *right jump of rectangle r_j by d steps*, denoted $R' = \overrightarrow{J}(R, j, d)$, where $2 \leq j \leq n$ and $d > 0$, if one of the following conditions holds; see Figure 10:

- $j = n$, and we have $p(R) = p(R') =: P \in \mathcal{R}_{n-1}$, $R = c_k(P)$ and $R' = c_{k+d}(P)$ for some $k > 0$;
- $j < n$, and R and R' are either both bottom-based or both right-based, and $p(R')$ differs from $p(R)$ in a right jump of rectangle r_j by d steps.

In words, the first condition asserts that the first $n-1$ rectangles in R and R' form the same rectangulation $P \in \mathcal{R}_{n-1}$, and R and R' are obtained by insertion from P using the kth and $(k+d)$th insertion point, respectively. The second condition asserts that R and R' agree in the rectangle r_n, which either forms the bottom boundary or the right boundary of those rectangulations, and $p(R')$ differs from $p(R)$ in a right jump with the same parameters.

A right jump as before is called *minimal* w.r.t. to a set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$, if in the first condition above there is no index ℓ with $k < \ell < k+d$ such that $c_\ell(P) \in \mathcal{C}_n$.

A *(minimal) left jump*, denoted $R' = \overleftarrow{J}(R, j, d)$, is defined analogously by replacing c_{k+d} by c_{k-d} and $k < \ell < k+d$ by $k > \ell > k-d$ in the definitions above. Clearly, if R' differs from R by a right jump of rectangle r_j by d steps, then R differs from R' by a left jump of rectangle r_j by d steps, and vice versa, i.e., we have $R' = \overrightarrow{J}(R, j, d)$ if and only if $R = \overleftarrow{J}(R', j, d)$. We sometimes simply say that R and R' differ in a jump, without specifying the direction left or right.

We state the following simple observations for further reference; see Figure 9.

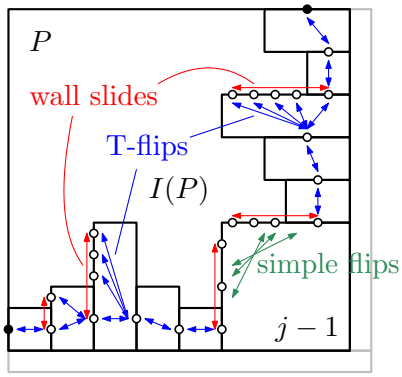


FIGURE 9. Jumps generalize wall slides, simple flips and T-flips.

Lemma 3. *Consider two rectangulations $R, R' \in \mathcal{R}_n$ that differ in a jump of rectangle r_j, define $P := R^{[j-1]} = R'^{[j-1]} \in \mathcal{R}_{j-1}$, and let q_k and q_ℓ be the insertion points in $I(P)$ such that $R^{[j]} = c_k(P)$ and $R'^{[j]} = c_\ell(P)$.*

(a) If q_k and q_ℓ are consecutive (w.r.t. $I(P)$) on a common wall of P, then R and R' differ in a wall slide.

(b) If q_k lies on the last vertical wall and q_ℓ on the first horizontal wall of P (w.r.t. $I(P)$), then R and R' differ in a simple flip.

(c) If q_k lies on a vertical wall and q_ℓ is the first insertion point on the next vertical wall of P (w.r.t. $I(P)$), or if q_k lies on a horizontal wall and q_ℓ is the last insertion point on the previous horizontal wall, then R and R' differ in a T-flip.

For any rectangulation $R \in \mathcal{R}_n$, we say that two insertion points from $I(R)$ belong to the same *vertical or horizontal group*, if they lie on the same vertical or horizontal wall in R, respectively. In the sequence $I(R)$, insertion points belonging to the same group appear consecutively.

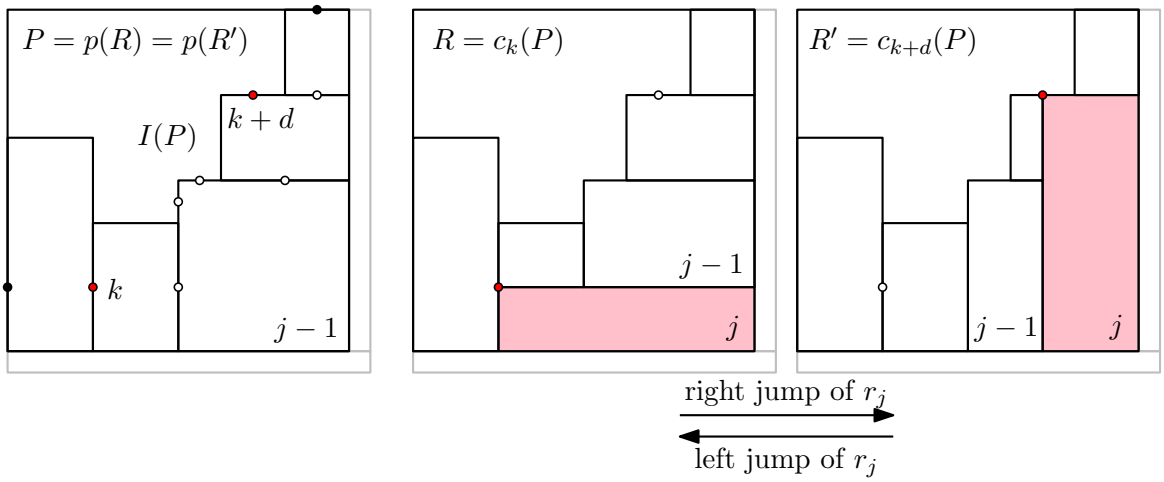


FIGURE 10. Illustration of jumps.

3.2. **Generating rectangulations by minimal jumps.** Consider the following algorithm that attempts to greedily generate a set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$ using minimal jumps.

Algorithm J$^\square$ *(Greedy minimal jumps).* This algorithm attempts to greedily generate a set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$ using minimal jumps starting from an initial rectangulation $R_0 \in \mathcal{C}_n$.

J1. [Initialize] Visit the initial rectangulation R_0.

J2. [Jump] Generate an unvisited rectangulation from $\mathcal{C}_n$ by performing a minimal jump of the rectangle with largest possible index in the most recently visited rectangulation. If no such jump exists, or the jump direction is ambiguous, then terminate. Otherwise visit this rectangulation and repeat J2.

To illustrate how Algorithm J$^\square$ works, we consider the set of five rectangulations $\mathcal{C}_4 = \{R_1, \ldots, R_5\} \subseteq \mathcal{R}_4$ shown in Figure 11. If initialized with $R_0 := R_1$, then the algorithm performs a left jump of rectangle 4 by one step (a right jump of rectangle 4 is impossible) to reach R_2, i.e., we have $R_2 = \overleftarrow{J}(R_1, 4, 1)$. In R_2, there are two options, either a right jump of rectangle 4 by one step, leading back to R_1, which has been visited before, or a left jump of rectangle 4 by two steps, leading to R_3, so we visit $R_3 = \overleftarrow{J}(R_2, 4, 2)$. In R_3, the jumps involving rectangle 4 lead to rectangulations that were visited before (R_1 and R_2). Moreover, a jump of rectangle 3 does not lead to a rectangulation in $\mathcal{C}_4$. However, a right jump of rectangle 2 by one step leads to R_4 (a left jump of rectangle 2 is impossible), so we visit $R_4 = \overrightarrow{J}(R_3, 2, 1)$. Finally, in R_4 a right jump of rectangle 4 by two steps leads to $R_5 = \overrightarrow{J}(R_4, 4, 2)$ (a left jump of rectangle 4 is impossible). In this example, Algorithm J$^\square$ successfully visits every rectangulation from $\mathcal{C}_4$ exactly once.

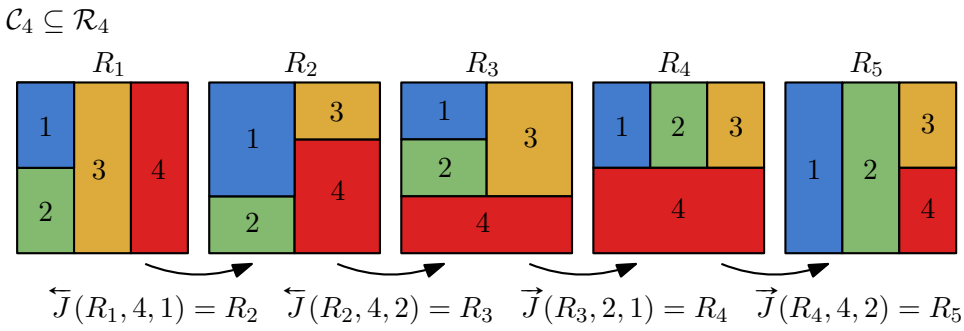


FIGURE 11. Example execution of Algorithm J$^{\square}$.

On the other hand, suppose we instead initialize the algorithm with $R_0 := R_3$. The algorithm will then visit $R_2 := \overrightarrow{J}(R_3, 4, 2)$ followed by $R_1 := \overrightarrow{J}(R_2, 4, 1)$, and then terminates without success, as from R_1 no jump leads to an unvisited rectangulation from $\mathcal{C}_4$. Lastly, suppose we initialize Algorithm J$^{\square}$ with $R_0 := R_2$. As before, in R_2, there are two possibilities, either a right jump or a left jump of rectangle 4, both leading to an unvisited rectangulation from $\mathcal{C}_4$. Both are minimal jumps in opposite directions, and as the jump direction is ambiguous, the algorithm terminates immediately without success.

Remark 4. *We do not recommend using Algorithm J$^{\square}$ in the stated form to generate a set of rectangulations efficiently!* This is because the algorithm requires to maintain the list of all previously visited rectangulations (possibly exponentially many), and to look up this list in each step to check whether a rectangulation obtained by a jump from the current one has been visited before. For us, Algorithm J$^{\square}$ is merely a tool to define a Gray code ordering of the rectangulations in the given set $\mathcal{C}_n$ in way that is easy to remember (cf. [Wil13]). In fact, in Section 5 we will present a modified algorithm that dispenses with the costly lookup operations, and that computes the very same sequence of rectangulations.

3.3. **A guarantee for success.** By definition, Algorithm J$^{\square}$ visits every rectangulation from a given set $\mathcal{C}_n \subseteq \mathcal{R}_n$ at most once, but it may terminate before having visited all. We now provide a sufficient condition guaranteeing that Algorithm J$^{\square}$ visits every rectangulation from $\mathcal{C}_n$ exactly once.

A set of generic rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$ is called *zigzag*, if either $n = 1$ and $\mathcal{C}_1 = \{\square\}$, or if $n \geq 2$ and $\mathcal{C}_{n-1} := \{p(R) \mid R \in \mathcal{C}_n\}$ is zigzag and for every $R \in \mathcal{C}_{n-1}$ we have $c_1(R) \in \mathcal{C}_n$ and $c_{\nu(R)}(R) \in \mathcal{C}_n$. In words, the set $\mathcal{C}_n$ must be closed under repeatedly deleting bottom-right rectangles and replacing them by rectangles inserted either below or to the right of the remaining ones; recall Figure 1. The name 'zigzag' does not refer to the shape of a rectangulation, but to the order in which they are visited by Algorithm J$^{\square}$, which will become clear momentarily. We also say that $\mathcal{C}_n$ is *symmetric*, if reflection at the main diagonal is an involution of $\mathcal{C}_n$, i.e., if $R \in \mathcal{C}_n$, then the rectangulation obtained from R by reflection at the main diagonal is also in $\mathcal{C}_n$. We write $\left\|\begin{smallmatrix} n \\ \vdots \end{smallmatrix}\right\|$ for the rectangulation that consists of n vertically stacked rectangles.

Theorem 5. *Given any zigzag set of rectangulations $\mathcal{C}_n$ and initial rectangulation $R_0 = \left\|\begin{smallmatrix} n \\ \vdots \end{smallmatrix}\right\|$, Algorithm J$^{\square}$ visits every rectangulation from $\mathcal{C}_n$ exactly once. Moreover, if $\mathcal{C}_n$ is symmetric, then the ordering of rectangulations generated by Algorithm J$^{\square}$ is cyclic, i.e., the first and last rectangulation differ in a minimal jump.*

The proof of Theorem 5 is provided in Section 8.

Note that the rectangulation $R_0 = \boxed{\begin{smallmatrix}n\\ \cdots\end{smallmatrix}}$ is contained in every zigzag set by definition, so this is a valid initialization for Algorithm J$^\square$. We write $J^\square(\mathcal{C}_n)$ for the sequence of rectangulations generated by Algorithm J$^\square$ for a zigzag set $\mathcal{C}_n$ when initialized with $R_0 = \boxed{\begin{smallmatrix}n\\ \cdots\end{smallmatrix}}$.

It is easy to see that the number of distinct zigzag sets of generic rectangulations is at least $2^{|\mathcal{R}_n|(1-o(1))} \geq 2^{\Omega(11.56^n)}$ (the latter estimate uses the best known lower bound on $|\mathcal{R}_n|$ from [ANY07]), i.e., at least double-exponential in n. In other words, Algorithm J$^\square$ exhaustively generates a given set of generic rectangulations in a vast number of cases. Moreover, many natural classes of rectangulations are in fact zigzag. In particular, *all* the different classes introduced in Section 2.2 and shown in Table 1 satisfy the aforementioned closure property. Moreover, all of these classes are symmetric, so for each of them we obtain cyclic jump orderings. Several such Gray codes are visualized in the appendix.

3.4. **Tree of rectangulations.** The notion of zigzag sets and the operation of Algorithm J$^\square$ can be interpreted combinatorially in the so-called *tree of rectangulations*, which is an infinite rooted tree, defined recursively as follows; see Figure 12: The root of the tree is a single rectangle $\square \in \mathcal{R}_1$. For any node $R \in \mathcal{R}_{n-1}$, $n \geq 2$, of the tree we consider all insertion points of the rectangulation R, and the set of children of R in the tree is $\{c_i(R) \in \mathcal{R}_n \mid i = 1, \ldots, \nu(R)\}$. Conversely, the parent of each $R \in \mathcal{R}_n$, $n \geq 2$, is $p(R) \in \mathcal{R}_{n-1}$. In words, insertion leads to the children of a node, and deletion leads to the parent of a node. By Lemma 2, each generic rectangulation appears exactly once in the tree, and the set of nodes at distance n from the root of the tree is precisely the set $\mathcal{R}_{n+1}$ of generic rectangulations with $n+1$ rectangles. We emphasize that this tree is *unordered*, i.e., there is no specified ordering among the children of a node.

By Lemma 1, a node $R \in \mathcal{R}_n$ in the tree has at most $n+1$ children, i.e., we have $|\mathcal{R}_n| \leq n!$. As we see from Figure 12, this inequality is tight up to $n = 4$, but starting from $n = 4$, there are nodes $R \in \mathcal{R}_n$ with strictly less than $n+1$ children, i.e., we have $|\mathcal{R}_5| < 5!$. In fact, it was shown in [ANY07] that $|\mathcal{R}_n| = \mathcal{O}(28.3^n)$.

A subset $\mathcal{C}_n \subseteq \mathcal{R}_n$ of nodes in depth $n-1$ of this tree is zigzag, if and only if it arises from the full tree of rectangulations by pruning some subtrees whose roots are neither bottom-based nor right-based rectangulations. In Figure 12, all bottom-based or right-based rectangulations are highlighted by gray boxes, and can therefore not be pruned, while all other nodes can possibly be pruned. If no nodes are pruned, then we have $\mathcal{C}_n = \mathcal{R}_n$, and if all possible nodes are pruned, then $\mathcal{C}_n$ is the set $\mathcal{B}_n$ of 2^{n-1} rectangulations obtained by repeatedly stacking a new rectangle either below or to the right of the previous ones, i.e., $\mathcal{B}_n = \{c_1(R), c_{\nu(R)}(R) \mid R \in \mathcal{B}_{n-1}\}$ for $n \geq 2$ and $\mathcal{B}_1 = \{\square\}$. Moreover, we have $\mathcal{B}_n \subseteq \mathcal{C}_n \subseteq \mathcal{R}_n$ for any zigzag set $\mathcal{C}_n$.

The operation of Algorithm J$^\square$ for a zigzag set $\mathcal{C}_n$ as input can be interpreted as follows: Given the pruned tree corresponding to $\mathcal{C}_n$, we consider the set of nodes on all previous levels of the tree, i.e., the sets $\mathcal{C}_{i-1} := \{p(R) \mid R \in \mathcal{C}_i\}$ for $i = n, n-1, \ldots, 2$, which are all zigzag sets by definition. Moreover, we consider the orderings $J^\square(\mathcal{C}_i)$, $i = 1, \ldots, n$, defined by Algorithm J$^\square$ for each of these sets. These sequences turn the unordered tree corresponding to $\mathcal{C}_n$ into an ordered tree, where the children $c_i(R)$ of each node R from left to right appear alternatingly in increasing order $i = 1, \ldots, \nu(R)$ or in decreasing order $i = \nu(R), \nu(R)-1, \ldots, 1$. Consequently, in the sequence $J^\square(\mathcal{C}_i)$, $i \geq 2$, which forms the left-to-right sequence of all nodes in depth $i-1$ of this ordered tree, the rectangle r_i alternatingly jumps left and right between the first and last insertion point, which motivates the name 'zigzag' set; see also the animations provided in [cos].

It is important to realize that these orderings are not consistent with respect to taking subsets, i.e., if we have two zigzag sets $\mathcal{C}'_n \subseteq \mathcal{C}_n$, then the entries of the sequence $J^\square(\mathcal{C}'_n)$ do not necessarily appear in the same relative order in the sequence $J^\square(\mathcal{C}_n)$.

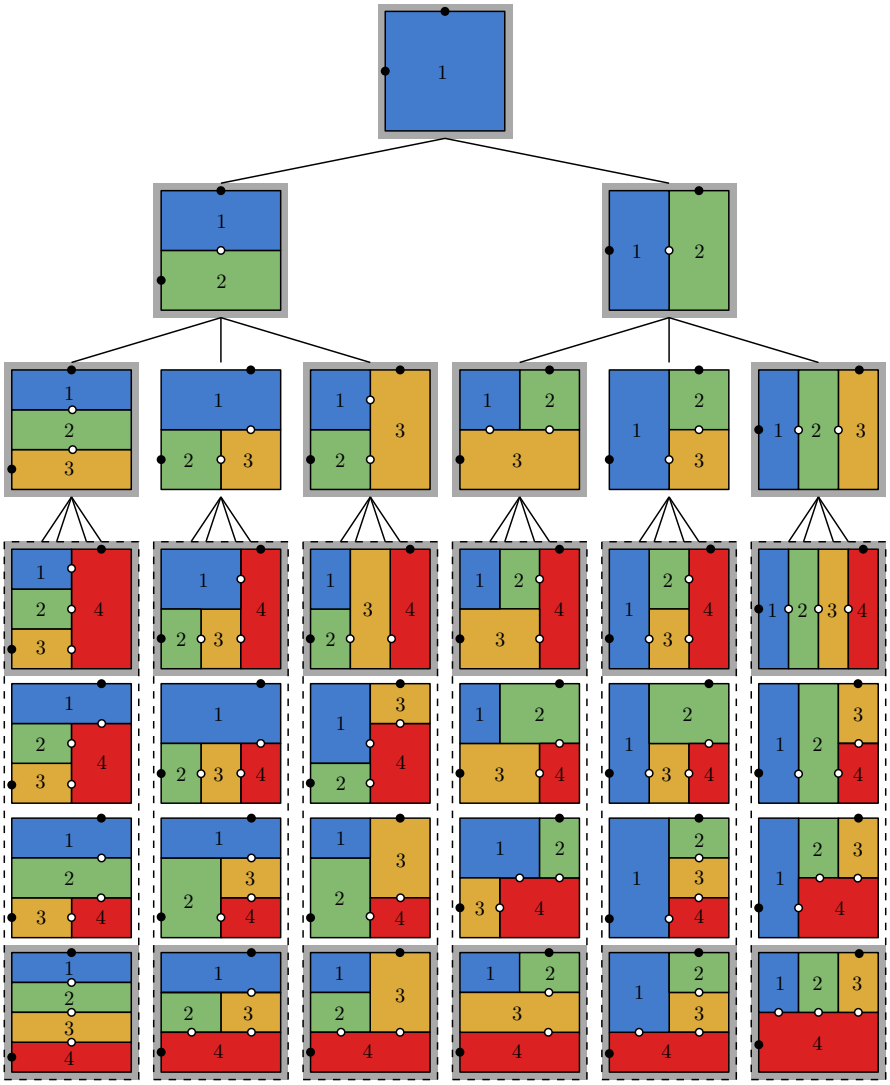


FIGURE 12. Tree of generic rectangulations up to depth 3 with insertion points highlighted, where first and last insertion point are filled. The rectangulations in the dashed boxes at the bottom level $\mathcal{R}_4$ are stacked on top of each other due to space constraints, but they are children of a common parent node. Bottom- or right-based rectangulations, corresponding to insertion at the first or last insertion point, are marked by gray boxes.

4. Pattern-avoiding rectangulations

In this section we show that Algorithm J$^\square$ applies to a large number of rectangulation classes that are defined by pattern avoidance, under some very mild conditions on the patterns; recall Table 1.

A *rectangulation pattern* is a configuration of walls with prescribed directions and incidences. For example, the windmill patterns  and  describe four walls such that when considering the walls in clockwise or counterclockwise order, respectively, the end vertex of one wall lies in the interior of the next wall. We can also think of a pattern as the rectangulation formed by the given walls and incidences. For example, we can think of the windmill patterns as rectangulations with 5 rectangles. We say that a rectangulation R *contains* the pattern P, if R contains a subset of walls with the directions and incidences specified by P. Otherwise we say that R *avoids* P. For

any set of rectangulation patterns $\mathcal{P}$ and for any set of rectangulations $\mathcal{C}$, we write $\mathcal{C}(\mathcal{P})$ for the rectangulations from $\mathcal{C}$ that avoid each pattern from $\mathcal{P}$. For example, diagonal rectangulations are given by $\mathcal{D}_n = \mathcal{R}_n(\{$  $\})$. Examples of rectangulations containing and avoiding various patterns are shown in Figure 4.

We say that a rectangulation pattern P is *tame*, if for any rectangulation R that avoids P, we also have that $c_1(R)$ and $c_{\nu(R)}(R)$ avoid P. In words, inserting a new rectangle below R or to the right of R must not create the forbidden pattern P. The next lemma follows directly from these definitions.

Lemma 6. *If a rectangulation pattern is neither bottom-based nor right-based, then it is tame. In particular, each of the patterns*  *is tame.*

The following powerful theorem allows to obtain many new zigzag sets of rectangulations from a given zigzag set $\mathcal{C}_n \subseteq \mathcal{R}_n$ by forbidding one or more tame patterns. All of these zigzag sets can then be generated by our Algorithm $\mathrm{J}^{\square}$.

Theorem 7. *Let $\mathcal{C}_n \subseteq \mathcal{R}_n$ be a zigzag set of rectangulations, and let $\mathcal{P}$ be a set of tame rectangulation patterns. Then $\mathcal{C}_n(\mathcal{P})$ is a zigzag set of rectangulations. Moreover, if $\mathcal{P}$ is symmetric, then $\mathcal{C}_n(\mathcal{P})$ is symmetric.*

Recall that $\mathcal{P}$ is symmetric if for each pattern $P \in \mathcal{P}$, we have that the pattern obtained from P by reflection at the main diagonal is also in $\mathcal{P}$. The significance of the second part of the theorem is that if $\mathcal{C}_n(\mathcal{P})$ is symmetric, then the ordering of rectangulations of $\mathcal{C}_n(\mathcal{P})$ generated by Algorithm $\mathrm{J}^{\square}$ is cyclic by Theorem 5.

Proof. As $\mathcal{C}_n$ is a zigzag set of rectangulations, we know that $\mathcal{C}_{i-1} := \{p(R) \mid R \in \mathcal{C}_i\}$ for $i = n, n-1, \ldots, 2$ are also zigzag sets. We argue by induction that $\mathcal{C}_i(\mathcal{P})$ is also a zigzag set for all $i = 1, \ldots, n$. For the induction basis $i = 1$ note that the rectangulation $\square$ that consists of a single rectangle has no walls, so it avoids any pattern, showing that $\mathcal{C}_1(\mathcal{P}) = \mathcal{C}_1 = \{\square\}$. For the induction step we assume that $\mathcal{C}_i(\mathcal{P})$, $i \in \{1, \ldots, n-1\}$, is a zigzag set, and we prove it for $\mathcal{C}_{i+1}(\mathcal{P})$. Note that $\{p(R) \mid R \in \mathcal{C}_{i+1}(\mathcal{P})\} = \mathcal{C}_i(\mathcal{P})$, and so we only need to check that $c_1(R)$ and $c_{\nu(R)}(R)$ are in $\mathcal{C}_{i+1}(\mathcal{P})$ for all $R \in \mathcal{C}_i(\mathcal{P})$, which is guaranteed by the assumption that each pattern $P \in \mathcal{P}$ is tame. This proves the first part of the theorem.

It remains to prove the second part. If $R \in \mathcal{C}_n(\mathcal{P})$, then R avoids every pattern from $\mathcal{P}$. Let R' be the rectangulation obtained from R by reflection at the main diagonal. R' must also avoid every pattern from $\mathcal{P}$, because if it contained a pattern P from $\mathcal{P}$, then R would contain the corresponding reflected pattern P', which is in $\mathcal{P}$ because of the assumption that $\mathcal{P}$ is symmetric. It follows that $R' \in \mathcal{C}_n(\mathcal{P})$, completing the proof. $\square$

5. Efficient computation

Recall from Remark 4 that Algorithm $\mathrm{J}^{\square}$ in its stated form is unsuitable for efficient implementation. We now discuss how to make the algorithm efficient, so as to achieve the time bounds claimed in Table 1 for several interesting classes of rectangulations.

5.1. **Memoryless algorithm.** Consider Algorithm $\mathrm{M}^{\square}$ below, which takes as input a zigzag set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$ and generates them exhaustively by minimal jumps in the same order as Algorithm $\mathrm{J}^{\square}$, i.e., in the order $J^{\square}(\mathcal{C}_n)$. After initialization in line M1, the algorithm loops over lines M2–M5, visiting the current rectangulation R at the beginning of each iteration (line M2), until it terminates (line M3).

The key idea of the algorithm is to track explicitly which rectangle jumps in each step, and the direction of the jump. With this information, the jump is determined by the condition that it must be minimal w.r.t. $\mathcal{C}_n$, i.e., starting from the current insertion point of the given rectangle, we choose the first insertion point (w.r.t. their linear ordering) for that rectangle in the given direction that creates the next rectangulation from $\mathcal{C}_n$.

Algorithm M$^\square$ *(Memoryless minimal jumps).* This algorithm generates all rectangulations of a zigzag set $\mathcal{C}_n \subseteq \mathcal{R}_n$ by minimal jumps in the same order as Algorithm J$^\square$. It maintains the current rectangulation in the variable R, and auxiliary arrays $o = (o_1, \ldots, o_n)$ and $s = (s_1, \ldots, s_n)$.

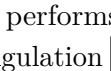
M1. [Initialize] Set $R \leftarrow$  , and $o_j \leftarrow \triangleleft$, $s_j \leftarrow j$ for $j = 1, \ldots, n$.

M2. [Visit] Visit the current rectangulation R.

M3. [Select rectangle] Set $j \leftarrow s_n$, and terminate if $j = 1$.

M4. [Jump rectangle] In the current rectangulation R, perform a jump of rectangle r_j that is minimal w.r.t. $\mathcal{C}_n$, where the jump direction is left if $o_j = \triangleleft$ and right if $o_j = \triangleright$.

M5. [Update o and s] Set $s_n \leftarrow n$. If $o_j = \triangleleft$ and $R^{[j]}$ is bottom-based set $o_j \leftarrow \triangleright$, or if $o_j = \triangleright$ and $R^{[j]}$ is right-based set $o_j \leftarrow \triangleleft$, and in both cases set $s_j \leftarrow s_{j-1}$ and $s_{j-1} \leftarrow j - 1$. Go back to M2.

Specifically, the jump directions are maintained by an array $o = (o_1, \ldots, o_n)$, where $o_j = \triangleleft$ means that rectangle r_j performs a left jump in the next step, and $o_j = \triangleright$ means that rectangle r_j performs a right jump in the next step (line M4). All sub-rectangulations of the initial rectangulation  are right-based, so the initial jump directions are $o_j = \triangleleft$ for $j = 1, \ldots, n$ (line M1). Whenever rectangle r_j jumps left and reaches the first insertion point, which means that $R^{[j]}$ is bottom-based, or if it jumps right and reaches the last insertion point, which means that $R^{[j]}$ is right-based, then the jump direction o_j is reversed (line M5).

The array $s = (s_1, \ldots, s_n)$ is used to determine which rectangle jumps in each step. Specifically, the last entry s_n determines the rectangle that jumps in the current iteration (line M3). This array simulates a stack in a loopless fashion, following an idea first used by Bitner, Ehrlich, and Reingold [BER76]. The stack is initialized by $(s_1, \ldots, s_n) = (1, 2, \ldots, n)$ (line M1), with s_n being the value on the top of the stack. The stack is popped (by the instruction $s_j \leftarrow s_{j-1}$ in line M5) when rectangle r_j reaches its first or last insertion point in this step, meaning that this rectangle is not eligible to jump in the next step, but becomes eligible again after the next step, which is achieved by pushing the value j on the stack again (by the instructions $s_n \leftarrow n$ and $s_{j-1} \leftarrow j - 1$ in line M5).

