

DISS. ETH NO. 28654

On Two Combinatorial Reconfiguration Problems: Reachability and Hamiltonicity



Phuc Hung Hoang

DISS. ETH NO. 28654

On Two Combinatorial Reconfiguration Problems: Reachability and Hamiltonicity

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZÜRICH

(Dr. sc. ETH Zürich)

presented by

Phuc Hung Hoang

M.Sc. London School of Economics and Political Science
in Management Science (Operational Research)

born on 2 November 1989
citizen of Vietnam

accepted on the recommendation of

Prof. Dr. Bernd Gärtner, examiner
Prof. Dr. Jean Cardinal, co-examiner
Prof. Dr. Torsten Mütze, co-examiner
Prof. Dr. Emo Welzl, co-examiner

2022

Abstract

This thesis considers two problems under the reconfiguration framework: ARRIVAL and exhaustive generation of combinatorial objects. Both problems can be viewed in terms of reconfiguration graphs. In the first case, the state space of ARRIVAL is the vertex set, and the transitions between states define the edges. The ARRIVAL problem then asks a reachability question on this graph. In the second case, the combinatorial objects to be generated are the vertices, and two vertices are adjacent if they differ by a small change. The task of exhaustive generation is then equivalent to finding a Hamilton path on this graph.

More specifically, in ARRIVAL, a train performs a deterministic, pseudorandom walk on a directed graph, and we need to determine which sink the train will eventually reach. The problem has been shown to be in $\text{NP} \cap \text{coNP}$ but is not known to be in P . In this thesis, we provide the first subexponential algorithm for ARRIVAL, derived from a new general framework for this problem. From this framework, we also obtain a polynomial-time algorithm, if the graph is almost acyclic. Additionally, we develop a different polynomial-time algorithm for another class of graphs that corresponds to the worst asymptotic runtime of many existing algorithms. All of the

above algorithms rely on a generalization of ARRIVAL with multiple trains, called G-ARRIVAL, and can be extended to solve G-ARRIVAL as well.

In the context of exhaustive generation, we present a general and versatile algorithmic framework that can generate a wide range of combinatorial objects, by encoding them as permutations. This framework unifies many known classical Gray codes, such as the Steinhaus-Johnson-Trotter algorithm for permutations, the binary reflected Gray code for bitstrings, the Lucas-Roelants van Baronaigien-Ruskey algorithm for binary trees, and the Gray code for set partitions by Kaye. Moreover, the framework also allows us to obtain many new Gray codes.

This thesis presents two main applications of the framework above. The first is the generation of many families of permutations avoiding certain classical patterns, (bi)vincular patterns, barred patterns, boxed patterns, Bruhat-restricted patterns, mesh patterns, monotone and geometric grid classes, and many others. By the bijection of these permutations with other combinatorial objects, we also obtain new Gray codes for these objects. The second application pertains to the lattice congruences of the weak order of the symmetric group S_n . These objects can be realized as polytopes, called quotientopes, that take the hypercubes, associahedra, permutahedra, as special cases. For each lattice congruence, our algorithm generates a Gray code for its equivalence classes, which translates to a Hamilton path on the skeleton of the corresponding quotientope.

Zusammenfassung

Diese Arbeit betrachtet zwei Rekonfigurationsprobleme: ARRIVAL und die vollständige Aufzählung von kombinatorischen Objekten. Beide Probleme können mit Hilfe von Rekonfigurationsgraphen betrachtet werden. Im ersten Fall ist der Zustandsraum von ARRIVAL die Knotenmenge, und die Übergänge zwischen den Zuständen definieren die Kanten. Das ARRIVAL-Problem stellt dann eine Erreichbarkeitsfrage für diesen Graphen. Im zweiten Fall sind die zu erzeugenden kombinatorischen Objekte die Knoten, und zwei Knoten sind benachbart, wenn sie sich in einer kleinen Änderung unterscheiden. Die Aufgabe der vollständigen Aufzählung ist dann gleichbedeutend mit der Suche nach einem Hamilton-Pfad in diesem Graphen.

Genauer gesagt, führt ein Zug beim ARRIVAL-Problem eine deterministische, pseudozufällige Irrfahrt auf einem gerichteten Graphen durch, und es gilt zu entscheiden, welche Senke der Zug schliesslich erreichen wird. Es wurde gezeigt, dass das Problem in $\text{NP} \cap \text{coNP}$ liegt, aber es ist nicht bekannt, ob es in P liegt. In dieser Arbeit stellen wir den ersten subexponentiellen Algorithmus für ARRIVAL vor, der sich aus einem neuen allgemeinen Ansatz für dieses Problem ableitet. Aus diesem Ansatz erhalten wir auch einen Polynomialzeit-Algorithmus, für den Fall dass der Graph fast azyklisch ist. Zusätzlich

entwickeln wir einen anderen Polynomialzeit-Algorithmus für eine andere Klasse von Graphen, die zur schlechtesten asymptotischen Laufzeit vieler bekannter Algorithmen korrespondiert. Alle oben genannten Algorithmen beruhen auf einer Verallgemeinerung des ARRIVAL-Problems auf mehrere Zügen, genannt G-ARRIVAL und können erweitert werden, um auch G-ARRIVAL zu lösen.

Bezüglich der vollständigen Aufzählung stellen wir einen allgemeinen und vielseitigen algorithmischen Ansatz vor, der eine breite Vielfalt von kombinatorischen Objekten erzeugen kann, indem er diese als Permutationen kodiert. Dieser Ansatz vereinigt viele bekannte klassische Gray-Codes, wie den Steinhaus-Johnson-Trotter-Algorithmus für Permutationen, den binären reflektierten Gray-Code für Bitfolgen, den Lucas-Roelants van Baronaigien-Ruskey-Algorithmus für binäre Bäume und den Gray-Code für Mengenpartitionen von Kaye. Darüber hinaus ermöglicht uns der Ansatz auch, viele neue Gray-Codes zu erhalten.

Diese Arbeit stellt zwei Hauptanwendungen des obigen Ansatzes vor. Die erste ist die Erzeugung vieler Familien von Permutationen, die bestimmte klassische Muster, (bi)vinkuläre Muster, Gittermuster, Kastenmuster, Bruhat-beschränkte Muster, Netzmuster, monotone und geometrische Netzklassen und viele andere vermeiden. Durch die Bijektion dieser Permutationen mit anderen kombinatorischen Objekten erhalten wir auch neue Gray-Codes für diese Objekte. Die zweite Anwendung bezieht sich auf die Gitterkongruenzen der schwachen Ordnung der symmetrischen Gruppe S_n . Diese Objekte können als Polytope, sogenannte Quotientope, realisiert werden, welche den Hyperwürfel, Assoziaeder und Permutoaeder als Spezialfälle ergeben. Für jede Gitterkongruenz generiert unser Algorithmus einen Gray-Code für ihre Äquivalenzklassen, der sich in einen Hamilton-Pfad auf dem Skelett des entsprechenden Quotientopes übersetzt.

Acknowledgments

I wish to express my sincere gratitude to Bernd Gärtner and Emo Welzl for welcoming me to their research group. They have maintained an excellent research environment in the group and allowed me many opportunities to grow in the past four years. Especially, I was incredibly fortunate to have Bernd as my supervisor. He gave me the freedom to explore different research directions and provided continuous support in whatever endeavor I pursued. Moreover, he always made time to discuss any aspect of my doctoral career.

I am immensely grateful to Torsten Mütze for inviting me to the combinatorial generation project. Many structural insights that we encountered have allowed me glimpses of the exceptional beauty of mathematics. I also thank him for inviting me to Warwick and his annual workshops and for coming to Zurich for my examination.

Special thanks to Jean Cardinal and Emo Welzl for reviewing my thesis and serving as members of my committee, and to Bernd Gärtner, Elizabeth Hartung, Sebastian Haslebacher, Torsten Mütze, and Aaron Williams for our fruitful collaborations, which resulted in the content of this thesis.

I am indebted to Gregory Sorkin, who welcomed me to mathematical research through my master's dissertation. He has encouraged me to

pursue a PhD and also provided support and guidance afterwards.

Next, I would like to thank all my colleagues at ETH Zürich. Special thanks to Michael Hoffmann for great advice and amazing workshops; to Malte Milatz, Ahad N. Zehmakan, Jerri Nummenpalo, Patrick Schnider, and Manuel Wettstein for their help in navigating the various aspects of ETH life; to Saeed Ilchi for our semester exploring the spectrahedra; to Luis Barba, Daniel Bertschinger, Nicolas El Maalouly, Nicolas Grelier, Chih-Hung Liu, Meghana M. Reddy, Tim Taubner, and Simon Weber for companionship in and outside of the office; to Matilda Backendal, Tobias Grosser, Florian Meier, and Alexander Viand for the great time with the mental strength initiative; and to Andrea Salow for Mi-Lu-Di/Mi/Dos, German practice, and incredible help in administrative matters.

Thanks to the collaborators of my four other papers [3, 26, 40, 88]: Oswin Aichholzer, Kristóf Bérczi, Sriram Bhyravarapu, Kenny Chiu, Tim Hartmann, Michael Hoffmann, Subrahmanyam Kalyanasundaram, Stefan Lendl, Yannic Maus, I. Vinod Reddy, Lilla Tóthmérész, Birgit Vogtenhuber, Alexandra Weinberger, and Lasse Wulf.

Many of the papers above came from visits to other research institutions. I want to thank Kristóf for the invitation to Budapest and to Hausdorff Research Institute of Mathematics for the invitation to their trimester program on discrete optimization. I am also grateful to Alexandra, Birgit, Lasse, Oswin, and Alexander Pilz for arranging desk spaces for me during my many visits to Graz. Special thanks to Uwe Yacine Schwarze for being the reason of these visits.

Last but not least, I would like to thank my family and friends for their support. Special thanks to Đỗ Công Lý for the amazing cover art of this thesis and to Christoph Ribbe whose presence in the flat helped keep me sane during the time of Corona.

Contents

1	Overview	1
1.1	ARRIVAL	7
1.2	Combinatorial generation via permutation languages	10
1.3	Outline and summary of contributions	12
1.4	Notations	14
I	ARRIVAL	17
2	ARRIVAL and its generalization	19
2.1	ARRIVAL	20
2.2	G-ARRIVAL	22
2.3	Abelian networks	26
2.3.1	Abelian networks	26
2.3.2	Switching systems are Abelian networks . . .	29
2.3.3	Other unary networks	30
2.4	Switching systems vs. rotor-routing	32
2.4.1	Abelian networks as reconfiguration graphs .	35
2.5	G-ARRIVAL is well-defined	38
2.5.1	Switching flows	38
2.5.2	Termination and unique run profile	39

2.6	Decision complexity of G-ARRIVAL	41
2.6.1	G-ARRIVAL is in $\text{NP} \cap \text{coNP}$	41
2.6.2	G-ARRIVAL is in $\text{UP} \cap \text{coUP}$	43
2.7	Search complexity of G-ARRIVAL	45
2.7.1	UEOPL	46
2.7.2	Difficulties in adapting the proof for ARRIVAL	48
2.7.3	Partial switching flow	52
2.7.4	GS-ARRIVAL is in UEOPL	55
3	Subexponential algorithm for G-ARRIVAL	59
3.1	Naïve simulation of the train runs	60
3.2	Layer decomposition	62
3.3	Greedy simulations of the train runs	64
3.4	A general framework	69
3.4.1	The idea	69
3.4.2	Candidate switching flows and guessing switching system	70
3.4.3	Tarski fixed points	73
3.5	Subexponential algorithm for G-ARRIVAL	75
3.6	Feedback vertex sets	78
3.7	Discussion and open questions	79
4	G-ARRIVAL with two vertices per layer	83
4.1	Preliminaries	84
4.2	Algorithm Overview	86
4.3	Hitting probabilities	87
4.4	Initial analysis	91
4.5	2-ladder with a cross	93
4.6	Algorithm L	98
4.7	Solving G-ARRIVAL with ladder	99
4.8	Discussions	100

II	Combinatorial generation via permutations	105
5	A framework for combinatorial generation	107
5.1	Preliminaries	109
5.2	The basic algorithm	110
5.3	Zigzag languages	111
5.3.1	Characterization via the tree of permutations	112
5.3.2	Characterization via nuts	114
5.3.3	Proof of Theorem 5.1	114
5.4	Further properties of Algorithm J	117
5.5	A general recipe with classical examples	118
5.5.1	Permutations (Steinhaus-Johnson-Trotter) . .	119
5.5.2	Binary strings (BRGC)	119
5.5.3	Binary trees (Lucas-Roelants van Baronaigien- Ruskey)	121
5.5.4	Set partitions (Kaye)	122
5.6	Discussion	124
6	Pattern-avoiding permutations	127
6.1	Classical patterns and other variants	128
6.1.1	Preliminaries	128
6.1.2	Tame patterns	131
6.1.3	Vincular patterns	132
6.1.4	Barred patterns	133
6.1.5	Boxed patterns	135
6.1.6	Patterns with Bruhat restrictions	135
6.1.7	Bivincular patterns	136
6.1.8	Mesh patterns	136
6.1.9	Proof of Lemmas 6.4–6.9	142
6.2	Patterns with multiplicities	143
6.3	Algebra with patterns	144
6.3.1	Elementary transformations	144
6.3.2	Partially ordered patterns	148
6.3.3	Barred patterns with multiple bars	150

6.3.4	Weak avoidance of barred patterns and dotted patterns	151
6.3.5	Monotone and geometric grid classes	155
6.4	Limitations of our approach	158
7	Lattice congruences of the weak order	161
7.1	Generating lattice congruences of the weak order . .	165
7.1.1	Modified zigzag languages	165
7.1.2	Preliminaries	166
7.1.3	Combinatorics of lattice congruences of the weak order	168
7.1.4	Restrictions, rails, ladders, and projections .	172
7.1.5	Jumping through lattice congruences	179
7.2	Regular, vertex-transitive, and bipartite lattice quotients	185
7.2.1	Preliminaries	186
7.2.2	Exact counts for small dimensions	190
7.2.3	Counting quotient graphs	191
7.2.4	Regular quotient graphs	193
7.2.5	Maximum degree	208
7.2.6	Vertex-transitive quotient graphs	210
7.2.7	Bipartite quotient graphs	227
7.3	Pattern-avoiding permutations and lattice congruences	234
7.4	Open questions	238
	Bibliography	241
	Curriculum Vitae	263
	About the cover art	264

*... there ain't no journey what don't
change you some.*

—David Mitchell,
Cloud Atlas

CHAPTER 1

Overview

In the depth of my childhood memory, there was a three-way junction. One road led to my home, while another to my schools. For the third, I rarely took it, even when I was older and could go anywhere on my own. To be precise, I took it only once, when I tagged along my mother on her visit to an old friend. It must have been exciting at the time to venture finally into the path. However, I remember nothing about the trip, except for the destination. A house with many curious objects: strange comics, thin furniture, and a colorful cube. While the adults were occupied with their boring conversation, I examined the cube, and after a while, I figured that I could rotate certain parts. Somehow, a natural thing to aim for is to make every side uniform of one color. For the length of the visit, I did not succeed, nor did I learn that it was called the Rubik's cube.

The deceptive simplicity of the cube has captivated the interests of many children and adults alike. The main task is intuitive: Given a scrambled cube (a *position*), how can one make every face of the same color? Still, there are many other questions we can ask about the puzzle, such as the least number of moves required, the number of possible positions, an automation of the solving process, etc. These types of questions have sparked the interests of many “speed-cubers”, robotic engineers, and recreational mathematicians. For example, Rokicki, Kociemba, Davidson, and Dethridge [148] stated that the number of solvable positions is 43,252,003,274,489,856,000 or 164,604,041,664 up to certain notions of symmetry. Further, they showed that it is enough to solve any position with at most 20 *face turns*, i.e., 90- or 180-degree rotations of a face. In fact, they show a stronger statement: With at most 20 face turns, we can turn a position into *any* other position. Using graph theoretic notations, many of the questions above can be phrased as questions on a special graph. The vertices of this graph are the solvable positions (up to symmetries), and an edge connects two positions that differ by a face turn. Then we can view the result by Rokicki et al. as follows: Although the graph has 164,604,041,664 vertices with the maximum degree at most 18, its *diameter*, i.e. the maximum edge-distance between any two vertices, is at most (and in fact exactly) 20.

The discussion above shows a common template for *reconfiguration problems*. Van den Heuvel [166] and Nishimura [126] wrote detailed surveys about these problems. Here, we will give a brief overview. Throughout the overview, we refer to two examples as illustrated in Figure 1.1. On the left of the figure, we have the reconfiguration of the *triangulations of a convex polygon*, i.e., different ways to partition the polygon into triangles. On the right of the figure, we have the reconfiguration of the *k-sized independent sets* of a graph, i.e, sets of k pairwise non-adjacent vertices in the graph. The details of these reconfigurations will be explained below.

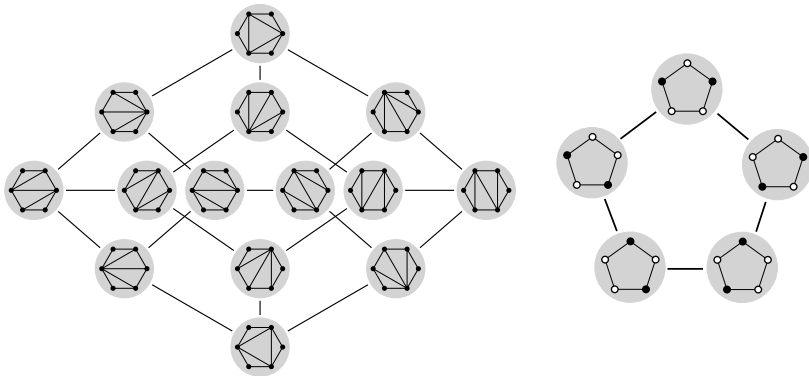


Figure 1.1: Two reconfiguration graphs: the flip graph of triangulations of a convex 6-gon by edge flips (left) and the flip graph of 2-sized independent sets of the 5-cycle by token sliding (right); the vertices in the independent sets are marked black

Generally speaking, a reconfiguration problem concerns with a state space, where we repeatedly travel from a state to its neighbor, under some notion of adjacency. More specifically, to define a reconfiguration problem, we need three main ingredients: the feasible configurations (i.e. the states), the transformations (i.e. the notion of adjacency), and a question.

Firstly, we assume that the set of feasible configurations can be described concisely, such as “the set of all triangulations of a given convex polygon”. Otherwise, if the full list of all possible configurations are given as the input, most problems would trivially be solvable in polynomial time. In fact, we usually assume that we can efficiently determine the feasibility of a configuration (i.e., checking if a given configuration is consistent with the concise description). When the context is clear, we will assume that the concerned configurations are feasible and drop the term “feasible”. Sometimes, these configurations are associated with a *source problem*, and they are the *fea-*

sible solutions of an instance of this problem. For example, for the k -sized independent sets as configurations, the source problem is the well known independence set problem: given a graph G and a positive integer k , we need to find an independent set with size k of G .

Secondly, we also assume some concise description of the transformations, the allowable operations that change a configuration into another. For example, we can transform a triangulation of a convex polygon by an *edge flip* described as follows: In the triangulation, we consider two triangles abc and dbc whose union forms a convex quadrilateral; we remove the common edge bc and add the edge ad . It is easy to see that the result is another triangulation. As another example, we can view the elements of an independent set as pairwise non-adjacent (unlabelled) tokens on the underlying graph, and a transformation can be the operation of sliding a token along an edge such that the resulting set of tokens is still pairwise non-adjacent. We call this operation *token sliding*. An intuitive way to formulate these settings is through the use of *reconfiguration graphs*, which are often also referred to as *flip graphs*. Here, the vertex set is the set of all configurations, and two vertices are connected if the corresponding configuration of one can be transformed into that of the other by one operation. The flip graphs for our two running examples are shown in Figure 1.1.

Finally, the question of the problem can be viewed as a question on the reconfiguration graph. The common questions are as follows:

- **Connectivity:** Can any configuration be transformed into another?
- **Diameter:** If the answer to the connectivity question is yes, what is the maximum number of transformations required to change a configuration to another?
- **Reachability:** If the answer to the connectivity question is no, can a given configuration be transformed into another

given configuration?

- **Shortest path:** What is the smallest number of transformations required to change a given configuration to another?

These questions can be structural (e.g., what the diameter of a flip graph is) or algorithmic (e.g., how to compute the diameter).

Let us consider some of these questions on our two examples. We start with the flip graph of the triangulations of an n -gon. Firstly, the flip graph is always connected, because it is the skeleton of the $(n - 3)$ -dimensional associahedron, assuming $n \geq 3$ (e.g., see [114]). In fact, this skeleton is the flip graphs of many objects, such as binary trees by tree rotations and Dyck paths by hill flips, and also appears in many other contexts in geometry [44]. We will give yet another of its realization in this thesis, when we discuss the quotientopes in Chapter 7. Secondly, the diameter question is known: Pournin [135] showed that the n -dimensional associahedron has the diameter of $2n - 4$ for $n > 9$. However, despite its connection with many contexts, the complexity of computing the shortest path between two vertices of an associahedron is still open.

The situations are generally more difficult for the k -sized independent sets of a graph G . The corresponding reconfiguration graph is not always connected (see [14] for a summary of known results). Even when G is a split graph, deciding the connectivity of the reconfiguration graph is coNP-hard [33]. Moreover, the shortest path problem is NP-complete [101]. The reachability problem is even harder to solve: it is PSPACE-complete, even when G is planar [86] or bipartite [118].

We note that the above examples are only a small fraction of the literature on reconfiguration problems. Even for independent sets, we can study many different other settings, such as labelled tokens and different transformations (e.g., see [29, 56]). For more examples, see the aforementioned surveys [166, 126].

While combinatorial games and puzzles such as the Rubik’s cube are a rich source of reconfiguration problems (see [87] for an extensive survey), we can find reconfiguration problems in many other contexts. For example, they naturally arise from the study of local changes in combinatorial objects, such as rotations of binary trees [120] and flipping of planar and plane graphs (e.g., see [123, 168]). Many other reconfiguration problems are derived from the structural examinations of the solution space of many graph problems, such as independent set, vertex cover, dominating set, clique, shortest path, maximum cut, etc.

These problems can have applications in many fundamental tasks. For example, we can solve the sampling task by defining a suitable Markov chain on a reconfiguration graph. If the chain satisfies certain property called *rapid mixing*, performing a random walk on the graph for a reasonable number of steps can give an almost uniformly random configuration. Further, being able to sample this way also allows us to approximately count the configurations. See [98] for full details on this method. In this thesis, we will discuss another important task: exhaustive generation, which can also be defined as a reconfiguration problem.

Reconfiguration also has connections with local search problems. The general method is as follows: we assign values to the configurations (or as discussed before, solutions to a source problem) typically to measure their “quality”, and in each step, we apply a transformation to obtain a configuration with better value. This corresponds to a walk on the reconfiguration graph with increasing values. Examples of such methods are the popular k -OPT heuristic for the travelling salesperson problem (e.g., see [23]) and the simplex methods for linear programming [55].

In this thesis, we will explore two aspects under the framework of the reconfiguration problems. The first part concerns about a problem called ARRIVAL, which decides whether a deterministic process

terminates. The problem can be phrased as a reachability question on a state space. A special feature of the problem is that it has a directed reconfiguration graph. Many examples above have reversible transformations, such as face turns of the Rubik's cube, token sliding for independent sets, or edge flips of triangulations. Hence, the reconfiguration graphs for these examples are naturally undirected. In contrast, the transformation between two adjacent configurations of ARRIVAL only works in one direction. (See the connection with reconfiguration problems in Section 2.4.1.) The main goal for this problem is to resolve its complexity status, that is specifically, whether it is in P.

The second part is related to one of the aforementioned applications of the reconfiguration problems: exhaustive generation. In particular, we consider the *Gray codes* of many sets of configurations, which are essentially sets of combinatorial objects. Originally, the term “Gray code” refers to the binary-reflected Gray code named after Frank Gray [81]. It is a cyclic sequence of all bitstrings of size n such that two adjacent bitstrings differ only by one bit. Nowadays, “Gray code” is a more general term for a listing of combinatorial objects, where two consecutive elements differ by small changes (see surveys by Savage [151] and Mütze [125]). Naturally, if we consider the reconfiguration graph where these small changes are the transformations, then a Gray code translates to a *Hamilton path* or sometimes a *Hamilton cycle*, a path or a cycle that visits every vertex exactly once. We will provide a general framework for generating such a Gray code for many classes of combinatorial objects.

We will now introduce these two parts in more detail.

1.1 ARRIVAL

Dohrau, Gärtner, Kohler, Matoušek, and Welzl [62] describes the ARRIVAL problem informally as follows:

Suppose that a train is running along a railway network, starting from a designated origin, with the goal of reaching a designated destination. The network, however, is of a special nature: every time the train traverses a switch, the switch will change its position immediately afterwards. Hence, the next time the train traverses the same switch, the other direction will be taken, so that directions alternate with each traversal of the switch.

Given a network with origin and destination, what is the complexity of deciding whether the train, starting at the origin, will eventually reach the destination?

Variants of this simple setting have been discovered and rediscovered in many contexts. Most notable is the study of deterministic simulations of a random walk under different names, such as *Eulerian walkers* [136], *rotor-router walks* [93], and *Propp machines* [53]. It also naturally arises from combinatorial games. The ARRIVAL problem, for example, is inspired by an online game called *Looping Piggy* developed by Gärtner for *Kinderlabor*, an initiative for computer science education in kindergartens. The latest reporting of a similar problem is in a blog post by Aaronson [1], also motivated by a children's game. As the author pointed out in the post, this in turn is a rediscovery of a result back in 1994 [46].

In the first context, the main focus has been to examine the close connection between this deterministic simulation and the corresponding random walk. In the second context, the interests lie primarily on the complexity of these games, which shares some flavor of the automata theory: Can these simple models be used to solve complex problems? Towards this end, the reachability question as quoted above has an interesting complexity status. It is shown in [62] that the problem is in $\text{NP} \cap \text{coNP}$. Subsequent papers have shown more specific results, such as the membership in $\text{UP} \cap \text{coUP}$ [76], NL-hardness [71], and CC-hardness [122], but one key question remains

wide open: Is ARRIVAL in P?

Two common approaches to tackle this question are finding a faster algorithm for a general instance and expanding the classes of graphs where a polynomial-time algorithm for ARRIVAL exists. On the first approach, the benchmark is the trivial algorithm of simply simulating the train run. On a graph with n vertices, Dohrau et al. [62] showed an upper bound of $\mathcal{O}(n2^n)$ steps and a lower bound of $\Omega(2^n)$. Since then, there have been other algorithms, but all of them have exponential runtime. Before the main result in this thesis, the best algorithm by Rote [149] runs in time $\mathcal{O}(p(n)2^{n/3})$, for some polynomial p . On the second approach, Haslebach [85] examined many classes of graph on which the trivial simulation runs in polynomial time. Recently, Auger, Coucheney, and Duhaze [10] considered the multigraph and higher out-degree variant of the problem and designed a nontrivial polynomial-time algorithm for the class of graphs whose underlying undirected simple graphs are trees.

This thesis contributes towards both approaches. The main result is a subexponential algorithm for general instances with the runtime of $2^{\mathcal{O}(\sqrt{n} \log n)}$. This is an instantiation of a general framework that can be tailored to produce other algorithms for ARRIVAL. One such algorithm achieves polynomial runtime for graphs that are close to acyclic. The main idea of the framework is to guess an NP or coNP certificate systematically. The approach of obtaining such a certificate also sets this framework apart from most of other algorithms to-date.

Additionally, we can solve in polynomial time another subclass of the ARRIVAL problem. This subclass captures a hard case of the above subexponential algorithm and also the other deterministic algorithms. We can exploit the structure of the instances in this subclass to derive a recursive procedure, where at every iteration, we can either decide the original problem or reduce the size of the underlying graph.

All of the results above rely on a generalization of ARRIVAL, called G-ARRIVAL. In this *multi-run* variant, we allow multiple trains, multiple origins, and multiple destinations. This variant contains ARRIVAL as a special case and has been discussed in a different form in the original paper [77], on which this part of the thesis is based on. Here, we expand the discussion about this generalized problem and show that it shares the same decision and search complexity with ARRIVAL. Moreover, it turns out that this generalization does not only allow us to design faster algorithms to solve ARRIVAL but also even G-ARRIVAL itself.

1.2 Combinatorial generation via permutation languages

Combinatorial algorithms constitute an important part of computer science. This is not least due to the ubiquity of the underlying combinatorial objects, such as graphs, binary strings, and permutations, in the field. In his seminal series *The Art of Computer Programming*, Knuth dedicated a volume to these algorithms and especially devoted a significant portion to exhaustive generation, one of the fundamental tasks on the combinatorial objects [108]. However, despite the vast number of algorithms, the literature still lacks a unifying theory for exhaustive generation.

As a contribution toward this goal, we present a general and versatile algorithmic framework that can generate a wide range of combinatorial objects. The framework requires the objects to be encoded by permutations and provides a simple greedy algorithm, called Algorithm J, to generate these permutations. We characterize a closure property that serves as a sufficient condition for a set of permutations to be successfully generated Algorithm J. As the condition is very mild, the number of such sets are double-exponential in the size of the permutations.

We highlight two key features of our framework. The first one is related to the complexity of the algorithm. The ultimate goal of exhaustive generation, besides listing all objects of a set, is to take little (ideally constant) time to produce each object. In order to do so, consecutive objects should differ only by a small change, or in other words, the listing should be a Gray code. In our framework, the listing of permutations generated by Algorithm J translates to a Gray code of the original objects, and the changes between two consecutive objects are smallest in a provable sense.

The second feature of the framework is its wide variety of applications. As noted above, Algorithm J can generate a double-exponential number of sets of permutations. Since many of these permutations can be used to encode objects, this implies a large number of combinatorial objects that can be generated by the framework. In particular, we recover the following four classical Gray codes: (1) the Steinhaus-Johnson-Trotter algorithm to generate all permutations of $[n] := \{1, 2, \dots, n\}$ by adjacent transpositions, also known as plain change order [162, 99]; (2) the binary reflected Gray code (BRGC) to generate all binary strings of length n by flipping a single bit in each step [81]; (3) the Gray code for generating all n -vertex binary trees by rotations due to Lucas, Roelants van Baronaigien, and Ruskey [121]; (4) the Gray code for generating all set partitions of $[n]$ by exchanging an element in each step due to Kaye [104]. Furthermore, the framework also allows us to generate many new classes of objects. In this thesis, we present two main applications.

The first application pertains to pattern-avoiding permutations. The framework can generate permutations avoiding certain classical patterns, vincular and bivincular patterns [16, 36], barred patterns [170], boxed patterns [12], Bruhat-restricted patterns [172], mesh patterns [37], monotone and geometric grid classes [94, 5], and many more. The new Gray codes for these permutations also translate to Gray codes for all the combinatorial objects that can be encoded

by them, including the five different types of geometric rectangulations [2, 144, 8, 43], also known as floorplans, which are tilings of a square by a fixed number of rectangles.

The second application of our framework is the generation of the lattice congruences of the weak order on permutations. These objects are found at the meeting point of combinatorics, discrete geometry, and algebra, and have been extensively studied in recent years [143, 146, 147]. Lattice congruences define lattice quotients, which take many well-known lattices as special cases, such as the Boolean lattice, the Tamari lattice [157], and certain Cambrian lattices [142, 47]. Moreover, the cover graphs of these lattice quotients can be realized as the skeletons of certain polytopes, called quotientopes [134], which generalize hypercubes, associahedra, permutahedra, etc. For each lattice congruence, our algorithm generates a listing of its equivalence classes, which corresponds to a Hamilton path on the cover graph of these corresponding lattice quotients and the skeleton of the corresponding quotientope.

The above two applications form the first two installments of a paper series [83, 89]. The other installments provide additional applications of the frameworks, such as elimination trees [42] and different types of rectangulations (including the aforementioned geometric rectangulations) [124].

1.3 Outline and summary of contributions

The remainder of the thesis is divided into two parts, which cover the two topics introduced above and can be read independently from each other.

The first part addresses the ARRIVAL problem and is primarily based on unpublished joint work with Bernd Gärtner and the following paper published in peer-reviewed proceedings:

- *A subexponential algorithm for ARRIVAL* by Bernd Gärtner, Sebastian Haslebacher, and Hung P. Hoang [77].

This part consists of three chapters. In Chapter 2, we formally define ARRIVAL and its generalization, G-ARRIVAL and establish G-ARRIVAL within the literature of Abelian networks, which provides the foundation for many results on this problem. We also extend the decision and search complexities of ARRIVAL to G-ARRIVAL in this chapter. Next, Chapter 3 presents a general framework to solve G-ARRIVAL and two applications of this framework, a subexponential algorithm for general instances and a polynomial-time algorithm for a subclass of almost acyclic graphs. Finally, Chapter 4 discusses another subclass that is hard for some existing algorithms and how we can take advantage of its structure to design a polynomial-time algorithm.

The second part of the thesis concerns with the exhaustive generation of combinatorial objects and is based on the following published papers in peer-reviewed journals:

- *Combinatorial generation via permutation languages: I. Fundamentals* by Elizabeth Hartung, Hung P. Hoang, Torsten Mütze, and Aaron Williams [83],
- *Combinatorial generation via permutation languages: II. Lattice congruences* by Hung P. Hoang and Torsten Mütze [89].

An extended abstract of these two papers was published in the proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms [84]. This part also consists of three chapters. Chapter 5 introduces the general framework for exhaustive generation and explains how four classical Gray codes arise as special cases. The last two chapters, Chapters 6 and 7, are dedicated to the two applications in pattern-avoiding permutations and lattice congruences of the weak order of permutations. In the latter chapter, we also discuss a few graph theoretical results for these lattice congruences.

1.4 Notations

In this section, we introduce a few common notations that are used throughout this thesis. Other specific notations will be introduced as they are required.

We denote by \log the binary logarithm and by \ln the natural logarithm. For any two real numbers a and b with $a \leq b$ we define $[a, b] := \{\lceil a \rceil, \lceil a \rceil + 1, \dots, \lfloor b \rfloor\}$ as the set of all integers between (and including) $\lceil a \rceil$ and $\lfloor b \rfloor$. We denote $]a, b[:= [a, b] \setminus \{a, b\}$ and introduce the abbreviation $[n] := [1, n]$. We use ε to denote the empty permutation or the empty string.