Table 2 shows the execution of Algorithm M$^\square$ with input $\mathcal{C}_4 = \mathcal{D}_4$ being the set of all diagonal rectangulations with 4 rectangles.

Theorem 8. *For any zigzag set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$, Algorithm M$^\square$ visits every rectangulation from $\mathcal{C}_n$ exactly once, in the order $J^\square(\mathcal{C}_n)$ defined by Algorithm J$^\square$.*

The proof of Theorem 8 is provided in Section 8.

To make meaningful statements about the running time of Algorithm M$^\square$, we need to specify the data structures used to represent the current rectangulation R, and the operations on this data structure to perform jumps in line M4 and to check the bottom-based and right-based property in line M5. Most importantly, we will develop oracles which efficiently compute the next minimal jump w.r.t. $\mathcal{C}_n$ for some interesting zigzag sets $\mathcal{C}_n$. One should think of $\mathcal{C}_n$ here as a class of rectangulations specified by some properties or forbidden patterns, such as 'diagonal guillotine rectangulations', and not as a large precomputed set of rectangulations. All of these

TABLE 2. Execution of Algorithm M$^{\square}$ for the set $\mathcal{C}_4 = \mathcal{D}_4$ of diagonal rectangulations with 4 rectangles. Empty entries in the o and s column are unchanged compared to the previous row.

	$J^{\square}(\mathcal{C}_4)$	jump	$o_1o_2o_3o_4$	$s_1s_2s_3s_4$		$J^{\square}(\mathcal{C}_4)$	jump	$o_1o_2o_3o_4$	$s_1s_2s_3s_4$
1		$\overleftarrow{J}(R,4,1)$	$\triangleleft\triangleleft\triangleleft\triangleleft$	1 2 3 4	12		$\overrightarrow{J}(R,4,1)$	$\triangleright$	1 1 4
2		$\overleftarrow{J}(R,4,1)$		4	13		$\overrightarrow{J}(R,4,1)$		4
3		$\overleftarrow{J}(R,4,1)$		4	14		$\overrightarrow{J}(R,4,1)$		4
4		$\overleftarrow{J}(R,3,1)$	$\triangleright$	3 3	15		$\overrightarrow{J}(R,3,1)$	$\triangleleft$	3 3
5		$\overrightarrow{J}(R,4,1)$		4	16		$\overleftarrow{J}(R,4,1)$		4
6		$\overrightarrow{J}(R,4,1)$		4	17		$\overleftarrow{J}(R,4,1)$		4
7		$\overrightarrow{J}(R,4,1)$		4	18		$\overleftarrow{J}(R,4,1)$		4
8		$\overleftarrow{J}(R,3,1)$	$\triangleleft$	3 3	19		$\overrightarrow{J}(R,3,1)$	$\triangleright$	3 3
9		$\overleftarrow{J}(R,4,1)$	$\triangleright$	2 2 4	20		$\overrightarrow{J}(R,4,1)$	$\triangleleft$	2 1 4
10		$\overleftarrow{J}(R,4,2)$		4	21		$\overrightarrow{J}(R,4,2)$		4
11		$\overleftarrow{J}(R,2,1)$	$\triangleright$	3 2	22			$\triangleleft$	3 1

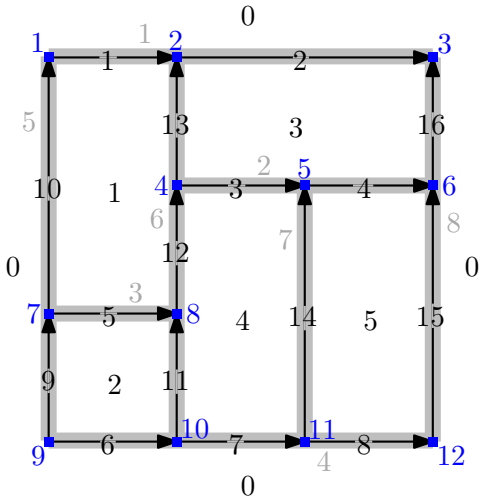
details are described in the following sections, and they are part of our C++ implementation provided in [cos].

6. Data structures and basic functions

In the following we describe the data structures we use to represent and manipulate generic rectangulations, and the efficient implementation of jump operations using those data structures.

6.1. **Data structures.** We represent a generic rectangulation with n rectangles as follows; see Figure 13: Rectangles are stored in the variables $r_1, \ldots, r_n$, indexed by the reverse deletion order described in Section 2.3 (recall Figure 6). Vertices and edges are stored in variables $v_1, \ldots, v_{2n+2}$ and $e_1, \ldots, e_{3n+1}$, respectively (indexed in no particular order).

Each vertex v points to the edges incident to it in the four directions by v.`north`, v.`east`, v.`south`, and v.`west`. Some of these can be 0, indicating that no edge is incident. This information determines the type v.`type`, which is one of $\top$, $\vdash$, $\bot$, $\dashv$, or 0 at the corner vertices of the rectangulation. We give all edges a default orientation from left to right, or from bottom to top. The `dir` entry of each edge e specifies its direction, which is either e.`dir` $= \triangleright$ for a



e_i	dir	tail	head	prev	next	left	right	wall
1	▷	1	2	0	2	0	1	1
2	▷	2	3	1	0	0	3	1
3	▷	4	5	0	4	3	4	2
4	▷	5	6	3	0	3	5	2
5	▷	7	8	0	0	1	2	3
6	▷	9	10	0	7	2	0	4
7	▷	10	11	6	8	4	0	4
8	▷	11	12	7	0	5	0	4
9	△	9	7	0	10	0	2	5
10	△	7	1	9	0	0	1	5
11	△	10	8	0	12	2	4	6
12	△	8	4	11	13	1	4	6
13	△	4	2	12	0	1	3	6
14	△	11	5	0	0	4	5	7
15	△	12	6	0	16	5	0	8
16	△	6	3	15	0	3	0	8

v_i	north	east	south	west	type
1	0	1	10	0	0
2	0	2	13	1	⊤
3	0	0	16	2	0
4	13	3	12	0	⊢
5	0	4	14	3	⊤
6	16	0	15	4	⊣
7	10	5	9	0	⊢
8	12	0	11	5	⊣
9	9	6	0	0	0
10	11	7	0	6	⊥
11	14	8	0	7	⊥
12	15	0	0	8	0

w_i	first	last
1	1	3
2	4	6
3	7	8
4	9	12
5	9	1
6	10	2
7	11	5
8	12	3

r_i	neast	seast	swest	nwest
1	2	8	7	1
2	8	10	9	7
3	3	6	4	2
4	5	11	10	4
5	6	12	11	5

FIGURE 13. Data structures used to represent a generic rectangulation with $n = 5$ rectangles. Edges are labeled black, and walls are labeled gray.

horizontal edge or $e.\texttt{dir} = \triangle$ for a vertical edge. Each edge e points to its two end vertices, specifically to its tail by $e.\texttt{tail}$ and to its head by $e.\texttt{head}$ (with respect to the default orientation). It also points to the previous and next edge, in the direction of its orientation, by $e.\texttt{prev}$ and $e.\texttt{next}$, respectively, which can be 0 if no such edge exists. The rectangle to the left and right side of an edge e, in the direction of its orientation, are stored in $e.\texttt{left}$ and $e.\texttt{right}$, which can be 0 at the boundary of the rectangulation. Each rectangle r points to its four corner vertices by $r.\texttt{neast}$, $r.\texttt{seast}$, $r.\texttt{swest}$, and $r.\texttt{nwest}$ in the corresponding directions.

For some rectangulation classes it is useful to store information about walls, i.e., maximal sequences of edges between two vertices that are not corners of the rectangulation. These are stored in $w_1, \ldots, w_{n+3}$, where for simplicity we also keep track of the four maximal line segments between corners of the rectangulation (which are not walls in our definition). We also think of walls having a default orientation from left to right, or from bottom to top, and each wall w

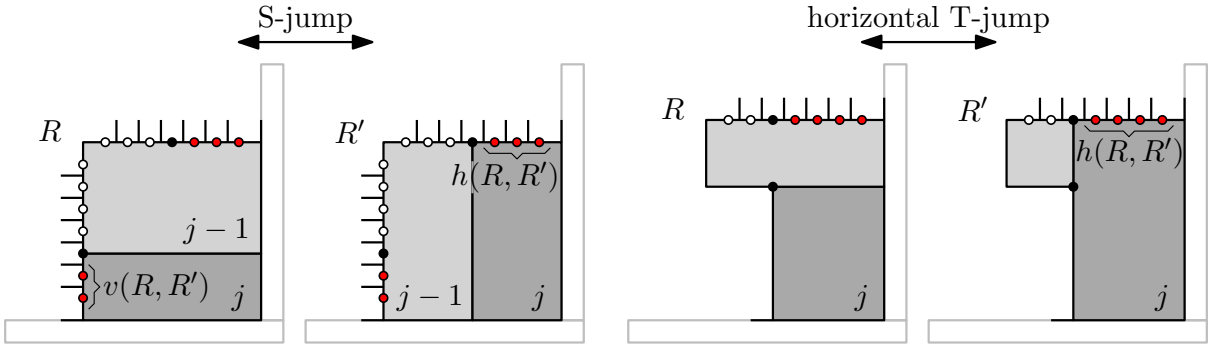


FIGURE 14. Illustration of Lemma 10.

points to its first and last vertex by $w.\texttt{first}$ and $w.\texttt{last}$, respectively, in the direction of its orientation. Moreover, each edge e has an entry $e.\texttt{wall}$ pointing to the wall that contains it.

Remark 9. The aforementioned data structures are natural in the sense that they also capture the dual graph of the rectangulation, i.e., the graph obtained by replacing every rectangle by a vertex, and by joining any two vertices that correspond to rectangles sharing a common edge. This allows constructing the so-called *transversal structure* [Fus09] (also known as *regular edge labeling* [KH97]), which is useful for computing a layout of the rectangulation; see Felsner's survey [Fel13]. Our data structures also allow to easily extract the twin binary tree representation of diagonal rectangulations described in [YCCG03].

We now use these data structures for implementing jumps efficiently. Recall the conditions stated in Lemma 3 when a jump is one of the three flip operations shown in Figure 3. We refer to a jump as in (a), (b) or (c) in the lemma as a *W-jump*, *S-jump*, or *T-jump*, respectively. By these definitions, a W-jump is a special wall slide, an S-jump is a special simple flip, and a T-jump is a special T-flip. We refer to W-, S- and T-jump as *local* jumps collectively. Moreover, a W-jump or T-jump between two horizontal insertion points, or between two vertical insertion points, is referred to as a *horizontal or vertical W- or T-jump*, respectively.

Consider two rectangulations R and R' that differ in a jump of rectangle r_j. If the jump is an S-jump or a horizontal T-jump, we let $h(R, R')$ denote the number of horizontal insertion points of $I(R^{[j-1]}) = I(R'^{[j-1]})$ that lie in the interior of the top side of rectangle r_j in R or R'; see Figure 14. Similarly, if the jump is an S-jump or a vertical T-jump, we let $v(R, R')$ denote the number of vertical insertion points of $I(R^{[j-1]}) = I(R'^{[j-1]})$ that lie in the interior of the left side of rectangle r_j in R or R'.

Lemma 10. *Local jumps can be implemented with the following time guarantees:*

(a) A W-jump takes time $\mathcal{O}(1)$.
(b) An S-jump between rectangulations R and R' takes time $\mathcal{O}(h(R, R') + v(R, R') + 1)$.
(c) A horizontal T-jump between rectangulations R and R' takes time $\mathcal{O}(h(R, R') + 1)$ and a vertical T-jump takes time $\mathcal{O}(v(R, R') + 1)$.

Clearly, every jump can be performed as a sequence of local jumps, and then the time bounds given by Lemma 10 can be added up.

Proof. The time bounds follow from the number of incidences that change during local jumps. The crucial point is that during a jump of rectangle r_j between R and R', all rectangles r_k for $k = j+1, \ldots, n$ are right-based or bottom-based in $R^{[k]}$ and $R'^{[k]}$, entailing that only constantly many incidences with such rectangles have to be modified. □

6.2. **Auxiliary functions.** In Section 6.3 we provide implementations of local jumps with the runtime guarantees stated in Lemma 10. Before doing so, we introduce some auxiliary functions to add and remove edges from a rectangulation. These auxiliary functions only update the incidences between edges, vertices and walls, but not the incidences between rectangles and any other objects and the type of vertices (this will be done separately later).

The following function $\texttt{remHead}(\beta)$ removes the edge e_β together with its head vertex.

Function $\texttt{remHead}(\beta)$ *(Remove e_β and its head).*

1. [Prepare] Set $\alpha \leftarrow e_\beta.\texttt{prev}$, $\gamma \leftarrow e_\beta.\texttt{next}$, $a \leftarrow e_\beta.\texttt{tail}$.

2. [Update edges/vertices] If $\alpha \neq 0$, set $e_\alpha.\texttt{next} \leftarrow \gamma$. If $\gamma \neq 0$, set $e_\gamma.\texttt{prev} \leftarrow \alpha$ and $e_\gamma.\texttt{tail} \leftarrow a$. If $e_\beta.\texttt{dir} = \triangleright$, set $v_a.\texttt{east} \leftarrow \gamma$. Otherwise we have $e_\beta.\texttt{dir} = \triangle$ and set $v_a.\texttt{north} \leftarrow \gamma$.

3. [Update wall] Set $x \leftarrow e_\beta.\texttt{wall}$. If $e_\beta.\texttt{head} = w_x.\texttt{last}$, set $w_x.\texttt{last} \leftarrow a$.

After defining some auxiliary variables in the first step, the function $\texttt{remHead}(\beta)$ updates the incidences between edges and vertices in the second step, and the incidences between walls and edges in the third step. We also define an analogous function $\texttt{remTail}(\beta)$ that removes e_β and its tail instead of its head. For details, see our C++ implementation [cos].

The following two functions $\texttt{insBefore}(\beta, a, \gamma)$ and $\texttt{insAfter}(\alpha, a, \beta)$ insert the edge e_β with head v_a or tail v_a, respectively, before or after the edge e_γ or e_α.

Function $\texttt{insBefore}(\beta, a, \gamma)$ *(Insertion of e_β with head v_a before e_γ).*

1. [Prepare] Set $\alpha \leftarrow e_\gamma.\texttt{prev}$ and $b \leftarrow e_\gamma.\texttt{tail}$.

2. [Update edges/vertices] Set $e_\beta.\texttt{tail} \leftarrow b$, $e_\beta.\texttt{head} \leftarrow a$, $e_\beta.\texttt{prev} \leftarrow \alpha$, $e_\beta.\texttt{next} \leftarrow \gamma$, $e_\gamma.\texttt{tail} \leftarrow a$, $e_\gamma.\texttt{prev} \leftarrow \beta$, and if $\alpha \neq 0$ set $e_\alpha.\texttt{next} \leftarrow \beta$. If $e_\gamma.\texttt{dir} = \triangleright$, set $e_\beta.\texttt{dir} \leftarrow \triangleright$, $v_a.\texttt{west} \leftarrow \beta$, $v_a.\texttt{east} \leftarrow \gamma$ and $v_b.\texttt{east} \leftarrow \beta$. Otherwise we have $e_\gamma.\texttt{dir} = \triangle$ and set $e_\beta.\texttt{dir} \leftarrow \triangle$, $v_a.\texttt{south} \leftarrow \beta$, $v_a.\texttt{north} \leftarrow \gamma$ and $v_b.\texttt{north} \leftarrow \beta$.

3. [Update wall] Set $e_\beta.\texttt{wall} \leftarrow e_\gamma.\texttt{wall}$.

Function $\texttt{insAfter}(\alpha, a, \beta)$ *(Insertion of e_β with tail v_a after e_α).*

1. [Prepare] Set $\gamma \leftarrow e_\alpha.\texttt{next}$ and $b \leftarrow e_\alpha.\texttt{head}$.

2. [Update edges/vertices] Set $e_\beta.\texttt{tail} \leftarrow a$, $e_\beta.\texttt{head} \leftarrow b$, $e_\beta.\texttt{prev} \leftarrow \alpha$, $e_\beta.\texttt{next} \leftarrow \gamma$, $e_\alpha.\texttt{head} \leftarrow a$, $e_\alpha.\texttt{next} \leftarrow \beta$, and if $\gamma \neq 0$ set $e_\gamma.\texttt{prev} \leftarrow \beta$. If $e_\alpha.\texttt{dir} = \triangleright$, set $e_\beta.\texttt{dir} \leftarrow \triangleright$, $v_a.\texttt{west} \leftarrow \alpha$, $v_a.\texttt{east} \leftarrow \beta$ and $v_b.\texttt{west} \leftarrow \beta$. Otherwise we have $e_\alpha.\texttt{dir} = \triangle$ and set $e_\beta.\texttt{dir} \leftarrow \triangle$, $v_a.\texttt{south} \leftarrow \alpha$, $v_a.\texttt{north} \leftarrow \beta$ and $v_b.\texttt{south} \leftarrow \beta$.

3. [Update wall] Set $e_\beta.\texttt{wall} \leftarrow e_\alpha.\texttt{wall}$.

6.3. **Local jumps.** Armed with these auxiliary functions, we now tackle the implementation of local jumps with the time guarantees stated in Lemma 10. Each of the functions $\texttt{Wjump}(R, j, d, \alpha)$, $\texttt{Sjump}(R, j, d, \alpha)$ and $\texttt{Tjump}(R, j, d, \alpha)$ below takes as input the current rectangulation R in which the jump is performed, the index j of the rectangle r_j to be jumped, the direction $d \in \{\triangleleft, \triangleright\}$ of the jump, and the index α of the edge e_α which contains the insertion point of $R^{[j-1]}$ that will become the top-left vertex of r_j after the jump. In the pseudocode of these algorithms, all references to rectangles r_i, edges e_i, vertices v_i or walls w_i are with respect to the current rectangulation R.

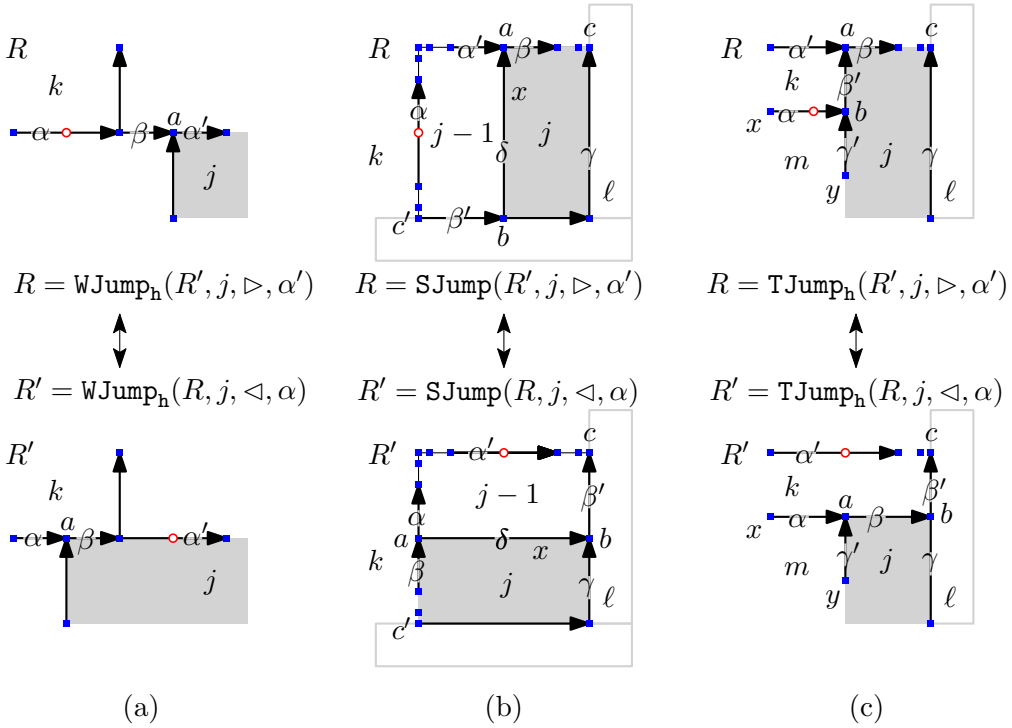


FIGURE 15. Implementation of local jumps: (a) horizontal W-jumps; (b) S-jumps; (c) horizontal T-jumps. Vertices are drawn as squares, insertion points as circles.

We first present the implementation of W-jumps. For simplicity, we only show the implementation of left horizontal W-jumps in the function $\texttt{Wjump}_\texttt{h}(R, j, \lhd, \alpha)$ below; see Figure 15 (a). The implementation of right horizontal W-jumps, and of left and right vertical W-jumps in a function $\texttt{Wjump}_\texttt{v}(R, j, d, \alpha)$ is very similar; we omit the details here.

Function $\texttt{Wjump}_\texttt{h}(R, j, \lhd, \alpha)$ *(left horizontal W-jump).*

1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$, $\beta \leftarrow v_a.\texttt{west}$ and $k \leftarrow e_\alpha.\texttt{left}$.
2. [Flip and update rectangles] Call $\texttt{remHead}(\beta)$ and $\texttt{insAfter}(\alpha, a, \beta)$. Then set $e_\beta.\texttt{left} \leftarrow k$ and $e_\beta.\texttt{right} \leftarrow j$.

The running time of $\texttt{Wjump}_\texttt{h}(R, j, \lhd, \alpha)$ is clearly $\mathcal{O}(1)$, as claimed in part (a) of Lemma 10.

We proceed with the implementation of S-jumps. For simplicity, we only provide the implementation of left S-jumps in the function $\texttt{Sjump}(R, j, \lhd, \alpha)$ below; see Figure 15 (b). The implementation of right S-jumps is very similar.

Function $\texttt{Sjump}(R, j, \lhd, \alpha)$ *(left S-jump).*

1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$, $b \leftarrow r_j.\texttt{swest}$, $c \leftarrow r_j.\texttt{neast}$, $\alpha' \leftarrow v_a.\texttt{west}$, $\beta \leftarrow v_a.\texttt{east}$, $\beta' \leftarrow v_b.\texttt{west}$, $\gamma \leftarrow v_c.\texttt{south}$, $\delta \leftarrow v_a.\texttt{south}$, $c' \leftarrow e_{\beta'}.\texttt{tail}$, $k \leftarrow e_\alpha.\texttt{left}$, $\ell \leftarrow e_\gamma.\texttt{right}$ and $x \leftarrow e_\delta.\texttt{wall}$.
2. [Flip] Call $\texttt{remTail}(\beta)$, $\texttt{remHead}(\beta')$, $\texttt{insBefore}(\beta, a, \alpha)$ and $\texttt{insAfter}(\gamma, b, \beta')$. Then set $e_\delta.\texttt{dir} \leftarrow \triangleright$, $e_\delta.\texttt{tail} \leftarrow a$, $e_\delta.\texttt{head} \leftarrow b$, $v_a.\texttt{east} \leftarrow \delta$, $v_a.\texttt{west} \leftarrow 0$, $v_a.\texttt{type} \leftarrow \vdash$, $v_b.\texttt{east} \leftarrow 0$, $v_b.\texttt{west} \leftarrow \delta$, $v_b.\texttt{type} \leftarrow \dashv$, $w_x.\texttt{first} \leftarrow a$ and $w_x.\texttt{last} \leftarrow b$.

3. [Update rectangles] Set r_j.`neast` $\leftarrow b$, r_j.`swest` $\leftarrow c'$, r_{j-1}.`neast` $\leftarrow c$, and r_{j-1}.`swest` $\leftarrow a$. Set $\nu \leftarrow v_c$.`west`, and while $\nu \neq \alpha'$ repeat e_ν.`right` $\leftarrow j-1$ and $\nu \leftarrow e_\nu$.`prev`. Set $\nu \leftarrow v_{c'}$.`north`, and while $\nu \neq \alpha$ repeat e_ν.`right` $\leftarrow j$ and $\nu \leftarrow e_\nu$.`next`. Also set e_β.`left` $\leftarrow k$, e_β.`right` $\leftarrow j$, $e_{\beta'}$.`left` $\leftarrow j-1$ and $e_{\beta'}$.`right` $\leftarrow \ell$.

Let R' be the rectangulation obtained from R by one call of $\texttt{Sjump}(R, j, \lhd, \alpha)$. The running time of this call is $\mathcal{O}(h(R,R') + v(R,R') + 1)$, as claimed in part (b) of Lemma 10. This time is incurred by the while-loops in step 3. Specifically, the first while-loop is iterated exactly $h(R,R')$ times, and the second while-loop is iterated exactly $v(R,R')+1$ times.

We complete this section by presenting the implementation of T-jumps; see Figure 15 (c). For simplicity, we only provide the implementation of left horizontal T-jumps in the function $\texttt{Tjump}_\texttt{h}(R, j, \lhd, \alpha)$ below. The implementation of right horizontal T-jumps, and of left and right vertical T-jumps in a function $\texttt{Tjump}_\texttt{v}(R, j, d, \alpha)$ is very similar.

Function $\texttt{Tjump}_\texttt{h}(R, j, \lhd, \alpha)$ *(left horizontal T-jump).*

1. [Prepare] Set $a \leftarrow r_j$.`nwest`, $b \leftarrow e_\alpha$.`head`, $c \leftarrow r_j$.`neast`, $\alpha' \leftarrow v_a$.`west`, $\beta \leftarrow v_a$.`east`, $\beta' \leftarrow v_a$.`south`, $\gamma \leftarrow v_c$.`south`, $\gamma' \leftarrow v_b$.`south`, $k \leftarrow e_{\beta'}$.`left`, $\ell \leftarrow e_\gamma$.`right`, $m \leftarrow e_\alpha$.`right`, $x \leftarrow e_\alpha$.`wall` and $y \leftarrow e_{\gamma'}$.`wall`.

2. [Flip] Call $\texttt{remTail}(\beta)$, $\texttt{remTail}(\beta')$, $\texttt{insAfter}(\alpha, a, \beta)$ and $\texttt{insAfter}(\gamma, b, \beta')$. Then set e_β.`head` $\leftarrow b$, $e_{\gamma'}$.`head` $\leftarrow a$, v_a.`south` $\leftarrow \gamma'$, v_b.`west` $\leftarrow \beta$, w_x.`last` $\leftarrow b$ and w_y.`last` $\leftarrow a$.

3. [Update rectangles] Set r_j.`neast` $\leftarrow b$, r_k.`neast` $\leftarrow c$ and r_m.`neast` $\leftarrow a$. Set $\nu \leftarrow v_c$.`west`, and while $\nu \neq \alpha'$ repeat e_ν.`right` $\leftarrow k$ and $\nu \leftarrow e_\nu$.`prev`. Also set e_β.`left` $\leftarrow k$ and $e_{\beta'}$.`right` $\leftarrow \ell$.

Let R' be the rectangulation obtained from R by one call of $\texttt{Tjump}_\texttt{h}(R, j, \lhd, \alpha)$. The running time of this call is $\mathcal{O}(h(R,R')+1)$, as claimed in part (c) of Lemma 10. This time is incurred by the while-loop in step 3, which is iterated exactly $h(R,R')+1$ times.

7. Minimal jump oracles

A *minimal jump oracle* is a function that is called in line M4 of Algorithm M$^\square$ to compute a jump in the current rectangulation R that is minimal respect to the given zigzag set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$. In this section we specify such oracles for the zigzag sets $\mathcal{C}_n$ mentioned in Table 1, which allows us to establish the runtime bounds for Algorithm M$^\square$ stated in the last column of the table (except for block-aligned rectangulations, which are handled in Section 9). A minimal jump oracle has the form $\texttt{next}_{\mathcal{C}_n}(R, j, d)$, and this function call performs in the current rectangulation R a jump of rectangle r_j in direction d that is minimal w.r.t. $\mathcal{C}_n$, and the function will modify R accordingly. Depending on $\mathcal{C}_n$, our minimal jump oracles perform a suitable W-, S-, or T-jump, or a combination thereof, as implemented in the previous section.

7.1. **Generic rectangulations.** We first consider the case $\mathcal{C}_n = \mathcal{R}_n$ of generic rectangulations. Given the current rectangulation R, upon a jump of rectangle r_j in direction d, *every* insertion from $I(R^{[j-1]})$ is used, so we simply need to detect the next one.

By Lemma 3, a W-jump occurs between any two consecutive (w.r.t. $I(R^{[j-1]})$) insertion points belonging to the same horizontal or vertical group, an S-jump occurs between the last vertical insertion point and the first horizontal insertion point, and a T-jump occurs between the last insertion point of a group and the first insertion point of the next group, if both groups are vertical or horizontal. Specifically, suppose there are λ vertical groups and μ horizontal groups with cardinalities g_k, $k = 1, \ldots, \lambda$, and h_k, $k = 1, \ldots, \mu$, respectively in $I(R^{[j-1]})$ (note that

$g_1 = h_\mu = 1$). Then the jump sequence consisting of letters $\{W, S, T\}$ that specifies the types of jumps performed with rectangle r_j from the first to the last insertion point is

$$(TW^{g_2-1})(TW^{g_3-1})\cdots(TW^{g_\lambda-1})\,S\,(W^{h_1-1}T)(W^{h_2-1}T)\cdots(W^{h_{\mu-1}-1}T); \tag{1}$$

see Figure 16 (a). Of course, during Algorithm M$^\square$, these jump operations are not consecutive, but they are interleaved with the jump sequences of other rectangles r_k, $k > j$.

The details are spelled out in the function $\texttt{next}_{\mathcal{R}_n}(R, j, d)$.

$\texttt{next}_{\mathcal{R}_n}(R, j, d)$ *(Minimal jump oracle for generic rectangulations).*

N1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$. If $d = \lhd$ and $v_a.\texttt{type} = \top$, set $\alpha \leftarrow v_a.\texttt{west}$, $\beta \leftarrow v_a.\texttt{south}$, $b \leftarrow e_\beta.\texttt{tail}$ and $c \leftarrow e_\alpha.\texttt{tail}$ and goto N2. If $d = \rhd$ and $v_a.\texttt{type} = \top$, set $\alpha \leftarrow v_a.\texttt{east}$, $b \leftarrow e_\alpha.\texttt{head}$ and goto N3. If $d = \rhd$ and $v_a.\texttt{type} = \vdash$, set $\alpha \leftarrow v_a.\texttt{north}$, $\beta \leftarrow v_a.\texttt{east}$, $b \leftarrow e_\beta.\texttt{head}$ and $c \leftarrow e_\alpha.\texttt{head}$ and goto N4. If $d = \lhd$ and $v_a.\texttt{type} = \vdash$, set $\alpha \leftarrow v_a.\texttt{south}$, $b \leftarrow e_\alpha.\texttt{tail}$ and goto N5.

N2. [Horizontal left jump] If $v_c.\texttt{type} = \bot$, set $\gamma \leftarrow v_c.\texttt{west}$ and call $\texttt{Wjump}_\texttt{h}(R, j, \lhd, \gamma)$. Else if $v_b.\texttt{type} = \dashv$, set $\gamma \leftarrow v_b.\texttt{west}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \lhd, \gamma)$. Otherwise we have $v_b.\texttt{type} = \bot$, set $\gamma \leftarrow v_c.\texttt{south}$ and call $\texttt{Sjump}(R, j, \lhd, \gamma)$. Return.

N3. [Horizontal right jump] If $v_b.\texttt{type} = \bot$, set $\gamma \leftarrow v_b.\texttt{east}$ and call $\texttt{Wjump}_\texttt{h}(R, j, \rhd, \gamma)$. Otherwise we have $v_b.\texttt{type} = \dashv$, set $k \leftarrow e_\alpha.\texttt{left}$, $c \leftarrow r_k.\texttt{nwest}$ and $\gamma \leftarrow v_c.\texttt{east}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \rhd, \gamma)$. Return.

N4. [Vertical right jump] If $v_c.\texttt{type} = \dashv$, set $\gamma \leftarrow v_c.\texttt{north}$ and call $\texttt{Wjump}_\texttt{v}(R, j, \rhd, \gamma)$. Else if $v_b.\texttt{type} = \bot$, set $\gamma \leftarrow v_b.\texttt{north}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \rhd, \gamma)$. Otherwise we have $v_b.\texttt{type} = \dashv$, set $\gamma \leftarrow v_c.\texttt{east}$ and call $\texttt{Sjump}(R, j, \rhd, \gamma)$. Return.

N5. [Vertical left jump] If $v_b.\texttt{type} = \dashv$, set $\gamma \leftarrow v_b.\texttt{south}$ and call $\texttt{Wjump}_\texttt{v}(R, j, \lhd, \gamma)$. Otherwise we have $v_b.\texttt{type} = \bot$, set $k \leftarrow e_\alpha.\texttt{left}$, $c \leftarrow r_k.\texttt{nwest}$ and $\gamma \leftarrow v_c.\texttt{south}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \lhd, \gamma)$. Return.

The four distinct cases treated in lines N2–N4 come from the directions $d \in \{\lhd, \rhd\}$ and whether the jump is horizontal or vertical. The latter condition is determined in line N1 by querying the type of the top-left vertex of r_j, which is either $\top$ or $\vdash$. Note that the code in lines N2 and N4 is symmetric by reflecting all directions at the main diagonal. The same holds for the code in lines N3 and N5.

Lemma 11. *Consider a rectangulation $P \in \mathcal{R}_{n-1}$ with $\nu = \nu(P)$ insertion points. Then calling $\texttt{next}_{\mathcal{R}_n}(R, n, \rhd)$ exactly $\nu - 1$ times with initial rectangulation $R = c_1(P)$, yields $c_i(P)$ for $i = 1, \ldots, \nu$, and the total time for these calls is $\mathcal{O}(\nu)$. An analogous statement holds for $\texttt{next}_{\mathcal{R}_n}(R, n, \lhd)$.*

Proof. If the sequence of insertion points $I(P)$ has λ vertical groups and μ horizontal groups of cardinalities g_k, $k = 1, \ldots, \lambda$, and h_k, $k = 1, \ldots, \mu$, respectively, then the sequence of jumps performed by the calls to $\texttt{next}_{\mathcal{R}_n}$ has the form (1). We clearly have $\nu = \sum_{k=1}^{\lambda} g_k + \sum_{k=1}^{\mu} h_k$. We use Lemma 10 to bound the overall time to perform those operations; see Figure 16 (a). The number of W-jumps in (1) is $w := \sum_{k=1}^{\lambda}(g_k - 1) + \sum_{k=1}^{\mu}(h_k - 1) \le \nu$. The sum of the terms $v(R, R') + 1$ and $h(R, R') + 1$ over any two consecutive rectangulations R, R' in this sequence that differ in a T-jump is $t := \sum_{k=1}^{\lambda-1} g_k$ and $t' := \sum_{k=2}^{\mu} h_k$, respectively. The sum $v(R, R') + h(R, R') + 1$ for the two consecutive rectangulations R, R' in this sequence that differ in an S-jump is $s := g_\lambda + h_1 - 1$. Clearly, we have $s + t + t' \le \nu$. Consequently, the overall time for those operations is $\mathcal{O}(w + s + t + t') = \mathcal{O}(\nu)$, as claimed. $\square$

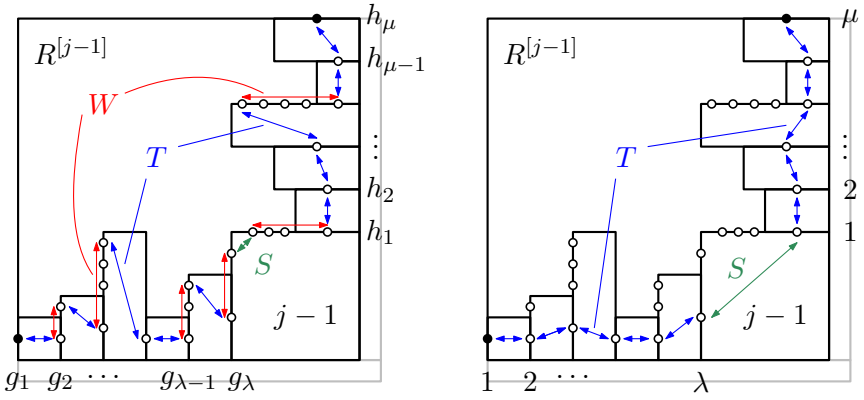


FIGURE 16. Illustration of the proofs of (a) Theorem 12 for generic rectangulations and (b) Theorem 15 for diagonal rectangulations.

Theorem 12. *Algorithm $M^\square$ with the minimal jump oracle $\texttt{next}_{\mathcal{R}_n}$ takes time $\mathcal{O}(1)$ on average to visit each generic rectangulation.*

Proof. For some fixed $j \in \{2, \dots, n\}$, we consider all jumps of rectangle r_j. Whenever rectangle r_j jumps in a rectangulation $R \in \mathcal{R}_n$, then $R^{[k]}$ is either bottom-based or right-based for all $k = j+1, \dots, n$. Moreover, none of the rectangles $k = j-1, j-2, \dots, 2$ jumps unless $R^{[j]}$ is bottom-based or right-based. Consequently, we can partition the jumps of r_j in the entire jump sequence into subsequences, such that in each subsequence, for each rectangulation $R \in \mathcal{R}_n$ of the subsequence, $R^{[j-1]} \in \mathcal{R}_{j-1}$ is the same subrectangulation and in R the rectangle r_j jumps to the next insertion point of $I(R^{[j-1]})$. By Lemma 11, the total time for visiting the $\nu = \nu(R^{[j-1]})$ many rectangulations of this subsequence is $\mathcal{O}(\nu)$, which is $\mathcal{O}(1)$ on average. □

Remark 13. By slightly modifying our data structures, we could even obtain a loopless algorithm for generic rectangulations. The idea is to introduce an additional data structure called *sides*. Each rectangle is subdivided into four sides, and in the incidence relations, sides sit between edges and rectangles, i.e., edges do not point to the two touching rectangles directly, but to the relevant sides of those rectangles, and each side points to the rectangle it belongs to. During S-jumps and T-jumps, a rectangle can be broken up into its four sides and the sides of two rectangles can be interchanged in constant time, avoiding the while-loops in the functions `Sjump` and `Tjump` that need to update possibly linearly many incidences between edges and rectangles. To keep the presentation simple, we do not show these modifications. Also, the resulting improvement is not substantial, and sides are a somewhat artificial concept.

7.2. **Diagonal rectangulations.** Recall that in a diagonal rectangulation $R \in \mathcal{D}_n$, every rectangle intersects the main diagonal, or equivalently, R avoids the patterns  and  . Consequently, during a jump of rectangle r_j in the current rectangulation R, we need to consider precisely the insertion points from $I(R^{[j-1]})$ that are the first insertion point of a vertical group, or the last insertion point of a horizontal group, as any other insertion point from each group would create one of the forbidden pattern; see Figure 16 (b). Consequently, if the sequence $I(R^{[j-1]})$ has λ vertical groups and μ horizontal groups, then the jump sequence that specifies the types of jumps with rectangle r_j from the first to the last insertion point is

$$T^{\lambda-1} S T^{\mu-1}.$$

In particular, we do not perform any wall slides.

An implementation of this is provided in the function $\texttt{next}_{\mathcal{D}_n}(R, j, d)$.

$\texttt{next}_{\mathcal{D}_n}(R, j, d)$ *(Minimal jump oracle for diagonal rectangulations).*

N1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$. If $d = \triangleleft$ and $v_a.\texttt{type} = \top$, set $\alpha \leftarrow v_a.\texttt{south}$ and $b \leftarrow e_\alpha.\texttt{tail}$ and goto N2. If $d = \triangleright$ and $v_a.\texttt{type} = \top$, set $\alpha \leftarrow v_a.\texttt{east}$ and goto N3. If $d = \triangleright$ and $v_a.\texttt{type} = \vdash$, set $\alpha \leftarrow v_a.\texttt{east}$ and $b \leftarrow e_\alpha.\texttt{head}$ and goto N4. If $d = \triangleleft$ and $v_a.\texttt{type} = \vdash$, set $\alpha \leftarrow v_a.\texttt{south}$ and goto N5.

N2. [Horizontal left jump] If $v_b.\texttt{type} = \dashv$, set $\gamma \leftarrow v_b.\texttt{west}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \triangleleft, \gamma)$. Otherwise we have $v_b.\texttt{type} = \perp$, set $c \leftarrow r_{j-1}.\texttt{swest}$ and $\gamma \leftarrow v_c.\texttt{north}$ and call $\texttt{Sjump}(R, j, \triangleleft, \gamma)$. Return.

N3. [Horizontal right jump] Set $k \leftarrow e_\alpha.\texttt{left}$, $b \leftarrow r_k.\texttt{neast}$ and $\gamma \leftarrow v_b.\texttt{west}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \triangleright, \gamma)$. Return.

N4. [Vertical right jump] If $v_b.\texttt{type} = \perp$, set $\gamma \leftarrow v_b.\texttt{north}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \triangleright, \gamma)$. Otherwise we have $v_b.\texttt{type} = \dashv$, set $c \leftarrow r_{j-1}.\texttt{neast}$ and $\gamma \leftarrow v_c.\texttt{west}$ and call $\texttt{Sjump}(R, j, \triangleright, \gamma)$. Return.

N5. [Vertical left jump] Set $k \leftarrow e_\alpha.\texttt{left}$, $b \leftarrow r_k.\texttt{swest}$ and $\gamma \leftarrow v_b.\texttt{north}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \triangleleft, \gamma)$. Return.

Similarly to before, the code in lines N2 and N4, and in lines N3 and N5 is symmetric by reflecting all directions at the main diagonal. For diagonal rectangulations the runtime analysis is straightforward and gives a loopless algorithm.

Lemma 14. *Each call* $\texttt{next}_{\mathcal{D}_n}(R, j, d)$ *takes time* $\mathcal{O}(1)$.

Proof. Let R' be the rectangulation after the call $\texttt{next}_{\mathcal{D}_n}(R, j, d)$, which differs from R in an S-jump or T-jump. As we only consider the first insertion point of each vertical group and the last insertion point of each horizontal group of $I(R^{[j-1]})$, we have $v(R, R') = 0$ and $h(R, R') = 0$. The claim now follows from Lemma 10 (b)+(c). □

Lemma 14 immediately yields the following result.

Theorem 15. *Algorithm* $M^\square$ *with the minimal jump oracle* $\texttt{next}_{\mathcal{D}_n}$ *takes time* $\mathcal{O}(1)$ *to visit each diagonal rectangulation.*

Remark 16. Jumps as performed by the oracles $\texttt{next}_{\mathcal{R}_n}$ and $\texttt{next}_{\mathcal{D}_n}$ and shown in Figure 16 correspond to cover relations in the lattice of generic rectangulations and the lattice of diagonal rectangulations described by Meehan [Mee19] and Law and Reading [LR12], respectively, which both arise as lattice quotients of the weak order on the symmetric group. Consequently, our cyclic Gray codes correspond to Hamilton cycles in the cover graphs of those lattices. In [HM21] we showed that our permutation language framework can be used to generate a Hamilton path on every lattice quotient of the weak order, which also yields a Hamilton path on the skeleton of the corresponding polytope [PS19] (see also [PPR21]).

7.3. **Pattern-avoiding rectangulations.** For any zigzag set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$, and any set of tame patterns $\mathcal{P}$, Theorem 7 guarantees that the set $\mathcal{C}_n(\mathcal{P})$ of rectangulations that avoid all patterns from $\mathcal{P}$ is also a zigzag set. We now describe how we can obtain a minimal jump oracle for $\mathcal{C}_n(\mathcal{P})$ from a minimal jump oracle $\texttt{next}_{\mathcal{C}_n}(R, j, d)$ for $\mathcal{C}_n$. The idea is simply to perform a minimal jump of r_j w.r.t. $\mathcal{C}_n$, and to test after each jump whether the resulting rectangulation contains any pattern from $\mathcal{P}$, repeating this process until we arrive at a rectangulation that avoids all patterns from $\mathcal{P}$. This is guaranteed to terminate after at most $j \leq n$ iterations, as

the first and last insertion point of $I(R^{[j-1]})$ will produce rectangulations that avoid all patterns from $\mathcal{P}$, due to the zigzag property.

$\texttt{next}_{\mathcal{C}_n(\mathcal{P})}(R, j, d)$ *(Minimal jump oracle for pattern-avoiding permutations).*
N1. [Fast forward] While R contains a pattern from $\mathcal{P}$ repeat $\texttt{next}_{\mathcal{C}_n}(R, j, d)$.

We immediately obtain the following generic runtime bounds.

Theorem 17. *Let $\mathcal{P}$ be a finite set of tame rectangulation patterns. If the zigzag set $\mathcal{C}_n$ has a minimal jump oracle $\texttt{next}_{\mathcal{C}_n}$ that runs in time f_n, and containment of any pattern from $\mathcal{P}$ in R can be tested in time t_n, then $\texttt{next}_{\mathcal{C}_n(\mathcal{P})}$ is a minimal jump oracle for $\mathcal{C}_n(\mathcal{P})$ that runs in time $\mathcal{O}(n \cdot (f_n + t_n))$.*

In some cases the runtime bound for at most n consecutive calls of $\texttt{next}_{\mathcal{C}_n}(R, j, d)$ or several consecutive pattern containment tests can be improved upon the trivial bounds $\mathcal{O}(n \cdot f_n)$ and $\mathcal{O}(n \cdot t_n)$, respectively (see the proof of Theorem 19 below). Moreover, for some patterns further optimizations of the function $\texttt{next}_{\mathcal{C}_n(\mathcal{P})}(R, j, d)$ are possible. For example, the property of R to be guillotine is invariant under W-jumps and S-jumps, so if R is found to contain one of the windmill patterns, then we only need to check containment after the next T-jump performed by the call $\texttt{next}_{\mathcal{C}_n}(R, j, d)$. Most importantly, when testing for containment of a pattern, we only need to check incidences of walls involving the rectangle r_j. In the following, we provide functions $\texttt{contains}(R, j, P)$ that test whether R contains one of the tame patterns P listed in Lemma 6 after a sequence of jumps of rectangle r_j from a rectangulation that avoids P. We emphasize here that these functions only work under these assumptions, and are not suitable for general pattern containment testing of arbitrary rectangulations, but only for use within our algorithm $\mathrm{M}^{\square}$.

We first present an implementation of such a containment testing function $\texttt{contains}(R, j, P)$ for the clockwise windmill $P =$  ; see Figure 17 (a). It uses the wall data structure $w_1, \ldots, w_{n+3}$ to quickly move to the end vertex of a wall (without traversing the possibly many edges along the wall).

$\texttt{contains}(R, j,$  $)$ *(Check for clockwise windmill pattern after jump of rectangle r_j).*
C1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$. If $v_a.\texttt{type} = \top$, return `false`. Otherwise we have $v_a.\texttt{type} = \vdash$ and proceed with C2.
C2. [Check] Set $\alpha \leftarrow v_a.\texttt{north}$, $x \leftarrow e_\alpha.\texttt{wall}$, $b \leftarrow w_x.\texttt{last}$, $\beta \leftarrow v_b.\texttt{east}$, $y \leftarrow e_\beta.\texttt{wall}$, $c \leftarrow w_y.\texttt{last}$, $\gamma \leftarrow v_c.\texttt{south}$, $z \leftarrow e_\gamma.\texttt{wall}$, $d \leftarrow w_z.\texttt{first}$ and $\delta \leftarrow v_d.\texttt{west}$. If $e_\delta.\texttt{right} = j$, return `true`, otherwise return `false`.

The function that tests for the counterclockwise windmill $P =$  is symmetric, and is not shown here for simplicity.

The next two functions test for containment of the patterns $P =$  and $P =$  , respectively; see Figure 17 (b)+(c).

$\texttt{contains}(R, j,$  $)$ *(Check for*  *after jump of rectangle r_j).*
C1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$. If $v_a.\texttt{type} = \top$, return `false`. Otherwise we have $v_a.\texttt{type} = \vdash$ and proceed with C2.
C2. [Check] Set $\alpha \leftarrow v_a.\texttt{south}$ and $b \leftarrow e_\alpha.\texttt{tail}$. If $v_b.\texttt{type} = \dashv$, return `true`, otherwise return `false`.

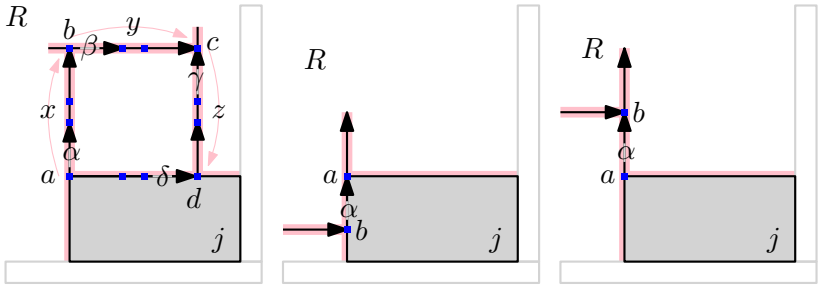


FIGURE 17. Testing containment of the patterns (a)  , (b)  and (c)  . The forbidden configuration of walls is highlighted.

`contains`$(R, j,$  $)$ *(Check for*  *after jump of rectangle* r_j*).*

C1. [Prepare] Set $a \leftarrow r_j$.`nwest`. If v_a.`type` $= \top$, return `false`. Otherwise we have v_a.`type` $= \vdash$ and proceed with C2.

C2. [Check] Set $\alpha \leftarrow v_a$.`north` and $b \leftarrow e_\alpha$.`head`. If v_b.`type` $= \dashv$, return `true`, otherwise return `false`.

Similarly to before, testing for the patterns $P =$  and $P =$  is symmetric to the previous two cases, so we omit those implementations.

It remains to provide containment testing for the patterns $P =$  and $P =$  . We only show the first case, as the other is symmetric; see Figure 18.

`contains`$(R, j,$  $)$ *(Check for*  *after jump of rectangle* r_j*).*

C1. [Prepare] Set $a \leftarrow r_j$.`nwest`. If v_a.`type` $= \top$, return `false`. Otherwise we have v_a.`type` $= \vdash$, set $b \leftarrow r_j$.`swest` and proceed with C2.

C2. [Go up] While v_b.`type` $\notin \{\top, 0\}$ repeat: goto C3; [*] set $\beta \leftarrow v_b$.`north` and $b \leftarrow e_\beta$.`head`. Return `false`.

C3. [Go left] Set $c \leftarrow b$. While v_c.`type` $\notin \{\vdash, 0\}$ repeat: if v_c.`type` $= \bot$ goto C4; [**] set $\gamma \leftarrow v_c$.`west` and $c \leftarrow e_\gamma$.`tail`. Go back to [*].

C4. [Go up] Set $d \leftarrow c$. While $d \neq b$ and v_d.`type` $\notin \{\top, 0\}$ repeat: if v_d.`type` $= \dashv$ return `true`; set $\delta \leftarrow v_d$.`north` and $d \leftarrow e_\delta$.`head`. Go back to [**].

Lines C2–C4 are essentially a triply nested loop that moves along the edges of the vertical wall x that contains the left side of r_j (line C2), the edges of each horizontal wall y whose right end vertex lies on x (line C3), and the edges of each vertical wall z whose bottom end vertex lies on y, searching for a vertex of type $\dashv$ on z (line C4); see Figure 18. This is realized by repeatedly calling line C3 from within the while-loop in line C2, and upon completion returning to from where the call occurred. Similarly, line C4 is repeatedly called from within the while-loop in line C3, and upon completion it returns to from where it was called.

The aforementioned functions have the following runtime guarantees.

Lemma 18. *The function* `contains`(R, j, P) *takes time* $\mathcal{O}(1)$ *for the patterns* $P =$  ,  ,  ,  ,  ,  *and time* $\mathcal{O}(n)$ *for the patterns* $P =$  ,  .

Proof. For the first six patterns the statement is obvious, as the specified functions only make constantly many changes to our data structures. For the pattern $P =$  , note that each edge

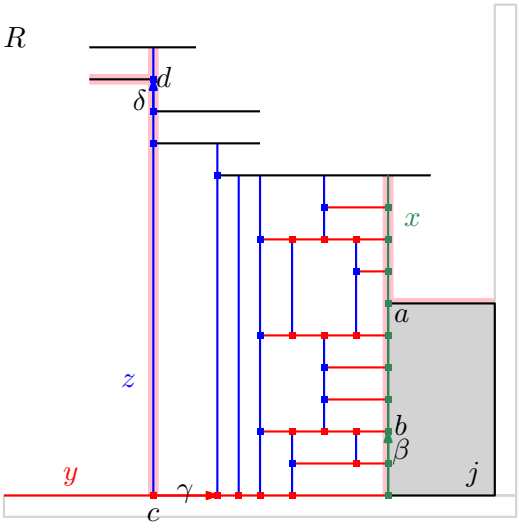


FIGURE 18. Testing containment of the pattern  . The forbidden configuration of walls is highlighted.

index of the rectangulation R is assigned to one of the variables β, γ, δ at most once during the call $\texttt{contains}(R, j, P)$ in lines C2–C4, and the number of edges of R is $3n + 1 = \mathcal{O}(n)$. For the pattern $P =$  the argument is the same. □

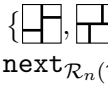
The next theorem combines all the observations from this section, thus establishing most of the runtime bounds stated in Table 1.

Theorem 19. *Let* $\mathcal{P}_1 :=$ {  } *and* $\mathcal{P}_2 :=$ {  }. *Algorithm* $M^{\square}$ *with the minimal jump oracle* $\texttt{next}_{\mathcal{R}_n(\mathcal{P})}$ *or* $\texttt{next}_{\mathcal{D}_n(\mathcal{P})}$ *visits each rectangulation from* $\mathcal{R}_n(\mathcal{P})$ *or* $\mathcal{D}_n(\mathcal{P})$*, respectively, in time* $\mathcal{O}(n)$ *for any set of patterns* $\mathcal{P} \subseteq \mathcal{P}_1$ *and in time* $\mathcal{O}(n^2)$ *for any set of patterns* $\mathcal{P} \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$.

All the bounds stated in Theorem 19 hold in the worst case (not just on average).

Proof. We first consider the minimal jump oracle $\texttt{next}_{\mathcal{D}_n(\mathcal{P})}$, which repeatedly calls the function $\texttt{next}_{\mathcal{D}_n}$ described in Section 7.2. Applying Theorem 17 with the bound $f_n = \mathcal{O}(1)$ from Lemma 14 and the bounds $t_n = \mathcal{O}(1)$ for $\mathcal{P} \subseteq \mathcal{P}_1$ and $t_n = \mathcal{O}(n)$ for $\mathcal{P} \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$ from Lemma 18, the term $n \cdot (f_n + t_n)$ evaluates to $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$, respectively, as claimed.

We now consider the minimal jump oracle $\texttt{next}_{\mathcal{R}_n(\mathcal{P})}$, which repeatedly calls $\texttt{next}_{\mathcal{R}_n}$ described in Section 7.1. In this case applying Theorem 17 directly would not give the desired bounds, so we have to refine the analysis of the while-loop in the algorithm $\texttt{next}_{\mathcal{R}_n(\mathcal{P})}$. Specifically, we consider the sequence of calls to $\texttt{next}_{\mathcal{R}_n}(R, j, d)$ from one rectangulation $R \in \mathcal{R}_n$ avoiding all patterns from $\mathcal{P}$ until the next one. The length of this sequence is at most $\nu = \nu(R^{[j-1]}) \leq n$ (recall Lemma 1), and by Lemma 11 the total time of all calls to $\texttt{next}_{\mathcal{R}_n}(R, j, d)$ is $\mathcal{O}(\nu) = \mathcal{O}(n)$. The total time of all pattern containment tests is at most $\nu \cdot t_n \leq n \cdot t_n$, which is $\mathcal{O}(n)$ for $\mathcal{P} \subseteq \mathcal{P}_1$ and $\mathcal{O}(n^2)$ for $\mathcal{P} \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$ by Lemma 18. This proves the claimed bounds, completing the proof of the theorem. □

Remark 20. For $\mathcal{P} :=$ {  } we have $\mathcal{R}_n(\mathcal{P}) = \mathcal{D}_n$, so we could use, or rather 'misuse', the minimal jump oracle $\texttt{next}_{\mathcal{R}_n(\mathcal{P})}$ to generate $\mathcal{D}_n$. However, this would give a worse guarantee of $\mathcal{O}(n)$ time per visited diagonal rectangulation, rather than $\mathcal{O}(1)$ for $\texttt{next}_{\mathcal{D}_n}$ as guaranteed by Theorem 15.

Remark 21. We write $c(\mathcal{C}_{n-1}(\mathcal{P}))$ for the set of all rectangulations from $\mathcal{C}_n$ that are obtained by inserting a rectangle into a rectangulation from $\mathcal{C}_{n-1}(\mathcal{P})$. To assess the runtime bounds stated in Theorem 19, one may try to investigate the quantity $|c(\mathcal{C}_{n-1}(\mathcal{P})|/|\mathcal{C}_n(\mathcal{P})|$. This is a lower bound for the average number of iterations of the while-loop of the algorithm $\texttt{next}_{\mathcal{C}_n(\mathcal{P})}$ before it returns a rectangulation from $\mathcal{C}_n(\mathcal{P})$. Experimentally, we found that this ratio grows with n in many cases, though maybe not linearly with n, hinting at the possibility that the time bounds stated in Theorem 19 are too pessimistic and can be improved in an average case analysis.

8. Proofs of Theorems 5 and 8

In this section we present the proofs of Theorems 5 and 8. For this purpose we first recap the exhaustive generation framework for permutation languages developed in [HHMW22]. Definitions and terminology intentionally parallel the corresponding definitions given for rectangulations before, and the connection between rectangulations and permutations will be made precise in Lemma 28 below.

8.1. **Permutation basics.** For any two integers $a \leq b$ we define $[a, b] := \{a, a+1, \ldots, b\}$, and we refer to a set of this form as an *interval*. We also define $[n] := [1, n] = \{1, \ldots, n\}$. We write S_n for the set of permutations on $[n]$, and we write $\pi \in S_n$ in one-line notation as $\pi = \pi(1)\pi(2)\cdots\pi(n) = a_1 a_2 \cdots a_n$. Moreover, we use $\varepsilon \in S_0$ to denote the empty permutation, and $\mathrm{id}_n = 12\cdots n \in S_n$ to denote the identity permutation.