Next, we discuss a few terminologies for directed graphs. For a directed edge (u, v) from a vertex u to a vertex v , u and v are called the *tail* and the *head* of the edge, respectively. We say v is an *out-neighbor* of u and u an *in-neighbor* of v . The edge (u, v) is then an *outgoing edge* from u and an *incoming edge* to v . The *out-degree* and *in-degree* of a vertex v are the numbers of outgoing edges from and incoming edges to v , respectively. A vertex with in-degree 0 is a *source*, and a vertex with out-degree 0 is a *sink*. Unless otherwise stated, all directed graphs in this thesis are simple (i.e., there are no loops and no multiple edges with the same head and tail).

For a vertex v and a set of edges E in a directed graph, we denote the set of outgoing edges of v by $E^+(v)$. Analogously, we denote the set of incoming edges of v by $E^-(v)$. Furthermore, for a function $x : E \rightarrow \mathbb{R}$, we also use the notation x_e instead of $x(e)$ to denote the value of x at some edge $e \in E$. Lastly, given some vertex v , edges E and a function $x : E \rightarrow \mathbb{R}$, we use $x^+(v) := \sum_{e \in E^+(v)} x_e$ to denote the *outflow* of x at v and $x^-(v) := \sum_{e \in E^-(v)} x_e$ to denote the *inflow* of x at v . For two functions $x, x' : E \rightarrow \mathbb{R}$, we write $x \leq x'$ if this holds componentwise, i.e. $x_e \leq x'_e$ for all $e \in E$.

For undirected graphs, we only discuss simple graphs (i.e., with no loops and no multiple edges between two vertices), unless stated otherwise. The *degree* of a vertex in an undirected graph is the number of edges incident to the vertex. An undirected graph G is *regular*, if all vertices have the same degree. It is *vertex transitive*, if for any two vertices u, v of G , there is an automorphism $f : G \rightarrow G$, such that $f(u) = v$. It is *k -partite* or *k -colorable*, for some $k \geq 2$, if the vertices of the graph can be partitioned into k disjoint subsets, such that no two vertices are adjacent in each subset. When $k = 2$, we call the graph bipartite.

Lastly, \mathbb{N}_0 denotes the set of all natural numbers (including 0), and \mathbb{R} the set of all real numbers.

Part I

ARRIVAL

"Men," said the little prince, "set out on their way in express trains, but they do not know what they are looking for. Then they rush about, and get excited, and turn round and round . . ."

—Antoine de Saint-Exupéry,
The Little Prince

CHAPTER 2

ARRIVAL and its generalization

Except for Sections 2.3 and 2.7, which have not been published, this chapter is based on [77], which is joint work with Bernd Gärtner and Sebastian Haslebacher.

The ARRIVAL problem was introduced by Dohrau et al. [62] as the problem of deciding whether the train arrives at a given destination or runs forever. Here, we work in a different but equivalent setting (implicitly established by Dohrau et al. already) in which the train always arrives at one of two destinations, and we have to decide at which one. The definitions and results from Dohrau et al. [62] easily adapt to this setting. We will first formally define the ARRIVAL problem. We will then introduce a generalization called G-ARRIVAL, where there are multiple trains, multiple origins, and multiple destinations. This generalization will be the focus for the rest of the chapter. Specifically, we place it in the broader frame-

work of Abelian networks (Lemma 2.8) and hence, the problem inherits an important “Abelian property” (Corollary 2.16) that is useful for later algorithmic results. We also extend the current upper bounds in terms of decision and search complexities of ARRIVAL to G-ARRIVAL (Theorems 2.21 and 2.33).

2.1 ARRIVAL

Given a finite set of vertices V , an origin $o \in V$, two destinations $d, \bar{d} \notin V$ and two functions $s_{\text{even}}, s_{\text{odd}} : V \rightarrow V \cup \{d, \bar{d}\}$, the 6-tuple $A = (V, o, d, \bar{d}, s_{\text{even}}, s_{\text{odd}})$ is an *ARRIVAL instance*. The vertices $s_{\text{even}}(v)$ and $s_{\text{odd}}(v)$ are called the even and the odd successor of v .

An ARRIVAL instance A defines a directed graph, connecting each vertex $v \in V$ to its even and its odd successor. We call this the *switch graph* of A and denote it by $G(A)$. To avoid special treatment of the origin in certain analysis, we introduce an artificial vertex $Y \notin V \cup \{d, \bar{d}\}$ (think of it as the “train yard”) that only connects to the origin o . Formally, $G(A) = (V(A), E(A))$ where $V(A) = V \cup \{Y, d, \bar{d}\}$ and $E(A) = \{(Y, o)\} \cup \{(v, s_{\text{even}}(v)) : v \in V\} \cup \{(v, s_{\text{odd}}(v)) : v \in V\}$. We also refer to $E(A)$ simply as the edges of A . An edge $e \neq (Y, o)$ is called *proper*.

The *run procedure* is the following. For every vertex we maintain a current and a next successor, initially the even and the odd one. We put a token (usually referred to as the train) at o and move it along switch graph edges until it reaches either d or \bar{d} . Whenever the train is at a vertex v , we move it to v ’s current successor and then swap the current and the next successor; see Algorithm 1 for a formal description.

Algorithm 1 (Run procedure) may cycle, but we can avoid this by assuming that from every vertex $v \in V$, one of d and \bar{d} is reachable along a directed path in $G(A)$. We call such an ARRIVAL in-

Algorithm 1 (*Run procedure*). Given an ARRIVAL instance $A = (V, o, d, \bar{d}, s_{\text{even}}, s_{\text{odd}})$, this algorithm decides if the destination of the train is d or \bar{d}

1. [*Initialize*] Let s_{curr} and s_{next} be arrays indexed by the vertices of V . For $v \in V$, set $s_{\text{curr}}[v] \leftarrow s_{\text{even}}(v)$ and $s_{\text{next}}[v] \leftarrow s_{\text{odd}}(v)$.
2. [*Traverse the edge* (Y, o)] Set $v \leftarrow o$.
3. [*Route train by one step*] While $v \neq d$ and $v \neq \bar{d}$,
 - 3.1. Set $w \leftarrow s_{\text{curr}}[v]$;
 - 3.2. Swap $s_{\text{curr}}[v]$ and $s_{\text{next}}[v]$;
 - 3.3. Set $v \leftarrow w$.
4. [*Output*] Return v .

stance *terminating*, since it guarantees that either d or \bar{d} is eventually reached, as shown later in Lemma 2.15 (for the generalization).

The ARRIVAL problem is to decide whether the train reaches d , i.e., whether Algorithm 1 (Run procedure) returns d .

Related work on complexity. In the introductory paper on ARRIVAL, Dohrau et al. [62] showed an upper bound of $\text{NP} \cap \text{coNP}$. This bound could be strengthened in various ways. ARRIVAL is in $\text{UP} \cap \text{coUP}$, meaning that there are efficient verifiers that accept *unique* proofs [76]. A search version of ARRIVAL has been introduced by Karthik C. S. and shown to be in PLS [103], then in CLS [76], and finally in UEOPL [76, 73]. The latter complexity class, established by Fearnley, Gordon, Mehta, and Savani [73], has an intriguing complete problem, but there is no evidence that ARRIVAL is complete for UEOPL.

Concerning lower bounds, ARRIVAL was first shown to be NL-hard [72]. This is not a very strong statement and means that every

problem that can be solved by a nondeterministic log-space Turing machine reduces (in log-space) to ARRIVAL. Recently, Manuell [122] strengthened this result to CC-hardness. CC is a complexity class between NL and P and contains a decision variant of stable marriage problem as a complete problem. He also showed that it is a hard problem for PL (probabilistic log-space).

Observe that ARRIVAL is a *zero player game*, i.e., a process that runs without a controller. Much more interesting complexity results are for the natural one- and two-player variants of ARRIVAL that have been introduced in the paper above by Fearnley et al. [72] and later expanded by Ani, Demaine, Hendrickson, and Lynch [7]. The one-player variants of ARRIVAL are NP-complete, and the two-player variants are PSPACE-hard [7, 72].

We can contrast these results with three well-known graph games in $\text{NP} \cap \text{coNP}$ that are not known to be in P, namely *simple stochastic games*, *mean-payoff games* and *parity games* [52, 176, 100]. These are two-player games. Moreover, it is stated in (or easily seen from) these papers that the one-player variants (the strategy of one controller is fixed) have polynomial-time algorithms. This shows that the p -player variant of ARRIVAL is probably strictly harder than the p -player variants of these graph games, for $p = 1, 2$. This makes it a bit less surprising that ARRIVAL itself ($p = 0$) could so far not be shown to lie in P.

2.2 G-ARRIVAL

We now define a generalized version of the ARRIVAL problem, where we allow multiple trains, multiple origins, and multiple destinations. A formal definition is as follows, where we use similar terminologies as for ARRIVAL.

Definition 2.1 (Switching system). *A switching system is a 5-tuple $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$, for*

- *a finite set of vertices $V = \{v_1, \dots, v_n\}$,*
- *a finite set of destinations $D = \{d_1, \dots, d_{|D|}\}$ with $D \cap V = \emptyset$,*
- *a supply function $s : V \rightarrow \mathbb{N}_0$ that indicates the number of starting trains at each vertex, and*
- *two functions $s_{\text{even}}, s_{\text{odd}} : V \rightarrow V \cup D$ that indicate the even and odd successors of each vertex in V , respectively.*

Definition 2.2 (Switch graph). *Given a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$, the train yard of A is an artificial vertex $Y = Y(A)$ that is connected to all vertices in V . The switch graph of A denoted by $G(A)$ is $(V(A), E(A))$, where $V(A) = \{Y\} \cup V \cup D$ and $E(A) = \{(Y, v) : v \in V\} \cup \{(v, s_{\text{even}}(v)) : v \in V\} \cup \{(v, s_{\text{odd}}(v)) : v \in V\}$. The edges in $E(A)$ are also called the edges of A . The proper edges are those that are not incident to Y .*

Definition 2.3 (Terminating switching system). *A switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ is terminating, if from every vertex $v \in V$, one of the vertices in D is reachable along a directed path in the switch graph of A .*

Note that we can define a switching system without using the train yard Y , and we indeed do not need this artificial vertex in most arguments. However, the train yard does help to ease the technicalities in a few places, such as in the proof of Theorem 2.33 in this chapter and as discussed in Section 3.7 in the next chapter.

Since every vertex in V has two outgoing edges, we can assume $|D| \leq 2n$, where $n = |V|$. Further, let $S = \sum_{v \in V} s(v)$ indicate the total number of starting trains. We assume $S \geq 1$, because otherwise, there is nothing to do. Moreover, we assume the encoding size of

S to be in $\mathcal{O}(\text{poly}(n))$, i.e., $\log S = \mathcal{O}(\text{poly}(n))$. In that way, when we talk about a polynomial-time algorithm for a problem involving a switching system, we mean that the runtime of the algorithm is polynomial in n .

Analogous to the run procedure, we describe the *multi-run procedure*, as follows. For each vertex v in V , we put $s(v)$ trains at v . By this, we mean that all S trains are at Y at the beginning, and we send $s(v)$ trains along the edge (Y, v) . Note that after this step, no trains are at Y , and since Y has no incoming edges, no train will reach Y . Trains that are now on vertices in V are called *waiting* (to move on). For every vertex, we initialize current and next successors as before in Algorithm 1 (Run procedure). Then we (nondeterministically) repeat the following until there are no more trains waiting.

We pick a vertex $v \in V$ where some trains are waiting and call the number of waiting trains $t(v)$. We choose a number $\rho \in \{1, \dots, t(v)\}$ of trains to move on; we move $\lceil \rho/2 \rceil$ of them to the current successor and $\lfloor \rho/2 \rfloor$ to the next successor. If ρ is odd, we afterwards swap the current and the next successor at v . We call the above operation *routing of ρ trains from v* . See Algorithm 2 for a formal description. Note that the algorithm considers as input a terminating switching system to ensure its termination, which will be shown subsequently in Lemma 2.15. However, the routing operation in general can be applied to non-terminating switching systems.

The output of the procedure is a tuple $(t[d_1], \dots, t[d_{|D|}])$, where each $t[d_i]$ indicates the number of trains arriving at d_i , for $i \in [|D|]$.

Definition 2.4 (G-ARRIVAL). *An instance of the G-ARRIVAL problem is (A, T) , for a terminating switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ and a tuple $T = (t_1, \dots, t_{|D|})$. The goal of the problem is to decide if there is an execution of Algorithm 2 (Multi-run procedure) with input A and output T .*

Algorithm 2 (*Multi-run procedure*). Given a terminating G-ARRIVAL instance $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ for $D = \{d_1, \dots, d_{|D|}\}$, this algorithm outputs the number of trains arriving at each vertex in D .

1. [*Initialize*] Let s_{curr} and s_{next} be arrays indexed by the vertices of V . For $v \in V$, set $s_{\text{curr}}[v] \leftarrow s_{\text{even}}(v)$ and $s_{\text{next}}[v] \leftarrow s_{\text{odd}}(v)$. Moreover, let t be a zero-initialized array indexed by the vertices of $V \cup D$.
2. [*Traverse edges from Y*] For $v \leftarrow V$, $t[v] \leftarrow s_v$.
3. [*Select*] Pick $v \in V$ such that $t[v] > 0$ and choose $\rho \in [t[v]]$.
4. [*Route ρ trains from v*] Do the following:
 - 4.1. Set $t[v] \leftarrow t[v] - \rho$;
 - 4.2. Set $t[s_{\text{curr}}(v)] \leftarrow t[s_{\text{curr}}(v)] + \lceil \rho/2 \rceil$;
 - 4.3. Set $t[s_{\text{next}}(v)] \leftarrow t[s_{\text{next}}(v)] + \lfloor \rho/2 \rfloor$;
 - 4.4. If ρ is odd, swap $s_{\text{curr}}[v]$ and $s_{\text{next}}[v]$.
5. [*Repeat*] If $\exists w \in V : t[w] > 0$, go to 3.
6. [*Output*] Return $(t[d_1], \dots, t[d_{|D|}])$.

If $\sum_{i \in [|D|]} t_i \neq S$, then we know for certain that this is a NO instance. Therefore, we always assume that $\sum_{i \in [|D|]} t_i = S$.

Note that in Algorithm 2 (Multi-run procedure), the trains are generally considered as unlabelled. However, for convenience, we sometimes treat them as labelled trains, and we can identify which trains we route from a vertex. That way, we can trace the sequence of vertices that a train visits from its starting vertex until it reaches a vertex in D .

Simple switch graph. Although we allow the switch graph of A to be a multi-graph, we can assume WLOG. that it is a simple graph, when A is terminating. We show this by considering the

three following cases that make $G(A)$ non-simple:

- If there exists $v \in V$ such that $s_{\text{even}}(v) = s_{\text{odd}}(v) = v$, this contradicts the assumption that the system is terminating.
- If there exists $v \in V$ such that $s_{\text{even}}(v) = s_{\text{odd}}(v) = u \neq v$, a train after reaching v will always go to u . Hence, we can contract v to u .
- If there exists $v \in V$ such that $\{s_{\text{even}}(v), s_{\text{odd}}(v)\} = \{v, u\}$, $u \neq v$, the train after reaching and looping at v will always go to u . Hence, we can contract v to u .

2.3 Abelian networks

Observe that Algorithm 1 (Run procedure) is completely deterministic, and hence, the ARRIVAL problem is well-defined. On the contrary, Algorithm 2 (Multi-run procedure) is nondeterministic. However, we proved in [77] that the output of this procedure is always the same, regardless of the choices during the execution of the multi-run procedure. It turns out that it was a rediscovery of a well-known “Abelian property” of more general models called Abelian networks. In this section, we introduce these models and show why the switching systems are one of them. We emphasize that this Abelian property is a key ingredient in the algorithm design for switching systems, as it allows us to prescribe freely a routing order of trains.

2.3.1 Abelian networks

We use the definition of an Abelian network by Bond and Levine [34]. Such a network contains automata that live at the vertices of a directed graph and communicate through the edges. It produces an output uniquely determined by the input and independent from the processing order of the automata. We note that these automata are

not the common finite-state automata, which recognize formal languages. Here, each automaton receives a string as the input, processes the letters of the string on a first-in-first-out basis, and produces many outputs. Each of the outputs, which is a string, is then appended to the input string of another automaton. In this sense, it is more related to a finite-state transducer. We call these automata *processors* and describe them more formally as follows.

Processor. Given a directed graph G , we associate each vertex v of G with a processor \mathcal{P}_v , which has one output port for each outgoing edge from v and a single input port. We can view the input port as containing a string, where any new string arriving at the port will be concatenated on the right of the existing string to form a new string. The processor then reads the letters in its input port from left to right. Typically, it reads one letter at a time, but we also allow it to read a substring, as this notion will be helpful in Definition 2.5 below.

The processor \mathcal{P}_v has an input alphabet A_v and a state space Q_v . For each out-neighbor u of v , the output alphabet of \mathcal{P}_v associated to the edge (v, u) is A_u . Let A_v^* be the free monoid of all finite words in the alphabet A_v . The *processing* of \mathcal{P}_v is defined by two functions. Firstly, a *transition function* $T_v : A_v^* \times Q_v \rightarrow Q_v$ computes the next state of the processor. Secondly, *message passing functions* $T_{(v,u)} : A_v^* \times Q_v \rightarrow A_u^*$ computes the output sent to each out-neighbor u of v . As the processor reads the input letters from left to right, these two functions must satisfy the following: For a word $w = aw'$ in A_v^* , a state q in Q_v , and an out-neighbor u of v , we have

$$\begin{aligned} T_v(w, q) &= T_v(w', T_v(a, q)), \\ T_{(v,u)}(w, q) &= T_{(v,u)}(a, q) \cdot T_{(v,u)}(w', T_v(a, q)), \end{aligned} \tag{2.1}$$

where the multiplication here denotes concatenation of words.

Definition 2.5 (Abelian processor [34]). *The processor \mathcal{P}_v is called Abelian, if for any words $w, w' \in A_v^*$ such that w is a permutation of w' , and for all $q \in Q_v$ and all out-neighbor u of v , we have*

- $T_v(w, q) = T_v(w', q)$, and
- $T_{(v,u)}(w, q)$ is a permutation of $T_{(v,u)}(w', q)$.

In words, permuting the letters inputted to \mathcal{P}_v results in the same final state and a permutation of the letters outputted to the processor \mathcal{P}_u of each neighbor u (but otherwise, the letters received by \mathcal{P}_u are unchanged).

Definition 2.6 (Abelian network [34]). *An Abelian network on the directed graph G is a collection of Abelian processors, one on each vertex of G .*

Processing of the network. Note that in the definition of an Abelian network above, we only have a local requirement about each individual processor in the network. We now describe the global processing of the network. The network processes sequentially, where at each step, an arbitrary processor processes a substring in its input port. The processing of the network stops when all processors have processed all the letters in their input ports. Observe that the order of processing of the network changes the order in which the letters arrive at an individual processor. However, if the network is Abelian, then we have the following “Abelian property”:

Theorem 2.7 (Abelian property of Abelian networks). [34, Theorem 4.6] *The following aspects of an Abelian network do not depend on the order of processing of the network:*

- *The halting status (i.e., whether the processing eventually stops),*
- *The final states of the processors,*

- The runtime (i.e., the total number of letters processed by all processors),
- The local runtimes (i.e., the number of letters processed by a given processor), and
- The detailed local runtimes (i.e., the number of times a given processor processes a given letter in its input alphabet).

2.3.2 Switching systems are Abelian networks

As noted in [34], a *unary processor* (i.e., a processor with input alphabet of cardinality 1) is trivially Abelian. Therefore, a network of unary processors is Abelian. We call such a network *unary*. In this section, we show that switching systems are unary and hence Abelian (Lemma 2.8).

Lemma 2.8. *A switching system is a unary network and hence is Abelian and has the Abelian property in Theorem 2.7.*

Proof. Suppose $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ is a switching system, with $V = \{v_1, \dots, v_n\}$. Let $S = \sum_{v \in V} s(v)$. We construct the processors on the switch graph $G(A)$ as follows. At each vertex $v \in V(A)$, the processor \mathcal{P}_v has an input alphabet consisting of a single letter a . The state space Q_v of \mathcal{P}_v is $\{0, 1\}$ for $v \in V$, and $[0, S]$ for $v \in D \cup \{Y\}$. The starting states of all processors are 0.

We define the sequence (b_0, b_1, \dots, b_n) , such that $b_0 = 0$ and $b_i = \sum_{j=1}^i s(v_j)$ for $i \in [n]$. Given a vertex v in $V(A)$ and a state $q \in Q_v$, the transition function is

$$T_v(a, q) = \begin{cases} q + 1 \pmod{2} & \text{if } v \in V, \\ \min\{q + 1, S\} & \text{if } v \in D \cup \{Y\}. \end{cases}$$

Given a vertex v in V , a state $q \in Q_v$, and an out-neighbor u of v ,

the message passing function is

$$T_{(v,u)}(a, q) = \begin{cases} a & \text{if } q = 0 \text{ and } u = s_{\text{even}}(v), \\ a & \text{if } q = 1 \text{ and } u = s_{\text{odd}}(v), \\ \varepsilon & \text{otherwise.} \end{cases}$$

Lastly, given $v_i \in V$ and a state $q \in Q_Y$,

$$T_{(Y,v_i)}(a, q) = \begin{cases} a & \text{if } q < S \text{ and } b_{i-1} \leq q < b_i, \\ \varepsilon & \text{otherwise.} \end{cases}$$

Observe that for input words of more than one letter, the values of the transition functions and message passing functions are defined by (2.1). Since there are no outgoing edges from a vertex v in D , there are no message passing functions associated with \mathcal{P}_v , and we can view the output of the processor to be always ε .

The states of 0 and 1 at a vertex $v \in V$ correspond to whether the current successor of v is the even and the odd successors, respectively. The state at Y represents the number of trains we have sent out from Y , where each time \mathcal{P}_Y processes, a train is sent from Y to a vertex v in V , and the sequence (b_0, b_1, \dots, b_n) plays a role at ensuring that at most $s(v)$ trains are sent to v . It is easy to see that we can simulate the routing of ρ trains from a vertex $v \in V$ by letting \mathcal{P}_v process ρ times. The number of waiting trains at each vertex v corresponds to the letters yet to be processed at the input to \mathcal{P}_v . Finally, the state at a destination in D counts the number of trains that have reached that destination. Based on the above correspondences, the theorem statement follows. \square

2.3.3 Other unary networks

Bond and Levine [34] also discuss two prominent unary networks that have a similar flavor as the switching systems. These networks are two simple combinatorial models that connect different branches

of mathematics, such as Tutte polynomial, random walk, and divisor theory in algebraic geometry. We introduce them briefly below and refer the reader to [34] for the proofs for why they are unary. These proofs are similar to that of Lemma 2.8 above.

Chip-firing. This is also called the Abelian sandpile model. An instance of chip-firing consists of a directed graph (V, E) and a *chip configuration*, a function $V \rightarrow \mathbb{N}_0$ that can be viewed as an allocation of tokens (commonly referred to as *chips*) to the vertices of the graph. If the number of chips at a non-sink v is at least its out-degree, the vertex can *fire*, transferring a chip along each incident edge. In other words, the number of chips at v decreases by the out-degree of v , while that at each out-neighbor of v increases by the in-degree from v . A *chip-firing game* is a sequence of firing, where at each step, we pick an arbitrary vertex that can fire and let it fire. The game halts if we get to a chip configuration where no vertex can fire.

Rotor-routing. On a high level, rotor-routing can be seen as a refined version of chip-firing and a deterministic simulation of a random walk on a directed graph. It has been studied under many names, such as Eulerian walkers [136] and Propp machine [53]. An instance of rotor-routing involves a ribbon graph and an initial chip-and-rotor configuration. A *ribbon graph* is a directed graph G with a fixed cyclic finite sequence of (not necessarily distinct) out-neighbors of each non-sink v . We denote this sequence by $(v^0, \dots, v^{d(v)-1})$, where $d(v)$ is a positive integer indicating the length of the sequence. A *rotor configuration* is a function r that maps each vertex v of G to a number in $[0, d(v) - 1]$. A *chip configuration* c is defined the same as for chip-firing. We can visualize that each vertex v has $c(v)$ chips and a *rotor* that points to $v^{r(v)}$. We call the pair (c, r) a *chip-and-rotor configuration*. A chip at a vertex v moves on G according to the *rotor-router operation*, defined as follows: if v has at least an

out-neighbor, we route the chip from v to $v^{r(v)}$ and increase $r(v)$ by one (mod $d(v)$). In other words, given the current chip-and-rotor configuration (c, r) and a vertex v where $c(v) > 0$, by performing the rotor-router operation on v , we obtain a new configuration (c', r') as follows:

$$c'(u) = \begin{cases} c(u) & \text{if } u \neq v, v^{r(v)}, \\ c(u) - 1 & \text{if } u = v, \\ c(u) + 1 & \text{if } u = v^{r(v)}. \end{cases}$$

$$r'(u) = \begin{cases} r(u) & \text{if } u \neq v, \\ r(u) + 1 \pmod{d(v)} & \text{if } u = v, \end{cases}$$

Note that in the literature on rotor-routing, the rotor-router operation changes the rotor before sending the chip to the out-neighbor that the new rotor is pointing. However, to compare easily with switching systems later, we define the rotor-router operation as above, where the chip is sent before the rotor is changed. This difference does not affect the comparability of the results in this thesis with other papers, as the two definitions of rotor-router operation are equivalent, subject to a cyclic shift in the cyclic sequence of the out-neighbors at every vertex.

Similar to a chip-firing game, at each step of a *rotor-routing game*, we pick an arbitrary non-sink that has at least a chip and perform the rotor-router operation on that vertex. The game halts if all non-sinks have no chips.

Lemma 2.9. [34] *Chip-firing and rotor-routing are unary networks and hence Abelian.*

2.4 Switching systems vs. rotor-routing

The setting of rotor-routing is very similar to switching systems. As mentioned earlier, rotor-routing have been studied under various

names, and switching systems are another variant. It is easy to see that switching systems are a special case of rotor-routing, since the main differences between rotor-routing and switching systems are as follows:

- A vertex in a switching system has at most two outgoing edges with a simple alternation between these two edges, while a vertex in a ribbon graph can have more outgoing edges and the order of the out-neighbours can be much more complicated.
- For the setting of a switching system, we are interested in graphs with at least a sink, whereas a ribbon graph can have no sinks. Observe that in a rotor-routing game, unless a chip reaches a sink, it will move forever.

However, the restrictions from the setting of switching systems can be imposed on a ribbon graph without loss of generality. Just like we can implement any Boolean functions with AND and NOT gates [91], we can “implement” rotor-routing with switching systems.

Lemma 2.10. *Given a rotor-routing instance, without loss of generality, we can assume that a ribbon graph has at least one sink, each non-sink has out-degree two, and its two neighbors appear exactly once in its cyclic sequence.*

Proof. Firstly, we can easily add an artificial sink in the graph without affecting the rotor-router operations on any vertex. More specifically, given a ribbon graph with no sink, we can add a sink d and a vertex w whose out-neighbors are d and an existing vertex in the graph. We put no chips on d and w . Since w has no incoming edges and is the only in-neighbor of d , no chips will come to and go from either of these two vertices.

Secondly, if there is a non-sink v that does not meet the requirements in the lemma statement, we can replace it with a few vertices that do. More precisely, we replace v with a complete binary tree. Suppose

the cyclic sequence at v has length $d(v)$. Then the depth of the binary tree is $\lceil \log d(v) \rceil$. In that way, there will be at least $d(v)$ (not necessarily distinct) leaves to serve $d(v)$ consecutive tokens entering the root. These leaves will correspond to the $d(v)$ outgoing vertices in the sequence of the original processor. For the remaining leaves, we connect them back to the root. Note that for each sequence of length $d(v)$, we add $\mathcal{O}(d(v))$ additional vertices and edges in the switching system. Hence, the encoding size only blows up by a linear factor. We can set the cyclic sequences of the new vertices such that the order of chips leaving the tree is exactly the same as the order in the original cyclic sequence at v . See Figure 2.1 for the illustration. \square

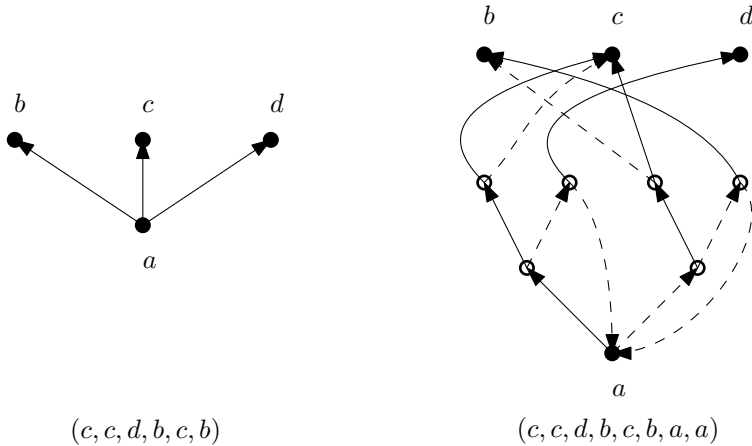


Figure 2.1: A vertex a with its out-neighbors and its cyclic sequence in a rotor-routing instance (left), and the corresponding replacement binary tree with the cyclic order of the leaves that consecutive chips leaving a will eventually visit (right). On the right, for each vertex, the chip will be routed via the solid edge before the dashed edge.

We remark that the technique of replacement by a binary tree above is also employed by Zwick and Paterson [177, Section 6] for a similar reduction for simple stochastic games. We also add that even though we can simulate a rotor-routing instance by a switching system, the resulting switching system is not necessarily terminating (i.e., there may be a vertex that does not have any directed path to a sink).

2.4.1 Abelian networks as reconfiguration graphs

The Abelian networks discussed so far easily fits into the framework of reconfiguration problems. We will look at some examples below.

G-ARRIVAL. To phrase G-ARRIVAL as a reconfiguration problem, we first need to define the configurations. Similar to a chip-and-rotor configuration of a rotor-routing instance, we can define analogously a train-and-switch configuration or simply a state of a switching system, as follows.

Definition 2.11 (State of a switching system). *Given switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$, a train-and-switch configuration or a state of A is a tuple (τ, σ) , where*

- *the function $\tau : V(A) \rightarrow \mathbb{N}_0$ called the train configuration records the number of trains on each vertex of $G(A)$ (including the train yard) and satisfies $\sum_{v \in V(A)} \tau(v) = \sum_{v \in V} s(v)$ (i.e., the total number of trains is preserved), and*
- *the function $\sigma : V \rightarrow \{0, 1\}$ called the switch configuration indicates whether the current successor of each vertex is the even successor (corresponding to the value 0) or the odd successor (corresponding to the value 1).*

We are now ready to define the reconfiguration graph. Suppose we have a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$, with $D = \{d_1, \dots, d_{|D|}\}$. The reconfiguration graph \mathcal{R}_A has all states of A as

its vertices, and the routing operations define its edges. Observe that the edges of \mathcal{R}_A are directed. Then, a G-ARRIVAL instance (A, T) with $T = (t_1, \dots, t_{|D|})$ asks a reachability question on \mathcal{R}_A . Specifically, we are given the starting state (τ_0, σ_0) , such that $\tau_0(v) = s(v)$ and $\sigma_0(v) = 0$ for $v \in V$, and $\tau_0(d) = 0$ for $d \in D$. Further, we are given a target set $\mathcal{T} = \{(\tau', \sigma') \mid \forall i \in [|D|], \tau'(d_i) = t_i\}$. The question is then whether there exists a directed path in \mathcal{R}_A from (τ_0, σ_0) to a state in \mathcal{T} . Note that in general, \mathcal{R}_A can have directed cycles. However, Lemma 2.15 and Corollary 2.16 in the next section imply that if the switching graph is terminating (i.e., from every vertex in V , there is a directed path in $G(A)$ to a destination in D), then \mathcal{R}_A is a union of directed rooted trees (i.e., in each tree, there is a vertex to which every other vertex has a directed path).

Rotor-routing reachability We can also ask the reachability question differently. For example, is there a directed path in \mathcal{R}_A from the starting state (τ_0, σ_0) to a given state (τ', σ') ? If the switching system is restricted to the setting of the ARRIVAL problem (i.e., there is only one train in the system), then a result by Gärtner, Hansen, Hubáček, Král, Mosaad, and Slívová [76] implies that this question can be answered in polynomial time, by solving a system of linear equations and verifying whether the solution of the system contains only natural numbers (see [76, Lemma 23]). Recently, Tóthmérész [160] extended this complexity to a more general problem: the *rotor-routing reachability problem*. In this problem, the reconfiguration graph has all possible chip-and-rotor configurations on a given ribbon graph as vertices and the rotor-router operations define the edges. Given two chip-and-rotor configurations, the problem asks to decide whether one can be reachable from the other in this reconfiguration graph.

Theorem 2.12. [160, Theorem 3.1] *The rotor-routing reachability problem is in P.*

Note that this result does not settle the complexity status of G-ARRIVAL or ARRIVAL. In the rotor-routing reachability problem, the instance includes an initial chip-and-rotor configuration and a target chip-and-rotor configuration. In the G-ARRIVAL and ARRIVAL problems, the instance includes an initial train-and-switch configuration and a target *train configuration*. That is, the target switch configuration is not part of the input. Hence, we have the set \mathcal{T} defined above. A naïve approach is to try all switch configurations (i.e., testing for every state in \mathcal{T}), but there are exponentially many of them.

Chip-firing reachability A similar problem has also been studied for chip-firing. The chip-firing reachability problem asks, given a directed graph and an initial chip configuration, whether we can reach another given chip configuration. The problem was shown to be in coNP [95] and is not in P unless $\text{NP} = \text{coNP}$ [160]. There are some possible intuitive reasons for why this problem is harder (in terms of complexity) than the rotor-routing reachability. For one, rotor-routing is more constrained, in the sense that its configuration involves also the rotor configuration in addition to the chip configuration. For another, the chips can move independently in rotor-routing, while in chip-firing, they have to gather sufficiently many at one place in order to move.