Given two permutations π and τ, we say that π *contains the pattern* τ, if there is a subsequence of π whose elements have the same relative order as in τ. Otherwise we say that π *avoids* τ. For example, $\pi = 6\,35\,41\,2$ contains the pattern $\tau = 231$, as the highlighted entries show, whereas $\pi = 654123$ avoids $\tau = 231$. In a *vincular* pattern τ, there is exactly one underlined pair of consecutive entries, with the interpretation that the underlined entries must match adjacent positions in π. For instance, the permutation $\pi = 3\,1\,42$ contains the pattern $\tau = 231$, but it avoids the vincular pattern $\tau = \underline{23}1$.

8.2. **Deletion, insertion and jumps in permutations.** For $\pi \in S_n$, $n \geq 1$, we write $p(\pi) \in S_{n-1}$ for the permutation obtained from π by deleting the largest entry n. We also define $\pi^{[i]} := p^{n-i}(\pi)$ for $i = 1, \ldots, n$. Moreover, for any $\pi \in S_{n-1}$ and any $1 \leq i \leq n$, 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 $\pi = a_1 \cdots a_{n-1}$ then $c_i(\pi) = a_1 \cdots a_{i-1}\, n\, a_i \cdots a_{n-1}$. For example, for $\pi = 412563$ we have $p(\pi) = 41253$ and $c_1(\pi) = 7412563$, $c_5(\pi) = 4125763$ and $c_7(\pi) = 4125637$. Given a permutation $\pi = a_1 \cdots a_n$ with a substring $a_i \cdots a_{i+d}$ with $d > 0$ and $a_i > a_{i+1}, \ldots, a_{i+d}$, a *right jump of the value* a_i *by* d *steps* is a cyclic left rotation of this substring by one position to $a_{i+1} \cdots a_{i+d} a_i$. Similarly, given a substring $a_{i-d} \cdots a_i$ with $d > 0$ and $a_i > a_{i-d}, \ldots, a_{i-1}$, a *left jump of the value* a_i *by* d *steps* is a cyclic right rotation of this substring to $a_i a_{i-d} \cdots a_{i-1}$. For example, a right jump of the value 5 in the permutation 265134 by 2 steps yields 261354.

We say that a jump is *minimal* w.r.t. a set of permutations $L_n \subseteq S_n$, if every jump of the same value in the same direction by fewer steps creates a permutation that is not in L_n.

8.3. **Generating permutations by minimal jumps.** Consider the following analogue of Algorithm J$^\square$ for greedily generating a set of permutations $L_n \subseteq S_n$ using minimal jumps.

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.

The following results were proved in [HHMW22]. 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 and for every $\pi \in L_{n-1}$ we have $c_1(\pi) \in L_n$ and $c_n(\pi) \in L_n$.

We now define a sequence $J(L_n)$ of all permutations from a zigzag language $L_n \subseteq S_n$. 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, in all possible positions that yield a permutation from L_n, skipping the positions that yield a permutation that is not in L_n. If $n = 0$ then we define $J(L_0) := \varepsilon$, and if $n \geq 1$ then we consider the finite sequence $J(L_{n-1}) =: \pi_1, \pi_2, \ldots$ and define

$$J(L_n) := \overleftarrow{c}(\pi_1), \overrightarrow{c}(\pi_2), \overleftarrow{c}(\pi_3), \overrightarrow{c}(\pi_4), \ldots, \tag{2}$$

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.

Theorem 22 ([HHMW22, Thm. 1+Lemma 4])**.** *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, in the order $J(L_n)$ defined by* (2)*. Moreover, if $|L_i|$ is even for all $2 \leq i \leq n-1$, then the sequence $J(L_n)$ is cyclic, i.e., the first and last permutation differ in a minimal jump.*

A permutation π is called *2-clumped* if it avoids each of the vincular patterns $3\underline{51}24$, $3\underline{51}42$, $24\underline{51}3$, and $42\underline{51}3$. We write $S'_n \subseteq S_n$ for the set of 2-clumped permutations.

Lemma 23 ([HHMW22, Thm. 8+Lemma 10])**.** *We have $S'_0 = \{\varepsilon\}$, and for every $n \geq 1$ we have $S'_{n-1} = \{p(\pi) \mid \pi \in S'_n\}$ and $S'_n \supseteq \{c_1(\pi), c_n(\pi) \mid \pi \in S'_{n-1}\}$. In particular, S'_n is a zigzag language for all $n \geq 0$.*

We also state the following observations for further reference.

Lemma 24. *The sequence of permutations $J(L_n)$ defined in* (2) *has the following properties:*

(a) The first permutation in $J(L_n)$ is the identity permutation id_n.
(b) For $j = 2, \ldots, n$, the first jump of the value j in $J(L_n)$ is a left jump.
(c) Every jump in the sequence $J(L_n)$ is minimal w.r.t. L_n.
(d) Given two consecutive permutations π, ρ in $J(L_n)$ that differ in a jump of some value j, then we have $\pi^{[k]} = c_1(\pi^{[k-1]})$ and $\rho^{[k]} = c_1(\rho^{[k-1]})$, or $\pi^{[k]} = c_k(\pi^{[k-1]})$ and $\rho^{[k]} = c_k(\rho^{[k-1]})$ for all $k = j+1, \ldots, n$.
(e) Let π, ρ be two consecutive permutations in $J(L_n)$ such that ρ is obtained from π by a left jump of some value j, and let π', ρ' be the next two consecutive permutations in $J(L_n)$ that differ in a jump of j. If j is not at the first position in ρ and the value left of it is smaller than j, then ρ' is obtained from π' by a left jump. Conversely, if j is at the first position in ρ or the value left of it is bigger than j, then ρ' is obtained from π' by a right jump. An analogous statement holds with left and right interchanged.

Proof. Properties (a) and (b) follow easily from the definition (2).

Property (c) follows from Theorem 22 and line J2 of Algorithm J.

We prove (d) and (e) by induction on n. Both statements are trivial for $n = 0$ and $n = 1$, which settles the induction basis. We now assume that $n \geq 2$, and suppose that $J(L_{n-1}) =: \pi_1, \pi_2, \ldots$ satisfies (a) and (b) for jumps of all values $j \in \{2, \ldots, n-1\}$.

We start with the induction step for (d). If π, ρ in $J(L_n)$ differ in a jump of the value $j = n$, then (d) is satisfied trivially, so it suffices to consider jumps of values $j \in \{2, \ldots, n-1\}$ in $J(L_n)$. However, by (2), such jumps only occur at the transitions between $\overleftarrow{c}(\pi_k)$ and $\overrightarrow{c}(\pi_{k+1})$ or between $\overrightarrow{c}(\pi_k)$ and $\overleftarrow{c}(\pi_{k+1})$ for some k. In the first case, $\pi = \pi^{[n]} := n\pi_k = c_1(\pi_k) = c_1(\pi^{[n-1]})$ is followed by $\rho = \rho^{[n]} := n\pi_{k+1} = c_1(\pi_{k+1}) = c_1(\rho^{[n-1]})$, and in the second case $\pi = \pi^{[n]} := \pi_k n = c_n(\pi_k) = c_n(\pi^{[n-1]})$ is followed by $\rho = \rho^{[n]} := \pi_{k+1} n = c_n(\pi_{k+1}) = c_n(\rho^{[n-1]})$, completing the proof.

We proceed with the induction step for (e). If π, ρ in $J(L_n)$ differ in a jump of the value $j = n$, then (e) follows from the definition (2) and of the sequences $\overleftarrow{c}(\pi_i)$ and $\overrightarrow{c}(\pi_i)$. On the other hand, consider π, ρ in $J(L_{n-1})$ that differ in a jump of some value $j \in \{2, \ldots, n-1\}$, and let π', ρ' be the next two permutations in $J(L_{n-1})$ that differ in a jump of j, satisfying (e) by induction. Then in $J(L_n)$, after the jump of j from $c_1(\pi)$ to $c_1(\rho)$ or from $c_n(\pi)$ to $c_n(\rho)$, the next jump of the value j is from $c_1(\pi')$ to $c_1(\rho')$ or from $c_n(\pi')$ to $c_n(\rho')$. If j is not at the first position in ρ and the value left of it is smaller than j, then j is not at the first position and the value left of it is smaller in both $c_1(\rho)$ and $c_n(\rho)$, and $c_1(\rho')$ and $c_n(\rho')$ are obtained from $c_1(\pi')$ and $c_n(\pi')$, respectively, by a left jump. On the other hand, if j is at the first position in ρ or the value left of it is bigger than j, then j is at the first position or the value left of it is bigger than j in both $c_1(\rho)$ and $c_n(\rho)$, and $c_1(\rho')$ and $c_n(\rho')$ are obtained from $c_1(\pi')$ and $c_n(\pi')$, respectively, by a right jump.

This completes the proof. □

8.4. **A surjection from permutations to generic rectangulations.** Observe that a diagonal rectangulation with n rectangles can be laid out canonically so that each rectangle intersects the main diagonal in a $1/n$-fraction. Specifically, rectangle r_i intersects the main diagonal in the ith such line segment counted from top-left to bottom-right, for $i = 1, \ldots, n$. We say that r_i is *left-fixed* or *left-extended*, if its left side touches or does not touch the diagonal, respectively. These notions are defined analogously for all the other three sides right, bottom, and top.

We begin by reviewing a mapping $\rho : S_n \to \mathcal{D}_n$, $n \geq 1$, from permutations to diagonal rectangulations, first described by Law and Reading [LR12]. Maps closely related to ρ have appeared previously in the literature, see e.g. [ABP06b, FFNO11]. We consider the outer rectangle and divide its main diagonal into n equally sized line segments numbered $1, \ldots, n$ from top-left to bottom-right. Given a permutation $\pi = a_1 \cdots a_n \in S_n$, the diagonal rectangulation $\rho(\pi)$ is obtained as follows; see Figure 19: For $i = 1, \ldots, n$, in step i we add the rectangle r_{a_i} such that it intersects the main diagonal precisely in the a_ith line segment, and such that the rectangle is maximal w.r.t. the property that the rectangles $r_{a_1} \cup \cdots \cup r_{a_i}$ form a *staircase*, which means that the bottom-left boundary of $r_{a_1} \cup \cdots \cup r_{a_i}$ is an L-shape, and the top-right boundary is a non-increasing polygonal line.

With any wall w of a generic rectangulation $R \in \mathcal{R}_n$ we associate a *wall shuffle* $\sigma(w)$, which is a permutation of a subset of the rectangles that share a side with w, defined as follows; see Figure 20. If the wall w is horizontal, we move from the left endpoint of w to the right endpoint, and whenever we encounter a vertical wall w' that is incident to w from the bottom, we record the rectangle whose top side lies on w and left side lies on w', and if we encounter a vertical wall w' that is incident to w from the top, we record the rectangle whose bottom side lies on w and right side lies on w'. Clearly, we record all rectangles whose bottom or top side lies on w, except the first rectangle below w and the last rectangle above w. On the other hand, if the

$\pi = (8, 13, 7, 5, 11, 2, 14, 6, 15, 9, 10, 3, 1, 4, 12)$

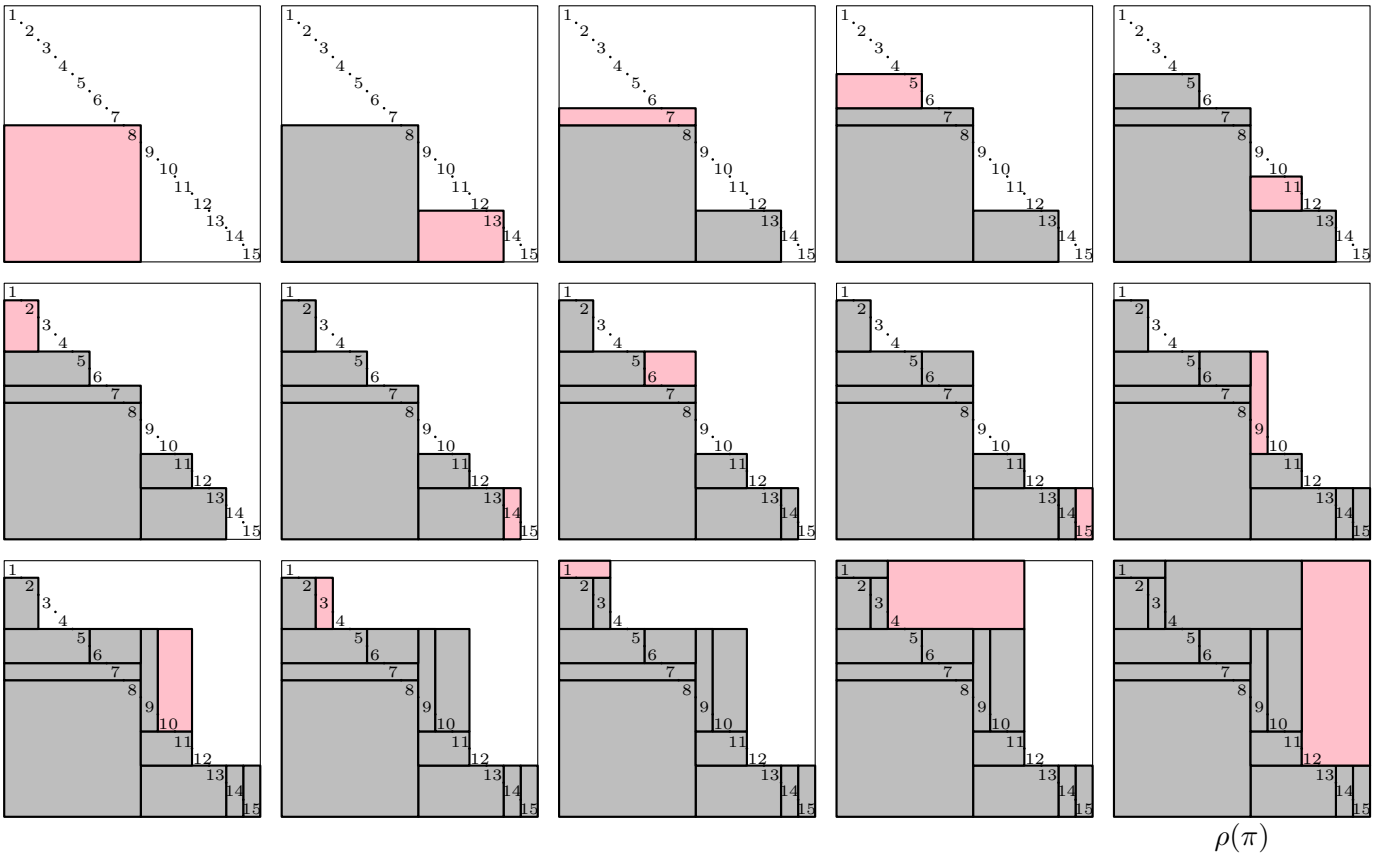


FIGURE 19. Illustration of the mapping $\rho : S_n \to \mathcal{D}_n$ for the permutation $\pi = (8, 13, 7, 5, 11, 2, 14, 6, 15, 9, 10, 3, 1, 4, 12)$ (example from [Rea12]).

wall w is vertical, we move from the bottom endpoint of w to the top endpoint, and whenever we encounter a horizontal wall w' that is incident to w from the left, we record the rectangle whose right side lies on w and bottom side lies on w', and if we encounter a horizontal wall w' that is incident to w from the right, we record the rectangle whose left side lies on w and top side lies on w'. In this case we record all rectangles whose left or right side lies on w, except the first rectangle to the left of w and the last rectangle to the right of w. Observe that wall slides do not affect the rectangles that appear in a wall shuffle, but only their relative order in the shuffle.

We are now in position to define the mapping $\gamma : S_n \to \mathcal{R}_n$, $n \geq 1$, from permutations to generic rectangulations; see Figure 21. From $\pi \in S_n$ we first construct the diagonal rectangulation $R := \rho(\pi) \in \mathcal{D}_n$. Let w be a horizontal wall in R, and consider the rectangles in R whose bottom side lies on w from left to right. By construction of ρ, these rectangles form an increasing subsequence of π. Similarly, the rectangles in R whose top side lies on w from left to right form an increasing subsequence of π. Thus, we can specify a wall shuffle $\sigma(w)$ by taking the subsequence of π that contains the appropriate rectangle numbers. On the other hand, for a vertical wall w in R, the rectangles in R whose left side lies on w from bottom to top form a decreasing subsequence of π, and the rectangles whose right side lies on w form a decreasing subsequence of π, so we can specify a wall shuffle $\sigma(w)$ by taking the subsequence of π containing the appropriate rectangle numbers. The rectangulation $\gamma(\pi) \in \mathcal{R}_n$ is obtained from $R \in \mathcal{D}_n$ by applying wall slides to it, so as to obtain the wall shuffles specified by π.

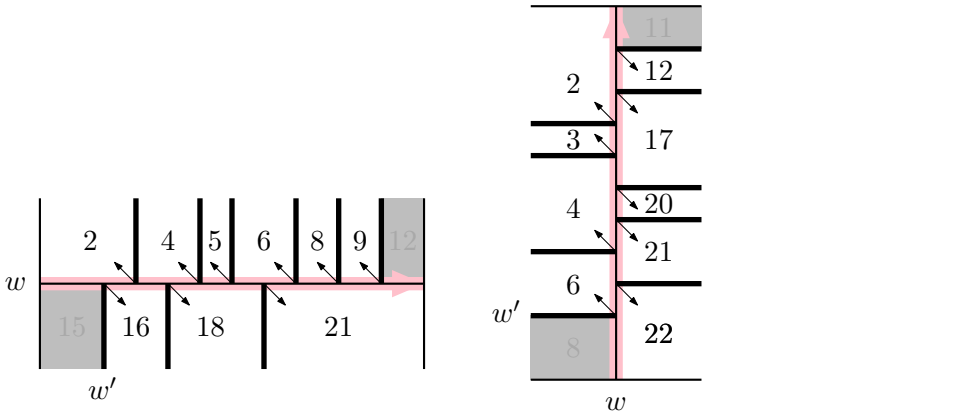


FIGURE 20. Illustration of wall shuffles.

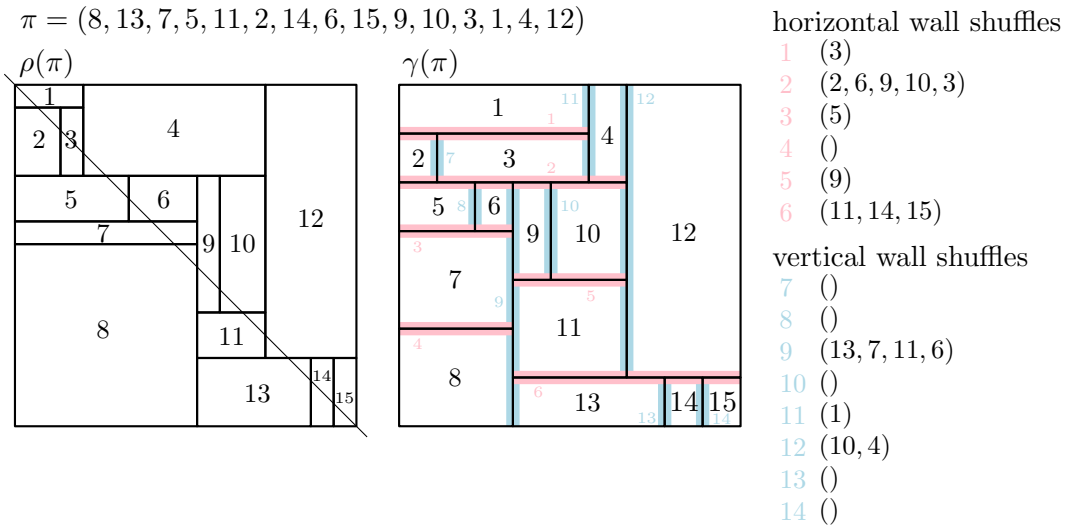


FIGURE 21. Illustration of the surjection $\gamma : S_n \to \mathcal{R}_n$. This example continues Figure 19.

Lemma 25 ([Rea12, Prop. 4.2])**.** *The map $\gamma : S_n \to \mathcal{R}_n$ is surjective.*

Even though γ is not a bijection, Reading [Rea12] showed that it becomes a bijection when restricting the domain to 2-clumped permutations.

Theorem 26 ([Rea12, Thm. 4.1])**.** *The map γ is a bijection when restricted to the set S'_n of 2-clumped permutations.*

8.5. **The connection between permutations and rectangulations.** The key lemma of this section, Lemma 28 below, asserts that deletion, insertion and jumps in permutations as defined in Section 8.2 are in bijective correspondence under γ to deletion, insertion and jumps in generic rectangulations as defined in Sections 2.3, 2.4 and 3.1. In order to prove it, we first establish a coarser version of this statement for the mapping ρ.

Lemma 27. *Let $\pi = a_1 \cdots a_n \in S_n$, $n \geq 1$, and define $P := \rho(\pi) \in \mathcal{D}_n$. In the sequence $I(P) = (q_1, \ldots, q_\nu)$, consider the subsequence $I'(P) = (q_{j_1}, \ldots, q_{j_\mu})$ consisting of the first insertion point of every vertical group, and the last insertion point of every horizontal group. Then we have*

$p(\rho(c_i(\pi))) = P$ for all $i = 1, \ldots, n+1$, and the sequence $1, \ldots, n+1$ can be partitioned into consecutive nonempty intervals $I_1, \ldots, I_\mu$ with the following properties:

- *(a) for every $k = 1, \ldots, \mu$ and every $i \in I_k$ we have $\rho(c_i(\pi)) = c_{j_k}(P)$;*
- *(b) for any interval $I_k = [\hat{\imath}, \check{\imath}]$, $1 < k < \mu$, such that the top-left vertex of r_{n+1} in $\rho(c_i(\pi))$, $i \in I_k$, has type $\vdash$, we have that the rectangle $r_{a_{\hat{\imath}-1}}$ is the unique rectangle left of r_{n+1}, and the rectangle $r_{a_{\check{\imath}}}$ is the leftmost rectangle above r_{n+1};*
- *(c) for any interval $I_k = [\hat{\imath}, \check{\imath}]$, $1 < k < \mu$, such that the top-left vertex of r_{n+1} in $\rho(c_i(\pi))$, $i \in I_k$, has type $\top$, we have that the rectangle $r_{a_{\hat{\imath}-1}}$ is the topmost rectangle left of r_{n+1}, and the rectangle $r_{a_{\check{\imath}}}$ is the unique rectangle above r_{n+1};*
- *(d) we have $I_1 = \{1\}$ and $I_\mu = \{n+1\}$.*

Proof. For the reader's convenience, the proof is illustrated in Figure 22. Recall the definition of the mapping ρ via the process described in Section 8.4 and illustrated in Figure 19. Using the definition of the rectangle insertion from Section 2.4, we first observe that $\rho(c_1(\pi)) = c_1(P)$ and $\rho(c_{n+1}(\pi)) = c_\nu(P)$. We consider the sequence of permutations $c_i(\pi)$ for $i = 1, \ldots, n+1$ and their images under ρ. Observe that $c_{i+1}(\pi)$ is obtained from $c_i(\pi)$ by the adjacent transposition $(n+1)a_i \longrightarrow a_i(n+1)$. We consider the construction of $R := \rho(c_i(\pi))$, specifically the two steps in which the rectangles r_{n+1} and r_{a_i} are added, and we analyze how the swapped insertion order of these two rectangles changes the resulting rectangulation $R' := \rho(c_{i+1}(\pi))$. Clearly, in both R and R', the rectangle r_{n+1} is either left-extended and top-fixed, or left-fixed and top-extended, and we treat both cases separately. In particular, these two cases are not symmetric.

Case (i): r_{n+1} is left-extended and top-fixed in R. This case is shown in the top and middle part of Figure 22. We distinguish two subcases, namely $a_i = n$ and $a_i < n$.

Case (ia): $a_i = n$. In this case the top side of r_{n+1} coincides with the bottom side of $r_{a_i} = r_n$ in R, as both rectangles left-extend to the same vertical line by the staircase property and the fact that every rectangle intersects the main diagonal; see Figure 22 (a2). Consequently, inserting them in the swapped order, first r_n and then r_{n+1}, produces a sub-rectangulation that differs in a simple flip of the wall between these two rectangles; see Figure 22 (a3). As the height of the top side of r_n is not determined by r_{n+1}, but only by previous rectangles $r_{a_1}, \ldots, r_{a_{i-1}}$, both insertion orders produce the same staircase $r_{a_1} \cup \cdots \cup r_n \cup r_{n+1}$, which by the definition of ρ is all that matters for the next construction steps. Consequently, R and R' differ in a simple flip of the rectangles r_{n+1} and r_n. Moreover, by the definition of rectangle deletion given in Section 2.3 we have $p(R) = p(R') = P$, and by the definition of rectangle insertion and the definition of $I'(P)$ we have $R = c_{j_k}(P)$ and $R' = c_{j_{k+1}}(P)$ for some index k. Note that in R, the top-left vertex of r_{n+1} has type $\vdash$ and r_{a_i}, $\check{\imath} := i$, is the leftmost rectangle above r_{n+1}. Furthermore, in R', the top-left vertex of r_{n+1} has type $\top$ and $r_{a_{\hat{\imath}-1}} = r_{a_i}$, $\hat{\imath} := i+1$, is the topmost rectangle left of r_{n+1}.

Case (ib): $a_i < n$. If r_{a_i} is bottom-fixed in R, or if r_{n+1} does not left-extend beyond the vertical line ℓ through the right endpoint of the a_ith interval on the main diagonal, then the rectangles r_{n+1} and r_{a_i} do not touch; see Figure 22 (a1). Consequently, inserting them in the swapped order, first r_{a_i} and then r_{n+1}, produces the same sub-rectangulation. It follows that $R = R'$, i.e., $\rho(c_i(\pi)) = \rho(c_{i+1}(\pi))$, and therefore trivially $p(R) = p(R')$.

On the other hand, if r_{a_i} is bottom-extended and r_{n+1} left-extends beyond ℓ, then r_{a_i} must be right-fixed because of $a_i < n$ (otherwise r_{a_i} would reach into the (a_i+1)st line segment on the main diagonal). Moreover, because of the staircase property the top side of r_{n+1} contains the bottom side of r_{a_i}, and both rectangles left-extend to the same vertical line; see Figure 22 (b2). Consequently, inserting them in the swapped order, first r_{a_i} and then r_{n+1} produces a sub-rectangulation that differs in a T-flip from $\perp$ to $\vdash$ around the top-right common vertex of

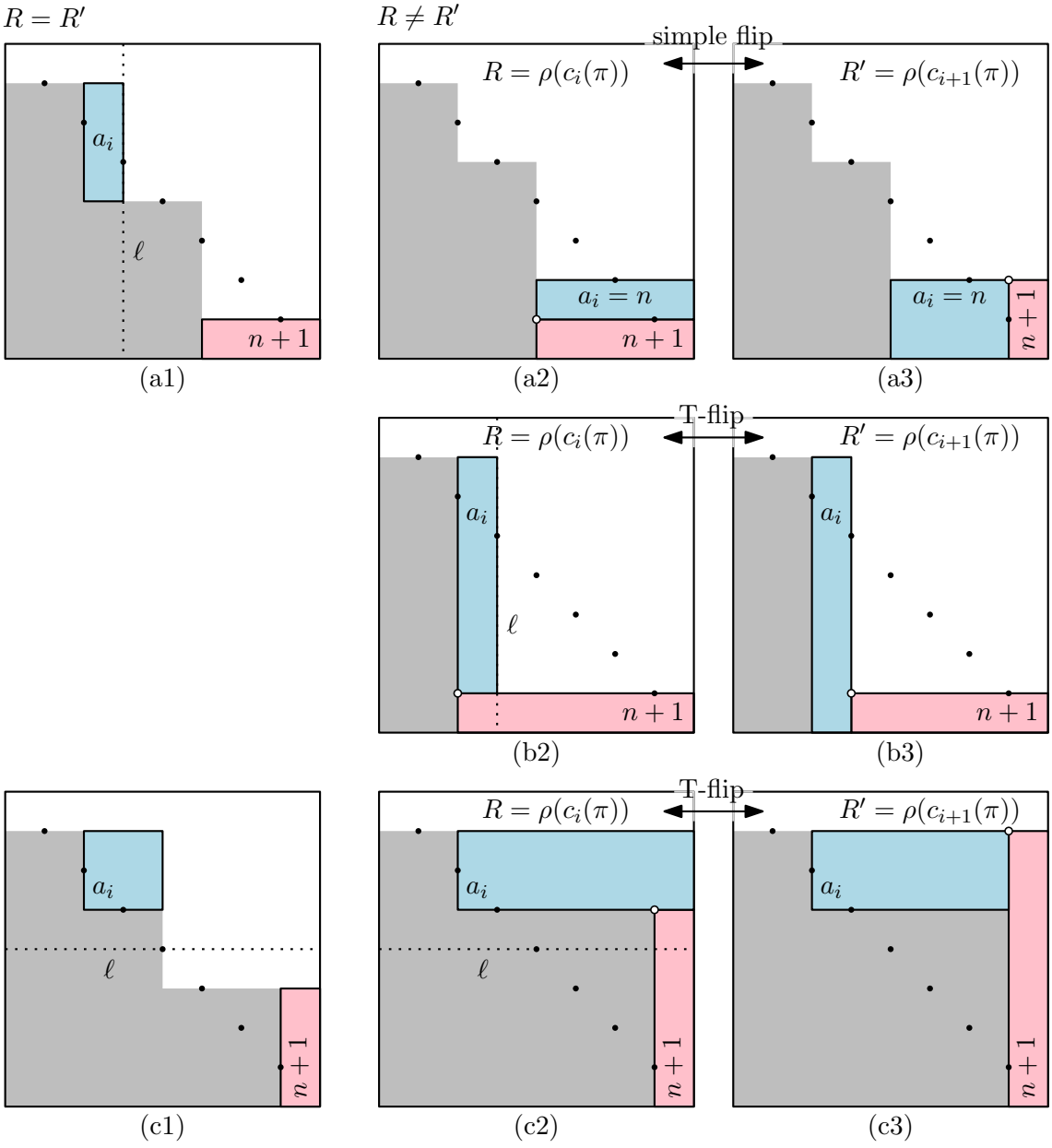


FIGURE 22. Illustration of the proof of Lemma 27.

these two rectangles; see Figure 22 (b3). As the height of the top side of r_{a_i} is not determined by r_{n+1}, and as r_{a_i} is right-fixed and r_{n+1} is top-fixed in both insertion orders, we obtain that R and R' differ in a T-flip around the top-right common vertex of the rectangles r_{n+1} and r_{a_i}. Moreover, by the definition of rectangle insertion and deletion and the definition of $I'(P)$, we have $p(R) = p(R') = P$, and $R = c_{j_k}(P)$ and $R' = c_{j_{k+1}}(P)$ for some index k. Note that in R, the top-left vertex of r_{n+1} has type $\vdash$ and r_{a_i}, $\check{\imath} := i$, is the leftmost rectangle above r_{n+1}. Furthermore, in R', the top-left vertex of r_{n+1} has type $\vdash$ and $r_{a_{i-1}} = r_{a_i}$, $\hat{\imath} := i + 1$, is the unique rectangle left of r_{n+1}.

Case (ii): r_{n+1} is left-fixed and top-extended in R. This case is shown in the bottom part of Figure 22. As r_{n+1} is top-extended, the staircase property implies that $a_i < n$, as n must be among the first entries $a_1, \ldots, a_{i-1}$ of π.

If r_{a_i} is right-fixed in R, or if r_{n+1} does not top-extend beyond the horizontal line ℓ through the right endpoint of the (a_i+1)st interval on the main diagonal, then the rectangles r_{n+1} and r_{a_i} do not touch; see Figure 22 (c1). Consequently, inserting them in the swapped order, first r_{a_i} and then r_{n+1}, produces the same sub-rectangulation. It follows that $R = R'$, i.e., $\rho(c_i(\pi)) = \rho(c_{i+1}(\pi))$, and therefore trivially $p(R) = p(R')$.

On the other hand, if r_{a_i} is right-extended and r_{n+1} top-extends beyond ℓ, then r_{a_i} must be bottom-fixed because of $a_i < n$ (otherwise r_{a_i} would reach into the (a_i+1)st line segment on the main diagonal). Moreover, because of the staircase property the top side of r_{n+1} must lie on the horizontal line through the bottom side of r_{a_i}, and therefore r_{a_i} right-extends to the right outer boundary, i.e., the bottom side of r_{a_i} contains the top side of r_{n+1}; see Figure 22 (c2). Consequently, inserting them in the swapped order, first r_{a_i} and then r_{n+1} produces a sub-rectangulation that differs in a T-flip from $\top$ to $\dashv$ around the bottom-left common vertex of these two rectangles; see Figure 22 (c3). As the height of the top side of r_{a_i} is not determined by r_{n+1} in the two insertion orders, we obtain that R and R' differ in a T-flip around the bottom-left common vertex of the rectangles r_{n+1} and r_{a_i}. Moreover, by the definition of rectangle insertion and deletion and the definition of $I'(P)$, we have $p(R) = p(R') = P$, and $R = c_{j_k}(P)$ and $R' = c_{j_{k+1}}(P)$ for some index k. Note that in R, the top-left vertex of r_{n+1} has type $\top$ and r_{a_i}, $\check{\imath} := i$, is the unique rectangle above r_{n+1}. Furthermore, in R', the top-left vertex of r_{n+1} has type $\top$ and $r_{a_{i-1}} = r_{a_i}$, $\hat{\imath} := i+1$, is the topmost rectangle left of r_{n+1}.

This proves (a), (b) and (c). For (d) observe that the rectangle in the bottom-left corner of $\rho(\pi)$ is r_{a_1}, whereas the rectangle in the top-right corner is r_{a_n}, so $\rho(c_1(\pi)) \neq \rho(c_2(\pi))$ and $\rho(c_n(\pi)) \neq \rho(c_{n+1}(\pi))$. This completes the proof of the lemma. □

Lemma 28. *Let $\pi = a_1 \cdots a_n \in S_n$, $n \geq 1$, and consider the rectangulation $P := \gamma(\pi) \in \mathcal{R}_n$ with $\nu = \nu(P)$ insertion points. Then we have $p(\gamma(c_i(\pi))) = P$ for all $i = 1, \ldots, n+1$, and the sequence $i = 1, \ldots, n+1$ can be partitioned into consecutive nonempty intervals $I_1, \ldots, I_\nu$, such that for every $k = 1, \ldots, \nu$ and every $i \in I_k$ we have $\gamma(c_i(\pi)) = c_k(P)$. Furthermore, if π is 2-clumped, then each interval I_k contains exactly one 2-clumped permutation.*

The proof of Lemma 28 shows that $I_1 = \{1\}$ and $I_\nu = \{n+1\}$.

Proof. For any $i = 1, \ldots, n+1$, consider the permutation $c_i(\pi)$, and the two diagonal rectangulations $P' := \rho(\pi) \in \mathcal{D}_n$ and $R' := \rho(c_i(\pi)) \in \mathcal{D}_n$. From Lemma 27 we know that $P' = p(R')$ and $R' = c_j(P')$ for some index j such that the jth insertion point in $I(P')$ is the first of a vertical group, or the last of a horizontal group. By the definitions from Section 8.4, γ consists of applying ρ plus wall slides that are determined by the wall shuffles of P' and R' and the relative order of the rectangle indices in those shuffles in π and $c_i(\pi)$, i.e., $P = \gamma(\pi)$ and $R := \gamma(c_i(\pi))$ are obtained from P' and R' by wall slides. Observe that in R' and R, the bottom-right rectangle r_{n+1} is contained in at most one wall shuffle. Specifically, if the top-left corner of r_{n+1} has type $\vdash$ and does not lie on the left boundary of the rectangulation, then r_{n+1} participates in a single wall shuffle of the vertical wall that contains the left side of r_{n+1}, whereas if the top-left corner of r_{n+1} has type $\top$ and does not lie on the upper boundary of the rectangulation, then r_{n+1} participates in a single wall shuffle of the horizontal wall that contains the top side of r_{n+1}. In particular, in the cases $j = 1$ and $j = \nu(P')$ the rectangle r_{n+1} is not contained in any wall shuffle. As the elements of all subsequences of π and $c_i(\pi)$ that do not contain $n+1$ appear in the same relative order in both permutations, all wall shuffles of P and R are the same, except the wall shuffle of R containing $n+1$, which is obtained from a wall shuffle of P by inserting the value $n+1$. We conclude that $p(R) = P$ and $R = c_k(P)$ for some index k.

The desired interval partition $I_1, \ldots, I_\nu$ of the indices $1, \ldots, n+1$ is obtained by refining the partition $I'_1, \ldots, I'_\mu$ guaranteed by Lemma 27, where μ is the number of vertical and horizontal groups of P', which is the same for P, as wall slides do not affect it. As $I'_1 = \{1\}$ and $I'_\mu = \{n+1\}$ by Lemma 27 (d), these two intervals are not refined, so we have $I_1 := I'_1 = \{1\}$ and $I_\nu := I_\mu = \{n+1\}$. It remains to consider the remaining intervals I'_k, $1 < k < \mu$.

First consider an interval $I'_k =: [\hat{\imath}, \check{\imath}]$, $1 < k < \mu$, such that in $R' := \rho(c_i(\pi)) \in \mathcal{D}_n$, $i \in I_k$, the top-left vertex has type $\vdash$ (recall Lemma 27 (a)). By Lemma 27 (b), in R' the rectangle $r_{a_{\hat{\imath}-1}}$ is the unique rectangle left of r_{n+1}, and the rectangle $r_{a_{\hat{\imath}}}$ is the leftmost rectangle above r_{n+1}. Consider the wall shuffle $\sigma(w)$ of the vertical wall w between $r_{a_{\hat{\imath}-1}}$ and $r_{a_{\hat{\imath}}}$ in P. It has the form $\sigma(w) = (b_1, \ldots, b_\lambda, a_{\check{\imath}}, \ldots)$, for some $\lambda \geq 0$, i.e., the rectangles $r_{b_1}, \ldots, r_{b_\lambda}$ are stacked on top of $r_{a_{\hat{\imath}-1}}$ and their bottom sides are incident with w below the incidence of the top side of $r_{\check{\imath}}$ with w. It follows that π contains the subsequence $a_{\hat{\imath}-1}, b_1, \ldots, b_\lambda, a_{\check{\imath}}$, and so the permutations $c_i(\pi)$ for $i = \hat{\imath}, \ldots, \check{\imath}$ have the value $n+1$ appear at every possible position within the subsequence $b_1, \ldots, b_\lambda$. Consequently, the interval I'_k is refined into λ subintervals such that $\gamma(c_i(\pi)) = \gamma(c_{i+1}(\pi))$ if $i, i+1$ are in the same subinterval and $\gamma(c_i(\pi)) = c_\ell(P)$ and $\gamma(c_{i+1}(\pi)) = c_{\ell+1}(P)$ for some index ℓ if $i, i+1$ are in consecutive subintervals.

Now consider an interval $I'_k =: [\hat{\imath}, \check{\imath}]$, $1 < k < \mu$, such that in $R' := \rho(c_i(\pi)) \in \mathcal{D}_n$, $i \in I_k$, the top-left vertex has type $\top$ (recall Lemma 27 (a)). By Lemma 27 (c), in R' the rectangle $r_{a_{\hat{\imath}-1}}$ is the topmost rectangle left of r_{n+1}, and the rectangle $r_{a_{\hat{\imath}}}$ is the unique rectangle above r_{n+1}. Consider the wall shuffle $\sigma(w)$ of the horizontal wall w between $r_{a_{\hat{\imath}-1}}$ and $r_{a_{\hat{\imath}}}$ in P. It has the form $\sigma(w) = (\ldots, a_{\hat{\imath}-1}, b_1, \ldots, b_\lambda)$, for some $\lambda \geq 0$, i.e., the rectangles $r_{b_\lambda}, \ldots, r_{b_1}$ are stacked to the left of $r_{a_{\hat{\imath}}}$ and their right sides are incident with w to the right of the incidence of the left side of $r_{a_{\hat{\imath}-1}}$ with w. It follows that π contains the subsequence $a_{\hat{\imath}-1}, b_1, \ldots, b_\lambda, a_{\check{\imath}}$, and so the permutations $c_i(\pi)$ for $i = \hat{\imath}, \ldots, \check{\imath}$ have the value $n+1$ appear at every possible position within the subsequence $b_1, \ldots, b_\lambda$. Consequently, the interval I'_k is refined into λ subintervals such that $\gamma(c_i(\pi)) = \gamma(c_{i+1}(\pi))$ if $i, i+1$ are in the same subinterval and $\gamma(c_i(\pi)) = c_\ell(P)$ and $\gamma(c_{i+1}(\pi)) = c_{\ell+1}(P)$ for some index ℓ if $i, i+1$ are in consecutive subintervals.

It remains to argue that each set of permutations $C_k := \{c_i(\pi) \mid i \in I_k\}$, $k = 1, \ldots, \nu$, contains exactly one 2-clumped permutation. Indeed, C_k can contain at most one 2-clumped permutation by Theorem 26, as all $\pi \in C_k$ have the same image under γ. Suppose for the sake of contradiction that C_k contains no 2-clumped permutation. Then by Theorem 26 there is another 2-clumped permutation $\rho \in S_{n+1}$ with $\rho \notin C := \{c_i(\pi) \mid i = 1, \ldots, n+1\}$ and $\gamma(\rho) = \gamma(c_i(\pi))$ for all $i \in C_k$. However, by Lemma 23, the permutation $\rho' := p(\rho) \in S_n$ is also 2-clumped, i.e., we have $\rho' \in S'_n$. Moreover, we have $\rho' \neq \pi$ as $\rho \notin C$, and by Lemma 28 we have $\gamma(\rho') = \gamma(\pi) = R$, a contradiction to the fact that γ is a bijection between S'_n and $\mathcal{R}_n$ by Theorem 26. □

8.6. **Proof of Theorem 5.** With Lemma 28 in hand, we are now in position to present the proof of Theorem 5.

Proof of Theorem 5. Consider a zigzag set of rectangulations $\mathcal{C}_n \subseteq \mathcal{R}_n$, and consider the zigzag sets $\mathcal{C}_{i-1} := \{p(R) \mid R \in \mathcal{C}_i\}$ for $i = n, n-1, \ldots, 2$. By Lemma 28, for all $i = 1, \ldots, n$ there is a set of 2-clumped permutations $L_i \subseteq S'_i$ such that γ restricted to L_i is a bijection between L_i and $\mathcal{C}_i$, and such that $L_{i-1} = \{p(\pi) \mid \pi \in L_i\}$ for all $i = 2, \ldots, n$. Moreover, as $\mathcal{C}_i$ is a zigzag set, we know that for all $R \in \mathcal{C}_{i-1}$ we have $c_1(R) \in \mathcal{C}_i$ and $c_{\nu(R)}(R) \in \mathcal{C}_i$, for all $i = 2, \ldots, n$. By Lemma 28, this implies that for all $\pi \in L_{i-1}$ we have $c_1(\pi) \in L_i$ and $c_i(R) \in L_i$, for all $i = 2, \ldots, n$, i.e., L_n is a zigzag language of 2-clumped permutations (using that $L_1 = \{1\}$ and $L_0 := \{p(\pi) \mid \pi \in L_1\} = \{\varepsilon\}$ are zigzag languages).

By Theorem 22, Algorithm J visits every permutation from L_n exactly once, in the order $J(L_n)$ defined by (2). From Lemma 24 (d) we obtain that if Algorithm J performs a jump of some value j in the current permutation $\pi \in L_n$, then the corresponding rectangulations $R^{[k]} := \gamma(\pi^{[k]})$ for $k = j+1, \ldots, n$ are either bottom-based or right-based. Using the definition of jumps from Section 3.1 and Lemma 28, a minimal left/right jump of the value j in $\pi \in L_n$, as performed by Algorithm J, corresponds to a minimal left/right jump of the rectangle r_j in $\gamma(\pi) \in \gamma(L_n) = \mathcal{C}_n$, as performed by Algorithm J$^{\square}$. This together with the observation that $\gamma(\mathrm{id}_n) = \boxed{\begin{smallmatrix}\boxed{}\boxed{n}\\ \boxed{\cdots}\end{smallmatrix}}$ proves the first part of the theorem. Specifically, the ordering of rectangulations generated by Algorithm J$^{\square}$ is

$$J^{\square}(\mathcal{C}_n) = \gamma(J(L_n)). \tag{3}$$

To prove the second part of the theorem, by Theorem 22 it suffices to show that $|\mathcal{C}_i| = |\gamma(L_i)|$ is even for all $2 \le i \le n-1$. For any rectangulation $R \in \mathcal{R}_n$, we write $\lambda(R) \in \mathcal{R}_n$ for the rectangulation obtained by reflection at the main diagonal. First observe that if $R, R' \in \mathcal{R}_n$, $n \ge 2$, satisfy $R' = \lambda(R)$, then we also have $p(R') = \lambda(p(R))$. Consequently, the assumption that $\mathcal{C}_n$ is symmetric implies that $\mathcal{C}_i$ is symmetric for all $i = 1, \ldots, n$. Consider a rectangulation $R \in \mathcal{R}_n$, $n \ge 2$, and observe that if the top-left vertex of the bottom-right rectangle r_n of R has type $\vdash$, then it has type $\top$ in $\lambda(R)$, and vice versa. It follows that λ is an involution without fixed points on $\mathcal{C}_i$ for all $i = 2, \ldots, n$, proving that $|\mathcal{C}_i|$ is even.

This completes the proof. □

8.7. **Memoryless generation of permutations.** Consider Algorithm M below, which takes as input a zigzag language of permutations $L_n \subseteq S_n$ and generates them exhaustively by minimal jumps in the same order as Algorithm J, i.e., in the order $J(L_n)$.

Algorithm M *(Memoryless minimal jumps).* This algorithm generates all permutations of a zigzag language $L_n \subseteq S_n$ by minimal jumps in the same order as Algorithm J. It maintains the current permutation in the variable π, and auxiliary arrays $o = (o_1, \ldots, o_n)$ and $s = (s_1, \ldots, s_n)$.

M1. [Initialize] Set $\pi \leftarrow \mathrm{id}_n = 12\cdots n$, and $o_j \leftarrow \triangleleft$, $s_j \leftarrow j$ for $j = 1, \ldots, n$.

M2. [Visit] Visit the current permutation π.

M3. [Select value] Set $j \leftarrow s_n$, and terminate if $j = 1$.

M4. [Jump value] In the current permutation π, perform a jump of the value j that is minimal w.r.t. L_n, where the jump direction is left if $o_j = \triangleleft$ and right if $o_j = \triangleright$.

M5. [Update o and s] Set $s_n \leftarrow n$. If $o_j = \triangleleft$ and j is at the first position in π or the value left of it is bigger than j set $o_j \leftarrow \triangleright$, or if $o_j = \triangleright$ and j is at the last position in π or the value right of it is bigger than j set $o_j \leftarrow \triangleleft$, and in both cases set $s_j \leftarrow s_{j-1}$ and $s_{j-1} \leftarrow j-1$. Go back to M2.

Theorem 29. *For any zigzag language of permutations $L_n \subseteq S_n$, $n \ge 1$, Algorithm M visits every rectangulation from L_n exactly once, in the order $J(L_n)$ defined by* (2).

The rest of this section is devoted to proving Theorem 29.

For any π in the sequence $J(L_n)$ we define a sequence $s_n^{\pi} = (s_{n,1}^{\pi}, \ldots, s_{n,n}^{\pi})$ as follows: If $n = 1$ we have $J(L_1) = \pi$ with $\pi := 1$ and we define $s_1^{\pi} = (1)$. If $n \ge 2$, we consider the permutation $\pi' := p(\pi) \in S_{n-1}$ in the sequence $J(L_{n-1})$, and we define $\overline{c}(\pi') := \overleftarrow{c}(\pi')$ if π' appears at an odd position in $J(L_{n-1})$, or $\overline{c}(\pi') := \overrightarrow{c}(\pi')$ if π' appears at an even position. If π

is not the last permutation in $\overline{c}(\pi')$ we define

$$s^{\pi}_{n,i} := \begin{cases} s^{\pi'}_{n-1,i} & \text{if } i \le n-1, \quad (4a) \\ n & \text{if } i = n, \quad (4b) \end{cases}$$

for $i = 1, \ldots, n$, and if π is the last permutation in $\overline{c}(\pi')$ we define

$$s^{\pi}_{n,i} := \begin{cases} s^{\pi'}_{n-1,i} & \text{if } i \le n-2, \quad (5a) \\ n-1 & \text{if } i = n-1, \quad (5b) \\ s^{\pi'}_{n-1,n-1} & \text{if } i = n, \quad (5c) \end{cases}$$

for $i = 1, \ldots, n$.

The following lemma captures important properties of the sequences defined in this way.

Lemma 30. *The sequences defined in* (4) *and* (5) *have the following properties.*

(a) For the first permutation $\pi = \mathrm{id}_n$ in the sequence $J(L_n)$, we have $s^{\pi}_n = (1, 2, \ldots, n)$.
(b) For any two consecutive permutations π, ρ in the sequence $J(L_n)$, ρ is obtained from π by a jump of the value $s^{\pi}_{n,n}$.
(c) For the last permutation π in $J(L_n)$ we have $s^{\pi}_{n,n} = 1$.

Moreover, for any three consecutive permutations π, ρ, σ in $J(L_n)$ we have:

(c) If π and ρ differ in a jump of n, and ρ and σ differ in a jump of n, then we have $s^{\rho}_{n,i} = s^{\pi}_{n,i}$ for $i = 1, \ldots, n-1$.
(d) If π and ρ differ in a jump of n, and ρ and σ differ in a jump of $j < n$, then we have $s^{\rho}_{n,i} = s^{\pi}_{n,i}$ for $i \in \{1, \ldots, n-2\}$ and $s^{\rho}_{n,n-1} = n-1$.
(e) If π and ρ differ in a jump of $j < n$, ρ and σ differ in a jump of n, and j is not at a boundary position in $p^{n-j}(\rho)$, then we have $s^{\rho}_{n,i} = s^{\pi}_{n,i}$ for $i = 1, \ldots, n-1$.
(f) If π and ρ differ in a jump of $j < n$, ρ and σ differ in a jump of n, and j is at a boundary position in $p^{n-j}(\rho)$, then we have $s^{\rho}_{n,i} = s^{\pi}_{n,i}$ for $i \in \{1, \ldots, n-1\} \setminus \{j-1, j\}$, $s^{\rho}_{n,j-1} = j-1$ and $s^{\rho}_{n,j} = s^{\pi}_{n,j-1}$.

Proof. We prove these properties by induction on n. The induction basis $n = 1$ is trivial. For the induction step let $n \ge 2$ and assume that all properties hold for the sequence $J(L_{n-1})$.

We first show the induction step for (a), (b) and (c).

Consider a permutation π in $J(L_n)$ and define $\pi' := p(\pi) \in S_{n-1}$.

To prove (a), let $\pi = \mathrm{id}_n$ be the first permutation in $J(L_n)$ (recall Lemma 24 (a)). We have $\pi' = \mathrm{id}_{n-1}$ and by induction and (a) we hence have $s^{\pi'}_n = (1, 2, \ldots, n-1)$. Using (4) we obtain $s^{\pi}_n = (1, 2, \ldots, n)$, as claimed.

To prove (b), let π be a permutation that is not the last in the sequence $J(L_n)$, and let ρ be the permutation succeeding π in $J(L_n)$. If π is not the last permutation in the subsequence $\overline{c}(\pi')$ of $J(L_n)$, then by (2) the permutation ρ is obtained from π by a jump of the value n, and then (b) follows directly from (4b). On the other hand, if π is the last permutation in the subsequence $\overline{c}(\pi')$, then ρ is obtained from π by a jump of the value $s^{\pi'}_{n-1,n-1}$ by induction and (b), and by (5c) we have $s^{\pi}_{n,n} = s^{\pi'}_{n-1,n-1}$, as desired.

To prove (c), let π be the last permutation in $J(L_n)$. Then the permutation π' is also the last permutation in $J(L_{n-1})$, so by induction we have $s^{\pi'}_{n-1,n-1} = 1$. Using (5c) we see that $s^{\pi}_{n,n} = s^{\pi'}_{n-1,n-1} = 1$, as desired.

To prove (d), note that $p(\pi) = p(\rho) = p(\sigma)$ and therefore ρ is not the last permutation in the subsequence $\overline{c}(p(\rho))$, so the claim follows directly from (4a).

To prove (e), note that $p(\pi) = p(\rho) \neq p(\sigma)$ and therefore ρ is the last permutation in the subsequence $\overline{c}(p(\rho))$, so the claim follows directly from (4a), (5a) and (5b).

To prove (f) and (g), let $\pi' := p(\pi)$, $\rho' := p(\rho)$ and $\sigma' := p(\sigma)$. Note that $\pi' \neq \rho' = \sigma'$ and therefore π is the last permutation in the subsequence $\overline{c}(\pi')$, whereas ρ is the first permutation in the subsequence $\overline{c}(\rho')$. Consequently, s_n^π is defined by (5) and s_n^ρ is defined by (4). In particular, we have $s_{n,n-1}^\pi = n-1$ by (5b) and $s_{n,n-1}^\rho = s_{n-1,n-1}^{\rho'}$ by (4a).

We first prove (f), and we distinguish whether $j = n-1$ or $j < n-1$. If $j = n-1$, then π' and ρ' differ in a jump of $j = n-1$, and as j is not at a boundary position in ρ', ρ' and σ' also differ in a jump of $n-1$ by (2). Consequently, we have $s_{n-1,i}^{\rho'} = s_{n-1,i}^{\pi'}$ for $i = 1, \ldots, n-2$ by induction and (d), and $s_{n-1,n-1}^{\rho'} = n-1$ by (4b), so (f) indeed holds in this case.

If $j < n-1$, then π' and ρ' differ in a jump of $j < n-1$, and ρ' and σ' differ in a jump of $n-1$ by (2). As j is not at a boundary position in $p^{n-j}(\rho) = p^{n-1-j}(\rho')$, we have $s_{n-1,i}^{\rho'} = s_{n-1,i}^{\pi'}$ for $i = 1, \ldots, n-2$ by induction and (f), and $s_{n-1,n-1}^{\rho'} = n-1$ by induction and (b), so (f) indeed holds in this case.

We now prove (g), and again we distinguish whether $j = n-1$ or $j < n-1$. If $j = n-1$, then π' and ρ' differ in a jump of $j = n-1$, in particular $p(\pi') = p(\rho')$, and as $j = n-1$ is at a boundary position in $p^{n-j}(\rho) = p(\rho) = \rho'$, ρ' and σ' differ in a jump of some value smaller than $n-1$ by (2). Consequently, we have $s_{n-1,i}^{\rho'} = s_{n-1,i}^{\pi'}$ for $i = 1, \ldots, n-3$ and $s_{n-1,n-2}^{\rho'} = n-2$ by induction and (e). Moreover, we have $s_{n-1,n-1}^{\rho'} = s_{n-2,n-2}^{p(\rho')} = s_{n-2,n-2}^{p(\pi')} = s_{n-1,n-2}^{\pi'}$ by (4a). Combining these observations shows that indeed (g) holds in this case.

If $j < n-1$, then π' and ρ' differ in a jump of $j < n-1$, and ρ' and σ' differ in a jump of $n-1$ by (2). Clearly, j is at a boundary position in $p^{n-j}(\rho) = p^{n-1-j}(\rho')$, and by induction and (g) we have $s_{n-1,i}^{\rho'} = s_{n-1,i}^{\pi'}$ for $i \in \{1, \ldots, n-2\} \setminus \{j-1, j\}$, $s_{n-1,j-1}^{\rho'} = j-1$ and $s_{n-1,j}^{\rho'} = s_{n-1,j-1}^{\pi'}$. Moreover, we have $s_{n-1,n-1}^{\rho'} = n-1$ by (4a). Combining these observations proves (g) in this last case. □

Proof of Theorem 29. We establish the following invariants about the permutation π visited in line M2 of the algorithm:

(A) For all $j = 2, \ldots, n$, the direction of the next jump of the value j after the permutation π in $J(L_n)$ is left if $o_j = \triangleleft$ and right if $o_j = \triangleright$.
(B) The values in the array $s = (s_1, \ldots, s_n)$ satisfy $s = s_n^\pi$ with s_n^π as defined in (4) and (5).

By Lemma 24 (a), the identity permutation $\pi := \mathrm{id}_n$ is the first permutation in the sequence $J(L_n)$. Moreover, by the initialization of π in line M1, $\pi = \mathrm{id}_n$ is also the first permutation visited in line M2. Combining this with the above invariants, we obtain by induction on the length of $J(L_n)$ that after visiting a permutation $\pi \in L_n$, the next permutation visited by Algorithm M is the permutation that succeeds π in $J(L_n)$. Indeed, by the instructions in line M3 and M4, the next permutation ρ visited by the algorithm is obtained from π by a jump of the value $j := s_n$ that is minimal w.r.t. L_n, and the jump direction is left if $o_j = \triangleleft$ and right if $o_j = \triangleright$. Applying (B) and Lemma 30 (b), and (A) and Lemma 24 (c), we obtain that ρ is indeed the permutation that succeeds π in $J(L_n)$. Also, the algorithm terminates correctly after visiting the last permutation in the sequence $J(L_n)$ by the condition in line M3 and Lemma 30 (c).

We prove (A)+(B) by double induction on n and the number of iterations of Algorithm M. The induction basis $n = 1$ is trivial. For the induction step, let $n \geq 2$, and assume that the invariants hold for the zigzag language $L_{n-1} = \{p(\pi) \mid \pi \in L_n\}$. We first verify that (A)+(B) hold in line M2 during the first iteration of the algorithm when $\pi = \mathrm{id}_n$. By line M1 we

have $o_j = \triangleleft$ for $j = 2, \dots, n$, so (A) is satisfied by Lemma 24 (b). By line M1 we also have $s = (s_1, \dots, s_n) = (1, \dots, n)$, which equals $s_n^\pi = s_n^{\mathrm{id}_n}$ by Lemma 30 (a).