Does the result above on chip-firing reachability give some indication that there may be no polynomial algorithm for G-ARRIVAL or maybe even for ARRIVAL? The answer is no, since the complexities of the two problems are quite different. To show the result in [160], Tóthmérész first showed that if there exists a polynomial-time for the chip-firing reachability problem, then there is an efficient NO certificate for the chip-firing halting problem, i.e., the problem of deciding the halting status of a chip-firing game. In other words, the chip-firing halting problem is then in coNP . However, this problem is already known to be NP -complete, which implies $\text{NP} = \text{coNP}$. On

the other hand, the halting variant of the ARRIVAL problem has the same complexity as the reachability variant and is known to be in $\text{NP} \cap \text{coNP}$ [62]. We will show later on that G-ARRIVAL also has the same complexity. Therefore, the same argument for chip-firing cannot be used for G-ARRIVAL.

2.5 G-ARRIVAL is well-defined

By Lemma 2.8, the order of applying the routing operations does not affect the halting status of the switching system. However, this does not mean that the halting status is YES (i.e., Algorithm 2 (Multi-run procedure) terminates). In this section, we show that this must be the case, if the switching system is terminating (i.e., from every vertex in the switch graph, at least a destination is reachable); see Lemma 2.15 below. Additionally, due to the Abelian property (Lemma 2.8), the number of traversals on each edge is the same, regardless of how the procedure is executed (Corollary 2.16 below). Hence, G-ARRIVAL is well-defined. One important concept for showing Lemma 2.15 and also for future discussions is the switching flows, defined below.

2.5.1 Switching flows

Switching flows were defined for ARRIVAL in [62]. Here, the definition carries straightforwardly to G-ARRIVAL, as follows.

Definition 2.13 (Switching flow). *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system with edges E . A function $x : E \rightarrow \mathbb{N}_0$ is a switching flow for A if*

$$\begin{aligned} x_{(Y,v)} &= s(v), & v \in V \\ x^+(v) - x^-(v) &= 0, & v \in V \quad (\text{flow conservation}) \\ x_{(v,s_{\text{even}}(v))} - x_{(v,s_{\text{odd}}(v))} &\in \{0, 1\}, & v \in V \quad (\text{switching behavior}). \end{aligned}$$

Summing all the flow conservation constraints, we obtain the following simple observation.

Observation 2.14. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system and x be a switching flow for A . Then,*

$$\sum_{d \in D} x^-(d) = \sum_{v \in V} s(v).$$

In other words, all flows emitted by the vertices in V are absorbed by the destinations in D . If we set x_e to the number of times the edge e is traversed in Algorithm 2 (Multi-run procedure), we obtain a switching flow where the inflows at the destinations correspond to the output of the algorithm. Indeed, every time the train enters $v \in V$, it also leaves it; this yields flow conservation. The alternation between the successors (beginning with the even one) yields switching behavior. We call this value of x a *run profile* of the corresponding switching system. (In the context of rotor-routing, this is called the *odometer* in [34].) In principle, a switching system may have many run profiles. However, due to the Abelian property (Lemma 2.8), the run profile is unique; see Corollary 2.16 below.

2.5.2 Termination and unique run profile

We begin with the termination guarantee.

Lemma 2.15. *Algorithm 2 (Multi-run procedure) terminates.*

Proof. Let $x : E \rightarrow \mathbb{N}_0$ record how many times each edge $e \in E$ has been traversed in total, at any given time of Algorithm 2 (Multi-run procedure). For $v \in V$, we always have $x^+(v) = x^-(v) - t(v)$, where $t(v)$ is the number of trains currently waiting at v . Suppose for a contradiction that the Multi-run procedure cycles. Then $x^-(v)$ is unbounded for at least one $v \in V$, which means that $x^+(v)$

is also unbounded, since $t(v)$ is bounded. This in turn means that $x^-(s_{\text{even}}(v))$ and $x^-(s_{\text{odd}}(v))$ are unbounded as well, since we distribute $x^+(v)$ evenly between the two successors. Repeating this argument, we see that $x^-(w)$ is unbounded for all vertices w reachable from v . But as the inflows at the vertices in D are bounded (by the number of trains that we started with), none of these vertices are reachable from v . This is a contradiction to A being terminating. \square

Then the following lemma follows the Abelian property of switching systems.

Corollary 2.16 (Abelian property of switching systems). *Given a terminating switching system A , the total number of edge traversals, the run profile, and the output of the algorithm do not depend on the choices in the execution of Algorithm 2 (Multi-run procedure).*

Proof. Firstly, by Lemma 2.15, Algorithm 2 (Multi-run procedure) always terminates. Let x and y be the run profiles with respect to two arbitrary executions of the algorithm. By Lemma 2.8, the local runtimes (i.e., the number of trains leaving each vertex) are the same in both executions. This implies that the total number of edge traversals in the two executions are the same. Due to the switching behavior at each vertex, we can conclude that the number of traversals on each edge is the same in both executions. This implies $x = y$, as claimed. Consequently, the inflow at each destination is the same in both executions. In other words, the outputs of the algorithm are the same. \square

Remark 2.19 below also sketches another proof, which uses the idea for an analogous statement for ARRIVAL in [62].

2.6 Decision complexity of G-ARRIVAL

In this section, we will extend the complexity of ARRIVAL to G-ARRIVAL. The arguments in this section are analogous to those for ARRIVAL in [62] and [76].

2.6.1 G-ARRIVAL is in $\text{NP} \cap \text{coNP}$

By Lemma 2.16, the run profile is unique and from that, we can deduce the unique output of Algorithm 2 (Multi-run procedure). Therefore, the run profile can certify that a G-ARRIVAL instance is a YES or a NO instance. It turns out that not only the run profile but any switching flow can be used as a certificate for both cases when the instance is either a YES or a NO instance. As a result, G-ARRIVAL is in $\text{NP} \cap \text{coNP}$.

Theorem 2.17 (Switching flows are certificates). *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system and let $D = \{d_1, \dots, d_{|D|}\}$. Then for any switching flow x , $(x^-(d_1), \dots, x^-(d_{|D|}))$ is the output of Algorithm 2 (Multi-run procedure).*

In fact, we will prove the following stronger statement, which implies the theorem above.

Theorem 2.18 (The run profile is the minimal switching flow). *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system with edge set E . Let \hat{x} be the run profile of A . Then $\hat{x} \leq x$ for all switching flows x . In particular, \hat{x} is the unique minimizer of the total flow $\sum_{e \in E} x_e$ over all switching flows.*

Proof. We prove this by the *pebble argument* [62]. Let x be any switching flow. For every edge e , we initially put x_e pebbles on e , and whenever a train traverses e in Algorithm 2 (Multi-run procedure), we let it collect a pebble. If we can show that we never run out of pebbles, $\hat{x} \leq x$ follows. By “running out of pebbles”, we concretely

mean that we are for the first time trying to collect a pebble from an edge with no pebbles left.

We show that we cannot run out of pebbles while processing a picked vertex $v \in V$. For this, we prove that we maintain the following additional invariants (which hold immediately at the beginning). Let $p : E \rightarrow \mathbb{N}_0$ record for each edge e the remaining number of pebbles on e . Then for all $v \in V$,

(a) $p^+(v) = p^-(v) + t(v)$, where $t(v)$ is the number of trains waiting at v ;

(b) $p((v, s_{curr}(v))) - p((v, s_{next}(v))) \in \{0, 1\}$.

Suppose that these invariants hold when picking a vertex $v \in V$. As we have not run out of pebbles before, $p^-(v) \geq 0$ and (a) guarantees that we have $q \geq t(v)$ pebbles on the outgoing edges; by (b), $\lceil q/2 \rceil$ of them are on $(v, s_{curr}(v))$ and $\lfloor q/2 \rfloor$ on $(v, s_{next}(v))$. From the former, we collect $\lceil \rho/2 \rceil$, and from the latter $\lfloor \rho/2 \rfloor$ where $\rho \leq t(v) \leq q$, so we do not run out of pebbles. We maintain (a) at v where both p^+ and t are reduced by ρ . We also maintain (a) at the successors; there, the gain in t exactly compensates the loss in p^- . Finally, we maintain (b) at v : If ρ is even, both $p((v, s_{curr}(v)))$ and $p((v, s_{next}(v)))$ shrink by $\rho/2$. If ρ is odd, we have $p((v, s_{curr}(v))) - p((v, s_{next}(v))) \in \{-1, 0\}$ after collecting one more pebble from $(v, s_{curr}(v))$ than from $(v, s_{next}(v))$, but then we reverse the sign by swapping s_{curr} and s_{next} . \square

Remark 2.19. *Observe that the proof above does not use the fact that the run profile is unique. In fact, it can be used as another proof of Corollary 2.16. Indeed, suppose the run profile is not unique. Then let \hat{x} and x' be two distinct run profiles, i.e., for some $d \in D$, $\hat{x}(d) \neq x'(d)$. WLOG, we assume $\hat{x}(d) > x'(d)$. Following the argument as in the proof of Theorem 2.18 for the run profile \hat{x} and the switching flow x' , we obtain that $\hat{x} \leq x'$. This contradicts the*

assumption $\hat{x}(d) > x'(d)$. Note that this proof is similar to the proof in [160, Lemma 3.6].

Proof of Theorem 2.17. Let \hat{x} be the run profile and x be any switching flow. By Observation 2.14, the sums of inflows at the vertices in D w.r.t. \hat{x} and x are the same. By Theorem 2.18, $\hat{x} \leq x$. Therefore, the inflow at each vertex in D is the same in both x and \hat{x} . The theorem then follows. \square

2.6.2 G-ARRIVAL is in $\text{UP} \cap \text{coUP}$

The result above can be tightened to a membership in $\text{UP} \cap \text{coUP}$. The complexity class UP is similar to NP , except that only at most one certificate can be accepted (i.e., exactly one if the instance is a YES instance and none if it is a NO instance). The class coUP is similar to coNP in the same way.

For G-ARRIVAL, as discussed before, the run profile is a certificate for both YES and NO instances, and it is also unique. The last ingredient we need is that it can indeed be verified efficiently. We establish this in the following lemma.

Lemma 2.20 (Efficient run profile verification). *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system with edge set E . Let $x : E \rightarrow \mathbb{N}_0$ be a function. Then we can verify whether x is the run profile of A in polynomial time.*

Proof. We adapt the proof for ARRIVAL in [76].

By Definition 2.13, we can verify in polynomial time whether x is a switching flow. Now consider the following subgraph $G_x = (V \cup D, E')$ of $G(A)$, where

$$\begin{aligned} E' = & \{(v, s_{\text{even}}(v)) : x_{(v, s_{\text{even}}(v))} = x_{(v, s_{\text{odd}}(v))} + 1\} \cup \\ & \{(v, s_{\text{odd}}(v)) : x_{(v, s_{\text{even}}(v))} = x_{(v, s_{\text{odd}}(v))} > 0\}. \end{aligned}$$

Intuitively, G_x contains the edges to the next successors of the vertices whose outflow is at least one.

We will show that x is the run profile if and only if G_x does not have any directed cycle. Then, since we can compute G_x and check if it has a directed cycle in polynomial time, we conclude that the run profile can be verified in polynomial time.

Suppose x is the run profile and, for the sake of contradiction, assume that G_x has a directed cycle. Then by the definition of E' , there is a positive flow along the cycle. If we reduce the flow in the edges along the cycle by 1, the flow conservation is still maintained, since the inflow and outflow at every vertex along the cycle is reduced by 1. Further, by the definition of E' , the switching behavior is also maintained. Hence, the resulting flow x' is a switching flow, and $x' \leq x$, where the inequality is strict for some components. However, this contradicts Theorem 2.18.

For the other direction, suppose G_x does not have a directed cycle and, for the sake of contradiction, assume that x is not the run profile. Then another flow $x^* \leq x$ is the run profile, where for some edge e' , $x_{e'}^* < x_{e'}$. Consider the flow $z = x - x^*$ (componentwise). Since flow conservation holds for both x and x^* , it also holds for z . Further, by Theorem 2.17, $z_e = 0$ for each incoming edge e to a destination. For every edge e , we put z_e pebbles on e . We start at an arbitrary vertex, such that there is a pebble in one of its two outgoing edges. (Such an edge exists, since $z_{e'} > 0$.) Whenever we are at a vertex v , if there is a pebble on the outgoing edge from v in G_x , we traverse that edge and take a pebble from there. Otherwise, we stop. Since G_x is acyclic, we traverse each edge at most once. Let u be the vertex where we stop, and let $e_1 \in E'$ and $e_2 \notin E'$ be the two outgoing edges of u in $G(A)$. By the stopping condition, we have $z_{e_1} = 0$, or $x_{e_1}^* = x_{e_1}$. If u is the starting vertex, by the choice of the starting vertex, there is a pebble in an outgoing edge from u in $G(A)$. Otherwise, we have taken a pebble from an incoming edge

to u ; so by flow conservation of z , there must also be a pebble in an outgoing edge from u in $G(A)$. In either case, we can conclude that $z_{e_2} > 0$, or $x_{e_2}^* < x_{e_2}$.

If $x_{(u, s_{\text{even}}(u))} = x_{(u, s_{\text{odd}}(u))} + 1$, then $e_1 = (u, s_{\text{even}}(u))$. However, that implies

$$x_{(u, s_{\text{even}}(u))}^* - x_{(u, s_{\text{odd}}(u))}^* = x_{e_1}^* - x_{e_2}^* > x_{e_1} - x_{e_2} = 1,$$

which violates the switching behavior constraint at u .

If $x_{(u, s_{\text{even}}(u))} = x_{(u, s_{\text{odd}}(u))}$, then $e_1 = (u, s_{\text{odd}}(u))$. We then have

$$x_{(u, s_{\text{even}}(u))}^* - x_{(u, s_{\text{odd}}(u))}^* = x_{e_2}^* - x_{e_1}^* < x_{e_2} - x_{e_1} = 0,$$

which also violates the switching behavior constraint at u .

Hence, we obtain a contradiction and can conclude that x must be a run profile, if G_x does not have a directed cycle. This completes the proof. \square

We are now ready to prove the membership in $\text{UP} \cap \text{coUP}$.

Theorem 2.21. *G-ARRIVAL is in $\text{UP} \cap \text{coUP}$.*

Proof. By Corollary 2.16, the run profile is unique, and from the run profile, we can recover the unique output of Algorithm 2 (Multi-run procedure). From this output, we can certify in polynomial-time whether a G-ARRIVAL instance is a YES or a NO instance (see also Theorem 2.17). By Lemma 2.20, we can verify the run profile efficiently. Hence, G-ARRIVAL is in $\text{UP} \cap \text{coUP}$, as claimed. \square

2.7 Search complexity of G-ARRIVAL

So far, we have recovered the computational complexity of the decision problem G-ARRIVAL. We now do the same for the search

variant. The variant for ARRIVAL is called S-ARRIVAL, defined by Karthik C.S. [103] as follows: (The switching flow for ARRIVAL is defined analogously as for switching systems.)

Definition 2.22 (S-ARRIVAL). *Given a terminating ARRIVAL instance A , find a switching flow of A .*

Similarly, we can define the search variant of G-ARRIVAL as follows:

Definition 2.23 (GS-ARRIVAL). *Given a terminating switching system A , find a switching flow of A .*

By Corollary 2.16, for any input, either we can verify efficiently that the input is not valid (i.e., it is not a terminating switching system), or there exists a solution (i.e., a switching flow) that can be verified efficiently, by Definition 2.13. This is exactly the description of a problem in the complexity class TFNP. A common interest in complexity theory is to pinpoint the complexity class for which a problem is hard. S-ARRIVAL has been shown to be a member of a subclass of TFNP called UEOPL [73], but so far, there is no indication that it is hard for this class. In this section, we extend this membership result to GS-ARRIVAL.

2.7.1 UEOPL

The complexity class UEOPL contains exactly all the problems that can be reduced in polynomial time to a problem called UniqueEOPL (i.e., Unique End of Potential Line). Informally, this problem provides a *promise* of a unique potential line, i.e., a graph of only a directed path with a potential function increasing along the path. For each vertex, we can query its successor and predecessor on the directed path, as well as its potential. The goal is to output a sink or a certificate that the promise is not satisfied, i.e., either (i) the potential function is not increasing along the path or (ii) the graph

is not a path (or equivalently, there is more than one maximal path in the graph). Note that by the way the problem is phrased, when the output is a sink, there is no guarantee that the promise is fulfilled. In other words, even when the promise is violated, any sink in the graph is still a valid output.

The formal definition is stated below, followed by an explanation about how it fits the informal description above.

Definition 2.24 (UniqueEOPL [73]). *Given Boolean circuits $S, P : \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$ and a Boolean circuit $V : \{0, 1\}^n \rightarrow \{0, 1, \dots, 2^m - 1\}$ such that $V(0^n) = 0$, find one of the following:*

- (U1) *A point $x \in \{0, 1\}^n$ such that $P(S(x)) \neq x$.*
- (UV1) *A point $x \in \{0, 1\}^n$ such that $S(x) \neq x$, $P(S(x)) = x$, and $V(S(x)) - V(x) \leq 0$.*
- (UV2) *A point $x \in \{0, 1\}^n$ such that $S(P(x)) \neq x \neq 0^n$.*
- (UV3) *Two points $x, y \in \{0, 1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < V(S(x))$.*

First, observe that for any vertex x of the n -dimensional hypercube, if $P(x) = S(x) = x$, then it is not a solution of any type listed above. Let X be the set of all other vertices of the hypercube, i.e., the vertices x such that either $P(x) \neq x$ or $S(x) \neq x$. We consider a directed graph G with the vertex set X and the edge set defined by the predecessor circuit P and successor circuit S as follows: For two distinct points x and y in X , if $S(x) = y$ and $P(y) = x$, then we add an edge (x, y) to the graph G . By definition, the sinks of this graph are exactly those vertices x such that $P(S(x)) \neq x$, i.e., solution of type (U1).

The other solution types encode violations of the promise. Firstly, a solution of type (UV1) indicates that the potential is not increasing

along the path. Secondly, a vertex x such that $S(P(x)) \neq x$ does not have any incoming edges in G , and hence it is a source. Since $x = 0^n$ satisfies $S(P(0^n)) \neq 0^n$ and is hence a source, if there is a solution y of type (UV2), y is a source distinct from 0^n . This is a violation of the promise that there is only one path (which implies that there can only be one source). Note that because each vertex of the graph has out-degree and in-degree at most one, if there is only one source then there can only one path. Therefore, the two solution types above already cover all the violations of the promise. However, as noted by the authors in [73], the last solution type (UV3), which clearly encodes a violation, makes the problem easier and distinct from another problem called EOPL, which defines a different complexity class. For more details on UniqueEOPL and EOPL, see [73].

Observe that there is always a solution of type (U1) or (UV1). If there is no solution of the first type, as discussed above, the graph has no sink. This implies the existence of a directed cycle. Since we cannot maintain an increasing potential along a cycle, there must be a solution of type (UV1).

2.7.2 Difficulties in adapting the proof for ARRIVAL

Although the decision complexity of ARRIVAL can be adapted easily to G-ARRIVAL, as we have seen in the previous section, the proof of S-ARRIVAL in UEOPL cannot be straightforwardly carried over to the setting of GS-ARRIVAL. To illustrate this point, we first look at the proof sketch for S-ARRIVAL.

Before going to the proof, we explain a few notations. Given a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$, we extend the definition of the routing operation also to describe the traversal of the edge (Y, v) from the train yard Y to a vertex $v \in V$, as long as this edge is not traversed more than $s(v)$ times in total. Recall the definition of a state of a switching system in Definition 2.11. We define further

terminologies related to states as follows:

Definition 2.25 (Initial state, final state, reachable state). *Given a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$, the initial state is the state (τ_0, σ_0) at the very beginning, i.e.,*

$$\begin{aligned}\tau_0(Y) &= S, \\ \tau_0(v) &= 0, \quad v \in V \cup D \\ \sigma_0(v) &= 0, \quad v \in V.\end{aligned}$$

The final state is the state obtained after Algorithm 2 (Multi-run procedure) terminates. We say a state z is reachable from a state z' , if we can encounter z after a sequence of routing starting from z' . A state z is reachable, if it is reachable from the initial state (τ_0, σ_0) .

We are now ready to discuss the proof of S-ARRIVAL in UEOPPL.

Theorem 2.26. [76, 73] *S-ARRIVAL is in UEOPPL.*

Proof sketch. We reduce S-ARRIVAL to UniqueEOPL. Suppose A is an ARRIVAL instance where the switch graph has n vertices. Note that we can describe the instance by a switching system with only one train. As such, we can extend the terminologies for switching systems to ARRIVAL.

For the UniqueEOPL instance, we define the directed path to be the sequence of all states that we encounter during the execution of Algorithm 1 (Run procedure). Since the run procedure is a deterministic process, the path is well defined and contains all the reachable states from the initial state. Note that the sink of this path is the final state, which corresponds to the run profile of A , which is also a switching flow for A . (We explain in Corollary 2.31 later how to obtain the run profile from the final state.)

Since there is only one train in the system, all states of A can be encoded by $M = \mathcal{O}(n)$ bits. We now describe how to evaluate the

successor, predecessor, and potential of a bitstring of length M in polynomial time. We first can efficiently verify whether a bitstring encodes a valid state. If it does not, we can set its successor and predecessor to itself, and its potential to 0 (i.e., it is never a solution). Otherwise, Gärtner et al. [76] described a process to verify in polynomial time whether the state is reachable (i.e., whether it lies on the directed path above). Further, the process also gives the number of edge traversals by the run procedure until we reach that state. This number of edge traversals is the potential of the state. Next, given the state, we know where the train is currently. In linear time, we can route the train and obtain the succeeding state. Lastly, for the predecessor, since there is only a linear number of in-neighbors of the current vertex of the train, we can calculate a linear number of candidates for the predeeding states, and verify whether each of them is on the path and whether its distance from the initial state is exactly one less than that of the current state. We can do the verification by the same process above by Gärtner et al. [76].

Since we define a directed path with increasing potential and hence fulfill the promise of UniqueEOPL, it is easy to see that there can only be a solution of type (U1) which corresponds to the run profile of A , as discussed above. Hence, the reduction is complete. \square

The proof sketch above relies on the fact that there is only one way to execute Algorithm 1 (Run procedure). However, this does not hold for Algorithm 2 (Multi-run procedure). Therefore, for the reduction from GS-ARRIVAL, we should define a “canonical” path. For example, we can define this as the path corresponding to the execution where we pick a train from the vertex with the lowest index and run this train until it reaches the destination; then we continue by picking a remaining train from the vertex with lowest index, and so on.

However, we are then faced with the issue of determining if a state lies on this canonical path. Even though we can efficiently determine if a state is reachable (by Theorem 2.12), the task of checking if a state is reachable via a specific execution of Algorithm 2 (Multi-run procedure) is not straightforward. To circumvent this issue, we utilize the two situations where we know with certainty that a state is reachable via *any* execution. For one, if a state is the *final state* of a switching system (i.e, the state obtained after the multi-run procedure terminates), then by the Abelian property (Lemma 2.8), any execution of the multi-run procedure will end at this state. For the other, if a switching system has only one train, then the multi-run procedure is deterministic, and hence, there is only one way to execute it (in fact, in this case, we recover the setting of ARRIVAL). With this in mind, besides the current state, we also encode an intermediate state that serves as a checkpoint. We will show that verifying whether this checkpoint is on the canonical path is equivalent to checking whether a state is the final state of a switching system. Moreover, verifying whether the current state is reachable along the canonical path from the checkpoint is equivalent to checking if a state is reachable in a switching system with only one train.

Another issue is the evaluation of the predecessor circuit. We will discuss in more detail in Remark 2.34 later how introducing the checkpoint hinders the method of determining the preceding state in the proof for S-ARRIVAL. To circumvent this issue, instead of reducing to UniqueEOPL, we reduce GS-ARRIVAL to the following problem UniqueForwardEOPL, which is also in UEOPL [73].

Definition 2.27 (UniqueForwardEOPL [73]). *Given a Boolean circuit $S : \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that $S(0^n) \neq 0^n$ and a Boolean circuit $V : \{0, 1\}^n \rightarrow \{0, 1, \dots, 2^m - 1\}$ such that $V(0^n) = 0$, find one of the following:*

(UF1) *A point $x \in \{0, 1\}^n$ such that $S(x) \neq x$ and either $S(S(x)) =$*

$$S(x) \text{ or } V(S(x)) \leq V(x).$$

(UFV1) Two points $x, y \in \{0, 1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < V(S(x))$.

(UFV2) Two points $x, y \in \{0, 1\}^n$, such that x is a solution of type (UF1), $y \neq S(y)$, and $V(x) < V(y)$.

The main differences from UniqueEOPL is that there is only a successor circuit (hence “forward”). The first solution type (UF1) gives either the end of the potential line or a certificate that the potential does not increase along the line. The other solution types (UFV1) and (UFV2) encodes a violation of the assumption that there is only one line.

We note that there is always a solution of type (UF1). We define a graph whose vertices are the points $x \in \{0, 1\}^n$. If $x \neq S(x)$, we add the edge $(x, S(x))$ to the graph. Since $S(0^n) \neq 0^n$, the graph has at least one edge. If there is a sink y with an in-neighbor x , then x satisfies $x \neq S(x) = y$ and $S(S(x)) = S(x)$. If there is no such sink, then there must be a directed cycle. Since we cannot have an increasing potential along the cycle, there must be a point x where $V(S(x)) \leq V(x)$.

We also remark that we can reduce S-ARRIVAL to UniqueForwardEOPL instead of UniqueEOPL.

2.7.3 Partial switching flow

The reduction will use Lemma 2.29 below on partial switching flow defined in Definition 2.28 below. Recall the definition of a state $z = (\tau, \sigma)$ of a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ in Definition 2.11. Note that $\tau(Y)$ needs not be 0 for z to be a state of A . To facilitate the reduction later on, we define a function $y : V \rightarrow \mathbb{N}_0$ to be a *train dispersal for the state z* , if $y(v) \leq s(v)$ for $v \in V$, and

$\sum_{v \in V} y(v) = \sum_{v \in V} \tau(v)$. We can interpret that some trains have left Y and been routed within the system, and the train dispersal records how many trains have traversed the edge (Y, v) for $v \in V$. Note that unless all trains have left Y , this may not correspond to a partial execution of Algorithm 2 (Multi-run procedure). We are now ready to define the partial switching flow.

Definition 2.28 (Partial switching flow). *Let $z = (\tau, \sigma)$ be a state of a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ with edge set E . A partial switching flow (for A) with respect to z and y is a function $x : E \rightarrow \mathbb{N}_0$ that satisfies*

$$\begin{aligned} x_{(Y, v)} &= y(v) & v \in V \\ x^-(v) - x^+(v) &= \tau(v), & v \in V \\ x_{(v, s_{\text{even}}(v))} - x_{(v, s_{\text{odd}}(v))} &= \sigma(v), & v \in V. \end{aligned}$$

If $y(v) = s(v)$ for $v \in V$ (and hence $\tau(Y) = 0$), then we simply call x a partial switching flow with respect to z .

It is easy to see that for a reachable state z , if functions $x : E \rightarrow \mathbb{N}_0$ and $y : V \rightarrow \mathbb{N}_0$ record the numbers of times an edge $e \in E$ and an edge (Y, v) for $v \in V$, respectively, are traversed in some sequence of routings until we encounter the state z , then x is a partial switching flow with respect to z and y . Lemma 2.29 below implies that for such z and y , there is only one partial switching flow, and hence, it is exactly the function x described above.

Lemma 2.29 (Partial switching flow is nonexistent or unique). *Let $z = (\tau, \sigma)$ be a state of a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ with the edge set E . Let y be a train dispersal for z . Then there is at most one partial switching flow with respect to z and y , and it can be computed in polynomial time.*

In order to prove Lemma 2.29, we use the following general lemma on switch graphs.

Lemma 2.30. *Let G be a directed graph that has at least a sink. Let V be the set of non-sinks in G . For any v in V , there exists a directed path from v to a sink of G . Further, v has two out-neighbors, denoted by $s_{\text{even}}(v)$ and $s_{\text{odd}}(v)$. Let τ, σ be two functions $V \rightarrow \mathbb{R}$. Then there exists a unique function $x : V \rightarrow \mathbb{R}$ such that:*

$$\begin{aligned} x^-(v) - x^+(v) &= \tau(v), & v \in V \\ x_{(v, s_{\text{even}}(v))} - x_{(v, s_{\text{odd}}(v))} &= \sigma(v), & v \in V. \end{aligned}$$

Proof. We can write the above system in the corresponding matrix form $Qx = b$, where we interpret x as a vector. Note that Q is a $2n$ -by- $2n$ square matrix, where $n := |V|$. For a special case where there are two sinks (i.e., G is the switch graph of a terminating ARRIVAL instance), Gärtner et al. showed that for certain vector b' , the system $Qx = b'$ has a unique *real* solution, and hence Q is invertible ([76, Lemma 23]). However, the linear system above does not have equalities corresponding to the sinks. Therefore, if there are more than three sinks, we can contract all sinks but one in G into a new sink \bar{d} . If G has only one sink, we add \bar{d} as an isolated vertex, which is also a sink. The new graph then only has two sinks. However, the matrix Q remains unchanged, except for a relabelling of each edge (v, d') for a contracted sink d' into (v, \bar{d}) . Hence, Q is invertible in the general case, and the lemma then follows. \square

We are now equipped to prove Lemma 2.29.

Proof of Lemma 2.29. Let $V = [n]$ be the vertex set and E be the edge set of the switch graph $G(A)$. The linear system in Definition 2.28 can be rewritten as

$$\begin{aligned} \sum_{e \in E^-(v) \setminus \{(Y, v)\}} x_e - x_v^+ &= \tau(v) - y(v), & v \in V, \\ x_{(v, s_{\text{even}}(v))} - x_{(v, s_{\text{odd}}(v))} &= \sigma(v), & v \in V. \end{aligned}$$

By Lemma 2.30, the above system has a unique solution over the reals. We can solve the system and obtain this solution in polynomial

time. If all the coordinates of the solution are natural numbers, then this defines a (and in fact, the only) partial switching flow with respect to z and y . Otherwise, there is no partial switching flow. \square

A consequence of the above lemma is that we can verify efficiently, if a state is the *final state* of a switching system.

Corollary 2.31. *Let z be a state of a terminating switching system A . In polynomial time, we can verify whether z is the final state of A and, if so, compute the run profile of A .*

Proof. Let $z = (\tau, \sigma)$. It can easily be seen that if z is the final state, then the run profile is a partial switching flow with respect to z . By Lemma 2.29, in polynomial time, we can either compute the unique partial switching flow with respect to z or conclude that it is nonexistent. If the partial switching flow does not exist, then we conclude that z is not the final state. Otherwise, by Lemma 2.20, we can verify if it is the run profile of A and accordingly decide whether z is the final state of A . \square

Remark 2.32. *Corollary 2.31 above implies that the final state of A is also an NP or coNP certificate for a switching system. Since it is also unique, by Lemma 2.8, it is also a UP or coUP certificate.*

2.7.4 GS-ARRIVAL is in UEOPL

This section is dedicated to a reduction from GS-ARRIVAL to UniqueForwardEOPL and hence prove the following theorem.

Theorem 2.33. *GS-ARRIVAL is in UEOPL.*

Proof. Suppose we are given a GS-ARRIVAL instance with the input of a terminating switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ with $S = \sum_{v \in V} s(v)$.

Assume $V = [n]$. We prescribe the following “canonical” procedure C , which is a modification of Algorithm 2 (Multi-run procedure). The procedure starts with S trains at the train yard Y and no trains in other vertices of $G(A)$. We route a train from Y to the lowest vertex $v \in V$ such that the edge (Y, v) has not been traversed $s(v)$ times. After that we route that train from vertices in V , until it reaches a destination in D . We repeat the above steps until all trains are at the destinations in D and output the number of trains at each destination. Procedure C is almost the same as the multi-run procedure, except that not all trains are sent out of Y at the beginning and that we prescribe an order of vertices and the number of trains to be processed at each iteration. As the switching system is Abelian (Lemma 2.8), the output of Procedure C is the same as the output of the multi-run procedure. Note that Procedure C is deterministic, and at any point in its execution, there is at most one train among the vertices in V . The sequence of states that we encounter along the execution of Procedure C defines uniquely a directed path, whose sink corresponds to the final state of A .

We call a state of A *legal*, if and only if we encounter it during the execution. A legal state (τ, σ) is a *checkpoint*, if there is no train in V (i.e., $\tau(v) = 0$ for all $v \in V$). Intuitively, the checkpoints separate the directed path into many lines. Each line goes from a checkpoint to the next and describes the trajectory of a train from Y to a destination in D . In other words, it corresponds to an ARRIVAL instance, although in a more relaxed sense, as there may be more than two destinations. For each legal state z , we denote by $p(z)$ the *last checkpoint of z* that is either z , if z is a checkpoint, or otherwise the checkpoint immediately preceeding z in the execution.

We label each node on the directed path by $(z, p(z))$ where z is the corresponding legal state. Since a train configuration and a switch configuration can be encoded by $\mathcal{O}(nS)$ and n bits, respectively, and since $S = \mathcal{O}(\text{poly}(n))$, all labels can be encoded with $M =$

$\mathcal{O}(\text{poly}(n))$ bits. Note that we can easily check if a bitstring of length M encodes a tuple (z, \hat{z}) where z and \hat{z} are (not necessarily legal) states of A . If it does not, then we set its successor to itself and its potential to 0 (and hence it is not a solution). From now on, we assume we have such a tuple.

In order to determine if (z, \hat{z}) is a label of a node on the directed path, we need to verify whether \hat{z} is a checkpoint and whether $\hat{z} = p(z)$ (which implies that z is legal). To verify the former, we observe that a checkpoint is the state obtained after sending some k trains from Y to the destinations. We can easily compute k from the state, and by the description of Procedure C, we know exactly how many trains have traversed the edge (Y, v) for each $v \in V$. In other words, the checkpoint corresponds to the final state of a well-defined switching system. Therefore, we can use Corollary 2.31 to verify if \hat{z} is a checkpoint.