For the induction step, consider three consecutive permutations $\hat\pi, \hat\rho, \hat\sigma$ in the sequence $J(L_n)$, and suppose that (A)+(B) are satisfied when Algorithm M visits $\pi = \hat\pi$. We need to verify that (A)+(B) still hold after one iteration through lines M2—M5, after which the algorithm visits $\pi = \hat\rho$ by the instructions in lines M3 and M4, as argued before.

Case (i): We first consider the case that $\hat\pi, \hat\rho$ satisfy $p(\hat\pi) = p(\hat\rho) =: \hat\pi' \in L_{n-1}$ and therefore both are contained in the subsequence $\overline{c}(\hat\pi')$ of $J(L_n)$. We only treat the case $\overline{c}(\hat\pi') = \overleftarrow{c}(\hat\pi')$, as the other case $\overline{c}(\hat\pi') = \overrightarrow{c}(\hat\pi')$ is symmetric. In this case $\hat\rho$ is obtained from $\hat\pi$ by a left jump of the value n. In particular, the variable j has the value $j = s_n = n$ and $o_n = \triangleleft$ in this iteration of the algorithm.

Case (ia): The value n is not at the first position in $\hat\rho$. Then by (2) the permutation $\hat\sigma$ is obtained from $\hat\rho$ by another left jump of the value n. In line M5, the value of s_n is set to n, which was the previous value, but none of the conditions in line M5 holds for $\pi = \hat\rho$, so overall none of the arrays s and o is modified. We conclude that (A) holds after this iteration for $\pi = \hat\rho$. Moreover, (B) holds by Lemma 30 (c) and (4b).

Case (ib): The value n is at the first position in $\hat\rho$, i.e., we have $\hat\rho = c_1(\hat\pi') = n\,\hat\pi'$. Then by (2) we have $\hat\sigma = c_1(\hat\rho') = n\,\hat\rho'$ where $\hat\rho' \in L_{n-1}$ succeeds $\hat\pi'$ in the sequence $J(L_{n-1})$. In line M5, the value of s_n is set to n, which is the same as the previous value, but since the first if-condition is satisfied, s_n is then overwritten by s_{n-1}, and s_{n-1} is set to $n-1$. Consequently, the new values are $s_{n-1} = n-1$ and $s_n = s^{\hat\pi}_{n,n-1} = s^{\hat\pi'}_{n-1,n-1}$ by induction and (B) and (4a). Moreover, the value of o_n is flipped to $o_n = \triangleright$. Using Lemma 24 (e), we conclude that (A) holds after this iteration for $\pi = \hat\rho$. Applying Lemma 30 (d) and using that $s^\pi_{n,n} = s^{\hat\pi'}_{n-1,n-1}$ by (5c), we obtain that (B) holds as well.

Case (ii): It remains to consider the case that $p(\hat\pi) \neq p(\hat\rho)$, i.e., both permutations have n at the first or last position. By symmetry, it suffices to consider the case that n is at the first position, i.e., $\hat\pi = c_1(\hat\pi') = n\,\hat\pi'$ and $\hat\rho = c_1(\hat\rho') = n\,\hat\rho'$, where $\hat\pi'$ and $\hat\rho'$ are consecutive permutations in $J(L_{n-1})$. They differ in a jump of the value $j := s_n < n$ by Lemma 30 (b) and (B). Then by (2), the permutation $\hat\sigma$ is obtained from $\hat\rho$ by a right jump of n. We proceed to show that (A) and (B) hold for $\pi = \hat\rho$, and for this we distinguish subcases.

Case (iia): The value j is not at a boundary position in $p^{n-j}(\hat\rho)$. Then by Lemma 24 (d) in $\hat\rho$ the value j is surrounded by the same values as in $p^{n-j}(\hat\rho)$, both smaller than j. In line M5, the value of s_n is set to n, and no other entries of s and o are modified. Using Lemma 24 (e), we conclude that (A) holds after this iteration for $\pi = \hat\rho$. Moreover, (B) holds by Lemma 30 (e), also using that $s^{\hat\rho}_{n,n} = n$ by (4b).

Case (iib): The value j is at a boundary position in $p^{n-j}(\hat\rho)$, and the value k next to it is smaller than j. Then by Lemma 24 (d) in $\hat\rho$ the value k is also next to j, and either j is at a boundary position (the right boundary, as n is at the first position in $\hat\rho$) or the other value next to in the direction o_j of the jump is bigger than j. In line M5, the value of s_n is set to n, the value of s_j is set to s_{j-1}, and s_{j-1} is set to $j-1$. Moreover, the value of o_j is flipped. Using Lemma 24 (e), we conclude that (A) holds after this iteration for $\pi = \hat\rho$. Moreover, (B) holds by Lemma 30 (f), also using that $s^{\hat\rho}_{n,n} = n$ by (4b). This completes the proof of the theorem. □

8.8. **Proof of Theorem 8.**

Proof of Theorem 8. In the proof of Theorem 5 we showed that the ordering of rectangulations generated by Algorithm J$^\square$ is given by (3) for some zigzag language L_n of 2-clumped permutations. The theorem hence follows by applying Theorem 29. □

9. S-equivalence of rectangulations

Recall that R-equivalence is the equivalence relation on $\mathcal{R}_n$ obtained from wall slides, i.e., any two generic rectangulations that differ in a sequence of wall slides are equivalent. It is well known that every equivalence class contains exactly one diagonal rectangulation, i.e., $\mathcal{D}_n$ is a set of representatives for R-equivalence (see e.g. [CSS18]).

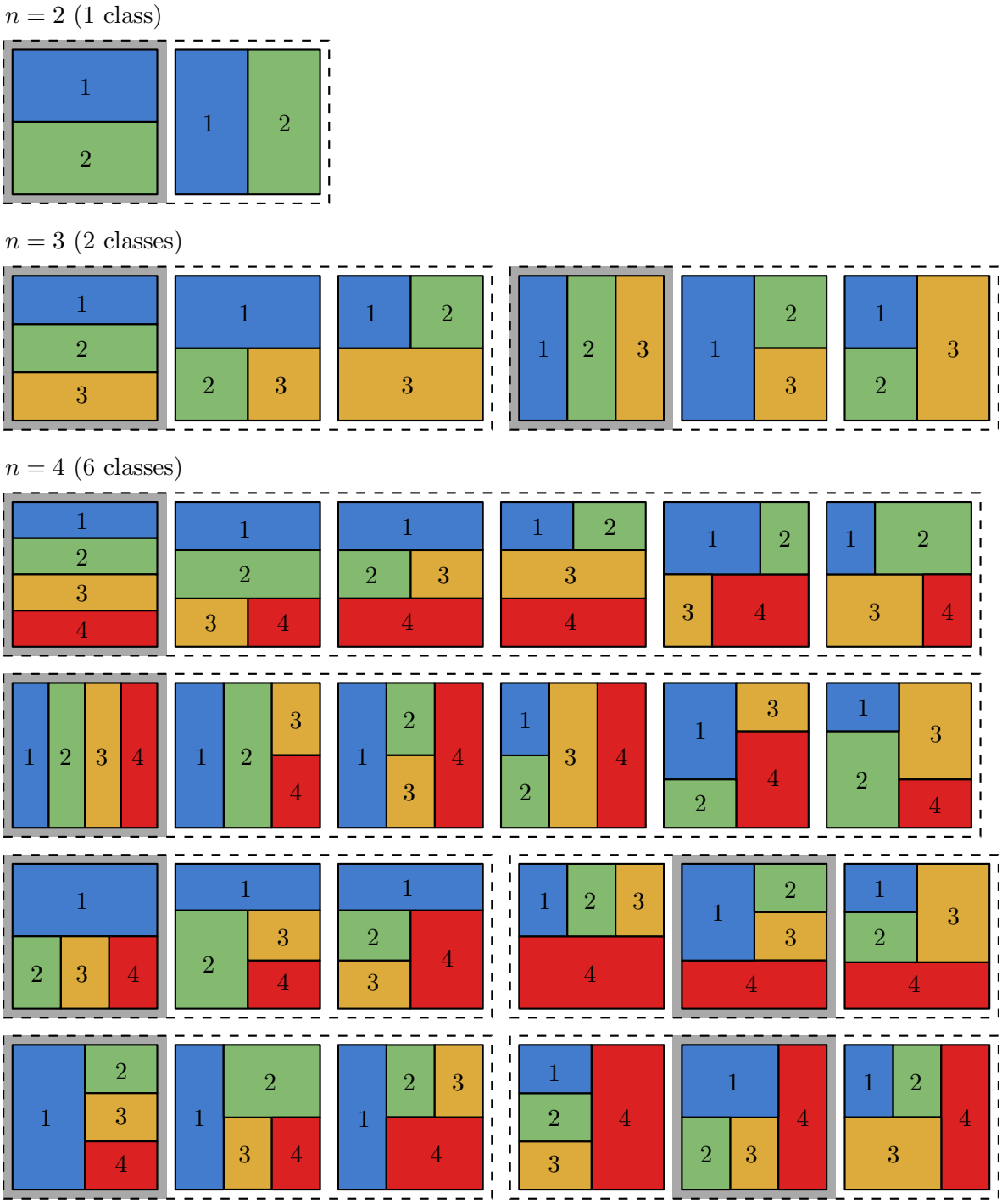


FIGURE 23. Equivalence classes of generic rectangulations under S-equivalence for $n = 2, 3, 4$. Block-aligned rectangulations as equivalence class representatives are highlighted.

We aim to do something analogous for S-equivalence, and to pick a suitable set of representatives for our generation algorithm. Recall that S-equivalence is the equivalence relation on $\mathcal{R}_n$ obtained from wall slides and simple flips, i.e., any two generic rectangulations that differ in a sequence of wall slides or simple flips are equivalent. Figure 23 shows the equivalence classes of generic rectangulations under S-equivalence for $n = 2, 3, 4$. Unfortunately, one can check that there is no choice of representatives $\mathcal{P}_2 \subseteq \mathcal{R}_2$, $\mathcal{P}_3 \subseteq \mathcal{R}_3$, $\mathcal{P}_4 \subseteq \mathcal{R}_4$ for those equivalence classes (i.e., $|\mathcal{P}_2| = 1$, $|\mathcal{P}_3| = 2$, $|\mathcal{P}_3| = 6$) that is consistent with the operations of rectangle deletion and insertion, i.e., such that $\mathcal{P}_2 = \{p(R) \mid R \in \mathcal{P}_3\}$ and $\mathcal{P}_3 = \{p(R) \mid R \in \mathcal{P}_4\}$. Consequently, for S-equivalence there is no set of unique representatives, one for each equivalence class, that would form a zigzag set, so our generation algorithms cannot be applied directly. However, we will show how to choose representatives for each equivalence class (highlighted in Figure 23), such that those representatives *and* the rectangulations obtained from them by a simple flip of the rectangle r_n admit a generation tree approach with our algorithms.

9.1. **Representatives for S-equivalence.** By definition, S-equivalence is a coarsening of R-equivalence, and we will therefore choose a subset of diagonal rectangulations as representatives.

We start with some definitions; see Figure 24. A rectangulation is *horizontally aligned*, or *H-aligned* for short, if all of its walls are horizontal. Moreover, a rectangulation is *almost horizontally aligned*, or *AH-aligned* for short, if all of its walls except one at the bottom are horizontal. Equivalently, it is obtained by gluing copies of $\square$ on top of $\boxplus$.

Similarly, a rectangulation is *vertically aligned*, or *V-aligned* for short, if all of its walls are vertical. Moreover, a rectangulation is *almost vertically aligned*, or *AV-aligned* for short, if all of its walls except one at the right are vertical. Equivalently, it is obtained by gluing copies of $\square$ on the left of $\boxminus$.

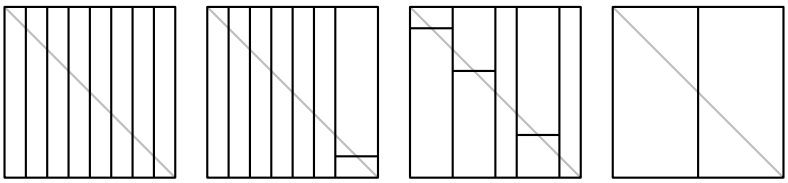


FIGURE 24. Illustration of aligned rectangulations. Rectangulation (a) is V-aligned, (b) is AV-aligned, (c) is V-alignable but neither V-aligned nor AV-aligned, (d) is V-aligned and AH-aligned.

The rectangulation $\square$ is H-aligned and V-aligned. The rectangulation $\boxminus$ is H-aligned and AV-aligned, and the rectangulation $\boxplus$ is V-aligned and AH-aligned.

A rectangulation is *H- or V-alignable*, if we can apply a sequence of simple flips to make it *H- or V-aligned*, respectively. Clearly, a rectangulation is H-alignable if it is obtained by vertically gluing together copies of $\square$ and $\boxplus$, and it is V-alignable if it is obtained by horizontally gluing together copies of $\square$ and $\boxminus$.

A *block* in a rectangulation is a subset of rectangles whose union is a rectangle. The *size* of a block is the number of rectangles of the block.

Lemma 31. *Every diagonal rectangulation can be partitioned uniquely into maximal alignable blocks.*

Proof. Suppose for the sake of contradiction that for some rectangulation $R \in \mathcal{D}_n$ there were two distinct block partitions P, P' of R. Consider a block B in P that is not a block in P'. Consider

one of the rectangles in B, and consider the block B' of P' containing this rectangle. As R does not have any points where 4 rectangles meet and $B' \neq B$, the block B' must be a proper subset or superset of B, contradicting the maximal choice of the blocks. □

Lemma 31 holds more generally for generic rectangulations and for maximal blocks with any additional property (such as alignable), but this is not needed here.

Lemma 32. *For any diagonal rectangulation, the partition into maximal alignable blocks is invariant under simple flips.*

Proof. Consider a wall that can be simple-flipped, and observe that the two rectangles to both sides of the wall must belong to the same alignable block due to the maximal choice of the blocks. □

From now on, whenever we refer to a block in a rectangulation, we mean a maximal alignable block. A block is a *base block*, if it contains the bottom boundary of the rectangulation.

Based on the partition of a diagonal rectangulation $R \in \mathcal{D}_n$ into blocks, which is unique by Lemma 31, we introduce the following definitions; see Figure 25. We refer to each block of R as an *H-block* or *V-block*, if it is H-alignable or V-alignable, respectively. We consider an H-block B of size at least 2 with rectangle r_i at the bottom-right. If $i = n$ or if rectangle r_{i+1} of R is below r_i we say that B is *free*, whereas if r_{i+1} is right of r_i we say that B is *locked*. Similarly, we consider a V-block B of size at least 2 with rectangle r_i at the bottom-right. If $i = n$ or if rectangle r_{i+1} of R is right of r_i we say that B is *free*, whereas if r_{i+1} is below r_i we say that B is *locked*.

We say that $R \in \mathcal{D}_n$ is *block-aligned* if for every block B of size at least 2 in R the following conditions hold: if B is a free H-block then B is H-aligned, if B is a free V-block then B is V-aligned, if B is a locked H-block then B is AH-aligned, and if B is a locked V-block then B is AV-aligned. A special rule applies if the rectangle r_n is contained in a block of size 2 (which is free and both H-alignable and V-alignable), and then we require this block to be V-aligned, unless it is a base block, in which case it must be H-aligned. Note that a block B of size exactly 2 that does not contain r_n is both an H-block and a V-block, however, if B is a locked/free H-block then B is a free/locked V-block, respectively, so this definition is consistent (as AH-aligned equals V-aligned and H-aligned equals AV-aligned for a block of size 2).

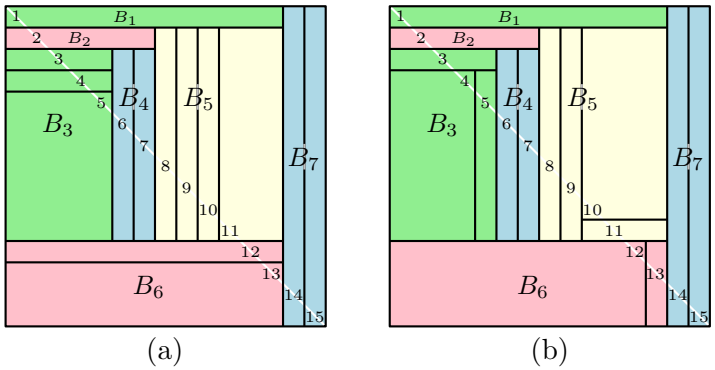


FIGURE 25. Illustration of blocks and block-aligned rectangulations. Blocks are highlighted by the same shading. H-blocks are B_1, B_2, B_3, B_4, B_6 and B_7, with B_7 free and B_3, B_4 and B_6 locked. V-blocks are B_1, B_2, B_4, B_5, B_6 and B_7, with B_4, B_6 and B_7 free and B_5 locked. Rectangulation (a) is not block-aligned, whereas (b) is block-aligned.

We write $\mathcal{B}_n \subseteq \mathcal{D}_n$ for the set of diagonal rectangulations that are block-aligned. We partition this set into $\mathcal{B}_n^{\square}$ and $\mathcal{B}_n^{\boxplus}$, respectively, according to whether the rectangle r_n is contained in a block of size 1 or at least 2, respectively. Note that if $R \in \mathcal{B}_n^{\square}$, then the wall between r_n and r_{n-1} does not admit a simple flip, whereas if $R \in \mathcal{B}_n^{\boxplus}$, then this wall admits a simple flip. For any $R \in \mathcal{B}_n^{\boxplus}$ we write $s(R) \in \mathcal{B}_n^{\boxplus}$ for the rectangulation obtained from r_n by a simple flip of this wall. The set $\mathcal{B}_n^{\boxplus}$ is partitioned into $\mathcal{B}_n^{\boxminus}$ and $\mathcal{B}_n^{\boxbar}$ according to whether this wall is horizontal or vertical, respectively.

As a consequence of Lemma 32, every equivalence class of generic rectangulations under S-equivalence contains exactly one block-aligned diagonal rectangulation; see Figure 23. Consequently, we will use the block-aligned rectangulations $\mathcal{B}_n \subseteq \mathcal{D}_n$ as representatives for S-equivalence.

9.2. **Insertion in block-aligned rectangulations.** The next two lemmas describe how to construct block-aligned rectangulations by rectangle insertion; see Figure 27.

For any diagonal rectangulation $R \in \mathcal{D}_n$, we let $I_v(R)$ denote the subsequence of $I(R)$ of the first insertion point of each vertical group. Similarly, we let $I_h(R)$ denote the subsequence of $I(R)$ of the last insertion point of each horizontal group.

Lemma 33. *Let $P \in \mathcal{B}_{n-1}^{\square}$, $I_v(P) =: (q_{i_1}, \ldots, q_{i_\lambda})$ and $I_h(P) =: (q_{j_1}, \ldots, q_{j_\mu})$. Then we have the following:*

- *For any $1 \leq k < \lambda$ we have $c_{i_k}(P) \in \mathcal{B}_n^{\square}$, and every $R \in \mathcal{B}_n^{\square}$ for which the top-left vertex of r_n has type $\vdash$ and r_{n-1} forms its own block is obtained by insertion from some $P \in \mathcal{B}_{n-1}^{\square}$ in this way.*
- *For any $1 < k \leq \mu$ we have $c_{j_k}(P) \in \mathcal{B}_n^{\square}$, and every $R \in \mathcal{B}_n^{\square}$ for which the top-left vertex of r_n has type $\top$ and r_{n-1} forms its own block is obtained by insertion from some $P \in \mathcal{B}_{n-1}^{\square}$ in this way.*
- *If $\lambda > 1$ we have $c_{j_1}(P) \in \mathcal{B}_n^{\boxbar}$, and every $R \in \mathcal{B}_n^{\boxbar}$ for which r_{n-1} and r_n form a V-aligned block of size 2 is obtained by insertion from some $P \in \mathcal{B}_{n-1}^{\square}$ in this way.*
- *If $\lambda = 1$ we have $c_{i_1}(P) \in \mathcal{B}_n^{\boxminus}$, and every $R \in \mathcal{B}_n^{\boxminus}$ for which r_{n-1} and r_n form an H-aligned base block of size 2 is obtained by insertion from some $P \in \mathcal{B}_{n-1}^{\square}$ in this way.*

Proof. The first and second part of the lemma are symmetric, so it suffices to prove the first one. For this we analyze how the blocks of $R := c_{i_k}(P)$ differ from the blocks of P, and prove that they are all aligned as required.

The rectangle r_{n-1} forms a block of size 1 in P, and as $k < \lambda$ this is also true in R. Similarly, as $k < \lambda$ the rectangle r_n forms a block of size 1 in R. Consequently, we only need to verify whether blocks of R not containing r_{n-1} or r_n in P are aligned as required. If $k = 1$, then the blocks of R are the same as those of P, plus the block containing r_n, so we are done; see Figure 26 (a). If $k > 1$, we let r_a and r_b be the rectangles in P to the left and right of the edge that contains the insertion point q_{i_k}. If r_a and r_b belong to two distinct blocks in P, then the blocks of R are the same as those of P, plus the block containing r_n, so we are done; see Figure 26 (b). On the other hand, if r_a and r_b belong to the same block B in P, then it must be a free V-block that is V-aligned or a locked H-block that AH-aligned, and we have $b = a + 1$. If B is a free V-block in P, then in R this block is split into two free V-blocks B' and B'', one containing r_a and the other one containing $r_b = r_{a+1}$, and both B' and B'' are V-aligned; see Figure 26 (c). If B is a locked H-block in P, then in R this block is split into the H-block $B \setminus \{r_a, r_{a+1}\}$, and two blocks of size 1 containing r_a or r_{a+1}, respectively; see Figure 26 (d). Moreover, if $|B| = 3$ then $|B \setminus \{r_a, r_{a+1}\}| = 1$, and otherwise $B \setminus \{r_a, r_{a+1}\}$ is a free H-block that is H-aligned in R. In all cases we obtain $R \in \mathcal{B}_n^{\square}$, as claimed.

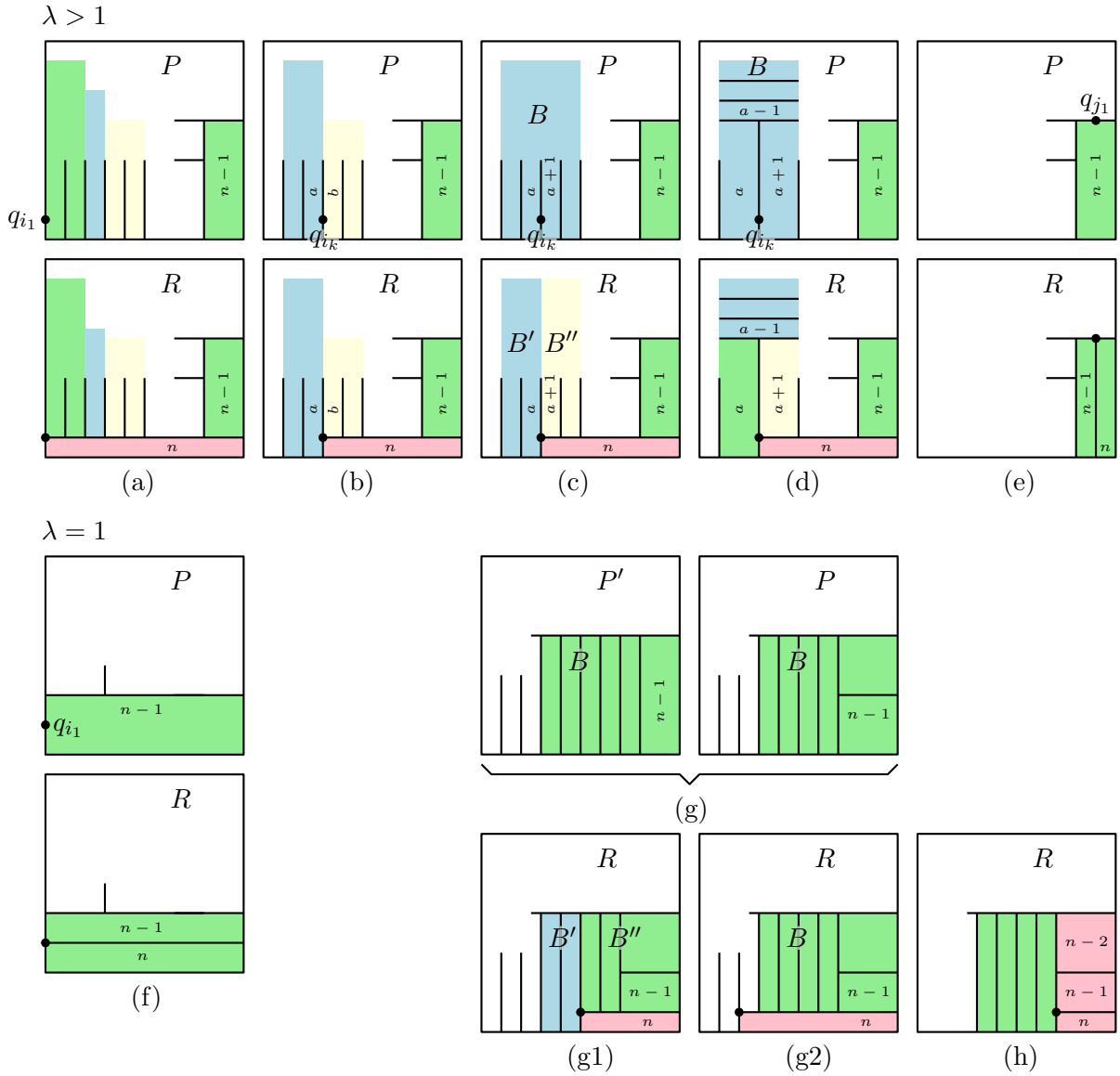


FIGURE 26. Illustration of the proofs of Lemmas 33 and 34.

We continue to prove the third part of the lemma about the rectangulation $R := c_{j_1}(P)$. The rectangle r_{n-1} forms a block of size 1 in P, and together with r_n it forms a block of size 2 in R; see Figure 26 (e). This block is V-aligned in R, so we have $R \in \mathcal{B}_n^{\boxplus}$, as claimed.

It remains to prove the fourth part of the lemma about the rectangulation $R := c_{i_1}(P)$. The rectangle r_{n-1} forms a block of size 1 in P, and together with r_n it forms a block of size 2 in R; see Figure 26 (f). This block is a base block and H-aligned in R, so we have $R \in \mathcal{B}_n^{\boxminus}$, as claimed. □

Lemma 34. *Let $P \in \mathcal{B}_{n-1}^{\boxminus}$ and $P' := s(P)$, or let $P' \in \mathcal{B}_{n-1}^{\boxplus}$ and $P := s(P')$, and define $I_v(P) =: (q_{i_1}, \ldots, q_{i_\lambda})$ and $I_h(P') =: (q_{j_1}, \ldots, q_{j_\mu})$. Then we have the following:*

- *For any $1 \le k < \lambda$ we have $c_{i_k}(P) \in \mathcal{B}_n^{\square}$, and every $R \in \mathcal{B}_n^{\square}$ for which the top-left vertex of r_n has type $\vdash$ and r_{n-1} is contained in a block of size at least 2 is obtained by insertion from some $P \in \mathcal{B}_{n-1}^{\boxminus}$ in this way.*

- *For any $1 < k \le \mu$ we have $c_{j_k}(P') \in \mathcal{B}_n^{\square}$, and every $R \in \mathcal{B}_n^{\square}$ for which the top-left vertex of r_n has type $\top$ and r_{n-1} is contained in a block of size at least 2 is obtained by insertion from some $P' \in \mathcal{B}_{n-1}^{\boxplus}$ in this way.*
- *We have $c_{i_\lambda}(P) \in \mathcal{B}_n^{\boxminus}$, and every $R \in \mathcal{B}_n^{\boxminus}$ for which r_{n-1} and r_n are contained in an H-aligned block of size at least 3 is obtained by insertion from some $P \in \mathcal{B}_{n-1}^{\boxminus}$ in this way.*
- *We have $c_{j_1}(P') \in \mathcal{B}_n^{\boxplus}$, and every $R \in \mathcal{B}_n^{\boxplus}$ for which r_{n-1} and r_n are contained in a V-aligned block of size at least 3 is obtained by insertion from some $P' \in \mathcal{B}_{n-1}^{\boxplus}$ in this way.*

Proof. The proof for the first part in the case $P \in \mathcal{B}_{n-1}^{\boxminus}$ and for the second part in the case $P' \in \mathcal{B}_{n-1}^{\boxplus}$ is analogous to the proof of Lemma 33. Therefore, by symmetry, to complete the proof of the first two parts, it suffices to argue about the case $P' \in \mathcal{B}_{n-1}^{\boxplus}$, $P := s(P')$ and the rectangulation $R := c_{i_k}(P)$ for $1 \le k < \lambda$; see Figure 26 (g). The V-block B in P' containing r_{n-1}, which is free and V-aligned in P', is AV-aligned and free in P. Consequently, in R the block B is either split into two blocks, a free V-block B' that is V-aligned to the left of a locked V-block B'' (Figure 26 (g1)) that is AV-aligned, or B remains a single locked V-block that is AV-aligned in R (Figure 26 (g2)), where the locking is due to the insertion of r_n. The remaining blocks of P' are treated as in the proof of Lemma 33. In all cases we obtain that $R \in \mathcal{B}_n^{\square}$, as claimed.

The third and fourth part of the lemma are symmetric, so it suffices to prove the third one about the rectangulation $R := c_{i_\lambda}(P)$. In this case $\{r_{n-2}, r_{n-1}, r_n\}$ is an H-block that is free and H-aligned in R, and either $|B \setminus \{r_{n-2}, r_{n-1}\}| = 1$ or $B \setminus \{r_{n-2}, r_{n-1}\}$ is a V-block that is free and V-aligned in R; see Figure 26 (h). It follows that $R \in \mathcal{B}_n^{\boxminus}$, as claimed. □

9.3. **Tree of block-aligned rectangulations.** By Lemmas 33 and 34, all block-aligned rectangulations $\mathcal{B}_n$ can be obtained by suitable rectangle insertions into all block-aligned rectangulations $\mathcal{B}_{n-1}$ and $s(\mathcal{B}_{n-1}^{\boxplus})$. We consider the subtree of the tree of rectangulations discussed in Section 3.4 induced by the rectangulations $\mathcal{B}_n$ and $s(\mathcal{B}_n^{\boxplus})$ for all $n \ge 1$. By gluing together pairs of nodes $(R, s(R))$ for all $R \in \mathcal{B}_n^{\boxplus}$, we obtain the tree shown in Figure 27.

For any $P \in \mathcal{B}_{n-1}^{\square}$, using the notation from Lemma 33 we define

$$c(P) := \begin{cases} \big(c_{i_1}(P), \dots, c_{i_{\lambda-1}}(P), c_{j_1}(P), c_{j_2}(P), \dots, c_{j_\mu}(P)\big) & \text{if } \lambda > 1, \\ \big(c_{i_1}(P), c_{j_2}(P), \dots, c_{j_\mu}(P)\big) & \text{if } \lambda = 1. \end{cases} \tag{6a}$$

For any $P \in \mathcal{B}_{n-1}^{\boxminus} \cup s(\mathcal{B}_{n-1}^{\boxplus})$ and $P' := s(P)$, using the notation from Lemma 34 we define

$$c(P) := \big(c_{i_1}(P), \dots, c_{i_{\lambda-1}}(P), c_{i_\lambda}(P), c_{j_1}(P'), c_{j_2}(P'), \dots, c_{j_\mu}(P')\big). \tag{6b}$$

These sequences define an ordering among the children of each node in the aforementioned (unordered) tree of block-aligned rectangulations.

Note that any two consecutive rectangulations in the sequence (6a) differ in a T-flip, except $c_{i_{\lambda-1}}(P)$ and $c_{j_1}(P)$, and $c_{i_1}(P)$ and $c_{j_2}(P)$, which differ in a T-flip plus a simple flip. Similarly, any two consecutive rectangulations in the sequence (6b) differ in a T-flip, except $c_{i_\lambda}(P)$ and $c_{j_1}(P')$, which differ in a simultaneous flip of the two walls between r_n, r_{n-1} and r_{n-2}. We refer to this operation as a *D-flip* (D like 'double').

9.4. **Next oracle for block-aligned rectangulations.** Using (6), we may modify the minimal jump oracle $\texttt{next}_{\mathcal{D}_n}$ for diagonal rectangulations described in Section 7.2 for the generation of block-aligned rectangulations within Algorithm M$^{\square}$ as follows. Some Gray code orderings produced by this algorithm are shown in the appendix.

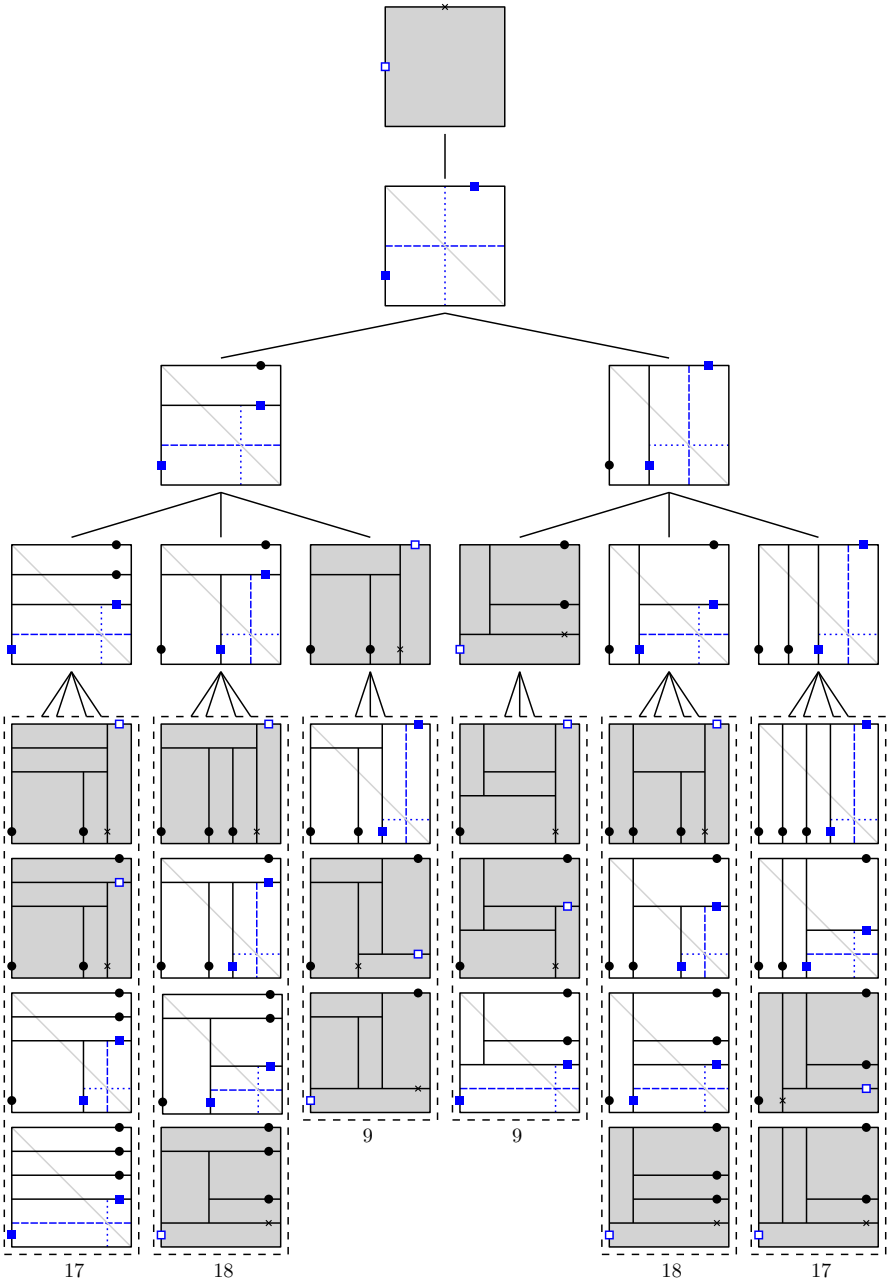


FIGURE 27. Tree of block-aligned rectangulations. Rectangulations from $\mathcal{B}_n^{\square}$ are drawn gray, and those from $\mathcal{B}_n^{\boxplus}$ are drawn white. For any $R \in \mathcal{B}_n^{\boxplus}$, the wall between rectangles r_n and r_{n-1} is drawn dashed, whereas the corresponding wall in $s(R)$ is drawn dotted. Each insertion point marked by a disk corresponds to one child of the current node as in the first two parts of Lemmas 33 and 34. Each insertion point marked by an empty square corresponds to one child as in the third or fourth part of Lemma 33, whereas crossed insertion points are not used. Each insertion point marked by a solid square corresponds to one child as in the third or fourth part of Lemma 34. The numbers at the bottom indicate the number of nodes in the next level of the tree, of which there are $2(17 + 18 + 9) = 88$ overall (i.e., we have $|\mathcal{B}_6| = 88$).

$\texttt{next}_{\mathcal{B}_n}(R, j, d)$ *(Next oracle for block-aligned rectangulations).*

N1. [Prepare] Set $a \leftarrow r_j.\texttt{nwest}$ and call $\texttt{unlock}(j, d)$. If $d = \lhd$ and $v_a.\texttt{type} = \top$, set $\alpha \leftarrow v_a.\texttt{south}$ and $b \leftarrow e_\alpha.\texttt{tail}$. If $v_b.\texttt{type} = \dashv$ goto N2, otherwise we have $v_b.\texttt{type} = \perp$ and goto N3. If $d = \rhd$ and $v_a.\texttt{type} = \top$, set $\alpha \leftarrow v_a.\texttt{east}$ and goto N4. If $d = \rhd$ and $v_a.\texttt{type} = \vdash$, set $\alpha \leftarrow v_a.\texttt{east}$ and $b \leftarrow e_\alpha.\texttt{head}$. If $v_b.\texttt{type} = \perp$ goto N5, otherwise we have $v_b.\texttt{type} = \dashv$ and goto N6. If $d = \lhd$ and $v_a.\texttt{type} = \vdash$, set $\alpha \leftarrow v_a.\texttt{south}$ and goto N7.

N2. [Horizontal left jump (T/TS)] Set $\gamma \leftarrow v_b.\texttt{west}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \lhd, \gamma)$. Then set $a \leftarrow r_j.\texttt{nwest}$, $\alpha \leftarrow v_a.\texttt{south}$, $b \leftarrow e_\alpha.\texttt{tail}$, $c \leftarrow r_{j-1}.\texttt{swest}$, $\gamma \leftarrow v_c.\texttt{north}$, $c' \leftarrow r_j.\texttt{seast}$ and if $v_b.\texttt{type} = \perp$ and $[v_{c'}.\texttt{type} = \dashv$ or $[j = n$ and $e_\gamma.\texttt{left} = 0]]$ call $\texttt{Sjump}(R, j, \lhd, \gamma)$. Call $\texttt{lock}(R, j, \rhd)$ and return.

N3. [Horizontal left jump (ST/D)] Set $c \leftarrow r_{j-1}.\texttt{swest}$ and $\gamma \leftarrow v_c.\texttt{north}$ and call $\texttt{Sjump}(R, j, \lhd, \gamma)$. Then set $\gamma \leftarrow v_c.\texttt{north}$, $k \leftarrow e_\gamma.\texttt{left}$, $c' \leftarrow r_k.\texttt{swest}$ and $\gamma' \leftarrow v_{c'}.\texttt{north}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \lhd, \gamma')$. Set $c \leftarrow r_{j-1}.\texttt{swest}$, $\gamma \leftarrow v_c.\texttt{north}$ and $a \leftarrow e_\gamma.\texttt{head}$. If $v_a.\texttt{type} = \top$ we have $k = j - 2$, set $c' \leftarrow r_{j-2}.\texttt{swest}$, $\gamma' \leftarrow v_{c'}.\texttt{north}$ and call $\texttt{Sjump}(R, j-1, \lhd, \gamma')$. Call $\texttt{lock}(R, j-1, \rhd)$ and return.

N4. [Horizontal right jump (T/TS)] Set $k \leftarrow e_\alpha.\texttt{left}$, $b \leftarrow r_k.\texttt{neast}$ and $\gamma \leftarrow v_b.\texttt{west}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \rhd, \gamma)$. Then set $a \leftarrow r_j.\texttt{nwest}$, $\alpha \leftarrow v_a.\texttt{south}$, $b \leftarrow e_\alpha.\texttt{tail}$, $\beta \leftarrow v_b.\texttt{south}$, $\gamma \leftarrow v_b.\texttt{west}$, $c \leftarrow e_\beta.\texttt{tail}$ and $c' \leftarrow e_\gamma.\texttt{tail}$, and if $v_c.\texttt{type} = \perp$ and $v_{c'}.\texttt{type} = \vdash$ we have $k = j - 2$, set $\gamma' \leftarrow v_a.\texttt{west}$ and call $\texttt{Sjump}(j-1, R, \rhd, \gamma')$. Call $\texttt{lock}(R, j, \triangle)$ and return.

N5. [Vertical right jump (T/TS)] Set $\gamma \leftarrow v_b.\texttt{north}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \rhd, \gamma)$. Then set $a \leftarrow r_j.\texttt{nwest}$, $\alpha \leftarrow v_a.\texttt{east}$, $b \leftarrow e_\alpha.\texttt{head}$, $c \leftarrow r_{j-1}.\texttt{neast}$, $\gamma \leftarrow v_c.\texttt{west}$, $c' \leftarrow r_j.\texttt{seast}$, $e \leftarrow r_{j-1}.\texttt{nwest}$ and if $v_b.\texttt{type} = \dashv$ and $[v_{c'}.\texttt{type} = \perp$ or $[j = n$ and not $[v_e.\texttt{type} = \vdash$ and $e_\gamma.\texttt{tail} = e]]$ call $\texttt{Sjump}(R, j, \rhd, \gamma)$. Call $\texttt{lock}(R, j, \triangle)$ and return.

N6. [Vertical right jump (ST/D)] Set $c \leftarrow r_{j-1}.\texttt{neast}$ and $\gamma \leftarrow v_c.\texttt{west}$ and call $\texttt{Sjump}(R, j, \rhd, \gamma)$. Then set $\gamma \leftarrow v_c.\texttt{west}$, $k \leftarrow e_\gamma.\texttt{left}$, $c' \leftarrow r_k.\texttt{neast}$ and $\gamma' \leftarrow v_{c'}.\texttt{west}$ and call $\texttt{Tjump}_\texttt{h}(R, j, \rhd, \gamma')$. Set $c \leftarrow r_{j-1}.\texttt{neast}$, $\gamma \leftarrow v_c.\texttt{west}$ and $a \leftarrow e_\gamma.\texttt{tail}$. If $v_a.\texttt{type} = \vdash$ we have $k = j - 2$, set $c' \leftarrow r_{j-2}.\texttt{neast}$, $\gamma' \leftarrow v_{c'}.\texttt{west}$ and call $\texttt{Sjump}(R, j-1, \rhd, \gamma')$. Call $\texttt{lock}(R, j-1, \triangle)$ and return.

N7. [Vertical left jump (T/TS)] Set $k \leftarrow e_\alpha.\texttt{left}$, $b \leftarrow r_k.\texttt{swest}$ and $\gamma \leftarrow v_b.\texttt{north}$ and call $\texttt{Tjump}_\texttt{v}(R, j, \lhd, \gamma)$. Then set $a \leftarrow r_j.\texttt{nwest}$, $\alpha \leftarrow v_a.\texttt{east}$, $b \leftarrow e_\alpha.\texttt{head}$, $\beta \leftarrow v_b.\texttt{east}$, $\gamma \leftarrow v_b.\texttt{north}$, $c \leftarrow e_\beta.\texttt{head}$ and $c' \leftarrow e_\gamma.\texttt{head}$, and if $v_c.\texttt{type} = \dashv$ and $v_{c'}.\texttt{type} = \top$ we have $k = j - 2$, set $\gamma' \leftarrow v_a.\texttt{north}$ and call $\texttt{Sjump}(j-1, R, \lhd, \gamma')$. Call $\texttt{lock}(R, j, \rhd)$ and return.

Lines N2–N4 are symmetric to lines N5–N7, so we only consider N2–N4; see the illustrations in Figure 28. Lines N2 and N4 perform a T-flip, possibly followed by a simple flip. Line N3 performs a simple flip followed by a T-flip, possibly followed by a simple flip, and in this case the combination of three flips, simple flip plus T-flip plus simple flip, yields a D-flip overall. The function $\texttt{lock}(R, j, \texttt{dir})$, $\texttt{dir} \in \{\rhd, \triangle\}$, called at the end of each of the lines N2–N7 checks whether rectangle r_j participates in an H-aligned block (if $\texttt{dir} = \rhd$) or V-aligned block (if $\texttt{dir} = \triangle$) that is locked and must be transformed to an AH-aligned block or AV-aligned block by a simple flip. The function $\texttt{unlock}(R, j, d)$, $d \in \{\lhd, \rhd\}$, called at the beginning in line N1 does the converse, namely checking whether r_j participates in an AH-aligned block or AV-aligned block that must be made H-aligned or V-aligned, respectively, before performing a jump of rectangle r_j in direction d. The implementation of these functions is shown below for the cases $\texttt{dir} = \rhd$ and $d = \rhd$. The other variants $\texttt{dir} = \triangle$ and $d = \lhd$ are omitted for simplicity.

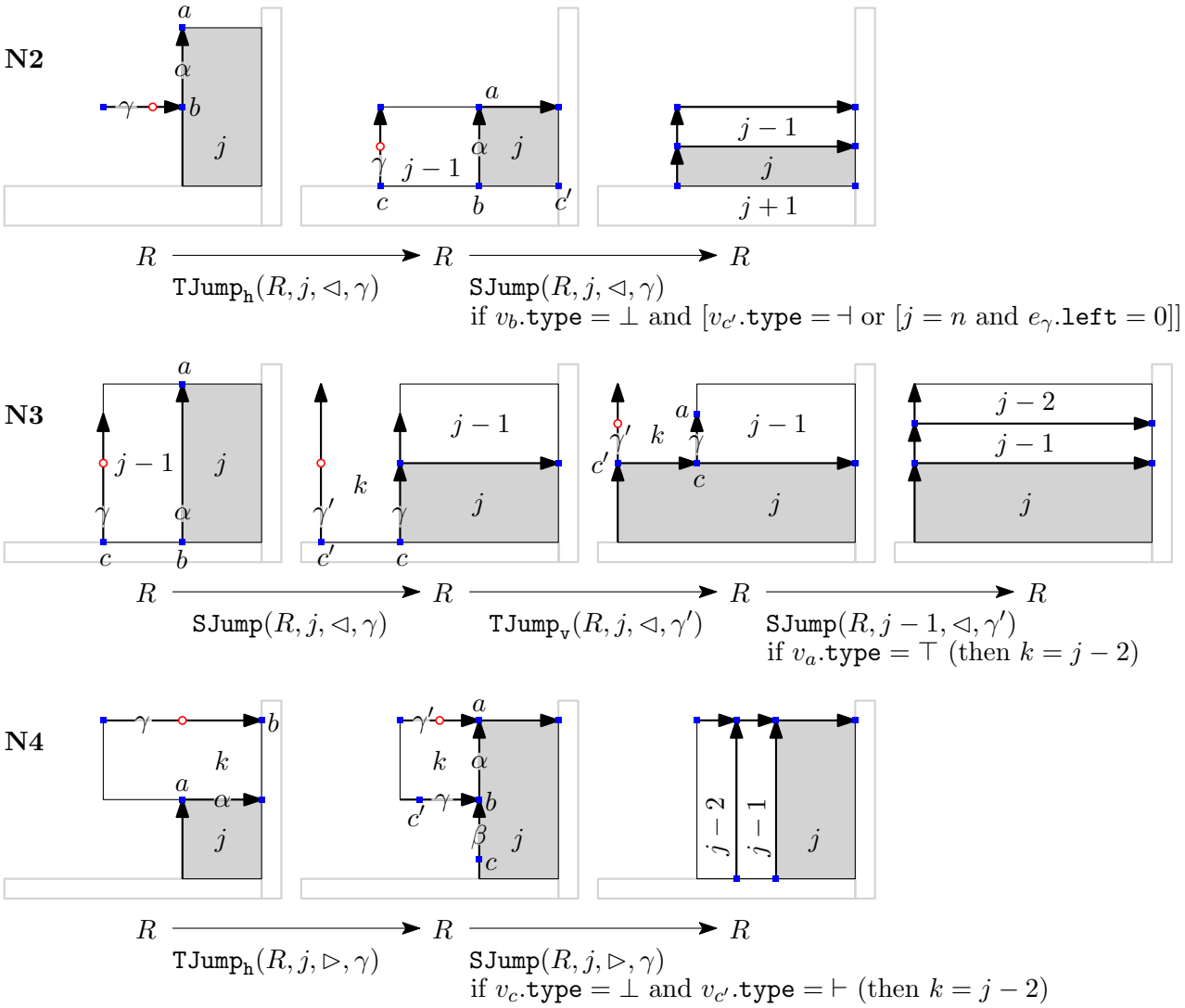


FIGURE 28. Flip operations in lines N2–N4 of the oracle $\texttt{next}_{\mathcal{B}_n}$.

lock$(R, j, \triangleright)$ *(Lock block if necessary).*

L1. [Prepare] Set $a \leftarrow r_j.\texttt{neast}$, $b \leftarrow r_j.\texttt{swest}$, $c \leftarrow r_j.\texttt{seast}$, $\alpha \leftarrow v_a.\texttt{west}$, $\beta \leftarrow v_b.\texttt{east}$ and return if $v_b.\texttt{type} \neq \vdash$ or $v_c.\texttt{type} \neq \dashv$ or $e_\beta.\texttt{head} \neq c$.

L2. [Lock if necessary] Set $d \leftarrow r_{j+1}.\texttt{seast}$ and if $v_d.\texttt{type} = \bot$ call $\texttt{Sjump}(R, j+1, \triangleright, \alpha)$.

unlock$(R, j, \triangleright)$ *(Unlock block if necessary).*

U1. [Prepare] Set $a \leftarrow r_j.\texttt{neast}$, $b \leftarrow r_j.\texttt{seast}$, $c \leftarrow r_j.\texttt{swest}$ and $\gamma \leftarrow v_c.\texttt{north}$.

U2. [Unlock if necessary] If $v_a.\texttt{type} = \top$ and $v_b.\texttt{type} = \bot$ call $\texttt{Sjump}(R, j+1, \triangleleft, \gamma)$.

To use Algorithm M$^\square$ with this oracle, in line M5 we also need to check whether $R^{[j-1]}$ is bottom-based or right-based (in addition to $R^{[j]}$), and whether $R^{[j-1]}$ or $R^{[j]}$ are one simple flip away from such a configuration. Similarly, to use this oracle in conjunction with the oracle $\texttt{next}_{\mathcal{B}_n(\mathcal{P})}$ defined in Section 7.3, we need to test containment of a pattern P in the rectangulation R after a jump of rectangle r_j not only via $\texttt{contains}(R, j, P)$, but also using

$\texttt{contains}(R, j-1, P)$ and $\texttt{contains}(R, j+1, P)$, as all three rectangles r_{j-1}, r_j and r_{j+1} may be modified through one call of $\texttt{next}_{\mathcal{B}_n}(R, j, d)$. For details see our C++ implementation [cos].

We obtain the following analogue of Theorems 7 and 8.

Theorem 35. *Let $n \geq 3$. For any set of patterns $\mathcal{P}$ that are neither bottom-based nor right-based nor simple-flippable to a bottom-based or right-based pattern, Algorithm $M^{\square}$ with the oracle $\texttt{next}_{\mathcal{B}_n(\mathcal{P})}$ defined in Section 7.3, which calls $\texttt{next}_{\mathcal{B}_n}$ as defined above, visits every rectangulation from $\mathcal{B}_n(\mathcal{P})$ exactly once, performing a sequence of one T- or D-flip plus at most three simple flips in each step.*

It remains to analyze the running time of this algorithm.

Lemma 36. *Each call $\texttt{next}_{\mathcal{B}_n}(R, j, d)$ takes time $\mathcal{O}(1)$.*

As we are dealing with a subset of diagonal rectangulations, the proof is very similar to the proof of Lemma 14.

Proof. Consider any of the calls $\texttt{Sjump}(R, j, d)$, $\texttt{Tjump}_{\texttt{h}}(R, j, d)$, $\texttt{Tjump}_{\texttt{v}}(R, j, d)$ in lines N2–N7 and let R' be the rectangulation after the call. As we only consider the first insertion point of each vertical group and the last insertion point of each horizontal group of $I(R^{[j-1]})$, we have $v(R, R') = 0$ and $h(R, R') = 0$, so the claim follows from Lemma 10 (b)+(c). □

Lemma 36 immediately yields the following result.

Theorem 37. *Algorithm $M^{\square}$ with the oracle $\texttt{next}_{\mathcal{B}_n}$ takes time $\mathcal{O}(1)$ to visit each block-aligned rectangulation.*

For the pattern avoidance version of this algorithm, we obtain the following runtime bounds.

Theorem 38. *For any set of patterns $\mathcal{P} \subseteq \{$  , $\}$, Algorithm $M^{\square}$ with the oracle $\texttt{next}_{\mathcal{B}_n(\mathcal{P})}$ visits each rectangulation from $\mathcal{B}_n(\mathcal{P})$ in time $\mathcal{O}(n)$.*

Proof. The oracle $\texttt{next}_{\mathcal{B}_n(\mathcal{P})}$ repeatedly calls the function $\texttt{next}_{\mathcal{B}_n}$. Applying Theorem 17 with the bound $f_n = \mathcal{O}(1)$ from Lemma 36 and the bound $t_n = \mathcal{O}(1)$ from Lemma 18, the term $n \cdot (f_n + t_n)$ evaluates to $\mathcal{O}(n)$, as claimed. □

10. Counting pattern-avoiding rectangulations

In this section we report on computer experiments that count pattern-avoiding rectangulations $\mathcal{C}_n(\mathcal{P})$ for all interesting subsets of patterns $\mathcal{P} \subseteq \{P_1, \ldots, P_8\}$ where $P_1 =$, $P_2 =$, $P_3 =$, $P_4 =$, $P_5 =$, $P_6 =$, $P_7 =$, $P_8 =$. Clearly, we can omit sets of patterns that are equivalent to another set of patterns under D_4 actions (rotations and mirroring vertically or horizontally). Table 3 shows the results for generic rectangulations $\mathcal{C}_n = \mathcal{R}_n$ as a base class, and Table 4 for block-aligned rectangulations $\mathcal{C}_n = \mathcal{B}_n$ as a base class. The set of patterns used in each row of the table is denoted by the pattern indices, omitting curly brackets and commas. For example, the row 1478 refers to the set $\mathcal{P} = \{P_1, P_4, P_7, P_8\}$. When counting block-aligned rectangulations with our algorithms, the patterns $P_3, \ldots, P_8$ cannot be used, as they do not satisfy the conditions of Theorem 35.

Several of these counting sequences appear in the OEIS [oei20], and are related to pattern-avoiding permutations (see e.g. [BGRR18]). The matching OEIS entries marked with ? are observed through numerical experiments, but no formal bijective proof has been obtained yet, even though finding one should be straightforward in some cases. The last two rows in Table 3 with ? are interesting, as the correspondence to the objects mentioned in those OEIS entries

TABLE 3. Counts for pattern-avoiding rectangulations with generic rectangulations as a base class.