Next, we verify if \hat{z} is the last checkpoint of z . We first can verify whether z is a checkpoint, and if it is, z must equal \hat{z} . If it is not a checkpoint, then there must have exactly one train in V , and when we remove this train and add it to Y , we should recover the train reconfiguration of \hat{z} . If this is not the case, then we know immediately that \hat{z} is not the last checkpoint for z . Now we suppose that it is the case. Then as discussed before, the subsequence of the directed line between \hat{z} and the next checkpoint corresponds to the trajectory of one train from Y to a destination. Hence, we can define another switching system with only this train, by making \hat{z} the initial state (after removing all other trains) and finding the appropriate vertex to route the train from Y . Since there is only one train in the system, there is only one order in which we route the train. We can then apply Theorem 2.12 to check if z is reachable from \hat{z} . Only if it is, then \hat{z} is the last checkpoint of z .

When we have verified that (z, \hat{z}) is a label of a node of the directed path, we now can find its potential, which is the number of edge

traversal in the execution until we encounter the state z . In other word, it is exactly $\sum_{e \in E(A)} x_e$, where the function $x : E(A) \rightarrow \mathbb{N}_0$ records the number of times we have traversed an edge until we encounter z . Then x is a partial switching flow with respect to z and a suitable train dispersal. By Lemma 2.29, there is only one such partial switching flow, and we can compute it in polynomial time. Therefore, we can compute the potential in polynomial time.

Lastly, we need to specify how to find the successor of a node on the path. In the corresponding state of the node, if there is a train at a vertex $v \in V$, we route that train by one step. If there is no train in V , then we route a train from Y (if any) according to the description of Procedure C . In either case, we obtain the successor, and this can be done in polynomial time. Lastly, we define the sink of the directed path to be its own successor.

By the definition of the successor and potential, it is easy to see that the only solution in the UniqueForwardEOPL instance is the node immediately before the sink in the path. We can then obtain the sink, which corresponds to the final state of A . By Corollary 2.31, from the final state, we can calculate the run profile of A , which is also a switching flow for A . This completes the reduction and the proof of the membership of GS-ARRIVAL in UEOPL. \square

Remark 2.34. *Consider a checkpoint z that is not the initial state. Then (z, z) is a label of a node in the directed path above. The predecessor of this node is $(z', p(z'))$, where z' is the state preceding z and $p(z')$ is the checkpoint immediately before z in the execution of Procedure C . From z , we cannot compute $p(z')$, especially its switch configuration (or rather we do not know how to). Because of this limitation, we reduce to UniqueForwardEOPL instead of UniqueEOPL, to avoid the treatment for the predecessors.*

*A process cannot be understood by
stopping it. Understanding must
move with the flow of the process,
must join in and flow with it.*

—Frank Herbert,
Dune

CHAPTER 3

Subexponential algorithm for G-ARRIVAL

This chapter is based on the results for ARRIVAL from [77], which is joint work with Bernd Gärtner and Sebastian Haslebach, and adapted for G-ARRIVAL.

We now move on to the algorithmic aspects of G-ARRIVAL. For ARRIVAL, as mentioned in Section 1.1, Dohrau et al. [62] showed an upper bound of $\mathcal{O}(n2^n)$ for the trivial algorithm of simulating the train run. This upper bound was improved to $\mathcal{O}(p(n)2^{n/2})$ (in expectation) for some polynomial p , by efficient sampling from the run [76]. The same bound can be achieved deterministically in Section 3.2 later. The same approach was also discovered independently and refined to yield a runtime of $\mathcal{O}(p(n)2^{n/3})$ by Rote [149].

The main result of this chapter is a subexponential algorithm for G-ARRIVAL that runs in time $n^{\mathcal{O}(\sqrt{n})}$ (Theorem 3.14). This algorithm

is an instantiation of a general framework to solve G-ARRIVAL (Section 3.4). Notably, this framework produces a switching flow and hence also solves GS-ARRIVAL. At the end of the chapter, we will show another application of the framework, which yields a polynomial-time algorithm, when the size of the feedback vertex set is small (see Theorem 3.15 and the definition of feedback vertex set in Section 3.6).

As a warm up, we will start by discussing variants of the trivial algorithm of simply simulating Algorithm 2 (Multi-run procedure) (Sections 3.1 and 3.3). In Section 3.2, we introduce the layer decomposition, a useful tool to design many algorithms for G-ARRIVAL.

3.1 Naïve simulation of the train runs

In this section, we analyze the runtime of a naïve simulation of the train runs. In this simulation, we run each of the starting trains individually and proceed with the next one only when the previous one has reached a destination. In Algorithm 2 (Multi-run procedure), this corresponds to always choosing $\rho = 1$ and the next vertex v as the head of the previously traversed edge, for each of the starting trains. Effectively, we perform Algorithm 1 (Run procedure) for all the trains. By Corollary 2.16, every execution of Algorithm 2 (Multi-run procedure) has the same number of edge traversals. Since we route only one train at every iteration in the naïve simulation, this simulation is the longest possible execution of the algorithm in terms of iteration count.

Following the discussion above, we have a simple lemma below.

Lemma 3.1. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system. Suppose that in any execution of Algorithm 2 (Multi-run procedure) with input A , each train traverses at most B edges before it reaches its destination. Further suppose that at the begin-*

ning of some iteration in the algorithm, W trains are still waiting. Then all subsequent iterations traverse at most WB edges in total.

Proof. By Corollary 2.16, the number of edge traversals is the same in any execution, including the naïve simulation. In this simulation, we sequentially route W trains, each of which takes at most B edge traversals to reach a destination. Hence, there are at most WB remaining edge traversals in total. \square

We will now find a bound B for the lemma above. For a switching system A and a vertex $v \in V(A)$, we denote by $\text{dist}_A(v)$ the length of the shortest path in $G(A)$ from v to a destination in D . When the switching system A is clear, we drop the subscript A .

Lemma 3.2. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system, $|V| = n$. For any train a and any vertex $v \in V$, a visits v at most $2^{\text{dist}(v)}$ times by Algorithm 2 (Multi-run procedure).*

Proof. Suppose $\text{dist}(v) = k$. Let $v = v_k, v_{k-1}, \dots, v_0 \in D$ be the sequence of vertices on a shortest path from v to D . Consider the first 2^k visits by a to v (if there are fewer, we are done). Once every two consecutive visits, the train moves on to v_{k-1} , so we can consider the first 2^{k-1} visits to v_{k-1} and repeat the argument from there to show that v_i is visited at least 2^i times for all i , before v exceeds 2^k visits. In particular, $v_0 \in D$ is visited, so the run indeed terminates within at most 2^k visits to v . \square

Lemma 3.3. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system, $|V| = n$. Let ℓ be the maximum length of the shortest path from a vertex in V to a destination. In any execution of Algorithm 2 (Multi-run procedure), each train traverses at most $(n - \ell + 2)2^\ell - 2$ proper edges before it reaches a destination.*

Proof. Fix a train. By Lemma 3.2, the total number of visits by the train to vertices $v \in V$ is bounded by $\sum_{i=1}^n n_i 2^i$, where n_i is the

number of vertices with a shortest path of length i to a destination. We have $n_i > 0$ if and only if $i \leq \ell$, and hence the sum is maximized if $n_i = 1$ for all $i < \ell$, and $n_\ell = n - \ell + 1$. In this case, the sum is $(n - \ell + 2)2^\ell - 2$. The number of edges being traversed (one after every visit of $v \in V$) is the same. \square

By Lemmata 3.1 and 3.3, we obtain the following runtime for the naïve simulation, which is also the upper bound of the runtime for any execution of Algorithm 2 (Multi-run procedure).

Corollary 3.4. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system, $|V| = n$, $S = \sum_{v \in V} s(v)$. Further, let ℓ be the maximum length of the shortest path from a vertex in V to a vertex in D . Then any execution of Algorithm 2 (Multi-run procedure) traverses at most $S((n - \ell + 2)2^\ell - 2)$ proper edges in total.*

A simple calculation shows that the function $f(\ell) = (n - \ell + 2)2^\ell - 2$ is increasing for $\ell \in [1, n]$. Hence, the number of edge traversals in the corollary above is $\mathcal{O}(S2^n)$.

3.2 Layer decomposition

In this section, we present the layer decomposition of the switch graph of a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$. The layer decomposition will be used in many algorithms later in this and the next chapter. The first of such algorithms is presented in this section. It involves a contraction procedure such that the naïve algorithm for G-ARRIVAL runs in the resulting graph in time $\mathcal{O}(S2^{n/2})$, an improvement from the time bound of $\mathcal{O}(S2^n)$ in the previous section. Independently, this contraction procedure has also been discovered by Rote for ARRIVAL (where $S = 1$), and was bootstrapped to a $\mathcal{O}(2^{n/3})$ -time algorithm [149].

We decompose the graph into layers, subsets of vertices (excluding the artificial vertex Y) with the same distance to the destinations. More formally, the layers are defined as

$$L_i := \{v \in V \cup D : \text{dist}(v) = i\} \text{ for } i \geq 0\}.$$

Observe that $L_0 = D$. The layer decomposition (L_0, L_1, \dots) can be computed in $\mathcal{O}(n)$ time using breadth-first search.

Let $k := \min\{i : |L_i| = 1\}$ and assume that such k exists. Let w be the only vertex in layer L_k . Consider the naïve simulation. Suppose at one iteration, a train visits a vertex u such that $\text{dist}(u) > k$. Observe that (i) the train only terminates when it visits a vertex in L_0 (i.e., a vertex in D); and (ii) by the construction of the layer decomposition, after visiting u , the train has to visit at least one vertex in each of the layers between $L_{\text{dist}(u)}$ and L_0 . This implies that it has to visit w , before it can visit any layer L_j for $j < k$. Further, before the first time it visits w (after this visit at u), the train only traverses vertices in layers L_j for $j > k$. Hence, in the sequence of the vertices that the train visits, we can remove the substring from u to the first subsequent occurrence of w . Note that we do not change the output of the multi-run procedure by doing so, since for that only the last vertex that the train visits counts. Effectively, we can redirect all edges whose head is u (including the edge (Y, u)) to w . We can do this for all applicable u . In other words, we can contract all vertices in layers L_j to w , for $j > k$.

Either after we have performed the contraction above or when there is no such k at the beginning, we obtain a switch graph where each layer has at least two vertices, except for the last layer, which may have only one. We call a switching system with such a switch graph *reduced*. In a reduced switching system, since the layers except for the last have at least two vertices, using an analogous argument as in Lemma 3.3, the bound for the number of proper edge traversals by one train is maximized, when each of these layers has exactly

two vertices and the last layer has $n - 2\ell + 2$ vertices, where $\ell = \max\{i : |L_i| > 0\}$ is the maximum length of the shortest path from a vertex in V to a destination. The number of proper edge traversals by one train is then bounded by $(n - 2\ell + 3)2^\ell - 4$. Coupled with Lemma 3.1, we obtain the following:

Corollary 3.5. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating reduced switching system, $|V| = n$, $S = \sum_{v \in V} s(v)$. Further, let ℓ be the maximum length of the shortest path from a vertex in V to a vertex in D . Then any execution of Algorithm 2 (Multi-run procedure) traverses at most $S((n - 2\ell + 3)2^\ell - 4)$ proper edges in total.*

By a simple calculation, the function $f(\ell) = (n - 2\ell + 3)2^\ell - 4$ is maximized when ℓ is roughly $n/2$. (Also note that as the switching system is reduced, ℓ is at most $n/2$). Hence, the worst-case time complexity for reduced switching system is $\mathcal{O}(S2^{n/2})$.

3.3 Greedy simulations of the train runs

One straightforward way to improve the runtime of the simulation is to pick the maximal number of trains at each iteration instead of only one train as in the naïve simulation. In other words, at every iteration of the while loop in Algorithm 2 (Multi-run procedure), after picking a vertex $v \in V$, we always choose $\rho = t[v]$. We call such a simulation *greedy*.

The next question is how to choose a vertex at each iteration. The greedy approach is to pick the vertex with the most waiting trains at every iteration. (Note that the greediness defined in the preceding paragraph is different from this greedy approach. The former refers to the choice of the number of trains, while the latter refers to the choice of a vertex. In fact, we can call this version the *doubly greedy simulation*.) We showed in [77] that this takes at most $\mathcal{O}((\ln S + n)nB)$ iterations, where $B = B(n)$ is an upper bound on

the number of iteration for one train from any vertex to a destination. Compared this bound to the bounds of the naïve simulation in the previous sections, we can see that this maximal simulation takes fewer iterations when $S = \Omega(n^2)$. However, we note that at every iteration, the greedy approach requires an overhead of finding the vertex with the most waiting train at every iteration. This introduces an additional $\mathcal{O}(\log(n))$ factor to the runtime, by using max heap. (It is unclear whether a more efficient implementation of priority queue can have a better runtime.)

In this section, we present a round robin version that repeatedly cycles through the vertices in some fixed order (v_1, \dots, v_n) . We use the order as discussed in the thesis of Ge [78]. This approach does not introduce the $\mathcal{O}(\log(n))$ overhead per iteration and also removes a factor n from the number of iterations. The order of the vertices is obtained from the layer decomposition. In particular, we process the vertices layer by layer, from those farthest to those nearest to the destinations. The order of the vertices within a layer is arbitrary. More formally, the only constraint for the order is that for two vertices $u \in L_i$ and $v \in L_j$, if $i > j$, then u precedes v .

For convenience, let R be the round robin maximal simulation described above. We now analyze the number of iterations. The idea of the analysis is similar to that in [78]. However, we present a different argument that uses a charging scheme on a slower procedure, which we call P and describe as follows.

We process the vertices in *rounds*, where each round corresponds to one cycle over the vertices in the order (v_1, \dots, v_n) above. At the beginning of each round, we mark all the waiting trains as *active* and each train at a vertex v is charged with $1/2^{\text{dist}(v)}$. (The trains in D are marked as inactive.) At iteration k of a round, we process vertex v_k . Suppose there are a active trains at v_k . We route $\lfloor a/2 \rfloor$ active trains to each successor of v_k . At least one successor u of v satisfies $\text{dist}(u) = \text{dist}(v) - 1$. For the $\lfloor a/2 \rfloor$ active trains routed

to u , we double their charges. For the other $\lfloor a/2 \rfloor$ active trains to the other successor, we remove their charges, and mark them as inactive. We also route all the inactive trains from v , and hence, the number of remaining trains at v is 0, if a is even, and 1, if a is odd. (The remaining train is then an active train.)

It is easy to see the following:

- At any point during the process, the charge of an active train at a vertex v is $1/2^{\text{dist}(v)}$.
- During each round, the total charge does not change.
- At the end of each round, each vertex $v \in V$ has at most one active train.

First, we observe that the procedure P above is slower than the round robin maximal simulation R in the following sense.

Lemma 3.6. *After the same number of rounds, the number of trains routed from each vertex in P is at most that in R .*

Proof. For $i \geq 1$ and $j \in [n]$, let p_{ij}^+ and r_{ij}^+ be the total number of trains routed from v_j up to the end of round i in P and in R , respectively. Further, for those i, j and $k \in [n]$, let p_{ijk} and r_{ijk} be the total number of trains routed from v_j to v_k up to the end of round i in P and in R , respectively. We define all numbers above to be 0, for the round index $i \leq 0$. By the switching behaviors at the vertices, for all applicable i, j, k , if $p_{ij}^+ \leq r_{ij}^+$ then $p_{ijk} \leq r_{ijk}$.

Now, we prove $p_{ij}^+ \leq r_{ij}^+$ by double induction on $i \geq 0$ and $j \in [n]$. This is true for $i = 0$ and any j , i.e., before the simulations start. In P , the total number p_{ij}^- of trains routed to v_j up to the point immediately before we process v_j in round i is the sum of

- the number $p_{(i-1)j'j}$ of trains routed from $v_{j'}$ to v_j up to round $i - 1$ for all $j' > j$ and

- the number $p_{ij''j}$ of trains routed from $v_{j''}$ to v_j up to round i for all $j'' < j$.

We analogously define r_{ij}^- and have

$$r_{ij}^- = \sum_{j' > j} r_{(i-1)j'j} + \sum_{j'' < j} r_{ij''j}.$$

Since the individual summand for p_{ij}^- is at most the corresponding summand for r_{ij}^- by the inductive hypothesis, we conclude that $p_{ij}^- \leq r_{ij}^-$. Recall that $s(v_j)$ is the number of starting trains at v_j . We then have

$$r_{ij}^+ = r_{ij}^- + s(v_j) \geq p_{ij}^- + s(v_j) \geq p_{ij}^+.$$

This completes the induction proof. \square

Now we analyze P . Let W_i be the total number of waiting trains at the beginning of round i for $i = 1, \dots, t$, where t is the number of iterations until the number of waiting trains is at most $2n$. For $j = 1, \dots, \ell$, let W_{ij} be the number of waiting trains in layer L_j at the beginning of round i , and let A_{ij} be the number of active trains in layer L_j at the end of round i . Then the number of trains reaching D in round i is precisely $W_i - W_{i+1}$. On the other hand, by the observations above, the active trains at the destinations have a charge of 1, and hence, the number of trains reaching D in round i is at least the difference between the total charge at the beginning of the round and the total charge in V at the end of the round, that is

$$\sum_{j=1}^{\ell} \frac{W_{ij}}{2^j} - \sum_{j=1}^{\ell} \frac{A_{ij}}{2^j}.$$

Let $X_i = W_i - n - \sum_j A_{i-1,j}/2^j$. Assume that $X_i > 0$. Then

$$X_i - X_{i+1} = (W_i - W_{i+1}) + \sum_{j=1}^{\ell} \frac{-A_{i-1,j} + A_{i,j}}{2^j}$$

$$\begin{aligned}
&\geq \sum_{j=1}^{\ell} \frac{W_{ij} - A_{ij}}{2^j} + \sum_{j=1}^{\ell} \frac{-A_{i-1,j} + A_{i,j}}{2^j} \\
&= \sum_{j=1}^{\ell} \frac{W_{ij} - A_{i-1,j}}{2^j} \geq \sum_{i=1}^{\ell} \frac{W_{ij} - A_{i-1,j}}{2^{\ell}} \quad (3.1)
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2^{\ell}} W_i - \frac{1}{2^{\ell}} \sum_{j=1}^{\ell} A_{i-1,j} \geq \frac{1}{2^{\ell}} W_i - \frac{1}{2^{\ell}} n \quad (3.2) \\
&\geq \frac{1}{2^{\ell}} X_i.
\end{aligned}$$

The inequality (3.1) holds, because the numerator of each summand $W_{ij} - A_{i-1,j}$ is the number of inactive trains in layer L_j at the end of round $i - 1$, and hence nonnegative. The inequality (3.2) holds, because at the end of each round, each vertex in V has at most 1 active train, and hence $\sum_j A_{i-1,j} \leq n$.

Therefore, as long as $X_i > 0$, it is reduced by a factor of at least $(1 - 1/2^{\ell})$ after the round. If $t \geq 2^{\ell} \ln S$, we have

$$\begin{aligned}
X_{t+1} &\leq \left(1 - \frac{1}{2^{\ell}}\right)^t X_1 \leq \left(1 - \frac{1}{2^{\ell}}\right)^{2^{\ell} \ln S} (S - n) \\
&< e^{-\ln S} (S - n) < 1.
\end{aligned}$$

Consequently, after at most $2^{\ell} \ln S$ rounds in P , the value of X_i must be nonpositive. Since $\sum_j A_{i-1,j} \leq n$, the number of waiting trains is then at most $2n$.

By Lemma 3.6, after the same number rounds, the number of trains that have been routed to the destinations in the procedure P is at most that in round robin maximal simulation R . Therefore, after at most $2^{\ell} \ln S$ rounds of R , we have at most $2n$ waiting trains. As each round has n iterations, the corresponding number of iterations up to this point is $\mathcal{O}(n2^{\ell} \ln S)$. By Lemmata 3.1 and 3.3, the

number of remaining iterations is at most $2n \cdot ((n - \ell + 2)2^\ell - 2) = \mathcal{O}((n - \ell)n2^\ell)$. Therefore, we obtain the following bound:

Lemma 3.7. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system, $|V| = n$, $S = \sum_{v \in V} s(v)$. Further, let ℓ be the maximum length of the shortest path from a vertex in V to a vertex in D . Then the round robin maximal simulation of of Algorithm 2 (Multi-run procedure) requires $\mathcal{O}((\ln S + n - \ell)n2^\ell)$ iterations.*

3.4 A general framework

We now move slightly away from simulating Algorithm 2 (Multi-run procedure) and consider an alternative approach. We can try to get hold of a switching flow; via Theorem 2.17, this allows us to decide G-ARRIVAL. By Definition 2.13, a switching flow can be obtained by finding a feasible solution to an integer linear program (ILP). This is a hard task in general, and it is unknown whether switching flow ILPs can be solved more efficiently than general ILPs.

In this section, we develop a framework that allows us to reduce the G-ARRIVAL problem to that of solving a number of G-ARRIVAL instances. However, these new G-ARRIVAL instances take less time to solve than the original G-ARRIVAL instance. Additionally, the framework also produces a switching flow and solves the search variant GS-ARRIVAL.

3.4.1 The idea

Given a terminating G-ARRIVAL instance, we consider the switching flow conditions in Definition 2.13. Given an arbitrary fixed subset $R = \{v_1, \dots, v_k\} \subseteq V$ of k vertices, we drop the flow conservation constraints at the vertices in R , but at the same time prescribe outflow values $x^+(v_1), \dots, x^+(v_k)$ that we can think of as guesses for their values in a switching flow.

If we find a flow that satisfies the switching behavior and the modified flow conservation subject to these guesses, we obtain unique inflow values $x^-(v_1), \dots, x^-(v_k)$ for the vertices in R (Lemma 3.10 (i) below). If we happen to stumble upon a fixed point of the mapping $x^+(v_1), \dots, x^+(v_k) \rightarrow x^-(v_1), \dots, x^-(v_k)$, we recover flow conservation also at R , which means that our guesses were correct and we have obtained a switching flow.

The crucial property is that the previously described mapping is *monotone* (Lemma 3.10 (ii) below), meaning that the theory of *Tarski fixed points* applies that guarantees the existence of a fixed point as well as efficient algorithms for finding it (Lemma 3.11).

Hence, we reduce the computation of a switching flow to a benign search problem (for a Tarski fixed point), where every search step requires us to find a solution of a “guessing” ILP. Instead of resorting to general purpose ILP solvers, we can obtain such a solution, by simply simulating Algorithm 2 (Multi-run procedure) on another switching system. For appropriate choices of the set R , it will be fast enough to bring the overall runtime down to subexponential regime and even further for a suitable original switching system.

3.4.2 Candidate switching flows and guessing switching system

Based on the idea described previously, we would like to find a *candidate switching flow* according to the following definition.

Definition 3.8 (Candidate switching flow). *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system with edges E , $R = \{v_1, v_2, \dots, v_k\} \subseteq V$, $w = (w_1, w_2, \dots, w_k) \in \mathbb{N}_0^k$.*

A function $x : E \rightarrow \mathbb{N}_0$ is a candidate switching flow for A (with respect to R and w) if

$$\begin{aligned}
x_{(Y,v)} &= s(v), & v \in V \\
x^+(v) - x^-(v) &= 0, & v \in V \setminus R \\
x_{(v,s_{\text{even}}(v))} - x_{(v,s_{\text{odd}}(v))} &\in \{0, 1\}, & v \in V \\
x^+(v_i) &= w_i, & i = 1, 2, \dots, k.
\end{aligned} \tag{3.3}$$

Note that the difference between the definition of a candidate switching flow from a switching flow (Definition 2.13) is that the flow conservation only holds for the vertices in $V \setminus R$ instead of V and the addition of the prescribed outflow w at R .

In order to find a candidate switching flow, we consider a *guessing switching system*, defined as follows.

Definition 3.9 (Guessing switching system). *Let A, R, w be as in Definition 3.8. A guessing switching system (A, R, w) defines a switching system $(V \setminus R, D \cup R, s', s'_{\text{even}}, s'_{\text{odd}})$, where*

- s' is a function $V \cup D \rightarrow \mathbb{N}_0$, such that $s'(v) = 0$ for $v \in D$, and for $v \in V$, the value of $s'(v)$ is

$$\mathbb{1}_{R(v)}s(v) + \sum_{i:s_{\text{even}}(v_i)=v} \left\lfloor \frac{s(v_i) + w_i}{2} \right\rfloor + \sum_{i:s_{\text{odd}}(v_i)=v} \left\lfloor \frac{s(v_i) + w_i}{2} \right\rfloor,$$

where $\mathbb{1}_R(v)$ takes value 1 if $v \in R$ and 0 otherwise.

- For $v \in V \setminus R$, $s'_{\text{even}}(v) = s_{\text{even}}(v)$ and $s'_{\text{odd}}(v) = s_{\text{odd}}(v)$.

Put differently, a guessing switching system is obtained from the original switching system as follows. For each vertex v_i in R , we route all $s(v_i) + w_i$ starting trains from v_i . After that, we consider the vertices in R as destinations (i.e., we move them from V to D and delete all outgoing edges from R while keeping the incoming edges intact).

Note that in the formal definition above, we abuse the definition of G-ARRIVAL a little by extending the domain of s' from $V \setminus R$

to $V \cup D$ to account for the case where a vertex in R may have a successor also in R . However, it is easy to see that this change only means that some trains are sent directly from the train yard Y to some destinations and does not invalidate any result that we have for G-ARRIVAL up to this point.

Lemma 3.10 (The mapping of outflows to inflows is monotone). *Let A, R, w be as in Definition 3.8 and let \hat{x} be a candidate switching flow for A with respect to R and w . For fixed A, R , define $F(w) = (\hat{x}^-(v_1), \dots, \hat{x}^-(v_k)) \in \mathbb{N}_0^k$. Then the following statements hold.*

- (i) *$F(w)$ does not depend on the choice of \hat{x} .*
- (ii) *The function $F : \mathbb{N}_0^k \rightarrow \mathbb{N}_0^k$ is monotone, meaning that $w \leq w'$ implies that $F(w) \leq F(w')$.*

Proof. For part (i), it is easy to see a one-to-one correspondence between a switching flow for the guessing switching system (A, R, w) and a candidate switching flow for A with respect to R and w . By Theorem 2.17, it follows that for any candidate switching flow, the inflows at R are the same. This is exactly the statement of part (i).

The proof of (ii) is now an easy consequence. By part (i), we can choose \hat{x} to correspond to the run profile of (A, R, w) , i.e., the switching flow obtained from a simulation of Algorithm 2 (Multi Run Procedure). $F(w)_i$ is the number of trains that arrive at v_i . If $w \leq w'$, we run Algorithm 2 (Multi Run Procedure) with the guessing switching system (A, R, w') as input, such that it first simulates a run with input (A, R, w) ; for this, we keep the extra trains corresponding to $w' - w$ waiting at the successors in $G(A)$ of vertices in R , until all other trains have terminated. At this point, we have inflows $f \geq F(w)$ at R . We finally run the extra trains that are still waiting, and as this can only further increase the inflows at R , we get $F(w') \geq f \geq F(w)$. \square

3.4.3 Tarski fixed points

Tarski fixed points arise in the study of order-preserving functions on complete lattices [158]. For our application, it suffices to consider finite sets of the form $L = \{0, 1, \dots, N\}^k$ for some $N, k \in \mathbb{N}^+$. For such a set, Tarski's fixed point theorem [158] states that any monotone function $D : L \rightarrow L$ has a fixed point, some $\hat{w} \in L$ such that $D(\hat{w}) = \hat{w}$. Moreover, the problem of finding such a fixed point has been studied: Dang, Qi and Ye [54] have shown that a fixed point can be found using $\mathcal{O}(\log^k N)$ evaluations of D . Recently, Fearnley, Pálvölgyi and Savani [74] improved this to $\mathcal{O}(\log^{2\lceil k/3 \rceil} N)$.

Via Lemma 3.10, we have reduced the problem of deciding a terminating G-ARRIVAL instance to the problem of finding a fixed point of a monotone function $F : \mathbb{N}_0^k \rightarrow \mathbb{N}_0^k$, assuming that we can efficiently evaluate F . Indeed, if we have such a fixed point, the corresponding candidate switching flow is an *actual* switching flow and hence decides the problem via Theorem 2.17.

The function F depends on a set $R \subseteq V$ of size k that we can choose freely (we will do so in the subsequent sections).

Here, we still need to argue that we can restrict F to a finite set $L = \{0, 1, \dots, N\}^k$ so that the Tarski fixed point theorem applies. We already know that outflow (and hence inflow) values never exceed $N = S2^n$ in *some* switching flow, namely the run profile (Lemma 3.2), where we recall that S denotes the number of starting trains. So we simply restrict F to this range and at the same time cap the function values accordingly.

Lemma 3.11. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating G-ARRIVAL instance, $R = \{v_1, \dots, v_k\} \subseteq V$, $|V| = n$, $S = \sum_{v \in V} s(v)$. Let F be the function defined in Lemma 3.10, let $N = S2^n$, and con-*

sider the function $g : \{0, 1, \dots, N\}^k \rightarrow \{0, 1, \dots, N\}^k$ defined by

$$g(w) = \begin{pmatrix} \min(N, F(w)_1) \\ \min(N, F(w)_2) \\ \vdots \\ \min(N, F(w)_k) \end{pmatrix}, \quad w \in \{0, 1, \dots, N\}^k.$$

Then g is monotone and has a fixed point \hat{w} that can be found with $\mathcal{O}(\log^{2\lceil k/3 \rceil} N)$ evaluations of g . Moreover, \hat{w} corresponds to also a fixed point of F , and hence a candidate switching flow for A w.r.t. R and \hat{w}) corresponds to a switching flow for A .

Proof. Monotonicity is clear: if $w \leq w'$, then $F(w) \leq F(w')$ by monotonicity of F ; see Lemma 3.10 (ii). But then also $g(w) \leq g(w')$ for the capped values. Hence, the Tarski fixed point theorem [158] yields a fixed point \hat{w} of g , and the algorithm of Fearnley, Pálvölgyi and Savani [74] finds it using $\mathcal{O}(\log^{2\lceil k/3 \rceil} N) = \mathcal{O}((n + \ln S)^{2\lceil k/3 \rceil})$ evaluations.

It remains to prove that \hat{w} is a fixed point of F . Suppose for a contradiction that it is not a fixed point. Then $g(\hat{w}) \leq F(\hat{w})$, i.e. some values were actually capped, and so $\hat{w}_j = g(\hat{w})_j = N < F(\hat{w})_j$ for at least one j . As we also have $\hat{w} = g(\hat{w}) \leq F(\hat{w})$, we get

$$\sum_{i=1}^k \hat{w}_i < \sum_{i=1}^k F(\hat{w})_i. \quad (3.4)$$

On the other hand, consider the candidate switching flow (3.3) with $w = \hat{w}$. Since the total flow emitted is absorbed at either R or D , (3.4) implies that the flow units arrive at D is less than S .

But this is a contradiction to $\hat{w}_j = N = S2^n$: By the same arguments as in the proof of Lemma 3.2, based on flow conservation (at all $v \neq v_j$) and switching behavior, out of these $S2^n$ outflow units from v_j , $S2^n/2^{\text{dist}(v)} \geq S$ units are guaranteed to arrive at D . \square

Remark 3.12. *Let A, R, \hat{w} be as defined in the theorem above. Observe that if we perform a simulation of Algorithm 2 (Multi-run procedure) for (A, R, \hat{w}) , we obtain a switching flow that is the run profile of this guessing switching system, and therefore it is flow-minimal for (A, R, \hat{w}) (Theorem 2.18). However, the corresponding switching flow for A is not necessarily flow-minimal, so we cannot argue that we obtain the run profile of A . The function g may have several fixed points, each of them leading to a different switching flow; to obtain the run profile, we would have to find a particular fixed point, the one that leads to the unique switching flow for smallest total flow. In other words, we want the least Tarski fixed point, which is an NP-hard problem even in one dimension [70]. The known Tarski fixed point algorithms cannot do this, and we do not know of any efficient method for computing the run profile from a given switching flow.*

3.5 Subexponential algorithm for G-ARRIVAL

In this section, we present our main application of the general framework developed in the previous section.

Given a terminating G-ARRIVAL instance A with $|V| = n$, the plan is to construct a set $R \subseteq V$ of size $\mathcal{O}(\sqrt{n})$ such that from any vertex, the length of the shortest path in $G(A)$ to a vertex in $R \cup D$ is also bounded by roughly $\mathcal{O}(\sqrt{n})$. Since R is that small, we can find a Tarski fixed point with a subexponential number of F -evaluations; and since shortest paths are that short, each F -evaluation can also be done in subexponential time using the round robin maximal simulation of Algorithm 2 (Multi-run procedure) (Lemma 3.7). An overall subexponential algorithm ensues.

Lemma 3.13. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system with $|V| = n$. Let $\phi \in (0, 1)$ be a real number. In $\mathcal{O}(n)$ time, we can construct a ϕ -set R , meaning a set $R \subseteq V$ such that*

- (i) $|R| \leq \phi \cdot (n + |D|) \leq 3\phi n$;
- (ii) *for all $v \in V$, the shortest path from v to $R \cup D$ in $G(A)$ has length at most $\log(n + |D|)/\phi \leq \log(3n)/\phi$.*

Proof. We adapt the ball-growing technique of Leighton and Rao [115], as explained by Trevisan [161].

We first compute the layer decomposition (L_0, \dots, L_ℓ) of the switch graph $G(A)$ in $\mathcal{O}(1)$ time, as explained in Section 3.2.

Algorithm 3: Procedure to compute a ϕ -set

Input: ARRIVAL instance with layer decomposition

$(L_0, \dots, L_\ell), \phi \in (0, 1)$

Output: a ϕ -set R

$R \leftarrow \emptyset$

$U \leftarrow L_0$

for $i = 1, \dots, \ell$ **do**

if $|L_i| < \phi|U|$ **then**

$R \leftarrow R \cup L_i$

$U \leftarrow \emptyset$

$U \leftarrow U \cup L_i$

return R

Consider Algorithm 3 that computes a ϕ -set as a union of layers. Intuitively, a “ball” is a union of consecutive layers. To grow a new ball U , we first add a layer with the lowest index that is not in a ball yet. Then we keep adding the succeeding layer into the ball, as long as doing this grows the size of the current ball by a factor of at least $1 + \phi$. The ϕ -set is then the union of the first layers added to the balls.

It is clear that the procedure is done in $\mathcal{O}(n)$ time. To prove (i), we observe that whenever we add a layer L_i to R , we have $|L_i| < \phi|U|$; moreover, the U 's considered in these inequalities are mutually disjoint subsets of $V \cup D$. Hence, $|S| < \phi \cdot (n + |D|)$.

For (ii), let $v \in V$. Then $v \in L_b$ for some $b \geq 1$. Let $0 \leq a \leq b$ be the largest index such that $L_a \subseteq R \cup D$. Then the shortest path from v to a vertex in $R \cup D$ has length at most $b - a$. It remains to bound $j := b - a$. The interesting case is $j > 0$.

Consider Algorithm 3. After the a -th iteration, $|U| = |L_a| \geq 1$. Moreover, $|L_i| \geq \phi|U|$ for $i = a + 1, \dots, b$, meaning that for each iteration i in this range, the size of U has grown by a factor of at least $1 + \phi$. Hence, after the b -th iteration, $(1 + \phi)^j \leq |U| \leq n + |D|$. This implies $j \leq \log(n + |D|) / \log(1 + \phi) < \log(n + |D|) / \phi$, where we use the inequality $\log(1 + \phi) > \phi$ for $\phi \in (0, 1)$.

The last inequalities in both (i) and (ii) follow the assumption that $|D| \leq 2n$. \square

Theorem 3.14. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating G-ARRIVAL instance with $|V| = n$. A can be decided in time $n^{\mathcal{O}(\sqrt{n})}$.*

Proof. By Lemma 3.13, we can find a ϕ -set R in $\mathcal{O}(n)$ time, for any $\phi \in (0, 1)$. As $|S| \leq \phi \cdot (3n)$, by Lemma 3.11, we can then decide A with $\mathcal{O}\left((n + \log S)^{2\lceil \phi n \rceil}\right)$ evaluations of the function g . Each evaluation in turn requires us to evaluate the function F in Lemma 3.10 (ii) for a given $w \in \{0, 1, \dots, S2^n\}^{|R|}$. We can do this by applying Algorithm 2 (Multi-run procedure). By Lemma 3.7 and the definition of a ϕ -set in Lemma 3.13, running this algorithm in a round robin maximal fashion requires at most $\mathcal{O}((\ln W + n - \ell)n2^\ell)$ iterations, where $W = S + \sum_{i=1}^{|R|} w_i$ and $\ell = \log(3n)/\phi$. Further, from the choice of w , we have $W \leq S2^n 3\phi n + S$. Therefore, the number of iterations is $\mathcal{O}(q(n)n^{\log 3/\phi})$ for some polynomial q . (Recall that we assume $\log(S)$ is polynomial in n .) Each iteration of

the procedure takes polynomial time.

In total, the runtime of the whole process is

$$\mathcal{O}\left((n + \log S)^{2\lceil\phi n\rceil} \cdot p(n)n^{\log 3/\phi}\right)$$

for some polynomial p . Choosing $\phi = \Theta(1/\sqrt{n})$ and noting that $\log S$ is polynomial in n , the runtime becomes $n^{\mathcal{O}(\sqrt{n})}$. \square

3.6 Feedback vertex sets

In the previous section, we used our framework to obtain an improved algorithm for G-ARRIVAL in general. In this section, we will instantiate the framework differently to obtain a polynomial-time algorithm for a certain subclass of G-ARRIVAL.

A subset $R \subseteq V$ of vertices in a directed graph $G = (V, E)$ is called a *feedback vertex set* if and only if the subgraph induced by $V \setminus R$ is acyclic (i.e. it contains no directed cycle). Karp [102] showed that the problem of finding a smallest feedback vertex set is NP-hard. However, there exists a parameterized algorithm by Chen et al. [48] which can find a feedback vertex set of size k in time $\mathcal{O}(n^4 4^k k^3 k!)$ in a directed graph on n vertices, or report that no such set exists.

It turns out that if the switch graph of a switching system is acyclic, the maximal simulation of Algorithm 2 (Multi-run procedure) takes polynomial time. This implies that we get a polynomial-time algorithm for G-ARRIVAL if there is a feedback vertex set of constant size k .

Theorem 3.15. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a terminating switching system with switch graph $G(A)$. If $G(A)$ has a feedback vertex set $R \subseteq V$ of size k (assumed to be fixed as $n = |V| \rightarrow \infty$), then A can be decided in time $\mathcal{O}(n^{2\lceil k/3 \rceil + 1}(n + \log S))$.*

Proof. Using the algorithm by Chen et al. [48], we can find a feedback vertex set S in $\mathcal{O}(n^4)$ time if it exists. According to Lemma 3.11, we can then decide A with $\mathcal{O}(n^{2\lceil k/3 \rceil})$ evaluations of the function g . Each evaluation in turn requires us to evaluate the function F in Lemma 3.10 for a given $w \in \{0, 1, \dots, S^{2^n}\}^k$. To do this, we apply Algorithm 2 (Multi-Run Procedure) where we pick vertices $v \in V \setminus S$ in topological order and choose $\rho = t[v]$ always. As we never send any trains back to vertices that have previously been picked, we terminate within $n - k$ iterations, each of which can be performed in time $\mathcal{O}(n + \log S)$ as it involves $\mathcal{O}(n + \log S)$ -bit numbers. Hence, $F(w)$ can be computed in $\mathcal{O}(n(n + \log S))$ time.

Overall, this gives a runtime of

$$\mathcal{O}\left(n^4 + n^{2\lceil k/3 \rceil + 1}(n + \log S)\right).$$

For $k \geq 1$, the second term dominates the first term. For $k = 0$, we can check if a graph is acyclic, for example, via topological sorting in $\mathcal{O}(n)$ time and do not need to employ the algorithm by Chen et al. [48]. The claimed runtime follows. \square

We remark that even if k is not constant, we can still beat the subexponential algorithm in Section 3.5, as long as $k = \mathcal{O}(n^\alpha)$ for some $\alpha < 1/2$.

3.7 Discussion and open questions

Other applications of the framework We have presented a general framework with three major steps: choosing a subset R , searching for a fixed point in the function $F(w)$ as defined in Lemma 3.10, and finding a candidate switching flow given a guessed w . In both applications in this chapter, we use Tarski fixed points for the second step and a form of maximal simulation for the third step. Further, as discussed earlier, the third step is essentially solving

some constrained G-ARRIVAL instances. Therefore, if we find a faster method to solve G-ARRIVAL or a suitable subclass of G-ARRIVAL, we may bootstrap that method by using it for the third step and choosing an appropriate subset R . Likewise, any improvement on algorithms to find Tarski fixed points will give a better time bound on any algorithms in this framework. However, we believe it is unlikely that this framework will give rise to a polynomial-time algorithm for a general G-ARRIVAL instance, as long as the query complexity in the second step of the framework has an $\Omega(|R|)$ term in the exponent.

The train yard The artificial vertex Y is used to avoid some technical details in applying Tarski fixed point theory. Without the train yard, the starting trains will be at the vertices in V at the beginning, and the system (3.3) in the definition of a candidate switching flow (3.8) becomes

$$\begin{aligned} x^+(v) - x^-(v) &= s(v), & v \in V \setminus R \\ x_{(v, s_{\text{even}}(v))} - x_{(v, s_{\text{odd}}(v))} &\in \{0, 1\}, & v \in V \\ x^+(v_i) &= w_i, & i = 1, 2, \dots, k. \end{aligned}$$

Since the net inflow at every vertex in V is not necessarily 0, we are no longer looking for a fixed point of function F as defined in Lemma 3.10. In order to apply Tarski fixed point theory, we need to define F differently to account for s . This makes the explanation of the framework more complicated.

Open questions

- The question of whether G-ARRIVAL is in P is still wide open. We would be interested to see any improvement on the upper bound (i.e., a faster algorithm) or on the lower bound, such as whether ARRIVAL is P-hard as conjectured in [7].

- As mentioned in Remark 3.12, our subexponential algorithm only produces a switching flow and not the run profile. This gives rise to the complexity question for the problem of finding the run profile (i.e., the flow-minimal switching flow), given a switching flow. We comment that this is related to the complexity of finding the rotor-routing action as defined by Holroyd, Levine, Mészáros, Peres, Propp, and Wilson [92].
- Tóthmérész [160] defined the following maximal r -bounded rotor-routing game, whose complexity is still open. We are given a rotor-routing game of a ribbon graph with edge set E and an initial chip-and-rotor configuration and a function $r : E \rightarrow \mathbb{N}_0$. We can move the chips as per the rotor-router rules, but each edge e can be traversed at most $r(e)$ times. The game ends when there is no further possible move. The goal is to find the *maximal odometer* that records the number of times we have traversed each edge until the game ends. This is analogous to finding the run profile of a switching system in the uncipitated setting of G-ARRIVAL.

Two roads diverged in a yellow wood,

—Robert Frost,
The Road Not Taken

CHAPTER 4

G-ARRIVAL with two vertices per layer

This chapter is based on unpublished joint work with Bernd Gärtner.

In the previous chapter, we present a subexponential algorithm for a general instance of G-ARRIVAL and a polynomial time algorithm for a subclass. Haslebacher [85] also identified some special subcases of ARRIVAL, where the simulation of Algorithm 1 (Run Procedure) runs in polynomial time. In particular, this happens when there are $\mathcal{O}(\log n)$ backward edges (that take the train further away from the destinations with respect to the distance of shortest path) or when maximum in-degree is two. Interestingly, he also showed that the simulation takes exponential time on an instance where only one vertex has in-degree three and the others at most two.

In this chapter, we discuss a different subclass where we can decide G-ARRIVAL in polynomial time without simulating the train runs.

This subclass is motivated by a “hard” case of the algorithm in Section 3.2. Observe that the worst possible case is when every layer (except for the layer of the destinations) in the layer decomposition has exactly two vertices. It is also easy to see that this is also one of the worst cases for the subexponential algorithm as discussed in Section 3.5. The method to solve this subclass is inspired by the integer programming approach to solving G-ARRIVAL. However, for this chapter, we present a different and more combinatorial approach for the analysis.

4.1 Preliminaries

Definition 4.1 (Ladder and 2-ladder). *A ladder is a terminating switching system A , such that in the layer decomposition of the switch graph $G(A)$, every layer has exactly two vertices, except for the layer furthest away from the destinations, which has at most two vertices, and the layer that contains the destinations, which can have any number of vertices. A 2-ladder is a ladder with exactly two destinations.*

For most of the remaining part of the chapter, we focus on solving a G-ARRIVAL instance (A, T) where $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ is a 2-ladder. We will discuss in Section 4.7 how to use the algorithm to solve an instance where A is a ladder.

Firstly, we can assume that the last layer contains a single vertex. If it does not, it must have two vertices. We then add a layer after it with one vertex, whose out-neighbors are the vertices in the layer above and whose s -value is 0. Denote by 0 the only vertex in the last layer.

Secondly, to ease the notations in this chapter, we define n differently than in the previous chapter. In particular, we denote $n := (|V|+1)/2$. Therefore, there are $n+1$ layers in the layer decom-

position, where layer L_0 is the set of destinations $D = \{n, -n\}$ and the last layer $L_n = \{0\}$. For $i \in [n-1]$, we denote the two vertices in layer L_i by $n-i$ and $-(n-i)$. See Figure 4.1 for an illustration.

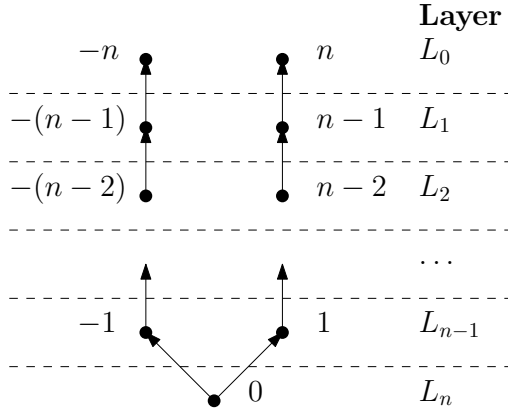


Figure 4.1: The layer decomposition of a 2-ladder

Thirdly, we can assume that there exist two vertex-disjoint directed paths from 0 to n and $-n$ respectively. Suppose there are no such paths. This implies that at some layer $L_i, i \geq 1$, all the edges from layer L_i to layer L_{i-1} end at only one vertex v in layer L_{i-1} . This is because we can otherwise construct the two vertex-disjoint paths, contradicting the hypothesis. We then observe that whenever the train goes to a layer $L_j, j \geq i$, it can only pass through v on its way to L_n . By similar arguments in Section 3.2, we can contract all the vertices in layer $L_j, j \geq i$ to v to obtain a smaller graph.

We assume that these two vertex-disjoint paths are $(0, 1, 2, \dots, n)$ and $(0, -1, -2, \dots, -n)$.

Lastly, we denote by \hat{E} the set of proper edges of $G(A)$ and define $[n]^* := \{-n, \dots, n\}$.

4.2 Algorithm Overview

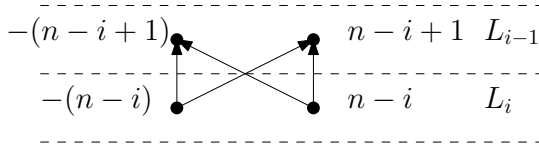
Suppose we have a G-ARRIVAL instance (A, T) where A a 2-ladder defined as above, and $T = (t_n, t_{-n})$. Recall that the goal is to decide whether T is the output of Algorithm 2 (Multi-run procedure) on input A .

Let x be a switching flow for A . In Section 4.4, we show that $x^-(n)$ has to lie in a range $[\alpha, \beta]$, where $\beta - \alpha \leq 1$. If this range contains exactly one integer, then we know exactly the value of $x^-(n)$ and decide (A, T) immediately. (Observe that we assume $t_n + t_{-n} = \sum_{v \in V} s(v)$, and so if $t_n = x^-(n)$, we must have $t_{-n} = x^-(-n)$.) Otherwise, the range contains two consecutive integers, and $x^-(n)$ is either of them. Therefore, if we can decide on the *parity* of $x^-(n)$, then coupled with the fact that $x^-(n) \in [\alpha, \beta]$, we can decide (A, T) .

Based on the above observation, we design Algorithm L that takes as input the 2-ladder A and a binary number σ . The algorithm outputs the (unique) value of $x^-(n)$ if $x^-(n) \equiv \sigma \pmod{2}$ and NO otherwise. Then we can call Algorithm L with the input A and the parity of t_n . It is easy to see that (A, T) is a YES instance, if and only if the output of Algorithm L is t_n .

Algorithm L is shown in Section 4.6. However, we will present the step-by-step explanation of the algorithm in the next few sections. We first calculate the hitting probabilities of the related Markov chain, which will be discussed in more details in Section 4.3. Based on these hitting probabilities, we can either output NO immediately (Section 4.4) or conclude that there is a *cross* at some layer L_k , defined below (see Figure 4.2 for illustration).

Definition 4.2 (Cross). *If for $i > 0$, the outgoing neighbors of the vertices of L_i are the vertices of L_{i-1} , we say that there is a cross at L_i .*

Figure 4.2: A cross at L_i

In the latter case, we can remove a part of the switch graph, obtain a smaller 2-ladder whose layer decomposition has one fewer cross, and recurse (Section 4.5).

4.3 Hitting probabilities

Consider the Markov chain induced by the switch graph $G(A)$, where every vertex in $V \cup D$ is a state, and from any $v \in V$, there is an equal probability that the next state is $s_{\text{even}}(v)$ or $s_{\text{odd}}(v)$. Here, the destinations are the only absorbing states, i.e., a state that once entered cannot be left. For $i \in [n]^*$, let h_i be the probability that a random walk starting from i will reach n . Therefore, h is the minimal nonnegative solution to the following linear system [128, Theorem 1.3.2]:

$$\begin{aligned} h_i &= 0.5h_{s_{\text{even}}(i)} + 0.5h_{s_{\text{odd}}(i)}, & \forall i \in [n-1]^* \\ h_n &= 1, \\ h_{-n} &= 0. \end{aligned} \tag{4.1}$$

Here, minimality means that if x is another nonnegative solution to the above system, then $x_i \geq h_i$ for all i . In this chapter, we do not need the h_i 's to be the minimal non-negative solution but only a solution to the above system.

Lemma 4.3 and Corollary 4.4 capture two useful properties of the h_i 's for a 2-ladder.

Lemma 4.3. $0 = h_{-n} \leq h_{-(n-1)} \leq \dots \leq h_{-1} \leq h_0 \leq h_1 \leq \dots \leq h_{n-1} \leq h_n = 1$.

Proof. We first prove that the sequence $(h_{-i}, \dots, h_{-1}, h_0, h_1, \dots, h_i)$ is monotone (i.e., it is either nonincreasing or nondecreasing), by induction on i for $i \in [n]$. After that, we will show that the sequence is nondecreasing.

Since 0 has two outgoing edges to 1 and -1 , we have $2h_0 = h_1 + h_{-1}$. This implies $h_{-1} \geq h_0 \geq h_1$ or $h_{-1} \leq h_0 \leq h_1$, and hence the monotonicity property is true for the base case $i = 1$.

Suppose that it is true for all $h_i, |i| \leq j$ for some $j \geq 1$, and for simplicity, suppose this is nondecreasing as we go through the same order as in the statement of the lemma. The proof in the case when it is nonincreasing is analogous.

The vertex $-j$ has two outgoing edges to $-(j+1)$ and another vertex ℓ_1 , where $|\ell_1| \leq j+1, \ell_1 \neq -(j+1)$. Similarly, j has two outgoing neighbors $j+1$ and ℓ_2 , where $|\ell_2| \leq j+1, \ell_2 \neq j+1$.

Suppose $\ell_1 \neq j+1$. As $\ell_1 \neq j+1$ and $\ell_1 \neq -(j+1)$, from the inductive hypothesis, $h_{-j} \leq h_{\ell_1}$. Since $2h_{-j} = h_{-j-1} + h_{\ell_1}$, we have $h_{-j-1} \leq h_{-j}$. As $\ell_2 \in \{-j-1, \dots, j-1\}$, we have $h_{\ell_2} \leq h_j$. Since $2h_j = h_{j+1} + h_{\ell_2}$, we then must have $h_j \leq h_{j+1}$.

Similarly, we can also extend monotonicity in the case where $\ell_2 \neq -(j+1)$.

Suppose now we have $\ell_1 = j+1$ and $\ell_2 = -(j+1)$. In that case, we have a cross and $2h_{-j} = h_{-j-1} + h_{j+1} = 2h_j$. In other words, $h_{-j} = h_j$. Due to the monotonicity established in the inductive hypothesis, we have all h_i 's are equal to each other for all $-j \leq i \leq j$. In that case, we can have the monotonicity property: $h_{-(j+1)} \leq h_{-j} = \dots = h_{-1} = h_0 = h_1 = \dots = h_j \leq h_{j+1}$ or $h_{-j-1} \geq h_{-j} = \dots = h_{-1} = h_0 = h_1 = \dots = h_j \geq h_{j+1}$.

We have completed the inductive proof for monotonicity. As $h_n = 1$ and $h_{-n} = 0$, the sequence must be non-decreasing. \square

For a vertex $v \in V$, by (4.1), $h_v - h_{s_{\text{even}}(v)} = h_{s_{\text{odd}}(v)} - h_v$. We define $g_v := h_v - h_{s_{\text{even}}(v)} = -(h_v - h_{s_{\text{odd}}(v)})$ to be the *gap* of v (with the even successor in terms of hitting probabilities). The following is a consequence of Lemma 4.3 above.

Corollary 4.4.

$$\sum_{v \in V} |g_v| = 1 - |g_0| \leq 1.$$

Proof. By (4.1), we have $h_n = 1$ and $h_{-n} = 0$. Hence,

$$\begin{aligned} 1 = h_n - h_{-n} &= \sum_{i=-n+1}^n (h_i - h_{i-1}) = \sum_{i=-n+1}^n |h_i - h_{i-1}| \quad (4.2) \\ &= \sum_{i=2}^n |h_i - h_{i-1}| + |h_1 - h_0| + \sum_{i=-n+1}^0 |h_i - h_{i-1}| \\ &= \sum_{i=2}^n |g_{i-1}| + |g_0| + \sum_{i=-n+1}^0 |g_i| \quad (4.3) \\ &= \sum_{i \in [n-1]^*} |g_i| + |g_0|, \end{aligned}$$

where the last equality in (4.2) follows Lemma 4.3 and the equality in (4.3) follows the facts that for i is a successor of $i-1$ for $i \in [2, n]$ and $i-1$ is a successor of i for $i \in [-n+1, 0]$.

From the equality above, together with the facts that $V = [n-1]^*$, the corollary then follows. \square

Now, we consider the case when the inequality in Corollary 4.4 above holds as equality. In that case, we can show the existence of a cross.

Lemma 4.5. *There exists a cross as defined in Definition 4.2, if and only if $g_0 = 0$ (i.e., $h_0 = h_1$). Further, if k is the smallest number such that there is a cross at L_k , then $g_v \neq 0$ for $v \in L_j$, $j \leq k$.*

Proof. We prove the first statement.

“ \Rightarrow ”: From the proof of Lemma 4.3, it is easy to see that if there is a cross at L_k then all the h_i ’s where $|i| \leq n - k$ have the same value. In particular, $h_0 = h_1$ and hence $g_0 = 0$.

“ \Leftarrow ”: Consider the maximal contiguous subsequence of equalities in the sequence in Lemma 4.3, such that h_0 is in this subsequence. Suppose the leftmost and rightmost terms are h_ℓ and h_r respectively. Since, $h_0 = h_1$, $\ell < 0$ and $r > 0$. Further, we should have either $|\ell| \geq |r|$ or $|\ell| \leq |r|$. WLOG, suppose it is the former. Let s be the other outgoing neighbor of r besides $r + 1$. Then $s < r$. We have $2h_r = h_{r+1} + h_s$. As the subsequence is maximal, $h_r \neq h_{r+1}$. Hence, $h_r \neq h_s$. For all $i \in \{\ell, \ell + 1, \dots, r - 1\}$, we have $h_i = h_r$. Therefore, $s < \ell < 0$. However, as $|\ell| \geq |r|$ and we have $(r, s) \in E$, we must have $|s| = |r + 1|$, or $s = -(r + 1)$ and $|\ell| = |r|$ (i.e., ℓ and r are in the same layer). By similar argument, we have the outgoing neighbors of ℓ are $\ell - 1$ and $r + 1$. In other words, there is a cross at L_{n-r} .

We follow the same arguments above for the second lemma statement. As mentioned in the “ \Rightarrow ” direction, if there is a cross at $L_{k'}$ for some $k' < n - r$, then $(h_{-(n-k')}, \dots, h_{n-k'})$ is a subsequence of equal hitting probabilities that is longer than the subsequence above, a contradiction to its maximality. Hence, $k := n - r$ is the smallest number such that there is a cross at L_k . Further, by the maximality of the subsequence, g_r and g_ℓ (or equivalently g_k and g_{-k}) are nonzero.

Lastly, we show that for $j < k$, $g_v \neq 0$ for $v \in L_j$. For the sake of contradiction, suppose the contrary. Let j be the highest number such that $g_v = 0$ for some $v \in L_j$. WLOG, we can assume $v = n - j$. By construction, either $s_{\text{even}}(v) < v$ or $s_{\text{odd}}(v) < v$. Since $g(v) = 0$,

we have $h_{s_{\text{even}}(v)} = h_v = h_{s_{\text{odd}}(v)}$. Couple with Lemma 4.3, this implies that $h_{v-1} = h_v$, or equivalently $g_{v-1} = 0$. Since $v-1 \in L_{j+1}$ and $g_k \neq 0$, this leads to a contradiction to the maximality of j . The lemma then follows. \square

4.4 Initial analysis

In this section, we show that for any switching flow of A , the inflow at a destination n falls into a range $[\alpha, \beta]$ for some α and β that satisfy $\beta - \alpha \leq 1$ (see Lemma 4.6 below). If the above range contains only one integer, then we know immediately the inflow at n .

Let x be a switching flow for A . Consider the following sum over the set \hat{E} of proper edges of $G(A)$

$$M := \sum_{(i,j) \in \hat{E}} x_{(i,j)} (h_i - h_j).$$

On one hand, we can sum over the vertices and obtain

$$\begin{aligned} M &= \sum_{i \in [n]^*} h_i \left(\sum_{e \in \hat{E}^+(i)} x_e - \sum_{e \in \hat{E}^-(i)} x_e \right) \\ &= \sum_{i \in [n]^*} h_i \left(\sum_{e \in E(A)^+(i)} x_e - x_{(Y,i)} - \sum_{e \in E(A)^-(i)} x_e \right) \\ &= -x^-(n) - \sum_{i \in [n-1]^*} h_i s(i). \end{aligned} \tag{4.4}$$

In the last equality, we use the equalities in the definition of a switching flow (Definition 2.13) and the facts that $h_n = 1$ and $h_{-n} = 0$.

On the other hand, we can pair up the outgoing edges of each vertex in M and obtain

$$M = \sum_{i \in [n-1]^*} (x_{(i, s_{\text{even}}(i))} (h_i - h_{s_{\text{even}}(i)}) + x_{(i, s_{\text{odd}}(i))} (h_i - h_{s_{\text{odd}}(i)}))$$

$$\begin{aligned}
&= \sum_{i \in [n-1]^*} (x_{(i, s_{\text{even}}(i))} g_i - x_{(i, s_{\text{odd}}(i))} g_i) \\
&= \sum_{x_{(i, s_{\text{even}}(i))} - x_{(i, s_{\text{odd}}(i))} = 1} g_i
\end{aligned} \tag{4.5}$$

In the second equality above, we use the definition of the gap g_i as defined in Section 4.3. In the third equality, we use the fact that $x_{(i, s_{\text{even}}(i))} - x_{(i, s_{\text{odd}}(i))} \in \{0, 1\}$ by definition of a switching flow.

Let $Q_x := \{i : x_{(i, s_{\text{even}}(i))} - x_{(i, s_{\text{odd}}(i))} = 1\}$, $P := \{i : g_i > 0\}$, and $N := \{i : g_i < 0\}$. Observe that the g_i 's are completely determined by the switch graph and do not depend on the starting trains or the switching flows. Hence, (4.5) is maximized, when the set $Q_x \supseteq P$ and $Q_x \cap N = \emptyset$. Likewise, it is minimized, when $Q_x \supseteq N$ and $Q_x \cap P = \emptyset$. Therefore, we have

$$M \in \left[\sum_{g_i < 0} g_i, \sum_{g_i > 0} g_i \right]. \tag{4.6}$$

Combining (4.4) and (4.6), we obtain $x^-(n) \in [\alpha, \beta]$, where

$$\begin{aligned}
\alpha &:= - \sum_{i \in [n-1]^*} h_i s(i) - \sum_{g_i > 0} g_i, \\
\beta &:= - \sum_{i \in [n-1]^*} h_i s(i) - \sum_{g_i < 0} g_i.
\end{aligned} \tag{4.7}$$

By Corollary 4.4, we have

$$\beta - \alpha = \sum_{g_i > 0} g_i - \sum_{g_i < 0} g_i = \sum_{i \in [n-1]^*} |g_i| = 1 - |g_0| \leq 1. \tag{4.8}$$

Combining with Lemma 4.5, we have the following lemma.

Lemma 4.6. *Let A be a 2-ladder with n being one of the destinations. Then there exist $\alpha = \alpha(G(A))$ and $\beta = \beta(G(A))$ such that $\beta - \alpha \leq 1$ and for any switching flow x for A , $x^-(n) \in [\alpha, \beta]$. Further, $\beta - \alpha = 1$, if and only if there is a cross in the layer decomposition of $G(A)$.*

Lemma 4.6 above immediately implies the following.

Corollary 4.7. *Let (A, T) be a G -ARRIVAL instance, where A is a 2-ladder as defined in Section 4.1 and $T = (t_n, t_{-n})$. Let α, β as guaranteed by Lemma 4.6. Suppose $\beta - \alpha$ contains exactly one integer; in particular, this holds when the layer decomposition of $G(A)$ does not contain a cross (equivalently, $\beta - \alpha < 1$). Then (A, T) is a YES instance, if and only if $t_n \in [\alpha, \beta]$.*

4.5 2-ladder with a cross

In this section, we consider the case where the range $[\alpha, \beta]$ contains more than one integer, where α, β are defined in the previous section. From (4.8), when this happens, the range contains exactly two integers α and β . Further, as stated in Lemma 4.6, the layer decomposition of $G(A)$ has a cross. From now on, we focus on deciding whether the inflow at n is α . If this is not true, then this inflow must be β . We first have the following lemma.

Lemma 4.8. *Let A be a 2-ladder as defined in Section 4.1, x be a switching flow for A , and α and β as defined in (4.7). Suppose that α and β are integers. Let k be the smallest number such that there is a cross at L_k in the layer decomposition of $G(A)$. If $x^-(n) = \alpha$, then for $j \leq k$ and $i \in L_j$, we have the gap $g_i \neq 0$ and*

$$x_{(i, s_{\text{even}}(i))} - x_{(i, s_{\text{odd}}(i))} = \gamma(i) := \begin{cases} 0 & \text{if } g_i < 0, \\ 1 & \text{if } g_i > 0. \end{cases}$$

The lemma above implies that if the inflow at n is α , then for any switching flow and for any vertex in the layers L_0, \dots, L_k , the difference in flows between its even and odd outgoing edges is the same for any switching flow.

Proof. As discussed above, when α and β are integers, by (4.8), we must have $g_0 = 0$. Then by Lemma 4.5, there exists a cross, and hence k exists.

Recall the definitions of M, Q_x, P , and N in the Section 4.4. Following the analysis in that section, when $x^-(n) = \alpha = \min[\alpha, \beta]$, M achieves the maximum value of the range in (4.6). That implies that $Q_x \supseteq P$ and $Q_x \cup N = \emptyset$. Equivalently, for $i \in V$, if $g_i \neq 0$ then

$$x_{(i, s_{\text{even}}(i))} - x_{(i, s_{\text{odd}}(i))} = \begin{cases} 0 & \text{if } g_i < 0, \\ 1 & \text{if } g_i > 0. \end{cases}$$

By Lemma 4.5, $g_i \neq 0$ for $i \in L_j, j \leq k$. The lemma follows. \square

From the above lemma, we can deduce something stronger: if the inflow at n is α , then the value of x_i is the same for any switching flow x and $i \in L_j, j < k$. Even more, we can compute this value efficiently.

Lemma 4.9 (Fixed flow values above highest cross). *Let A, α, β, γ be as in Lemma 4.8. Let x be a switching flow for A . If α and β are integers and $x^-(n) = \alpha$, then in polynomial time, we can compute x_e for all outgoing edges e from vertices in $L_j, j < k$.*

Proof. Let $\hat{W} := L_k \cup \dots \cup L_n$. Suppose we contract the vertices in $L_k \cup \dots \cup L_n$ of the switch graph $G(A)$ into the vertex 0 and remove the train yard Y . Let $G' = (V', E')$ be the resulting graph, where $V' = L_0 \cup \dots \cup L_{k-1} \cup \{w\}$. Since the edges going out of \hat{W} in $G(A)$ are the edges from $(n-k)$ and $-(n-k)$ to $n-k+1$ and

$-(n - k + 1)$, the outgoing edges of w in (V', E') are $(w, n - k + 1)$ and $(w, -(n - k + 1))$.

By Lemma 4.8, for $i \in [n - k, n - 1]$, we have

$$\begin{aligned} x_{(i, s_{\text{even}}(i))} - x_{(i, s_{\text{odd}}(i))} &= \gamma(i), \\ x_{(-i, s_{\text{even}}(-i))} - x_{(-i, s_{\text{odd}}(-i))} &= \gamma(-i). \end{aligned}$$

Define

$$\gamma' = \begin{pmatrix} x_{(n-k, n-k+1)} - x_{(n-k, -(n-k+1))} + \\ x_{(-(n-k), n-k+1)} - x_{(-(n-k), -(n-k+1))} \end{pmatrix}.$$

Note that $\gamma' \in \{\pm\gamma(k) \pm \gamma(-k)\}$ is a fixed number in $[-2, 2]$.

Consider the function $y : E' \rightarrow \mathbb{N}_0$ defined as follows:

$$\begin{aligned} y_e &:= x_e, & e \notin E'^+(w) \cup E'^-(w) \\ y_{(v, w)} &:= x_{(v, u)}, & (v, w) \in E'^-(w), u \in W, (v, u) \in E \\ y_{(w, v)} &:= \sum_{u \in L_k} x_{(u, v)}, & v \in L_{k-1}. \end{aligned}$$

Note that $y_{(w, n-k+1)} - y_{(w, -(n-k+1))} = \gamma'$.

Observe that for every vertex of G' has out-degree either 0 or 2 with at least one sink, and from every vertex, there is a directed path to a sink. Further, for every vertex, we prescribe the net inflow (which is governed by s) and the difference of flows between the two outgoing edges (which is governed by γ and γ'). Hence, by Lemma 2.30, y is unique and can be computed in polynomial time. Since x and y agree for all the edges going out from each vertex in L_1, \dots, L_{k-1} , the lemma then follows. \square

Reducing the switch graph. Again, suppose x is a switching flow x for A such that $x^-(n) = \alpha$, we consider the x -values for four disjoint sets of edges. Set (a) contains the edges incident to Y . Set (b) contains the edges going out from vertices in $L_{<k} :=$

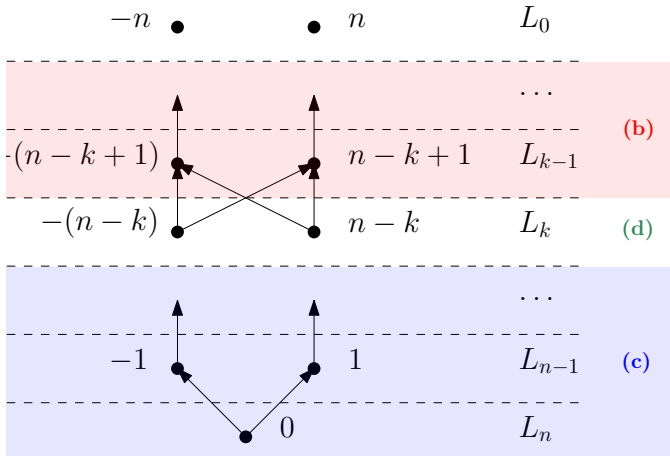


Figure 4.3: Four sets of edges for analysis of x grouped by their tails: Set (a) of edges whose tails are Y is not shown here. Set (b) contains edges whose tails are in $L_{<k}$; the corresponding x -values are fixed by Lemma 4.9. Set (c) contains edges whose tails are in $L_{>k}$; together with Y , it induces another switching system A' . Set (d) contains the edges going from L_k to L_{k-1} .