Patterns $\mathcal{P}$	Counts $\lvert\mathcal{R}_n(\mathcal{P})\rvert$ for $n = 1, \ldots, 12$	OEIS
$\emptyset$	1, 2, 6, 24, 116, 642, 3938, 26194, 186042, 1395008, 10948768, 89346128, ...	A342141
1	1, 2, 6, 24, 115, 624, 3712, 23704, 160140, 1132628, 8321372, 63129494, ...	
3	1, 2, 6, 23, 104, 530, 2958, 17734, 112657, 750726, 5207910, 37387881, ...	A117106 ?
7	1, 2, 6, 24, 115, 619, 3607, 22265, 143667, 960854, 6622454, 46841852, ...	
12	1, 2, 6, 24, 114, 606, 3494, 21434, 138100, 926008, 6418576, 45755516, ...	
13	1, 2, 6, 23, 103, 514, 2779, 15983, 96557, 607174, 3947335, 26393968, ...	
14	1, 2, 6, 23, 103, 514, 2779, 15983, 96557, 607174, 3947335, 26393968, ...	
17	1, 2, 6, 24, 114, 601, 3391, 20070, 123156, 777836, 5031860, 33225018, ...	
34	1, 2, 6, 22, 92, 422, 2074, 10754, 58202, 326240, 1882960, 11140560, ...	A001181
35	1, 2, 6, 22, 94, 450, 2349, 13128, 77533, 479250, 3077864, 20421177, ...	
36	1, 2, 6, 22, 92, 422, 2074, 10754, 58202, 326240, 1882960, 11140560, ...	A001181 ?
37	1, 2, 6, 23, 103, 514, 2779, 15987, 96664, 608933, 3970441, 26661194, ...	
38	1, 2, 6, 23, 103, 507, 2641, 14245, 78619, 441174, 2508688, 14429287, ...	
78	1, 2, 6, 24, 114, 596, 3276, 18396, 103718, 581636, 3229888, 17730584, ...	
123	1, 2, 6, 23, 102, 498, 2606, 14378, 82725, 492520, 3017043, 18933201, ...	
127	1, 2, 6, 24, 113, 583, 3183, 18077, 105813, 634838, 3889236, 24262094, ...	
134	1, 2, 6, 22, 91, 408, 1938, 9614, 49335, 260130, 1402440, 7702632, ...	A000139 ?
135	1, 2, 6, 22, 93, 436, 2209, 11889, 67159, 394692, 2397407, 14974319, ...	
136	1, 2, 6, 22, 91, 408, 1938, 9614, 49335, 260130, 1402440, 7702632, ...	A000139 ?
137	1, 2, 6, 23, 102, 498, 2605, 14362, 82567, 491285, 3008821, 18886524, ...	
138	1, 2, 6, 23, 102, 491, 2472, 12763, 66908, 354396, 1892049, 10169071, ...	
145	1, 2, 6, 22, 91, 408, 1938, 9614, 49335, 260130, 1402440, 7702632, ...	A000139 ?
147	1, 2, 6, 23, 102, 491, 2472, 12763, 66908, 354396, 1892049, 10169071, ...	
148	1, 2, 6, 23, 102, 498, 2605, 14362, 82567, 491285, 3008821, 18886524, ...	
178	1, 2, 6, 24, 113, 578, 3070, 16496, 88378, 468780, 2455332, 12694892, ...	
345	1, 2, 6, 21, 82, 346, 1547, 7236, 35090, 175268, 897273, 4690392, ...	A281784 ?
347	1, 2, 6, 22, 91, 406, 1905, 9264, 46288, 236364, 1229209, 6494549, ...	
357	1, 2, 6, 22, 93, 439, 2257, 12407, 71963, 436176, 2742686, 17791880, ...	
358	1, 2, 6, 22, 93, 427, 2044, 9975, 49089, 242458, 1199855, 5947447, ...	
367	1, 2, 6, 22, 91, 406, 1905, 9264, 46288, 236364, 1229209, 6494549, ...	
378	1, 2, 6, 23, 102, 491, 2462, 12534, 63842, 322875, 1615726, 7990347, ...	
1234	1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, 5293446, ...	A006318
1235	1, 2, 6, 22, 92, 422, 2073, 10738, 58029, 324648, 1869482, 11031813, ...	
1236	1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, 5293446, ...	A006318 ?
1237	1, 2, 6, 23, 101, 482, 2437, 12877, 70514, 397823, 2302074, 13614952, ...	
1238	1, 2, 6, 23, 101, 475, 2309, 11409, 56879, 285220, 1436772, 7267279, ...	
1278	1, 2, 6, 24, 112, 560, 2872, 14780, 75512, 381320, 1901292, 9366128, ...	
1345	1, 2, 6, 21, 81, 334, 1446, 6498, 30074, 142556, 689248, 3388453, ...	
1346	1, 2, 6, 21, 81, 334, 1446, 6498, 30074, 142556, 689248, 3388453, ...	
1347	1, 2, 6, 22, 90, 392, 1774, 8236, 38961, 187093, 909961, 4475961, ...	
1348	1, 2, 6, 22, 90, 392, 1774, 8236, 38961, 187093, 909961, 4475961, ...	
1357	1, 2, 6, 22, 92, 425, 2119, 11210, 62164, 358200, 2130760, 13019572, ...	
1358	1, 2, 6, 22, 92, 413, 1914, 8981, 42310, 199500, 940788, 4437867, ...	
1367	1, 2, 6, 22, 90, 392, 1774, 8236, 38961, 187093, 909961, 4475961, ...	
1378	1, 2, 6, 23, 101, 475, 2298, 11178, 54030, 258192, 1217964, 5673144, ...	
1457	1, 2, 6, 22, 90, 392, 1774, 8236, 38961, 187093, 909961, 4475961, ...	
1478	1, 2, 6, 23, 101, 475, 2298, 11178, 54030, 258192, 1217964, 5673144, ...	
3456	1, 2, 6, 20, 72, 274, 1088, 4470, 18884, 81652, 360054, 1614618, ...	
3457	1, 2, 6, 21, 81, 335, 1461, 6643, 31235, 150960, 746522, 3764017, ...	
3458	1, 2, 6, 21, 81, 330, 1386, 5925, 25614, 111638, 489937, 2164127, ...	
3478	1, 2, 6, 22, 90, 390, 1736, 7794, 34926, 155340, 683920, 2977794, ...	
3578	1, 2, 6, 22, 92, 416, 1952, 9270, 43986, 207340, 968862, 4486184, ...	
3678	1, 2, 6, 22, 90, 390, 1736, 7794, 34926, 155340, 683920, 2977794, ...	
12345	1, 2, 6, 21, 80, 322, 1347, 5798, 25512, 114236, 518848, 2384538, ...	A106228 ?
12347	1, 2, 6, 22, 89, 378, 1647, 7286, 32574, 146866, 667088, 3050619, ...	
12357	1, 2, 6, 22, 91, 411, 1985, 10099, 53547, 293602, 1655170, 9551440, ...	
12358	1, 2, 6, 22, 91, 399, 1788, 8057, 36291, 163158, 732385, 3285369, ...	
12367	1, 2, 6, 22, 89, 378, 1647, 7286, 32574, 146866, 667088, 3050619, ...	
12378	1, 2, 6, 23, 100, 459, 2140, 9944, 45676, 206855, 923746, 4073045, ...	
13456	1, 2, 6, 20, 71, 264, 1018, 4042, 16438, 68196, 287724, 1231514, ...	
13457	1, 2, 6, 21, 80, 323, 1362, 5941, 26628, 122036, 569781, 2702496, ...	
13458	1, 2, 6, 21, 80, 318, 1290, 5287, 21803, 90351, 376174, 1573975, ...	
13467	1, 2, 6, 21, 80, 318, 1290, 5287, 21803, 90351, 376174, 1573975, ...	
13468	1, 2, 6, 21, 80, 323, 1362, 5941, 26628, 122036, 569781, 2702496, ...	
13478	1, 2, 6, 22, 89, 376, 1610, 6878, 29094, 121498, 500688, 2037758, ...	
13578	1, 2, 6, 22, 91, 402, 1824, 8318, 37750, 169880, 757488, 3348274, ...	
13678	1, 2, 6, 22, 89, 376, 1610, 6878, 29094, 121498, 500688, 2037758, ...	
14578	1, 2, 6, 22, 89, 376, 1610, 6878, 29094, 121498, 500688, 2037758, ...	
34567	1, 2, 6, 20, 71, 263, 1006, 3949, 15839, 64700, 268477, 1129385, ...	
34578	1, 2, 6, 21, 80, 319, 1300, 5340, 21946, 89909, 366626, 1487463, ...	
123456	1, 2, 6, 20, 70, 254, 948, 3618, 14058, 55432, 221262, 892346, ...	A078482
123457	1, 2, 6, 21, 79, 311, 1265, 5275, 22431, 96900, 424068, 1876143, ...	A033321 ?
123458	1, 2, 6, 21, 79, 306, 1196, 4681, 18308, 71564, 279820, 1095533, ...	
123478	1, 2, 6, 22, 88, 362, 1488, 6034, 24024, 93830, 359824, 1357088, ...	
123578	1, 2, 6, 22, 90, 388, 1700, 7434, 32212, 138040, 585246, 2457712, ...	
123678	1, 2, 6, 22, 88, 362, 1488, 6034, 24024, 93830, 359824, 1357088, ...	
134567	1, 2, 6, 20, 70, 253, 938, 3553, 13708, 53736, 213588, 859335, ...	
134578	1, 2, 6, 21, 79, 307, 1206, 4738, 18532, 72070, 278718, 1072739, ...	
134678	1, 2, 6, 21, 79, 307, 1206, 4738, 18532, 72070, 278718, 1072739, ...	
345678	1, 2, 6, 20, 70, 252, 924, 3432, 12870, 48620, 184756, 705432, ...	A000984 ?
1234567	1, 2, 6, 20, 69, 243, 870, 3159, 11611, 43130, 161691, 611065, ...	
1234578	1, 2, 6, 21, 78, 295, 1114, 4166, 15390, 56167, 202738, 724813, ...	
1345678	1, 2, 6, 20, 69, 242, 858, 3068, 11050, 40052, 145996, 534888, ...	A026029 ?
12345678	1, 2, 6, 20, 68, 232, 792, 2704, 9232, 31520, 107616, 367424, ...	A006012

TABLE 4. Counts for pattern-avoiding rectangulations with block-aligned rectangulations as a base class.

Patterns $\mathcal{P}$	Counts $\lvert\mathcal{B}_n(\mathcal{P})\rvert$ for $n = 1, \ldots, 13$	OEIS
$\emptyset$	1, 1, 2, 6, 22, 88, 374, 1668, 7744, 37182, 183666, 929480, 4803018, ...	A214358
1	1, 1, 2, 6, 21, 79, 312, 1280, 5416, 23506, 104198, 470192, 2154204, ...	
12	1, 1, 2, 6, 20, 70, 254, 948, 3618, 14058, 55432, 221262, 892346, ...	A078482

is not obvious. This is true in particular for OEIS sequence A000984, which are the central binomial coefficients $\binom{2n}{n}$.

11. Open questions

The subject of pattern-avoiding rectangulations deserves further systematic investigation, and may still hold many undiscovered gems; recall Table 3. Understanding the number of pattern-avoiding rectangulations that are obtained by rectangle insertion may also help to improve the runtime bounds for our generation algorithms (recall Remark 21). Moreover, does the avoidance of a rectangulation pattern always correspond to the avoidance of a particular permutation pattern, and what is this correspondence?

In our paper we considered R-equivalence and S-equivalence of generic rectangulations $\mathcal{R}_n$, and these equivalence relations are induced by wall slides, or by wall slides and simple flips, respectively. Considering all three basic flip operations $F = \{W, S, T\}$, namely wall slides, simple flips, and T-flips, there are $2^3 = 8$ possible subsets of F to induce an equivalence relation on $\mathcal{R}_n$. Which of these equivalence relations are interesting (apart from $\emptyset$, $\{W\}$ and $\{W, S\}$ considered here), and what are suitable representatives that can be generated efficiently?

Another interesting question to investigate would be Gray codes for rectangulations of point sets as introduced by Ackerman, Barequet and Pinter [ABP06b]. Some first results in this direction have been obtained by Yamanaka, Rahman and Nakano [YRN18]. In particular, can we apply our permutation-based generation framework for this task?

Acknowledgements

We thankfully acknowledge several discussions in the early phases of this manuscript with Hung P. Hoang, which took place at the 17th Gremo Workshop on Open Problems in Switzerland. We thank the organizers for the invitation to the workshop, and the other participants for the pleasant and stimulating working atmosphere. Furthermore, we thank the reviewers for their numerous comments, which helped improving the manuscript.

References

[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 pp., 2013.

[ABP06a] 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.

[ABP06b] E. Ackerman, G. Barequet, and R. Y. Pinter. On the number of rectangulations of a planar point set. *J. Combin. Theory Ser. A*, 113(6):1072–1091, 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).

[AM10] A. Asinowski and T. Mansour. Separable d-permutations and guillotine partitions. *Ann. Comb.*, 14(1):17–43, 2010.

[ANY07] K. Amano, S. Nakano, and K. Yamanaka. On the number of rectangular drawings: Exact counting and lower and upper bounds. IPSJ SIG Technical Report 2007-AL-115 (5), 2007.

[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.

[BGRR18] M. Bouvel, V. Guerrini, A. Rechnitzer, and S. Rinaldi. Semi-Baxter and strong-Baxter: two relatives of the Baxter sequence. *SIAM J. Discrete Math.*, 32(4):2795–2819, 2018.

[CM14] J. Conant and T. Michaels. On the number of tilings of a square by rectangles. *Ann. Comb.*, 18(1):21–34, 2014.

[cos] The Combinatorial Object Server: Generate rectangulations. `http://www.combos.org/rect`.

[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.

[Ehr73] G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. Assoc. Comput. Mach.*, 20:500–513, 1973.

[EMSV12] D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek. Area-universal and constrained rectangular layouts. *SIAM J. Comput.*, 41(3):537–564, 2012.

[Fel13] S. Felsner. Rectangle and square representations of planar graphs. In *Thirty essays on geometric graph theory*, pages 213–248. Springer, New York, 2013.

[FFNO11] S. Felsner, É. Fusy, M. Noy, and D. Orden. Bijections for Baxter families and related objects. *J. Combin. Theory Ser. A*, 118(3):993–1020, 2011.

[FIT09] R. Fujimaki, Y. Inoue, and T. Takahashi. An asymptotic estimate of the numbers of rectangular drawings or floorplans. In *2009 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 856–859, 2009.

[FNT21] S. Felsner, A. Nathenson, and C. D. Tóth. Aspect ratio universal rectangular layouts. Manuscript, 2021.

[Fus09] É. Fusy. Transversal structures on triangulations: a combinatorial study and straight-line drawings. *Discrete Math.*, 309(7):1870–1894, 2009.

[He14] B. D. He. A simple optimal binary representation of mosaic floorplans and Baxter permutations. *Theoret. Comput. Sci.*, 532:40–50, 2014.

[HHC+00] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In E. Sentovich, editor, *Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, 2000, San Jose, California, USA, November 5-9, 2000*, pages 8–12. IEEE Computer Society, 2000.

[HHMW22] E. Hartung, H. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Trans. Amer. Math. Soc.*, 375(4):2255–2291, 2022.

[HM21] H. P. Hoang and T. Mütze. Combinatorial generation via permutation languages. II. Lattice congruences. *Israel J. Math.*, 244(1):359–417, 2021.

[ITF09] Y. Inoue, T. Takahashi, and R. Fujimaki. Counting rectangular drawings or floorplans in polynomial time. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 92-A(4):1115–1120, 2009.

[KH97] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoret. Comput. Sci.*, 172(1-2):175–193, 1997.

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

[Lei21] L. J. Leifheit. Combinatorial properties of rectangulations. Master's thesis, TU Berlin, 2021.

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

[Mee19] E. Meehan. The Hopf algebra of generic rectangulations. `https://arxiv.org/abs/1903.09874`, 2019.

[MSL76] W. J. Mitchell, J. P. Steadman, and R. S. Liggett. Synthesis and optimization of small rectangular floor plans. *Environment and Planning B: Planning and Design*, 3(1):37–70, 1976.

[Nak01] S. Nakano. Enumerating floorplans with n rooms. In *Algorithms and computation (Christchurch, 2001)*, volume 2223 of *Lecture Notes in Comput. Sci.*, pages 107–115. Springer, Berlin, 2001.

[oei20] OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2020. `http://oeis.org`.

[Ott82] R. H. J. M. Otten. Automatic floorplan design. In J. S. Crabbe, C. E. Radke, and H. Ofek, editors, *Proceedings of the 19th Design Automation Conference, DAC '82, Las Vegas, Nevada, USA, June 14-16, 1982*, pages 261–267. ACM/IEEE, 1982.

[PPR21] A. Padrol, V. Pilaud, and J. Ritter. Shard polytopes. To appear in *Proceedings of the 33rd International Conference on Formal Power Series and Algebraic Combinatorics*; preprint available at `https://arxiv.org/abs/2007.01008`, 2021.

[PS19] V. Pilaud and F. Santos. Quotientopes. *Bull. Lond. Math. Soc.*, 51(3):406–420, 2019.

[Rea12] N. Reading. Generic rectangulations. *European J. Combin.*, 33(4):610–623, 2012.

[Rus16] F. Ruskey. Combinatorial Gray code. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 342–347. Springer, 2016.

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

[SC03] Z. C. Shen and C. C. N. Chu. Bounds on the number of slicing, mosaic, and general floorplans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(10):1354–1361, 2003.

[TN04] M. Takagi and S. Nakano. Listing all rectangular drawings with certain properties. *Systems and Computers in Japan*, 35(4):1–8, 2004.

[vKS07] M. van Kreveld and B. Speckmann. On rectangular cartograms. *Comput. Geom.*, 37(3):175–187, 2007.

[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.

[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.

[YCYN06] S. Yoshii, D. Chigira, K. Yamanaka, and S. Nakano. Constant time generation of rectangular drawings with exactly n faces. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 89-A(9):2445–2450, 2006.

[YRN18] K. Yamanaka, M. S. Rahman, and S. Nakano. Enumerating floorplans with columns. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 101-A(9):1392–1397, 2018.

Appendix A. Visualization of Gray codes

In this section we visualize the Gray codes obtained from our algorithms for generic rectangulations, diagonal rectangulations and block-aligned rectangulations (without any forbidden patterns). The corresponding 2-clumped permutations are shown below each rectangulation.

A.1. **Generic rectangulations.**

Figure 29. $n = 1$

Figure 30. $n = 2$

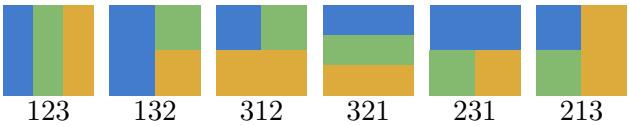


Figure 31. $n = 3$

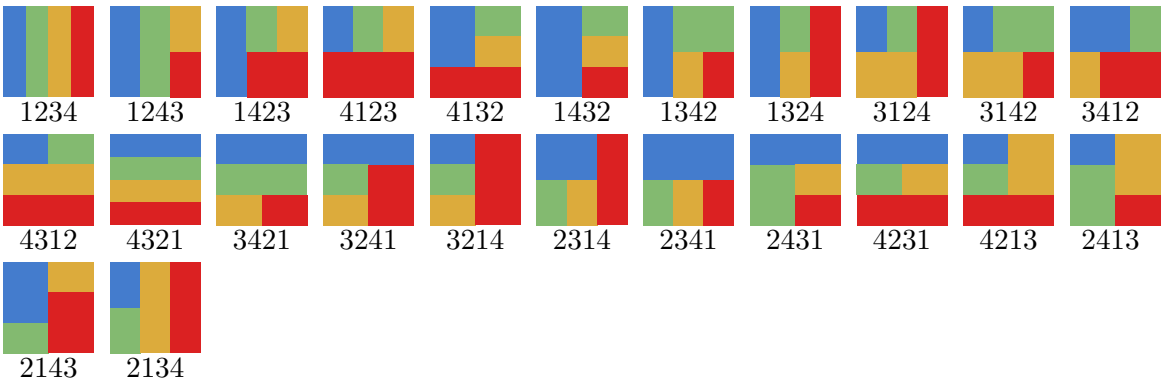


Figure 32. $n = 4$

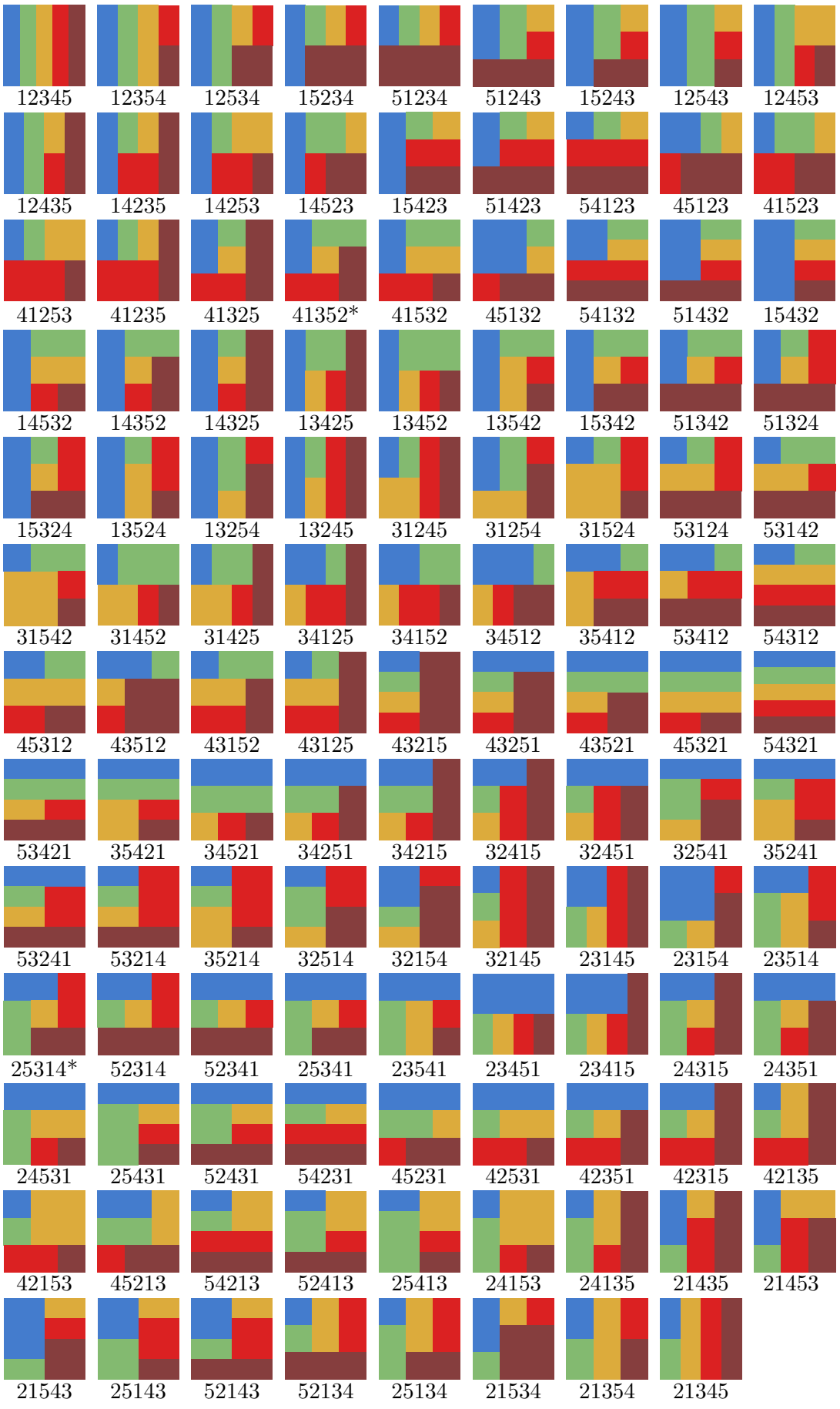


FIGURE 33. $n = 5$. The 2 non-guillotine rectangulations are marked by *.

A.2. **Diagonal rectangulations.**

FIGURE 34. $n = 1$

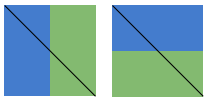


FIGURE 35. $n = 2$

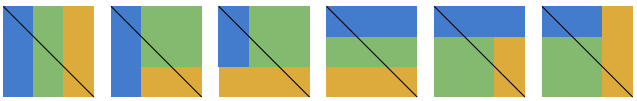


FIGURE 36. $n = 3$

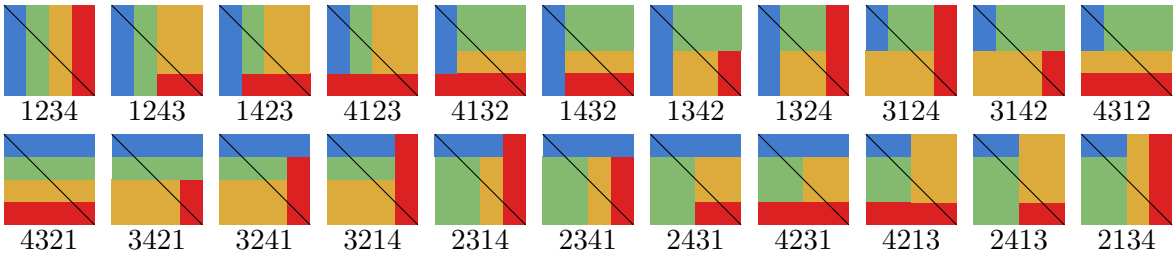


FIGURE 37. $n = 4$

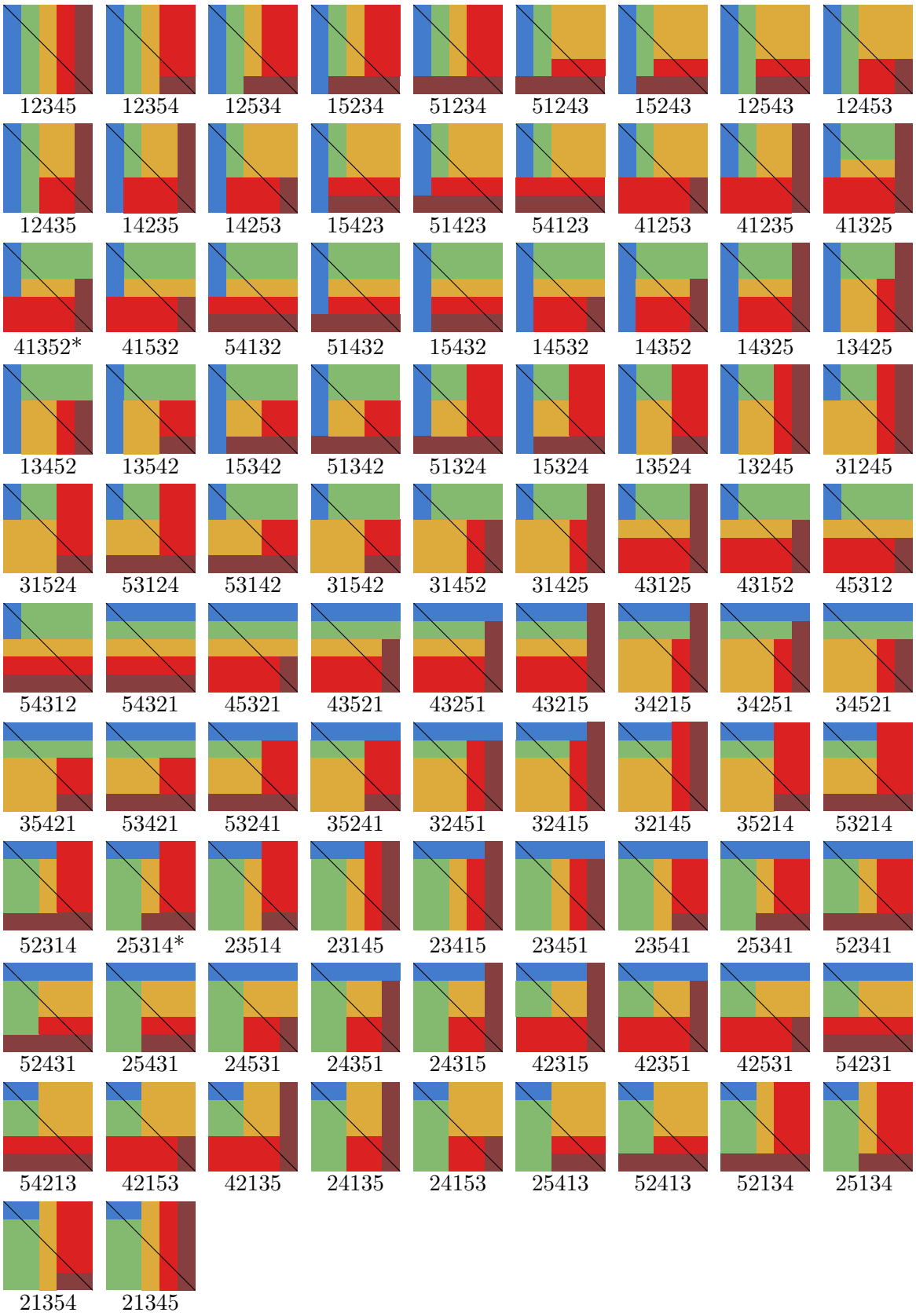


FIGURE 38. $n = 5$. The 2 non-guillotine rectangulations are marked by *.

A.3. **Block-aligned rectangulations.**

FIGURE 39. $n = 1$

FIGURE 40. $n = 2$

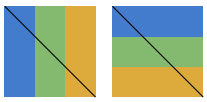


FIGURE 41. $n = 3$

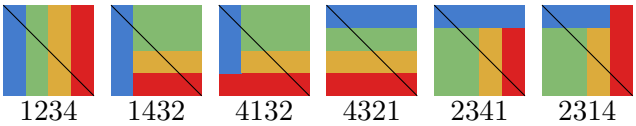


FIGURE 42. $n = 4$

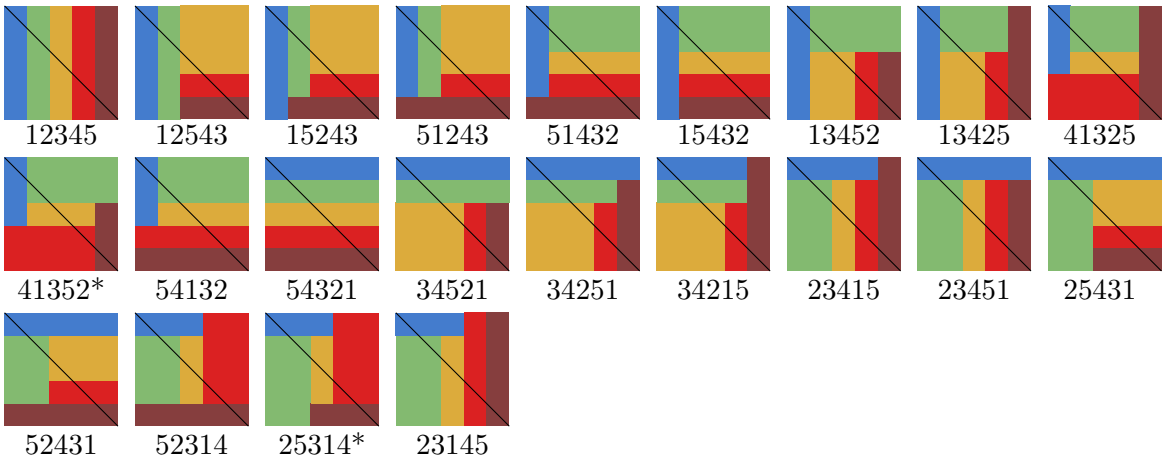


FIGURE 43. $n = 5$. The 2 non-guillotine rectangulations are marked by *.