$L_1 \cup \dots \cup L_{k-1}$. Set (c) contains the remaining edges, i.e., the edges going out from vertices in $L_{>k} := L_{k+1} \cup \dots \cup L_n$. The last set (d) contains the edges that form the cross at layer L_k , i.e., the edges from $n-k$ and $-(n-k)$ to $n-k+1$ and $-(n-k+1)$. See Figure 4.3 for an illustration.

The values in set (a) are fixed by the definition of switching flow. The values in set (b) are the same for any switching flow and can be computed easily by Lemma 4.9; hence we can consider them as fixed.

For set (c), consider a switching system A' whose switch graph is the induced subgraph on $\{Y\} \cup L_k \cup L_{>k}$. The destinations of A' are the vertices in L_k . We can consider the combined fixed x -values

of the edges in the first two sets coming into $L_{>k}$ as the starting trains for A' . It is easy to see that A' is a 2-ladder and that there is a one-to-one correspondence between a switching flow for A' and the x -values for the edges in the third set.

For set (d), because of the Abelian property of switching system (Lemma 2.8), for any switching flow for A' , the inflow to $n - k$ in A' is always the same. Given that the x -values in the first two sets are also fixed, we conclude that the total inflow at $n - k$ is fixed (i.e., it is the same for any switching flow for A). By flow conservation, the total outflow equals the total inflow at $n - k$. By Lemma 4.8, we know that the difference between the flows for its outgoing edges, which is $\gamma(n - k)$. As we know the sum and the difference, we can compute the flows for each outgoing edge from $n - k$. The same analysis can be done for $-(n - k)$.

So far, it seems that there is nothing in the way of constructing a switching flow x for A such that $x^-(n) = \alpha$. However, the catch is in the integrality condition. While the switching flow for A' is integral by definition, the other values may not be. For one, the values computed in Lemma 4.9 may not be integral. Further, in the last step, although we can compute the flows for the outgoing edges of $n - k$, these values may not be integers. In fact, they are only integers when the total inflow at $n - k$ has the same parity as $\gamma(n - k)$. Since the inflow from Y and from $L_{<k}$ to $n - k$ are known, this means that we can prescribe the parity for the inflow from $L_{>k}$ to $n - k$. In order to verify if this inflow indeed has the prescribed parity, we can use Algorithm L on A' which is a smaller 2-ladder than A (in terms of number of vertices). This suggests a recursive structure of the algorithm.

Note that if the outflows at the vertices in L_{k-1} and the flows from $n - k$ to each of these vertices are integral, by flow conservation, the flows from $-(n - k)$ to each of the two vertices in L_{k-1} are also integral. Therefore, we do not need to check for $-(n - k)$.

4.6 Algorithm L

Putting the above analyses together, we can devise Algorithm L that, given a 2-ladder A and a binary number σ , outputs the inflow to the destination n or outputs NO if the parity of this inflow is not σ .

We first compute h_i by solving the linear system (4.1). We can then easily compute the gaps g_i . Next, we compute α and β as in (4.7). If $[\alpha, \beta]$ contains only one number c , then we output c , if $c \equiv 2 \pmod{2}$, and NO otherwise.

If $[\alpha, \beta]$ contains two integers, then we try to construct a switching flow x for A , where $x^-(n) = \alpha$. We first find the smallest value k such that there is a cross at layer L_k . We then compute the values of γ_i for $i \in L_j$, $j < k$, as defined in Lemma 4.8. Next, we compute x_e for every outgoing edge e from a vertex in L_j for $j < k$, using Lemma 4.9. If any of the computed values are not integral, we know immediately that $x^-(n)$ must be β .

Otherwise, based on these values, we can construct the 2-ladder $A' = (V', D', s', s'_{\text{even}}, s'_{\text{odd}})$ as follows:

$$\begin{aligned} V' &:= L_{>k} = [n - k - 1]^*, \\ D' &:= L_k = \{n - k, -(n - k)\}, \\ s'(v) &:= s(v) + \sum_{u \in L_{<k}: (u,v) \in E} x_{(u,v)}, \quad v \in V' \\ s'_{\text{even}}(v) &:= s_{\text{even}}(v), \quad v \in V' \\ s'_{\text{odd}}(v) &:= s_{\text{odd}}(v), \quad v \in V'. \end{aligned}$$

Further we define a binary number $\sigma' \in \{0, 1\}$, such that

$$\sigma' \equiv \gamma(k) - s(k) - \sum_{u \in L_{<k}: (u,k) \in E} x_{(u,k)}.$$

Then we make a recursive call on Algorithm L with input A' and σ' . If the output of this call is NO, then we know the inflow at n

for A is β (i.e., there is no switching flow x such that $x^-(n) = \alpha$). Otherwise, the inflow at n is α .

In any case, once we determine whether the inflow at n is α or β , we can compute output the original Algorithm L appropriately.

In the recursive call, we remove vertices from layer L_n to L_{k+1} . As $k < n$, we remove at least two vertices. And hence, there are $\mathcal{O}(n)$ recursive calls. The running time of each iteration is dominated by solving two linear systems for h_i and in Lemma 4.9. These runs in time $O(n^\omega)$, where ω is the matrix multiplication exponent. Therefore, in total, the algorithm runs in time $O(n^{\omega+1})$.

4.7 Solving G-ARRIVAL with ladder

Now consider a G-ARRIVAL instance (A, T) where A is a ladder, (i.e., there may be more than two destinations). Note that since there are only two vertices at L_1 , we can assume that there are at most four destinations. Now assume that there are exactly four destinations d_1, d_2, d_3, d_4 , and the corresponding guessed inflows in T are t_1, t_2, t_3, t_4 . (The case where there are three destinations is analogous.) Then we can verify each guessed inflow separately. Suppose we start with d_1 . We consider another G-ARRIVAL instance (A'', T'') where A'' is obtained by A by contracting d_2, d_3, d_4 in $G(A)$ into a new destination \bar{d} . For T'' , the guessed inflow for d_1 is t_1 and that for \bar{d} is $t_2 + t_3 + t_4$. It is easy to see that A'' is a 2-ladder, and the inflow at d_1 in a switching flow for A'' is the same as the inflow at d_1 in a switching flow for A . Therefore, we can verify if t_1 is the correct inflow, by checking if (A'', T'') is a YES instance. We then continue for the other destinations of A . Overall, we can solve this in polynomial time.

Theorem 4.10. *Let (A, T) be a G-ARRIVAL instance such that A is a ladder. Then we can decide (A, T) in polynomial time.*

4.8 Discussions

Computation of a switching flow In the analysis, we try to construct a switching flow for the switching system. However, in the method above, we always determine the inflow at n before finishing computing the switching flow. The only edges for which we can compute the flow values are those above certain cross (Lemma 4.9). However, if we follow through the recursion, we will reach a switch graph with no cross. In that case, we cannot compute the γ 's in Lemma 4.8, because this lemma only holds when both α and β are integers, and this does not hold when there is no cross.

Towards a characterization of applicable graphs Ladders only impose sufficient conditions for the above method to work. It is still open what the necessary conditions are, i.e., which other graphs can the above method apply to. In the rest of this section, we provide some sufficient conditions.

We first extend the notion of reduction discussed in Section 3.2. Suppose we have a switching system $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$. If there is a subset R of $V(A)$ such that among the out-neighbors of the vertices in R , there is only one out-neighbor $v \notin R$. In other words, if a train in R visits a vertex outside R , it has to visit v first. Then following the same analysis as in Section 3.2, we can redirect all edges whose heads in R to v . We called a graph *reduced*, if there is no such subset R . Note that we can identify such a subset via the layer decomposition of $G(A)$. Also observe that the reduction above preserves the terminating property of the switching system.

Next, we give a characterization for switching systems with two destinations, where the initial analysis in Section 4.4 can apply.

Lemma 4.11. *Let $A = (V, D, s, s_{\text{even}}, s_{\text{odd}})$ be a reduced terminating switching system with $D = \{n, -n\}$. Let h be a solution of the*

system (4.1). For $v \in V$, define the gap $g_v := h_v - h_{\text{seven}(v)}$. Then the following are equivalent:

- (a) $\sum_{v \in V} |g_v| < 1$, and
- (b) There are two directed paths $(v_0, v_1, \dots, v_p = n)$ and $(v_0, v_{-1}, \dots, v_{-q} = -n)$, such that for any vertex $v \in V$, v is on at least one path, and

$$0 = h_{v_{-q}} < \dots < h_{v_{-1}} < h_{v_0} < h_{v_1} < \dots < h_{v_p} = 1.$$

Proof. (b) \Rightarrow (a): This follows the same argument in the proof of Corollary 4.4 and the observation that $g_{v_0} \neq 0$.

(a) \Rightarrow (b): Let u be a vertex that has the smallest nonzero absolute gap among all vertices in V , that is $|g_u| = \min\{|g_v| : g_v \neq 0\}$. From u , we take a maximal directed path such that the h -value along the path is nondecreasing. As A is terminating and $h_u > 0$, the end of the path must be n . Denote the path by $(u = v_0, v_1, \dots, v_p = n)$. Similarly, there exists a directed path $(u = v_0, v_{-1}, \dots, v_{-q} = -n)$, such that the h -value is nonincreasing along the path. Since v_1 and v_{-1} are the two out-neighbors of u with distinct h -values, all vertices in the two paths, except for u , are distinct.

We have $h_n = 1$ and $h_{-n} = 0$. Therefore,

$$1 = h_n - h_{-n} = \sum_{i=0}^{p-1} g_{v_i} + \sum_{i=1-q}^0 g_{v_i} = \sum_{i=1-q}^{p-1} g_{v_i} + g_{v_0}. \quad (4.9)$$

If there is a vertex w with nonzero gap other than the vertices in the two paths above, then we have

$$\sum_{v \in V} g_v \geq \sum_{i=1-q}^{p-1} g_{v_i} + g_w \geq \sum_{i=1-q}^{p-1} g_{v_i} + g_{v_0} = 1. \quad (4.10)$$

Therefore, this cannot happen if $\sum_{v \in V} g_v < 1$.

Next, we argue that there is no vertex with zero gap. We first observe that for any vertex whose h -value is 0, all directed paths from that vertex have the same h -value and end with $-n$. Therefore, we can contract these vertices to $-n$, which contradicts the assumption that A is reduced. Hence, there is no vertex in V with h -value 0, and analogously, no vertex in V with h -value 1. Now for the sake of another contradiction, suppose there exists a vertex a , such that $0 < h_a < 1$ and $g_a = 0$. Consider a directed path from a to a destination. Let b be the first vertex on this path with nonzero gap, i.e., all vertices between a and b on the path have the same h -value. As argued before, there is no other vertex with nonzero gap whose h -value is h_b (and h_a). Therefore, all the directed paths from a to a destination has to contain b , and so do all the vertices with zero gap along any path between a and b . Hence, we can contract all these vertices to b , contradicting the assumption that A is reduced.

The lemma then follows. \square

When we have $\sum_{v \in V} |g_v| < 1$, we can use the same analysis as in Section 4.4 and obtain $\beta - \alpha < 1$, for α, β defined as in (4.7). It then follows that we can decide G-ARRIVAL immediately, as there can be only one integer value in the range $[\alpha, \beta]$. Note that the characterization in Lemma 4.11 is not combinatorial, and it is interesting to understand what the condition (b) in the lemma means in graph theoretic terms.

Next, we can extend the characterization to the case $\sum_{v \in V} |g_v| = 1$.

Lemma 4.12. *Let A, h, g be defined as in Lemma 4.11. If $\sum_{v \in V} |g_v| = 1$, then there are two directed paths $(v_0, v_1, \dots, v_p = n)$ and $(v_0, v_{-1}, \dots, v_{-q} = -n)$, such that*

$$0 = h_{v_{-q}} < \dots < h_{v_{-1}} < h_{v_0} < h_{v_1} < \dots < h_{v_p} = 1.$$

Other than the vertices in the two paths above, there is a vertex w with nonzero gap, and for such w , $g_w = g_{v_0}$. Further, if $h_w = h_{v_i}$ for some $i \in]-q, p[$, then for any vertex with zero gap, any directed path from that vertex to a destination contains either w or v_i .

Proof. We follow the construction of the two paths as in the proof of Lemma 4.11. Further, suppose w is a vertex that is not on any path and has nonzero gap, from the two expressions (4.9) and (4.10), we obtain $g_w = g_{v_0}$ and there are no other vertices with nonzero gap.

Moreover, for a vertex a with zero gap, it is easy to see that it has to have the same h -value as a vertex with nonzero gap. If there is exactly one vertex b with that value and nonzero gap, then by the same argument in the proof of Lemma 4.11, we can contract a and some other vertices to b , a contradiction to the assumption that A is reduced. Otherwise, by construction, no vertices with nonzero gap on the two paths can have the same h -value. Hence, only w can share the same h -value with a vertex v_i on the paths. Since a shares the same h -value with more than one vertex with nonzero gap, we then have $h_a = h_w = h_{v_i}$. Therefore, on any directed path from a to a destination, the first vertex with nonzero gap can only be either w or v_i . The lemma then follows. \square

Observe that if there are no vertices with zero gaps, then we can use the arguments in the proofs of Lemmata 4.8 and 4.9 and compute the switching flow for the switching system. Otherwise, Lemma 4.12 states that there are two vertices w and v_i with the same h -value. Then we can again use Lemma 4.9 to fix the values of the switching flow for the vertices going out of vertices with nonzero gap. After that, following the analysis for Algorithm L, we can remove all these vertices from the switch graph, except for w and h_i , which serve as new destinations. In order to recurse, we require the new switch graph to exhibit the same structure (i.e., the sum of the absolute gaps is at most 1).

Overall, the above discussion outlines certain sufficient characteristics of graphs for which Algorithm L works. However, as mentioned earlier, it would be interesting to have a combinatorial description of these graphs.

Part II

Combinatorial generation via permutations

*When we hit our lowest point, we are
open to the greatest change.*

—Avatar Aang,
from *The Legend of Korra*

CHAPTER 5

A framework for combinatorial generation

This chapter is based on [83], which is joint work with Elizabeth Hartung, Torsten Mütze, and Aaron Williams.

In this chapter, we present a simple greedy algorithm, Algorithm J, for exhaustively generating a given set $L_n \subseteq S_n$, where S_n denotes the set of all permutations of $[n]$, and we show that the algorithm works successfully under very mild assumptions on the set L_n (Theorem 5.1). Further, in Section 5.5, we illustrate how the algorithm recovers four classical Gray codes to generate all permutations, binary strings, binary trees, and set partitions, as mentioned in Chapter 1.

Before going to our framework, we briefly mention some other existing frameworks on exhaustive generation of combinatorial objects.

Related work Avis and Fukuda [13] introduced *reverse-search* as a general technique for exhaustive generation. Their idea is to consider the set of objects to be generated as the nodes of a graph, and to connect them by edges that model local modification operations (for instance, adjacent transpositions for permutations). The resulting flip graph is equipped with an objective function, and the directed tree formed by the movements of a local search algorithm that optimizes this function is traversed backwards from the optimum node, using an adjacency oracle. The authors applied this technique successfully to derive efficient generation algorithms for a number of different objects; for instance, triangulations of a point set, spanning trees of a graph, etc. Reverse-search is complementary to our permutation based approach, as both techniques use fundamentally different encodings of the objects. The permutation encoding seems to allow for more fine-grained control (optimal Gray codes) and even faster generation algorithms.

Another method for combinatorial counting and exhaustive generation is the *ECO framework* introduced by Barcucci, Del Lungo, Pergola, and Pinzani [18]. The main tool is an infinite tree with integer node labels, and a set of production rules for creating the children of a node based on its label. Bacchelli, Barcucci, Grazzini, and Pergola [17] also used ECO for exhaustive generation, deriving an efficient algorithm for generating the corresponding root-to-node label sequences in the ECO tree in lexicographic order, which was later turned into a Gray code [24]. Dukes, Flanagan, Mansour, and Vajnovszki [64], Baril [20], and Do, Tran and Vajnovszki [61] used ECO for deriving Gray codes for different classes of pattern-avoiding permutations, which works under certain regularity assumptions on the production rules. Vajnovszki [164] also applied ECO for efficiently generating other classes of permutations, such as involutions and derangements. The main difference between ECO and our framework is that the change operations on the label sequences of the ECO tree do not necessarily correspond to Gray-code like changes on the

corresponding combinatorial objects. Minimal jumps in a permutation, on the other hand, always correspond to minimal changes on the combinatorial objects in a provable sense, even though they may involve several entries of the permutation.

Li and Sawada [116] considered another tree-based approach for generating so-called *reflectable languages*, yielding Gray codes for k -ary strings and trees, restricted growth strings, and open meandric systems (see also [173]). Ruskey, Sawada, and Williams [150, 152] proposed a generation framework based on binary strings with a fixed numbers of 1s, called *bubble languages*, which can generate e.g. combinations, necklaces, Dyck words, and Lyndon words. In the resulting cool-lex Gray codes, any two consecutive words differ by cyclic rotation of some prefix.

5.1 Preliminaries

We write a permutation $\pi \in S_n$ in one-line notation as $\pi = \pi(1)\pi(2) \cdots \pi(n) = a_1a_2 \cdots a_n$. We use $\text{id}_n = 12 \cdots n$ to denote the identity permutation. 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 $c_i(\pi) = a_1 \cdots a_{i-1} n a_i \cdots a_{n-1}$. Moreover, for $\pi \in S_n$, we write $p(\pi) \in S_{n-1}$ for the permutation obtained from π by removing the largest entry n . Here, c_i and p stand for the child and parent of a node in the tree of permutations discussed shortly.

Given a permutation $\pi = a_1 \cdots a_n$ with a substring $a_i \cdots a_j$ with $a_i > a_{i+1}, \dots, a_j$, a *right jump of the value a_i by $j - i$ steps* is a cyclic left rotation of this substring by one position to $a_{i+1} \cdots a_j a_i$. Similarly, given a substring $a_i \cdots a_j$ with $a_j > a_i, \dots, a_{j-1}$, a *left jump of the value a_j by $j - i$ steps* is a cyclic right rotation of this substring to $a_j a_i \cdots a_{j-1}$. For example, a right jump of the value 5 in the permutation 265134 by 2 steps yields 261354.

5.2 The basic algorithm

Our approach starts with the following simple greedy algorithm to generate a set of permutations $L_n \subseteq S_n$. We say that a jump is *minimal* (w.r.t. L_n), if every jump of the same value in the same direction by fewer steps creates a permutation that is not in L_n . Note that each entry of the permutation admits at most one minimal left jump and at most one minimal right jump.

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

J1. [Initialize] Visit the initial permutation π_0 .

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

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

ates $\pi_1 = 2134$ and then stops, as no further jump is possible. If we choose $\pi_0 = 1423$, then we may jump 4 to the left or right (by 1 step), but as the direction is ambiguous, the algorithm stops immediately. As mentioned before, the algorithm may stop before having visited the entire set L_n either because no minimal jump leading to a new permutation from L_n is possible, or because the direction of jump is ambiguous in some step. By the definition of step J2, the algorithm will never visit any permutation twice.

5.3 Zigzag languages

The following main result of this chapter provides a sufficient condition on the set L_n to guarantee that Algorithm J is successful (cf. Section 6.4). This condition is captured by the following closure property of the set L_n . A set of permutations $L_n \subseteq S_n$ is called a *zigzag language*, if either $n = 0$ and $L_0 = \{\varepsilon\}$, or if $n \geq 1$ and $L_{n-1} := \{p(\pi) \mid \pi \in L_n\}$ is a zigzag language satisfying the following condition:

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

Theorem 5.1. *Given any zigzag language of permutations L_n and initial permutation $\pi_0 = \text{id}_n$, Algorithm J visits every permutation from L_n exactly once.*

Remark 5.2. *It is easy to see that the number of zigzag languages is at least $2^{(n-1)!(n-2)} = 2^{2^{\Theta(n \log n)}}$, i.e., it is more than double-exponential in n . We will see that many of these languages do in fact encode interesting combinatorial objects. Moreover, minimal jumps as performed by Algorithm J always translate to small changes on those objects in a provable sense, i.e., our algorithm defines Gray codes for a large variety of combinatorial objects, and Hamilton paths/cycles on the corresponding flip graphs and polytopes.*

Before we present the proof of Theorem 5.1, we give two equivalent characterizations of zigzag languages.

5.3.1 Characterization via the tree of permutations

There is an intuitive characterization of zigzag languages via the *tree of permutations*. This is an infinite (unordered) rooted tree which has as nodes all permutations from S_n at distance n from the root; see Figure 5.1. Specifically, the empty permutation ε is at the root, and the children of any node $\pi \in S_{n-1}$ are exactly the permutations $c_i(\pi)$, $1 \leq i \leq n$, i.e., the permutations obtained by inserting the new largest value n in all possible positions. Consequently, the parent of any node $\pi' \in S_n$ is exactly the permutation $p(\pi')$ obtained by removing the largest value n . In the figure, for any node $\pi \in S_{n-1}$, the nodes representing the children $c_1(\pi)$ and $c_n(\pi)$ are drawn black, whereas the other children are drawn white. Any zigzag language of permutations can be obtained from this full tree by pruning subtrees, where by condition (z) a subtree may be pruned only if its root $\pi' \in S_n$ is neither the child $c_1(\pi)$ nor the child $c_n(\pi)$ of its parent $\pi = p(\pi') \in S_{n-1}$, i.e., only subtrees rooted at white nodes may be pruned. For any subtree obtained by pruning according to this rule and for any $n \geq 1$, the remaining permutations of size n form a zigzag language L_n ; see Figure 5.2.

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

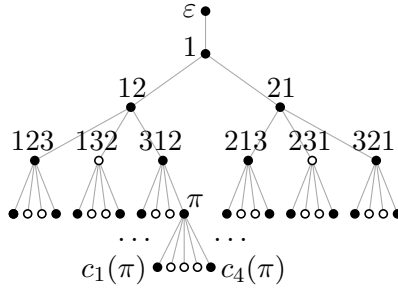


Figure 5.1: Tree of permutations, where the children $c_1(\pi)$ and $c_n(\pi)$ of any node $\pi \in S_{n-1}$ are drawn black, all others white.

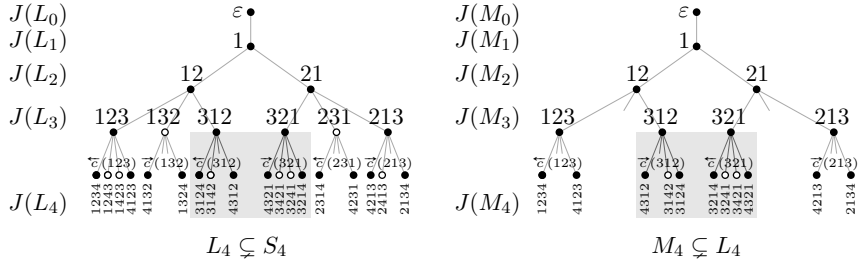


Figure 5.2: Ordered tree representation of two zigzag languages of permutations L_4 (left) and M_4 (right) with $M_4 \subsetneq L_4 \subsetneq S_4$. Both trees contain the same sets of permutations in the subtrees rooted at 312 and 321 (highlighted in gray), but in the corresponding sequences $J(L_4)$ and $J(M_4)$, those permutations appear in different relative order due to the node 132, which was pruned from the right tree.

5.3.2 Characterization via nuts

Given a permutation π , we may repeatedly remove the largest value from it as long as it is in the leftmost or rightmost position, and obtain what is called the *nut* of π . For example, given $\pi = 965214378$, we can remove 9, 8, 7, 6, 5, yielding 2143 as the nut of π . A left or right jump of some value in a permutation is *maximum* if there is no left jump or right jump of the same value with more steps. For example, in $\pi = 965214378$ a maximum right jump of the value 6 gives $\pi' = 952143678$. By unrolling the recursive definition of zigzag languages from before, we obtain that $L_n \subseteq S_n$ is a zigzag language if and only if for all $\pi \in L_n$ both the maximum left jump and the maximum right jump of the value i yield another permutation in L_n for all $k \leq i \leq n$, where k is the largest value in π 's nut (with $k = 2$ if the nut is empty).

5.3.3 Proof of Theorem 5.1

Given a zigzag language L_n , we define a sequence $J(L_n)$ of all permutations from L_n , and we prove that Algorithm J generates the permutations of L_n exactly in this order. For any $\pi \in L_{n-1}$ we let $\vec{c}(\pi)$ be the sequence of all $c_i(\pi) \in L_n$ for $i = 1, 2, \dots, n$, 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 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 . The sequence $J(L_n)$ is defined recursively as follows: If $n = 0$ then we define $J(L_0) := \varepsilon$, and if $n \geq 1$ then we consider the finite sequence $J(L_{n-1}) =: \pi_1, \pi_2, \dots$ and define

$$J(L_n) := \overleftarrow{c}(\pi_1), \vec{c}(\pi_2), \overleftarrow{c}(\pi_3), \vec{c}(\pi_4), \dots, \quad (5.1)$$

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; see Figure 5.2.

Remark 5.3. *Algorithm J thus defines a left-to-right ordering of the nodes at distance n of the root in the tree representation of the zigzag language L_n described before, and this ordering is captured by the sequence $J(L_n)$; see Figure 5.2. Clearly, the same is true for all the zigzag languages L_0, L_1, \dots, L_{n-1} that are induced by L_n through the rule $L_{k-1} := \{p(\pi) \mid \pi \in L_k\}$ for $k = n, n-1, \dots, 1$. The unordered tree is thus turned into an ordered tree, and it is important to realize that pruning operations change the ordering. Specifically, given two zigzag languages L_n and M_n with $M_n \subseteq L_n$, the tree for M_n is obtained from the tree for L_n by pruning, but in general $J(M_n)$ is not a subsequence of $J(L_n)$, as shown by the example in Figure 5.2. This shows that our approach is quite different from the one presented by Vajnovszki and Vernay [165], which considers only subsequences of the Steinhaus-Johnson-Trotter order $J(S_n)$.*

Proof of Theorem 5.1. For any $\pi \in L_n$, we let $J(L_n)_\pi$ denote the subsequence of $J(L_n)$ that contains all permutations up to and including π . An immediate consequence of the definition of zigzag language is that L_n contains the identity permutation $\text{id}_n = c_n(\text{id}_{n-1})$. Moreover, the definition (5.1) implies that id_n is the very first permutation in the sequence $J(L_n)$.

We now argue by double induction over n and the length of $J(L_n)$ that Algorithm J generates all permutations from L_n exactly in the order described by the sequence $J(L_n)$, and that when we perform a minimal jump with the largest possible value to create a previously unvisited permutation, then there is only one direction (left or right) to which it can jump. The induction basis $n = 0$ is clear. Now suppose the claim holds for the zigzag language $L_{n-1} := \{p(\pi) \mid \pi \in L_n\}$. We proceed to show that it also holds for L_n .

As argued before, the identity permutation id_n is the first permutation in the sequence $J(L_n)$, and this is indeed the first permutation visited by Algorithm J in step J1. Now let $\pi \in L_n$ be the permutation currently visited by the algorithm in step J2, and let $\pi' := p(\pi) \in L_{n-1}$. If π' appears at an odd position in $J(L_{n-1})$, then we define $\bar{c} := \overleftarrow{c}(\pi')$ and otherwise we define $\bar{c} := \overrightarrow{c}(\pi')$. By (5.1), we know that π appears in the subsequence \bar{c} within $J(L_n)$. We first consider the case that π is not the last permutation in \bar{c} . In this case, the permutation ρ succeeding π in $J(L_n)$ is obtained from π by a minimal jump (w.r.t. L_n) of the largest value n in some direction d , which is left if $\bar{c} = \overleftarrow{c}(\pi')$ and right if $\bar{c} = \overrightarrow{c}(\pi')$. Now observe that by the definition of \bar{c} , all permutations in L_n obtained from π by jumping n in the direction opposite to d precede π in $J(L_n)$ and have been visited by Algorithm J before by induction. Consequently, to generate a previously unvisited permutation, the value n can only jump in direction d in step J2 of the algorithm. Again by the definition of \bar{c} , the permutation ρ is obtained from π by a minimal jump (w.r.t. L_n), so the next permutation generated by the algorithm will indeed be ρ .

It remains to consider the case that π is the last permutation in the subsequence \bar{c} within $J(L_n)$. Let ρ' be the permutation succeeding π' in $J(L_{n-1})$. By induction, we have the following property (*): ρ' is obtained from π' by a minimal jump (w.r.t. L_{n-1}) of the largest possible value a by k steps in some direction d (left or right), and a can jump only into one direction. As π is the last permutation in \bar{c} , the largest value n of π is at the boundary, which is the left boundary if $\bar{c} = \overleftarrow{c}(\pi')$ or the right boundary if $\bar{c} = \overrightarrow{c}(\pi')$. By (5.1), the permutation ρ succeeding π in $J(L_n)$ also has n at the same boundary, i.e., ρ differs from π by a jump of the value a by k steps in direction d . Suppose for the sake of contradiction that when transforming the currently visited permutation π in step J2, the algorithm does not perform this jump operation, but another one. This could be a jump of a larger value $b > a$ to transform π into

some permutation $\tau \in L_n$ that is different from ρ and not in $J(L_n)_\pi$, or a jump of the value a in the direction opposite to d , or a jump of the value a in direction d by fewer than k steps. But in all those cases the permutation $\tau' := p(\tau) \in L_{n-1}$ is different from ρ' and not in $J(L_{n-1})_{\pi'}$, and it is obtained from π' by a jump of the value $b > a$, or a jump of the value a in the direction opposite to d , or a jump of the value a in direction d by fewer than k steps, respectively, a contradiction to property (*). This completes the proof. \square

5.4 Further properties of Algorithm J

The next lemma captures when Algorithm J generates a *cyclic* listing of permutations.

Lemma 5.4. *In the ordering of permutations $J(L_n)$ generated by Algorithm J, the first and last permutation are related by a minimal jump if and only if $|L_i|$ is even for all $2 \leq i \leq n-1$.*

For example, the conditions described by Lemma 5.4 are satisfied for the zigzag languages $L_n = S_n$ (all permutations) and $L_n = P_n$ (permutations without peaks), and the resulting cyclic orderings $J(L_n)$ are shown in Figures 5.3 and 5.4, respectively. Another cyclic Gray code is shown in Figure 6.2. In contrast to that, the Gray codes shown in Figures 5.5 and 5.6 violate the conditions of the lemma and are therefore not cyclic.

Proof. Let π_i be the last permutation in the ordering $J(L_i)$ for all $i = 0, 1, \dots, n$. For $i \geq 1$, we see from (5.1) that $\pi_i = c_i(\pi_{i-1})$ if $|L_{i-1}|$ is even and $\pi_i = c_1(\pi_{i-1})$ if $|L_{i-1}|$ is odd. As $|L_1| = 1$ is odd, we know that 1 and 2 are reversed in π_n , and so all numbers $|L_i|$, $2 \leq i \leq n-1$, must be even for id_n and π_n to be related by a minimal jump. \square

Remark 5.5. *It follows from the proof of Theorem 5.1 that instead of initializing the algorithm with the identity permutation $\pi_0 = \text{id}_n$, we may use any permutation without peaks as a seed π_0 .*

Let us make it very clear that in its stated form, Algorithm J is not an efficient algorithm to actually generate a particular zigzag language of permutations. The reason is that it requires storing the list of all previously visited permutations in order to decide which one to generate next. However, by introducing a few additional arrays, the algorithm can be made memoryless, so that such lookup operations are not needed anymore, and hence no permutations need to be stored at all. The efficiency of the resulting algorithm is then only determined by the efficiency with which we are able to compute minimal jumps with respect to the input zigzag language L_n for a given entry of the permutation. This leads to an algorithm that computes the next permutation to be visited in polynomial time. In many cases, this can be improved to a loopless algorithm that generates each new permutation in constant worst-case time. The key insight here is that any jump changes the inversion table of a permutation only in a single entry. By maintaining only the inversion table, jumps can thus be performed efficiently, even if the number of steps is big.

5.5 A general recipe with classical examples

Here is a step-by-step approach to apply our framework to the generation of a given family X_n of combinatorial objects. The first step is to establish a bijection f that encodes the objects from X_n as permutations $L_n \subseteq S_n$. If L_n is a zigzag language, which can be checked by verifying the closure property, then we may run Algorithm J with input L_n , and interpret the resulting ordering $J(L_n)$ in terms of the combinatorial objects, by applying f^{-1} to each permutation in $J(L_n)$, yielding an ordering on X_n . We may also ap-

ply f^{-1} to Algorithm J directly, which will yield a simple greedy algorithm for generating X_n . The final step is to make these algorithms efficient, by introducing additional data structures that allow the change operations on X_n (which are the preimages of minimal jumps under f) as efficiently as possible.

We now amply illustrate this approach by four examples of classical Gray codes.

5.5.1 Permutations (Steinhaus-Johnson-Trotter)

Consider the set $X_n = L_n = S_n$ of all permutations of $[n]$. The bijection f between X_n and L_n here is simply the identity, i.e., $f = \text{id}$. In this case, each jump is a jump by 1 step, i.e., it is an adjacent transposition. Algorithm J thus yields an ordering of permutations by adjacent transpositions, which coincides with the well-known Steinhaus-Johnson-Trotter order, also known as plain change order [162, 99], which can be implemented efficiently [108]. This ordering is shown in Figure 5.3. Algorithm J translates into the following simple greedy algorithm to describe this order (see [171]):

J1. Visit the identity permutation. **J2.** Perform a transposition of the largest possible value with an adjacent smaller entry that yields a previously unvisited permutation; then visit this permutation and repeat J2.

5.5.2 Binary strings (BRGC)

Consider the set X_n of binary strings of length $n - 1$. We map any binary string $x = x_2 \cdots x_n$ to a permutation $f(x) \in S_n$ by setting $f(\varepsilon) := 1$ and

$$f(x_2 \cdots x_n) := \begin{cases} c_n(f(x_2 \cdots x_{n-1})) & \text{if } x_n = 0, \\ c_1(f(x_2 \cdots x_{n-1})) & \text{if } x_n = 1, \end{cases}$$

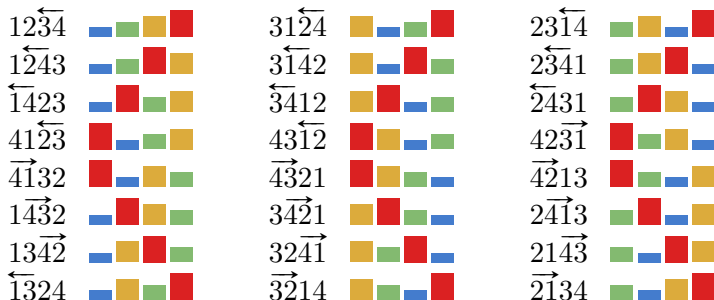


Figure 5.3: All permutations of size $n = 4$ generated by Algorithm J, coinciding with the Steinhaus-Johnson-Trotter ordering. Read the figure column by column, from left to right. Each arrows indicates an adjacent transposition (jump by 1 step) that creates the next permutation. The ordering is cyclic, so the last transposition creates the first permutation.

		x_2	x_3	x_4
$\overleftarrow{1234}$		0	0	0
$\overleftarrow{4123}$		0	0	1
$\overrightarrow{4312}$		0	1	1
$\overleftarrow{3124}$		0	1	0
$\overleftarrow{3214}$		1	1	0
$\overrightarrow{4321}$		1	1	1
$\overrightarrow{4213}$		1	0	1
$\overrightarrow{2134}$		1	0	0

Figure 5.4: Permutations without peaks of size $n = 4$ generated by Algorithm J, and the resulting Gray code for binary strings, coinciding with the BRGC. In this and subsequent figures, an arrow above a permutation indicates a jump, where the value below the tail of the arrow is the value that jumps, and the head of the arrow shows the position of the value after the jump.

i.e., we build the permutation $f(x)$ by inserting the values $i = 2, \dots, n$ one by one, either at the leftmost or rightmost position, depending on the bit x_i ; see Figure 5.4 for examples. Observe that $f(X_n)$ is exactly the set of permutations without peaks $P_n \subseteq S_n$ discussed previously in Section 5.3.1, and a jump of the value i in the permutation translates to flipping the bit x_i . Moreover, $f^{-1}(J(P_n))$ is exactly the well-known reflected Gray code (BRGC) for binary strings of length $n-1$ [81], which can be implemented efficiently [30]. This ordering is shown in Figure 5.4. Applying f^{-1} to Algorithm J yields the following simple greedy algorithm to describe the BRGC (see [171]): **J1.** Visit the all-zero string. **J2.** Flip the rightmost bit that yields a previously unvisited string; then visit this string and repeat J2.

5.5.3 Binary trees (Lucas-Roelants van Baronaigien-Ruskey)

Consider the set X_n of binary trees with n nodes, labelled with n distinct integers from $[n]$, that have the search tree property, i.e., for every node, all nodes in the left subtree are smaller than all nodes in the right subtree. We recursively map any such tree x with root node i , left subtree x_L , and right subtree x_R to a permutation $f(x) \in S_n$ by setting $f(x) := (i, f(x_L), f(x_R))$ and $f(\emptyset) := \varepsilon$ if $x = \emptyset$ has no nodes; see Figure 5.5 for examples. By the search tree property, $f(X_n)$ is exactly the set of permutations that avoid the pattern 231 (i.e., $f(X_n) = S_n(231)$), which we will prove to be a zigzag language. A jump of the value i in the permutation translates to a tree rotation involving the node i . Specifically, a right jump of the value i in the permutation corresponds to a right rotation at node i , and a left jump corresponds to a left rotation at the parent of node i , where node i is the right child of this parent, and these two operations are inverse to each other. Moreover, $f^{-1}(J(S_n(231)))$ is exactly the ordering of binary trees described by Lucas, Roelants van Baronaigien, and Ruskey [121], which they showed can be im-

plemented efficiently. This ordering is shown in Figure 5.5. Applying f^{-1} to Algorithm J yields the following simple greedy algorithm to describe this order (see [171]): **J1.** Visit the all-right tree, i.e., every node has exactly one child, namely a right child. **J2.** Perform a rotation involving the largest possible node that yields a previously unvisited tree; then visit this tree and repeat J2.

Via standard bijections, binary trees are equivalent to many other Catalan families (such as triangulations, Dyck paths, etc.), so we also obtain Gray codes for all these other objects; see Figure 5.5.

5.5.4 Set partitions (Kaye)

Consider the set X_n of partitions of the set $[n]$ into nonempty subsets. We can represent any such partition $x = \{x_1, \dots, x_k\}$, $x_i \subseteq [n]$, in a canonic way, by sorting all subsets in decreasing order of their minimum element, and the elements of each subset in increasing order. Then x is mapped to a permutation $f(x) \in S_n$ by writing out the elements from left to right in this canonic representation of x . For instance, the set partition $x = \{\{9\}, \{6\}, \{3, 4, 7\}, \{1, 2, 5, 8\}\}$ is encoded as the permutation $f(x) = 963471258$. Observe that $f(X_n)$ is the set of permutations with the property that for every descent $a_i a_{i+1}$, $a_i > a_{i+1}$, in the permutation, no value left of a_i is smaller than a_{i+1} . This notion of pattern-avoidance can be described concisely by the vincular permutation pattern $\underline{132}$, i.e., we have $f(X_n) = S_n(\underline{132})$ (the formal definition of vincular patterns is given in the next section), which we will prove to be a zigzag language. A jump of the value i in the permutation corresponds to moving the element i from its subset to the previous or next subset in the canonic representation, possibly creating a singleton set $\{i\}$. Moreover, $f^{-1}(J(S_n(\underline{132})))$ is exactly the ordering of set partitions described by Kaye [104], which he showed can be implemented efficiently. This ordering is shown in Figure 5.6. Applying f^{-1} to Algorithm J yields the following simple greedy algorithm to describe

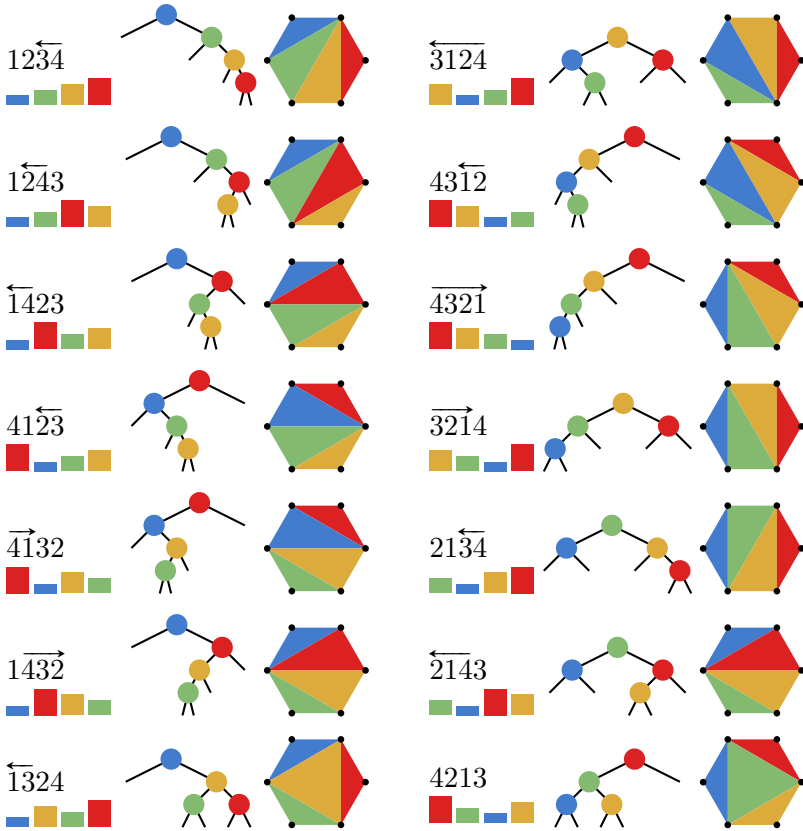


Figure 5.5: 231-avoiding permutations of size $n = 4$ generated by Algorithm J and the resulting Gray codes for Catalan families (binary trees, triangulations, Dyck paths), coinciding with the ordering described by Lucas, Roelants van Baronaigien, and Ruskey.

this order (see [171]): **J1.** Visit the set partition $\{\{1, \dots, n\}\}$. **J2.** Move the largest possible element from its subset to the previous or next subset so as to obtain a previously unvisited partition; then visit this partition and repeat J2.

$\overleftarrow{1}234$		1234	$\overrightarrow{4}3\overrightarrow{2}1$		4 3 2 1
$\overleftarrow{4}1\overrightarrow{2}3$		4 123	$\overrightarrow{4}2\overrightarrow{3}1$		4 23 1
$\overrightarrow{4}31\overrightarrow{2}$		4 3 12	$\overrightarrow{2}34\overrightarrow{1}$		234 1
$\overrightarrow{3}41\overrightarrow{2}$		34 12	$\overrightarrow{2}3\overrightarrow{1}4$		23 14
$\overleftarrow{3}1\overrightarrow{2}4$		3 124	$\overleftarrow{2}1\overrightarrow{3}4$		2 134
$\overleftarrow{3}21\overrightarrow{4}$		3 2 14	$\overleftarrow{2}41\overrightarrow{3}$		24 13
$\overrightarrow{3}241$		3 241	4213		4 2 13
$\overleftarrow{3}421$		34 2 1			

Figure 5.6: 132 -avoiding permutations of size $n = 4$ generated by Algorithm J and resulting Gray code for set partitions, coinciding with Kaye's Gray code. Set partitions are denoted compactly, omitting curly brackets and commas in the canonic representation, and using vertical bars to separate subsets.

5.6 Discussion

Theorem 5.1 asserts that if L_n is a zigzag language, then Algorithm J successfully visits every permutation from L_n . This condition is not necessary, however. For instance, the set $L_4 \subseteq S_4$ discussed in Section 5.2 is not a zigzag language, and still Algorithm J is successful when initialized suitably. From the proof of Theorem 5.1 we immediately see that condition (z) in the definition of zigzag language could be weakened as follows, and the same proof would still go through:

- (z') Given the sequence $J(L_{n-1})$, then for every permutation π in this sequence there is a direction $\chi(\pi) \in \{\leftarrow, \rightarrow\}$ satisfying the following: For every π from $J(L_{n-1})$ we define $I(\pi) := \{i \in [n] \mid c_i(\pi) \in L_n\}$, and we also define $\tilde{i}(\pi) := \max I(\pi)$ and $\hat{i}(\pi) := \min I(\pi)$ if $\chi(\pi) = \leftarrow$, and $\tilde{i}(\pi) := \min I(\pi)$ and $\hat{i}(\pi) := \max I(\pi)$ if $\chi(\pi) = \rightarrow$. Then for any two permutations π, ρ that appear consecutively in $J(L_{n-1})$ and that differ in a

jump from position i_1 to position i_2 , we have $\hat{i}(\pi) = \check{i}(\rho)$ and this number is not in the interval $] \min\{i_1, i_2\}, \max\{i_1, i_2\}]$.

Note that condition (z) implies (z'), as for a zigzag language we have $1, n \in I(\pi)$ for all $\pi \in L_{n-1}$, so $\min I(\pi) = 1$ and $\max I(\pi) = n$, and the sequence χ alternates between \leftarrow and \rightarrow in every step. This shows that Algorithm J succeeds to generate a much larger class of languages $L_n \subseteq S_n$. However, condition (z') is considerably more complicated, in particular as it depends on the ordering $J(L_{n-1})$ generated for L_{n-1} . More importantly, in the context of permutation patterns, which we will discuss shortly in the next chapter, we do not see any interesting languages L_n that would satisfy condition (z') but not condition (z), which is why we think condition (z) is the "correct" one.

CHAPTER 6

Pattern-avoiding permutations

This chapter is based on [83], which is joint work with Elizabeth Hartung, Torsten Mütze, and Aaron Williams.

The first main application of our framework is the generation of pattern-avoiding permutations. Pattern avoidance in permutations is a central topic in combinatorics, as illustrated by the books [106, 32], and by the conference “Permutation Patterns”, held annually since 2003. It is well known that many fundamental classes of combinatorial objects are in bijection with pattern-avoiding permutations (see Tables 6.1 and 6.5 and [159]). For instance, Knuth [107] first proved that all 123-avoiding and 132-avoiding permutations are counted by the Catalan numbers (see also [50]). With regards to counting and exhaustive generation, a few tree-based algorithms for pattern-avoiding permutations have been proposed [68, 64, 19, 20].

Our main results in this chapter are summarized in Theorem 6.3, Theorem 6.10 (and its corollaries Lemmata 6.4–6.9), and in Table 6.1. We emphasize that all our results can be generalized to bounding the number of appearances of patterns, where the special case with a bound of 0 appearances is pattern-avoidance; see Section 6.2 below. From these results, in Section 6.3, we further extend the classes of pattern-avoiding permutations that can be generated by our framework, by applying transformations and combinations to the patterns discussed in Section 6.1.

6.1 Classical patterns and other variants

6.1.1 Preliminaries

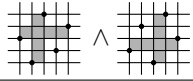

The following simple but powerful lemma follows immediately from the definition of zigzag languages given in Section 5.3. For any set $L_n \subseteq S_n$ we define $p(L_n) := \{p(\pi) \mid \pi \in L_n\}$.

Lemma 6.1. *Let $L_n, M_n \subseteq S_n$, $n \geq 1$, be two zigzag languages of permutations. Then $L_n \cup M_n$ and $L_n \cap M_n$ are also zigzag languages of permutations, and we have $p(L_n \cup M_n) = p(L_n) \cup p(M_n)$ and $p(L_n \cap M_n) = p(L_n) \cap p(M_n)$.*

We say that two sequences of integers σ and τ are *order-isomorphic*, if their elements appear in the same relative order in both sequences. For instance, 2576 and 1243 are order-isomorphic. Given two permutations $\pi \in S_n$ and $\tau \in S_k$, we say that π *contains the pattern* τ , if and only if $\pi = a_1 \dots a_n$ contains a subpermutation $a_{i_1} \dots a_{i_k}$, $i_1 < \dots < i_k$, that is order-isomorphic to τ . We refer to such a subpermutation as a *match* of τ in π . If π does not contain the pattern τ , then we say that π *avoids* τ . For example, $\pi = 635412$ contains the pattern $\tau = 231$, as the highlighted entries form a match of τ in π . On the other hand, $\pi = 654123$ avoids $\tau = 231$. We let $S_n(\tau)$ denote all permutations from S_n avoiding a given pattern τ .

Table 6.1: Tame permutation patterns and corresponding combinatorial objects and orderings generated by Algorithm J. Some patterns are interesting in their own right and have no ‘natural’ associated combinatorial objects, in which case the first two columns are merged. The patterns with underscores, bars, boxes, and those drawn as grids with some shaded cells, are defined and explained in Sections 6.1.3–6.1.8. See Table 6.5 for more patterns.

Tame patterns	Combinatorial objects and ordering	References/ OEIS [129]
none	permutations by adjacent transpositions \rightarrow plain change order	[99, 162], A000142
$231 = \underline{231}$	<i>Catalan families</i> • binary trees by rotations \rightarrow Lucas-Roelants van Baronaigien-Ruskey order • triangulations by edge flips • Dyck paths by hill flips	A000108 [121]
$\underline{132}$	<i>Bell families</i> • set partitions by element exchanges \rightarrow Kaye’s order	A000110 [104, 171]
$132 \wedge 231 = \underline{132} \wedge \underline{231}$: permutations without peaks	binary strings by bitflips \rightarrow reflected Gray code order (BRGC)	[81], A011782
1342	forests of $\beta(0, 1)$ -trees	[31, 11], A022558
2143: vexillary permutations		[112], A005802
conjunction of v_k tame patterns with $v_2 = 35, v_3 = 91, v_4 = 2346$ (see [27]): k -vexillary permutations ($k \geq 1$)		[28], A224318, A223034, A223905
$2143 \wedge 3412$: skew-merged permutations		[155, 9], A029759
$2143 \wedge 2413 \wedge 3142$		[58, 154], A033321
$2143 \wedge 2413 \wedge 3142 \wedge 3412$: X-shaped permutations		[169, 69], A006012

Tame patterns	Combinatorial objects and ordering	References/ OEIS [129]
2413 \wedge 3142: separable permutations	<i>Schröder families</i> • slicing floorplans (=guillotine partitions) • topological drawings of $K_{2,n}$	A006318 [15, 35, 2] [41]
2413 \wedge 3142: Baxter 2413 \wedge 3412: twisted Baxter 2143 \wedge 3142	mosaic floorplans (=diagonal rectangulations=R-equivalent rectangulations)	[174, 2, 113, 43], A001181
2143 \wedge 3412	S-equivalent rectangulations	[8], A214358
2143 \wedge 3412 \wedge 2413 \wedge 3142	S-equivalent guillotine rectangulations	[8], A078482
35124 \wedge 35142 \wedge 24513 \wedge 42513: 2-clumped permutations	generic rectangulations (=rectangular drawings)	[144]
conjunction of c_k tame patterns with $c_k = 2(k/2)!(k/2 + 1)!$ for k even and $c_k = 2((k+1)/2)!^2$ for k odd: k -clumped permutations		[144]
12543 \wedge 13254 \wedge 13524 \wedge 13542 \wedge 21543 \wedge 125364 \wedge 125634 \wedge 215364 \wedge 215634 \wedge 315264 \wedge 315624 \wedge 315642: permutations with 0-1 Schubert polynomial		[75]
2143 \wedge 2413 \wedge 3412 \wedge 314562 \wedge 412563 \wedge 415632 \wedge 431562 \wedge 512364 \wedge 512643 \wedge 516432 \wedge 541263 \wedge 541632 \wedge 543162: widdershins permutations		[38]
2431̄ (A051295), 25314 (A117106), 35241̄ (A137534), 42513 (A137535), 42513̄ (A110447), 42153 (A137536), 25134 (A137538), 41523 (A137539), 41253 (A137540), 35241̄ (A137542)		[138] (OEIS shown on the left)
31524̄ = 3142 \wedge 2413̄		[137, 36], A098569
[2143], [3142]		[12]
 permutations that characterize Schubert varieties which are Goren- stein		[172], A097483
	(2 + 2)-free posets	[130, 36] A022493

For propositional formulas F and G consisting of logical ANDs \wedge , ORs \vee , and patterns as variables, we define

$$\begin{aligned} S_n(F \wedge G) &:= S_n(F) \cap S_n(G), \\ S_n(F \vee G) &:= S_n(F) \cup S_n(G). \end{aligned} \tag{6.1}$$

For instance, $S_n(\tau_1 \wedge \cdots \wedge \tau_\ell)$ is the set of permutations avoiding each of the patterns τ_1, \dots, τ_ℓ , and $S_n(\tau_1 \vee \cdots \vee \tau_\ell)$ is the set of permutations avoiding at least one of the patterns τ_1, \dots, τ_ℓ .

Remark 6.2. *From the point of view of counting, we clearly have $|L_n \cup M_n| = |L_n| + |M_n| - |L_n \cap M_n|$, so the problem of counting the union of two zigzag languages can be reduced to counting the individual languages and the intersection. However, from the point of view of exhaustive generation, we clearly do not want to take this approach, namely generate all permutations in L_n , all permutations in M_n , all permutations in $L_n \cap M_n$, and then combine and reduce those lists. This shows that the problem of generating languages like $S_n(\tau_1 \vee \cdots \vee \tau_\ell)$ or $S_n(F)$ for more general formulas F is genuinely interesting in our context. We will see a few applications of this general setting below, and we feel that this direction of generalization deserves further investigation by the pattern-avoidance community.*

6.1.2 Tame patterns

We say that an infinite sequence of sets L_0, L_1, \dots is *hereditary*, if $L_{i-1} = p(L_i)$ holds for all $i \geq 1$. We say that a permutation pattern τ is *tame*, if $S_n(\tau)$, $n \geq 0$, is a hereditary sequence of zigzag languages. The hereditary property ensures that for a given set $S_n(\tau) =: L_n$, we can check whether a permutation π is in the sets $L_{i-1} := p(L_i)$ for $i = n, n-1, \dots, 1$ simply by checking for matches of the pattern τ in π . See also the discussion in Section 6.4.

The following theorem is an immediate consequence of Lemma 6.1 and the definition (6.1).

Theorem 6.3. *Let F be an arbitrary propositional formula consisting of logical ANDs \wedge , ORs \vee , and tame patterns as variables, then $S_n(F)$, $n \geq 0$, is a hereditary sequence of zigzag languages. Consequently, all of these languages can be generated by Algorithm J.*

In the following we provide simple sufficient conditions guaranteeing that a pattern is tame (cf. Section 6.4).

Lemma 6.4. *If a pattern $\tau \in S_k$, $k \geq 3$, does not have the largest value k at the leftmost or rightmost position, then it is tame.*

We prove Lemma 6.4 in Section 6.1.9.

Table 6.1 lists several tame patterns and the combinatorial objects encoded by the corresponding zigzag languages. The bijections between those permutations and the combinatorial objects are well-known and are described in the listed papers (recall also Section ??). The ordering of 231-avoiding permutations of size $n = 4$ generated by Algorithm J, and the corresponding Gray codes for three different Catalan objects are shown in Figure 5.5. We refer to the permutation patterns discussed so far as *classical* patterns. In the following we discuss some other important variants of permutation patterns appearing in the literature.

6.1.3 Vincular patterns

Vincular patterns were introduced by Babson and Steingrímsson [16]. In a *vincular* pattern τ , there is exactly one underlined pair of consecutive entries, with the interpretation that a match of τ in π requires that the underlined entries match adjacent positions in π . For instance, the permutation $\pi = \underline{3}1\underline{4}2$ contains the pattern $\tau = 231$, but it avoids the vincular pattern $\tau = \underline{23}1$.

Lemma 6.5. *If a vincular pattern $\tau \in S_k$, $k \geq 3$, does not have the largest value k at the leftmost or rightmost position, and the largest*

value k is part of the vincular pair, then it is tame.

We prove Lemma 6.5 in Section 6.1.9.

Table 6.1 also lists several tame vincular patterns and the combinatorial objects encoded by the corresponding zigzag languages, namely set partitions and different kinds of rectangulations. The ordering of 132-avoiding permutations of size $n = 4$ generated by Algorithm J, and the resulting Gray code for set partitions, is shown in Figure 5.6. The generated ordering of twisted Baxter permutations of size $n = 4$, and the resulting Gray code for diagonal rectangulations, is shown in Figure 6.2.

6.1.4 Barred patterns

Barred permutation patterns were first considered by West [170]. A *barred* pattern is a pattern τ with a number of overlined entries, e.g., $\tau = 25\bar{3}41$. Let τ' be the permutation obtained by removing the bars in τ , and let τ^- be the permutation that is order-isomorphic to the non-barred entries in τ . In our example, we have $\tau' = 25341$ and $\tau^- = 2431$. A permutation π *contains* a barred pattern τ if and only if it contains a match of τ^- that cannot be extended to a match of τ' by adding entries of π at the positions specified by the barred entries. For instance, $\pi = \boxed{35}2\boxed{41}$ contains $\tau = 25\bar{3}41$, as the highlighted entries form a match of $\tau^- = 2431$ that cannot be extended to a match of $\tau' = 25341$. We clearly have $S_n(\tau^-) \subseteq S_n(\tau)$.

The following lemma gives a sufficient condition for a single-barred pattern to be tame.

Lemma 6.6. *If for a single-barred pattern $\tau \in S_k$, $k \geq 4$, the permutation $\tau^- \in S_{k-1}$ does not have the largest value $k-1$ at the leftmost or rightmost position, and the barred entry in τ is smaller than k or at a position next to the entry $k-1$, then τ is tame.*

We prove Lemma 6.6 in Section 6.1.9. See Table 6.1 for several examples of tame single-barred patterns that were studied by Pud-

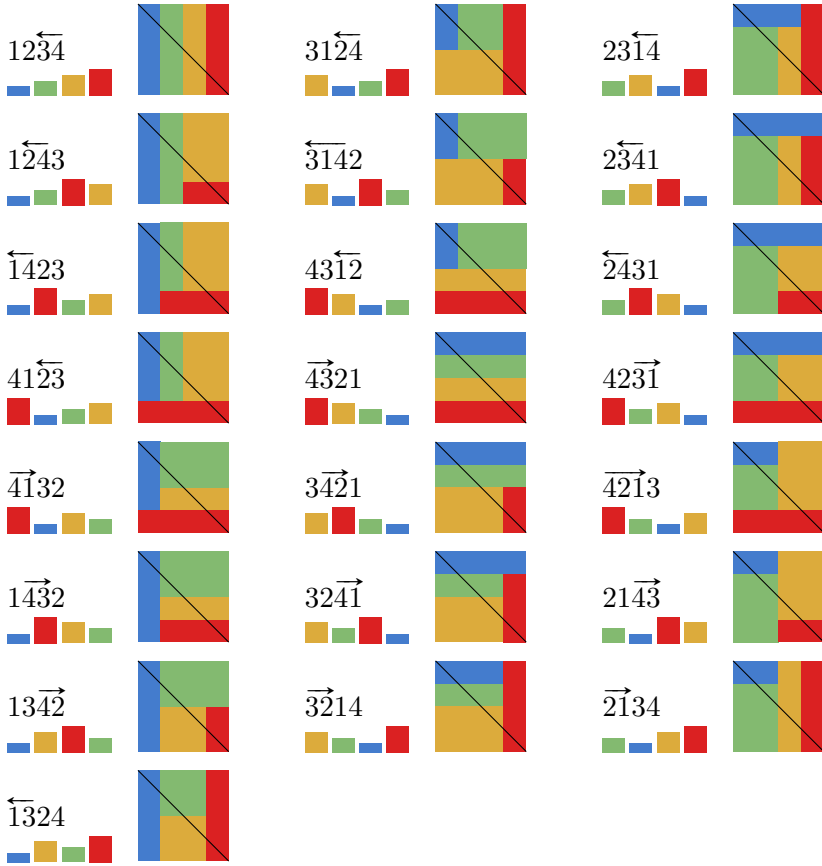


Figure 6.2: Twisted Baxter permutations ($2413 \wedge 3412$ -avoiding) for $n = 4$ generated by Algorithm J and resulting Gray code for diagonal rectangulations.

well [138]. As we will show in Section 6.3.3 below, in many cases patterns with multiple bars can be reduced to single-barred patterns.

6.1.5 Boxed patterns

Boxed patterns were introduced in the paper by Avgustinovich, Kitaev, and Valyuzhenich [12]. A permutation π *contains* the boxed pattern $\boxed{\tau}$ if and only if it contains a match of τ such that no entry of π at a position between the matched ones has a value between the smallest and largest value of the match. For example, the permutation $\pi = 431792865$ contains the boxed pattern $\boxed{2143}$, as the highlighted entries of π form a match of 2143, and the entries 1, 9, 2, 8 are either smaller than 3 or larger than 7. On the other hand, the permutation $\pi' = 351624$ avoids $\boxed{2143}$, as the only possible match of 2143 is at the highlighted positions in π' , but the entries 5 and 2 are both between 1 and 6.

Lemma 6.7. *Given a boxed pattern $\boxed{\tau}$ with $\tau \in S_k$ and $k \geq 3$, if τ does not have the largest value k at the leftmost or rightmost position, then it is tame.*

We prove Lemma 6.7 in Section 6.1.9. Table 6.1 shows two tame boxed patterns studied in [12].

6.1.6 Patterns with Bruhat restrictions

Patterns with Bruhat restrictions were introduced by Woo and Yong [172]. Such a pattern is a pair (τ, B) , where $\tau \in S_k$ and $B \subseteq [k]^2$ is a set of pairs of indices (a, b) with $a < b$ and $\tau(a) < \tau(b)$ such that for all $i \in \{a+1, \dots, b-1\}$ we either have $\tau(i) < \tau(a)$ or $\tau(i) > \tau(b)$. A permutation π *contains* this pattern if and only if it contains a match of τ , and for any pair of entries $\pi(i_a)$ and $\pi(i_b)$ that are matched by a corresponding pair of entries $\tau(a)$ and $\tau(b)$ with $(a, b) \in B$, we have that $\pi(i) < \pi(i_a)$ or $\pi(i) > \pi(i_b)$ for all $i \in \{i_a + 1, \dots, i_b - 1\}$.

Lemma 6.8. *Given a pattern with Bruhat restrictions (τ, B) with $\tau \in S_k$ and $k \geq 3$, if τ does not have the largest value k at the leftmost or rightmost position, then it is tame.*

We prove Lemma 6.8 in Section 6.1.9. Note that Lemma 6.8 does not impose any restrictions on the set B , and that it hence generalizes Lemma 6.4, which corresponds to the case $B = \emptyset$. Table 6.1 shows two patterns with Bruhat restriction studied in [172].

6.1.7 Bivincular patterns

Bivincular patterns were introduced by Bousquet-Mélou, Claesson, Dukes, and Kitaev [36]. Such a pattern is a pair (τ, B) , where $\tau \in S_k$ is a vincular pattern and $B \subseteq [k-1]$. A permutation π *contains* this pattern if and only if it contains a match of the vincular pattern τ (respecting the adjacency condition for the vincular pair), and in this match, the entries at positions $\tau^{-1}(i)$ and $\tau^{-1}(i+1)$ are consecutive values in π for all $i \in B$.

Lemma 6.9. *Given a bivincular pattern (τ, B) with $\tau \in S_k$ and $k \geq 3$, if the vincular pattern τ satisfies the conditions in Lemma 6.5 and if $k-1 \notin B$, then it is tame.*

We prove Lemma 6.9 in Section 6.1.9. Note that Lemma 6.9 generalizes Lemma 6.5, which corresponds to the case $B = \emptyset$. In Table 6.1, $(2+2)$ -free posets are mentioned as an example of a combinatorial class that is in bijection to permutations avoiding a tame bivincular pattern.

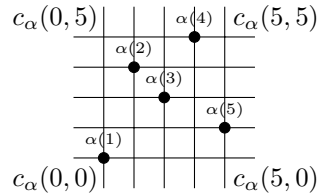
6.1.8 Mesh patterns

In the following we take a geometric viewpoint on permutations. For any pair of real numbers $P = (a, b)$, we define $P_x := a$ and $P_y := b$. Moreover, for any mapping $\alpha : [n] \rightarrow B$ and any subset $I \subseteq [n]$

with $|I| = k$ we write $\alpha|_I : [k] \rightarrow B$ for the function defined by $\alpha|_I(i) := \alpha(j)$, where j is the i th smallest element in I , for all $i \in [k]$.

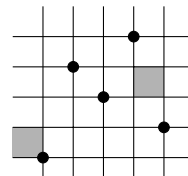
The *grid representation* of a permutation $\pi \in S_n$ is a mapping $\alpha : [n] \rightarrow \mathbb{Z}^2$ such that the sequence $\alpha(1)_x, \dots, \alpha(n)_x$ is strictly increasing, and the sequence $\alpha(1)_y, \dots, \alpha(n)_y$ is order-isomorphic to π . This representation is unique up to shifts that pre-

serve the relative order of the values $\alpha(1)_y, \dots, \alpha(n)_y$. We can think of the grid representation of π as a graphical representation of the permutation matrix. For instance, the permutation $\pi = 14352$ has the grid representation shown on the right. A *cell* in the grid representation α is a connected region from $\mathbb{R}^2 \setminus (\alpha([n])_x \times \mathbb{R} \cup \mathbb{R} \times \alpha([n])_y)$, and we denote these cells by $c_\alpha(i, j)$, $i, j \in \{0, \dots, n\}$, where the first index i increases with x , and the second index j increases with y , as shown in the figure on the right. By definition, every cell is a Cartesian product of two open intervals. For instance, we have $c_\alpha(1, 0) =]\alpha(1)_x, \alpha(2)_x[\times]-\infty, \alpha(1)_y[$. If the grid representation α is clear from the context, we sometimes refer to a cell $c_\alpha(i, j)$ simply by its index (i, j) .



Mesh patterns were introduced by Brändén and Claesson [37], and they generalize all the aforementioned types of patterns.

A *mesh pattern* is a pair $\sigma = (\tau, C)$, $\tau \in S_k$, with $C \subseteq \{0, \dots, k\}^2$. Each pair $(a, b) \in C$ encodes the cell with index (a, b) in the grid representation of τ , and in our figures we

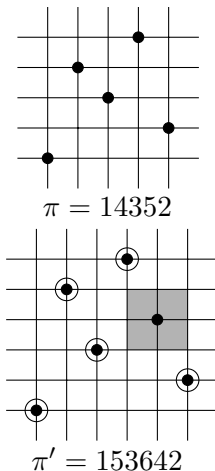


$$\begin{aligned}\sigma &= (\tau, C) \\ \tau &= 14352 \\ C &= \{(0, 1), (4, 3)\}\end{aligned}$$

visualize σ by drawing the grid representation of τ and by shading exactly the cells indexed by C . For instance, the mesh pattern $\sigma = (\tau, C) = (14352, \{(0, 1), (4, 3)\})$ has the graphical representation shown above.

A permutation $\pi \in S_n$ *contains* the mesh pattern $\sigma = (\tau, C)$, $\tau \in S_k$, if the grid representation α of π admits a subset $I \subseteq [n]$, $|I| = k$, such that $\beta := \alpha|_I$ is the grid representation of τ , and the cell $c_\beta(i, j)$ contains no points from $\alpha([n])$ for all $(i, j) \in C$. In this case, we refer to β as a *match* of the mesh pattern σ in the grid representation α of π . Note that the first condition in the definition of mesh pattern containment is equivalent to requiring that the subpermutation of π on the indices in I is order-isomorphic to τ , while the second condition requires that the cells of C in the grid representation of τ in π contain no points from π .

For example, consider the grid representation α of each of the two permutations $\pi = 14352$ and $\pi' = 153642$ shown on the right. While π clearly contains the mesh pattern σ from before, the permutation π' avoids it, as the only choice for $I \subseteq [5]$ such that $\beta := \alpha|_I$ is the grid representation of τ is $I = \{1, 2, 3, 4, 6\}$ (marked points in the figure), but then the cell $c_\beta(4, 3)$ (shaded gray) contains the point $\alpha(5)$ (non-marked).



The following main theorem of this section implies all the lemmas about classical, vincular, barred patterns, etc. stated in the previous sections.

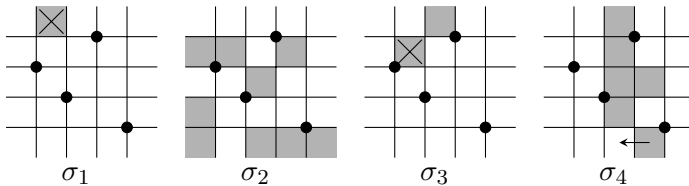
Theorem 6.10. *Let $\sigma = (\tau, C)$, $\tau \in S_k$, $k \geq 3$, be a mesh pattern, and let i be the position of the largest value k in τ . If the pattern satisfies each of the following four conditions, then it is tame:*

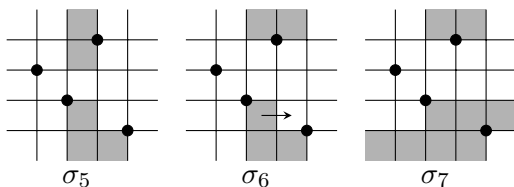
- (i) i is different from 1 and k .
- (ii) For all $a \in \{0, \dots, k\} \setminus \{i-1, i\}$, we have $(a, k) \notin C$.
- (iii) If $(i-1, k) \in C$, then for all $a \in \{0, \dots, k\} \setminus \{i-1\}$ we have $(a, k-1) \notin C$ and for all $b \in \{0, \dots, k-2\}$ we have that $(i, b) \in C$ implies $(i-1, b) \in C$.

- (iv) If $(i, k) \in C$, then for all $a \in \{0, \dots, k\} \setminus \{i\}$ we have $(a, k-1) \notin C$ and for all $b \in \{0, \dots, k-2\}$ we have that $(i-1, b) \in C$ implies $(i, b) \in C$.

The conditions in Theorem 6.10 can be understood in the grid representation of (τ, C) as follows; see the left hand side of Figure 6.3: Condition (i) asserts that the highest point of τ must not be the leftmost or rightmost point (the two crossed out grid points in the figure are forbidden). Condition (ii) asserts that none of the cells in the topmost row (above the points) must be shaded, with the possible exception of the cells next to the highest point (solid crossed out cells in the figure). Condition (iii) asserts that if the cell $(i-1, k)$ to the top left of the highest point is shaded (dark gray cell in the figure), then none of the cells in the row below except possibly $(i-1, k-1)$ must be shaded (dotted crossed out cells), and if one of the cells strictly below $(i, k-1)$ is shaded (dark gray questions marks), then the cell to the left of it must also be shaded (indicated by an arrow to the left). Symmetrically, condition (iv) asserts that if the cell (i, k) to the top right of the highest point is shaded (light gray cell in the figure), then none of the cells in the row below except possibly $(i, k-1)$ must be shaded (dashed crossed out cells), and if one of the cells strictly below $(i-1, k-1)$ is shaded (light gray questions marks), then the cell to the right of it must also be shaded (indicated by an arrow to the right).

To illustrate the conditions in Theorem 6.10 further, consider the following mesh patterns $\sigma_k = (3241, C_k)$, $k = 1, \dots, 7$, with different sets C_k :





All of these mesh patterns satisfy condition (i). The pattern σ_1 violates condition (ii) due to the cell $(1,4) \in C_1$ (crossed in the figure), while all other patterns satisfy this condition. The pattern σ_2 satisfies conditions (iii) and (iv) trivially, as the premises of each of the two conditions are not satisfied. The pattern σ_3 violates the first part of condition (iii) due to the cell $(1,3) \in C_3$ (crossed). The pattern σ_4 satisfies the first part of condition (iii), but violates the second part due to the cells $(3,0) \in C_4$ and $(2,0) \notin C_4$ (connected by an arrow in the figure). The pattern σ_5 satisfies conditions (iii) and (iv). The pattern σ_6 violates condition (iv) due to the cells $(2,1) \in C_6$ and $(3,1) \notin C_6$ (arrow). Finally, the pattern σ_7 satisfies conditions (iii) and (iv).

Proof. We show that if $\sigma = (\tau, C)$ satisfies the four conditions of the theorem, then $S_n(\sigma)$, $n \geq 0$, is a hereditary sequence of zigzag languages. We argue by induction on n . Note that $S_0(\sigma) = S_0 = \{\varepsilon\}$ is a zigzag language by definition, so the induction basis is clear. For the induction step let $n \geq 1$. We first show that if $\pi \in S_{n-1}(\sigma)$, then $c_1(\pi), c_n(\pi) \in S_n(\sigma)$. As $c_1(\pi)$ and $c_n(\pi)$ are obtained from π by inserting the new largest value n at the leftmost or rightmost position, respectively, the grid representation of these two permutations differs from the grid representation of π by adding a new highest point at the leftmost or rightmost position. However, as π avoids σ by assumption, condition (i) guarantees that both $c_1(\pi)$ and $c_n(\pi)$ also avoid σ , which is what we wanted to show.

To complete the induction step, we now show that if $\pi \in S_n(\sigma)$, then $p(\pi) \in S_{n-1}(\sigma)$. Recall that $p(\pi)$ is obtained from π by removing

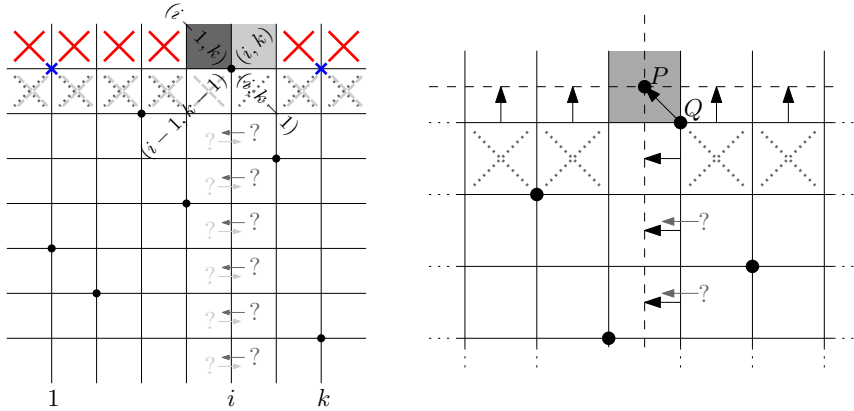


Figure 6.3: Illustration of the four conditions in Theorem 6.10 (left) and how they are used in the proof of the theorem (right).

the largest value n , so in the grid representation, we remove the highest point P . Our assumption is that π avoids the pattern σ , and we need to show that removing the highest point does not create a match of the pattern σ . For the sake of contradiction, suppose that removing P creates a match of the pattern σ in $p(\pi)$. Let Q be the highest point in this match of the pattern σ in $p(\pi)$. By condition (ii), we are in exactly one of the following two symmetric cases: (a) the cell $(i-1, k)$ is in C and P lies inside this cell of σ in this match of the pattern; (b) the cell (i, k) is in C and P lies inside this cell of σ in this match of the pattern. We first consider case (a), which is illustrated on the right hand side of Figure 6.3: We claim that we can exchange the point Q for the point P in the match of the pattern σ , and obtain another match of σ in π , which would contradict the assumption that π avoids σ . Indeed, this exchange operation strictly enlarges only the cells $(a, k-1)$ for all $a \in \{0, \dots, k\} \setminus \{i-1\}$ and the cells (i, b) for all $b \in \{0, \dots, k-2\}$. The first set of cells are not in C by the first part of condition (iii). The second set of cells are either not in C , or if they are, then

the corresponding cells to the left of it are also in C by the second part of condition (iii). Moreover, after the exchange the cell (i, k) contains no point from π , as P is the highest point (this is of course only relevant if $(i, k) \in C$). Furthermore, after the exchange the cell $(i - 1, k - 1)$ contains at most those points from π that were in the same cell before the exchange (clearly P is the only point inside the cell $(i - 1, k)$). So we indeed obtain a match of σ in π , a contradiction.

In the symmetric case (b), we apply the same exchange argument, using condition (iv) instead of (iii). This completes the proof. \square

6.1.9 Proof of Lemmas 6.4–6.9

With Theorem 6.10 in hand, the proofs of Lemmas 6.4–6.9 are straightforward. As noted before, Lemma 6.8 generalizes Lemma 6.4, and Lemma 6.9 generalizes Lemma 6.5, so we only need to prove Lemmas 6.6–6.9.

Proof of Lemma 6.6. Note that a barred pattern $\tau \in S_k$ with a single barred entry b at position a corresponds to the mesh pattern $\sigma = (\tau^-, \{(a - 1, b - 1)\})$, i.e., in the grid representation of σ a single cell is shaded. It follows that conditions (iii) and (iv) of Theorem 6.10 are trivially satisfied, and conditions (i) and (ii) translate into the conditions in the lemma. \square

Proof of Lemma 6.7. A boxed pattern $\boxed{\tau}$, $\tau \in S_k$, corresponds to the mesh pattern $\sigma = (\tau, C)$ with $C := \{(i, j) \mid 1 \leq i, j \leq k - 1\}$, i.e., in the grid representation of σ , all cells inside the bounding box of the points from τ are shaded. It follows that conditions (ii)–(iv) of Theorem 6.10 are trivially satisfied, and condition (i) corresponds exactly to the condition in the lemma. \square

Proof of Lemma 6.8. A pattern with Bruhat restrictions (τ, B) corresponds to the mesh pattern $\sigma = (\tau, C)$ where C is the union of

all the sets $R(a, b) := \{(i, j) \mid a \leq i < b \wedge \tau(a) \leq j < \tau(b)\}$ for $(a, b) \in B$, i.e., in the grid representation of σ , certain rectangles of cells inside the bounding box of the points from τ are shaded. It follows that conditions (ii)–(iv) of Theorem 6.10 are trivially satisfied, and condition (i) corresponds exactly to the condition in the lemma. \square

Proof of Lemma 6.9. A vincular pattern $\tau \in S_k$ where the entries at positions a and $a + 1$ are underlined corresponds to the mesh pattern $\sigma = (\tau, C)$ with $C := \{a\} \times \{0, \dots, k\}$, i.e., in the grid representation of σ , an entire column of cells is shaded. For the bivincular pattern (τ, B) we also have to add the sets $\{0, \dots, k\} \times \{b\}$ for all $b \in B$ to the set of cells C , i.e., in the grid representation we also have to shade the corresponding rows of cells. By the conditions stated in Lemma 6.5, conditions (i) and (ii) of Theorem 6.10 are satisfied. By the condition $k - 1 \notin B$, conditions (iii) and (iv) of the theorem are also satisfied, proving that the mesh pattern σ is tame. \square

6.2 Patterns with multiplicities

All the aforementioned notions and results in this section generalize straightforwardly to bounding the number of appearances of a pattern. Formally, a *counted pattern* is a pair $\sigma = (\tau, c)$, where τ is a mesh pattern, and c is a non-negative integer. Moreover, $S_n(\sigma)$ denotes the set of all permutations from S_n that contain *at most* c matches of the pattern τ , where the special case $c = 0$ is pattern-avoidance (cf. [127]).

By Theorem 6.3, we can form propositional formulas F consisting of logical ANDs \wedge , ORs \vee , and tame counted patterns (τ_i, c_i) as variables, with possibly different counts c_i for each variable. The tameness of each (τ_i, c_i) can be checked by verifying whether the pat-

terns τ_i satisfy the conditions stated in Theorem 6.10 or its corollaries Lemmas 6.4–6.9. We then obtain a hereditary zigzag language $S_n(F)$ that can be generated by Algorithm J.

A somewhat contrived example for such a language would be $F = ((231, 3) \wedge (2143, 5)) \vee (3\underline{142}, 2)$, the language of permutations that contain at most 3 matches of the pattern 231 AND at most 5 matches of the pattern 2143, OR at most 2 matches of the vincular pattern $3\underline{142}$.

6.3 Algebra with patterns

In this section we significantly extend the methods described in the previous section, by applying geometric transformations to permutation patterns, and by describing some other types of patterns as conjunctions and disjunctions of suitable mesh patterns (recall Theorem 6.3). Two particularly relevant additional types of permutations covered in this section are monotone and geometric grid classes; see Theorem 6.16 below.

6.3.1 Elementary transformations

We now consider three important *elementary transformations* of permutations that are important in the context of pattern-avoidance, as they preserve the cardinality of the set $S_n(F)$. Each of them corresponds to a geometric transformation of the grid representation of each of the patterns $\tau = a_1 \dots a_k$ in the formula F , and together these transformations form the dihedral group D_4 of symmetries of a regular 4-gon:

- *Reversal*, defined as $\text{rev}(\tau) := a_k \dots a_1$. This corresponds to a vertical reflection of the grid representation.
- *Complementation*, defined as $\text{cpl}(\tau)_i = k + 1 - a_i$ for all $i = 1, \dots, k$. This corresponds to a horizontal reflection of the grid

representation.

- *Inversion*, defined by $\text{inv}(\tau)_{\tau(i)} = i$ for all $i = 1, \dots, k$. This corresponds to a diagonal reflection of the grid representation along the south-west to north-east diagonal.

Note that a clockwise 90-degree rotation is obtained as $\text{rot}(\tau) := \text{inv}(\text{rev}(\tau)) = \text{cpl}(\text{inv}(\tau))$. Clearly, all these operations generalize to mesh patterns (τ, C) , by applying the aforementioned geometric transformations to the grid representation of (τ, C) . These operations and their relations are illustrated in Figure 6.4 for $(\tau, C) = (14352, \{(1, 0), (1, 1), (3, 3), (4, 3)\})$.

The following lemma is immediate.

Lemma 6.11. *Given any composition h of the elementary transformations reversal, complementation and inversion, and any propositional formula F consisting of logical ANDs \wedge , ORs \vee , and mesh patterns τ_1, \dots, τ_ℓ as variables, then the sets of pattern-avoiding permutations $S_n(F)$ and $S_n(h(F))$ are in bijection under h for all $n \geq 1$, where the formula $h(F)$ is obtained from F by replacing every pattern τ_i by $h(\tau_i)$ for all $i = 1, \dots, \ell$.*

Lemma 6.11 is very useful for the purpose of exhaustive generation, because even if τ_i is not tame, then maybe $h(\tau_i)$ is. So even if we cannot apply Algorithm J to generate $S_n(\tau)$ directly, we may be able to generate $S_n(h(\tau_i))$, and then apply h^{-1} to the resulting permutations. For instance, $\tau = 213$ is not tame, as the largest entry appears at the rightmost position. However, $\text{cpl}(\tau) = 231$ is tame by Lemma 6.4, and so we can use Algorithm J to generate $S_n(\text{cpl}(\tau))$.

As another example, consider so-called *2-stack sortable permutations* introduced by West [170] and later counted in [175, 80, 65]. These permutations are characterized by the pattern-avoidance formula $F = \tau_1 \wedge \tau_2$ with $\tau_1 := 2341$ and $\tau_2 := 3\bar{5}241$ (τ_2 is a barred pattern). Unfortunately, τ_2 is not tame (the barred entry $\bar{5}$ is not at a position

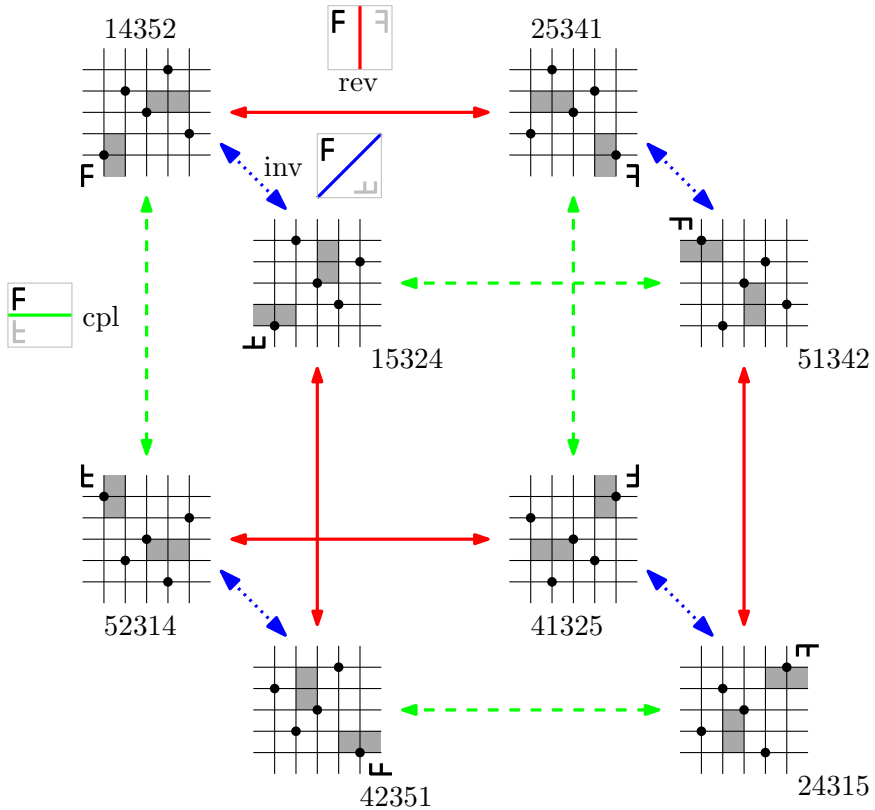


Figure 6.4: Elementary transformations between permutations.

next to the entry 4; recall Lemma 6.6), so Algorithm J cannot be used directly for generating $S_n(F)$. However, applying rotation, $h(\tau) := \text{rot}(\tau) = \text{inv}(\text{rev}(\tau))$, yields two tame patterns $h(\tau_1) = 1432$ and $h(\tau_2) = 13524$ and the formula $h(F) = h(\tau_1) \wedge h(\tau_2)$, which can be used for generating $S_n(h(F))$ via Algorithm J:

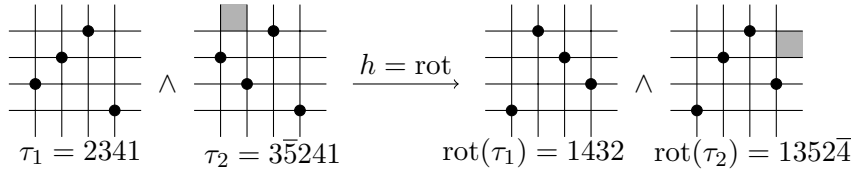


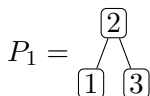
Table 6.5 lists several more permutations patterns that have been studied in the literature and that can be turned into tame patterns by such elementary transformations.

Table 6.5: Permutation patterns that become tame through elementary transformations, plus corresponding combinatorial objects.

Tame patterns and combinatorial objects	References/ OEIS [129]
$\text{rot}(2341 \wedge \bar{3}\bar{5}241) = 1432 \wedge 1352\bar{4}$ (2-stack sortable permutations), $2413 \wedge 41\bar{3}52$, $2413 \wedge 45\bar{3}12$, $2413 \wedge 21\bar{3}54$, $3241 \wedge \bar{2}4153$, $\text{rot}^{-1}(2413 \wedge 5\bar{1}324) = 2413 \wedge 1534\bar{2}$, $\text{cpl}(2413 \wedge \bar{4}2315) = 3142 \wedge \bar{2}4351$, $\text{cpl}(2314 \wedge \bar{4}2513) = 3241 \wedge \bar{2}4153$, $\text{cpl}(3214 \wedge \bar{2}4135) = 2341 \wedge \bar{4}2531$, $\text{rev}(2413 \wedge 41\bar{3}52) = 3142 \wedge \underline{2413}$: rooted non-separable planar maps	[170, 175, 80] [65], A000139 [66] [51]
conjunction of 20 patterns τ_i with tame $\text{cpl}(\tau_i)$: permutations generated by a stack of depth two and an infinite stack	[67], A245233
$\text{inv}(132 \wedge 312) = 132 \wedge 231$: Gilbreath permutations	[167, 59]
$\text{rot}(3\bar{1}\underline{4}2 \wedge 3\bar{1}\underline{2}4) = \text{rot} \left(\begin{array}{ c c c c } \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array} \wedge \begin{array}{ c c c c } \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array} \right) = \begin{array}{ c c c c } \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array} \wedge \begin{array}{ c c c c } \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}$: permutations that uniquely encode pile configurations in patience sorting	[111, 39], A129698

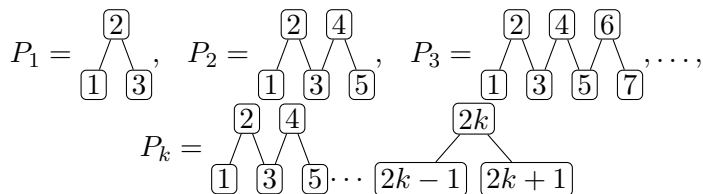
6.3.2 Partially ordered patterns

Partially ordered patterns were introduced by Kitaev [105]. A *partially ordered pattern (POP)* is a partially ordered set $P = ([k], \prec)$, and we say that a permutation π *contains* this pattern if and only if it contains a subpermutation $a_{i_1} \dots a_{i_k}$, $i_1 < \dots < i_k$, such that $k \prec l$ in the partial order implies that $a_{i_k} < a_{i_l}$. In particular, if \prec is a linear order, then this is equivalent to classical pattern avoidance. However, some other constraints can be expressed much more conveniently using POPs. For instance, avoiding the POP



is equivalent to avoiding peaks in the permutation, so $S_n(P_1)$ is the set of permutations without peaks discussed before, which satisfies $|S_n(P_1)| = 2^{n-1}$.

More generally, the POP P_k defined by



realizes the language $S_n(P_k)$ of permutations with at most $k - 1$ peaks.

We let $L(P)$ denote the set of all linear extensions of the poset P , and for any linear extension $x \in L(P)$, we consider the inverse permutation of x , as the i th entry of $\text{inv}(x)$ denotes the position of i in x . Moreover, $\text{inv}(x) \in S_k$, so $\text{inv}(x)$ is a classical pattern.

Lemma 6.12. *For any partially ordered pattern $P = ([k], \prec)$,*

$$S_n(P) = \bigcap_{x \in L(P)} S_n(\text{inv}(x)) = S_n\left(\bigwedge_{x \in L(P)} \text{inv}(x)\right). \quad (6.2)$$

In particular, if the poset P does not have 1 or k as a maximal element, then P is tame.

Proof. The first part of the lemma follows immediately from the definition of POPs and from (6.1). To prove the second part, suppose that P does not have 1 or k as a maximal element. Then in any linear extension $x \in L(P)$, 1 and k will not appear at the last position, and so in the inverse permutation $\text{inv}(x)$, the largest entry k will neither be at position 1 nor at position k . We can hence apply Lemma 6.4, and using Theorem 6.3 we obtain that P is tame. \square

For instance, for the POP P_1 from before we have $L(P_1) = \{132, 312\}$, and so $P_1 = 132 \wedge 231$, and for the POP P_2 we have $L(P_2) = \{13254, 13524, 13542, 15324, 15342, 31254, \dots\}$, a set of 16 linear extensions in total, so $P_2 = 13254 \wedge 14253 \wedge 15243 \wedge 14352 \wedge 15342 \wedge 23145 \wedge \dots$.

Moreover, we can create counted POPs with multiplicity c (recall Section 6.2), by taking the OR of conjunctions of counted classical patterns as described by Lemma 6.12, over all number partitions of c into the corresponding number of parts. For instance, the counted POP $\sigma = (P_1, c)$, which realizes the zigzag language $S_n(\sigma)$ of permutations with at most c triples of values forming a peak, is obtained by considering the partitions $c = c + 0 = (c - 1) + 1 = \dots = 1 + (c - 1) = 0 + c$, resulting in the formula

$$(P_1, c) = ((132, c) \wedge (231, 0)) \vee \dots \vee ((132, 0) \wedge (231, c))$$

with counted classical patterns on the right-hand side.

6.3.3 Barred patterns with multiple bars

Some patterns with multiple bars can be reduced to single-barred patterns (to which Lemma 6.6 applies) as shown by the following lemma.

Lemma 6.13 (cf. [163]). *Let $\tau \in S_k$, $k \geq 5$, be a pattern with $b \geq 2$ bars, such that no two barred entries are at neighboring positions or have adjacent values. Let $\tilde{\tau}_1, \dots, \tilde{\tau}_b \in S_{k-b+1}$ be the permutations with a single barred entry that are order-isomorphic to the sequences obtained from τ by removing all but one barred entry. Then we have*

$$S_n(\tau) = \bigcap_{1 \leq i \leq b} S_n(\tilde{\tau}_i) = S_n\left(\bigwedge_{1 \leq i \leq b} \tilde{\tau}_i\right).$$

Consequently, if $\tau^- \in S_{k-b}$ does not have the largest value $k-b$ at the leftmost or rightmost position, and the largest barred entry in τ is smaller than k or at a position next to the entry $k-1$, then τ is tame.

Proof. To prove the first part, observe that when no two barred entries are at neighboring positions or have adjacent values, then the definition of barred pattern avoidance is equivalent to avoiding each of the single-barred patterns $\tilde{\tau}_1, \dots, \tilde{\tau}_b$, so the claim follows using (6.1).

To prove the second part we show that each of the single-barred patterns $\tilde{\tau}_1, \dots, \tilde{\tau}_b$ satisfies the conditions of Lemma 6.6. Indeed, we know that $\tau^- = (\tilde{\tau}_i)^- \in S_{k-b}$, $1 \leq i \leq b$, does not have the largest value $k-b$ at the leftmost or rightmost position. Moreover, if $\tilde{\tau}_i$ is obtained from τ by removing all but the largest barred entry, then the barred entry in $\tilde{\tau}_i \in S_{k-b+1}$ is either smaller than $k-b+1$ or at a position next to the entry $k-b$. To see this note that if the largest entry k in τ is barred, then the second largest entry $k-1$ is not barred by the assumption that no two barred entries have

adjacent values. For the same reason, if $\tilde{\tau}_i$ is obtained from τ by removing barred entries including the largest one, then the barred entry in $\tilde{\tau}_i \in S_{k-b+1}$ is smaller than $k-b+1$. Consequently, we can apply Lemma 6.6 to each of the patterns $\tilde{\tau}_1, \dots, \tilde{\tau}_b$, and complete the proof by applying Theorem 6.3. \square

Lemma 6.13 applies for instance to the tame pattern $3\bar{1}52\bar{4} = 3\bar{1}42 \wedge 241\bar{3}$ listed in Table 6.1.







6.3.4 Weak avoidance of barred patterns and dotted patterns

Weak pattern avoidance and dotted patterns were introduced by Baril [21] (see also [60]). Given a single-barred pattern τ , we define τ' and τ^- as in Section 6.1.4, and we say that a permutation π *weakly contains* τ , if and only if it contains a match of τ^- that cannot be extended to a match of τ' by adding one entry of π , not necessarily at the position specified by the barred entry. Otherwise, we say that π *weakly avoids* τ . We let $S_n^w(\tau)$ denote the set of permutations that weakly avoid τ . For instance, $\pi = 1243$ contains the barred pattern $\tau = 12\bar{3}$, as for the increasing pair 24 in π , there is no entry in π to the right that extends this pair to an increasing triple. However, π weakly avoids τ , as each of the increasing pairs 12, 14, 13, 24 and 23 can be extended to an increasing triple, by adding an entry from π to the right, middle, middle, left and left of this pair, respectively.

The relation between weak pattern avoidance and mesh pattern avoidance is captured by the following lemma.

Lemma 6.14. *Let $\tau \in S_k$ be a single-barred pattern with barred entry b . Consider the longest increasing or decreasing substring of consecutive values in τ' that contains b , and let r and s be the start and end indices of this substring. Let σ be the mesh pattern defined*

Table 6.6: Illustration of Lemma 6.14.

τ	τ'	τ^-	r	s	σ
$\bar{1}45632$	145632	34521	1	1	
$1\bar{4}5632$	145632	14532	2	4	
$14\bar{5}632$	145632	14532	2	4	
$145\bar{6}32$	145632	14532	2	4	
1456 $\bar{3}2$	145632	13452	5	6	
14563 $\bar{2}$	145632	13452	5	6	

by $\sigma := (\tau^-, C)$ with

$$C := \{(r+i-1, \tau'(r+i)-1) \mid 0 \leq i \leq s-r\}. \quad (6.3)$$

Then we have $S_n^w(\tau) = S_n(\sigma)$.

Table 6.6 illustrates the mesh pattern $\sigma = (\tau^-, C)$ defined in Lemma 6.14 for six different single-barred patterns.

Proof. We only consider the case that the longest substring of consecutive values in τ' that contains b is increasing, as the other case is symmetric.

We first show that if a permutation π weakly contains τ , then it also contains σ . For this consider a match of τ^- that cannot be extended to a match of τ' in the grid representation of π ; see the left hand side of Figure 6.7. Consider the $s-r$ points P_r, \dots, P_{s-1} of π to which the entries at positions $r, \dots, s-1$ of τ^- are matched. We know that they form an increasing sequence, and all other points in this match are below or above them. Now consider the $s-r+1$ cells in C defined in (6.3) between and around these points. We need to show that none of them contains any points of π , demonstrating that this is a match of the mesh pattern σ . Indeed, if one of these regions did

contain a point Q from π , then Q together with P_r, \dots, P_{s-1} would form an increasing sequence of length $s - r + 1$, i.e., Q would extend the match of τ^- to a match of τ' in π , a contradiction.

It remains to show that if a permutation π contains the mesh pattern σ , then it weakly contains τ . For this consider a match of $\sigma = (\tau^-, C)$ in the grid representation of π ; see the right hand side of Figure 6.7. We label the points to which the entries of τ^- are matched by P_1, \dots, P_{k-1} . By the definition of σ , the points P_r, \dots, P_{s-1} form a longest increasing sequence of consecutive values in this match. In particular, P_{r-1} is not located at the bottom left corner of the leftmost cell of C , and P_s is not located at the top right corner of the rightmost cell of C , so there is a point P_a , $a \in [k-1] \setminus \{r-1, \dots, s-1\}$, on the same height as the lower boundary of the leftmost cell of C , and a point P_b , $b \in [k-1] \setminus \{r, \dots, s\}$, on the same height as the upper boundary of the rightmost cell of C . We know that no point of π lies within any of the cells in C , but we also need to show that a point Q of π contained in any of the other cells cannot be used to extend this match of τ^- to a match of τ' . For this we distinguish four cases: Suppose that Q is contained in a cell L to the left of the cells in C . Then P_1, \dots, P_{k-1} together with Q is not a match of τ' , as the longest increasing substring of consecutive values P_r, \dots, P_{s-1} contains only $s - r$ points. A symmetric argument works if Q is contained in a cell R to the right of the cells in C . Now suppose that Q is contained in a cell A above a cell from C , but not above the rightmost one, or below a cell from C , but not below the leftmost one. In this case the points P_r, \dots, P_{s-1} together with Q do not form an increasing substring, so this is not a match of τ' . It remains to consider the case that Q is contained in a cell B above the rightmost cell from C , or below the leftmost cell from C . In this case the points P_r, \dots, P_{s-1} together with Q form an increasing substring, but the values are not consecutive, as Q is separated from P_r, \dots, P_{s-1} by the point P_b or P_a , respectively, so this is not match of τ' either. This completes the proof.

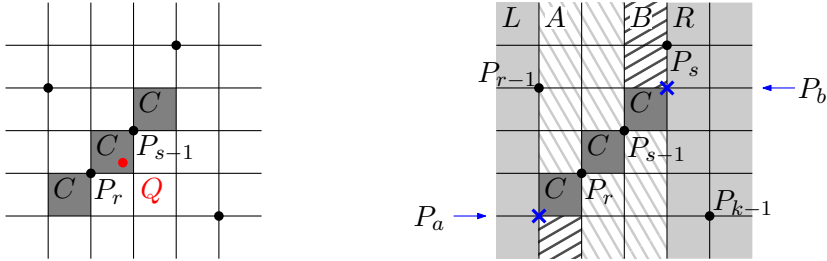


Figure 6.7: Illustration of the proof of Lemma 6.14.

□

Baril [21] also introduced the notion of a *dotted* pattern $\tau \in S_k$, which is a pattern with some entries at positions $I \subseteq [k]$ that have a dot above them. A permutation π *avoids* the pattern τ if and only if π weakly avoids every single-barred pattern obtained by putting a bar above every entry not in I , i.e., we have

$$S_n(\tau) := \bigcap_{j \in [k] \setminus I} S_n^w(\tau_j), \quad (6.4)$$

where τ_j is the barred pattern obtained by putting the bar above entry j .

Lemma 6.15. *Let $\tau \in S_k$, $k \geq 3$, be a dotted pattern with dots over all positions $I \subseteq [k]$. If τ does not have the largest value k at the leftmost or rightmost position, and all entries in the longest increasing or decreasing substring of consecutive values including k are dotted, then τ is tame.*

Proof. Let r and s be the start and end indices of the longest increasing or decreasing substring of consecutive values including k . The conditions of the lemma imply that $[r, s] \subseteq I$.

By (6.4), Theorem 6.3, and Lemma 6.14, to prove that τ is tame it is sufficient to show this for the mesh pattern $\sigma = (\rho^-, C)$ with C defined in (6.3) for every single-barred pattern ρ obtained from τ' by placing a bar over an entry at a position $[k] \setminus I$. As $[r, s] \subseteq I$, we obtain in particular that the largest value k in ρ is not barred, and if the second largest entry $k - 1$ is next to k , then it is also not barred. Combining this with the assumption that the largest value k is not at the leftmost or rightmost position in ρ , we obtain that in ρ^- , the largest entry $k - 1$ is not at the leftmost or rightmost position. Moreover, we obtain from the definition (6.3) that C does not have any cells in the topmost row, i.e., $(i, k - 1) \notin C$ for $i = 0, \dots, k - 1$. Therefore, applying Theorem 6.10 shows that σ is tame, completing the proof. \square

6.3.5 Monotone and geometric grid classes

Monotone grid classes of permutations were introduced by Huczynska and Vatter [94]. To define them, we consider a matrix M with entries from $\{0, +1, -1\}$, indexed first by columns from left to right, and then by rows from bottom to top. A permutation π of $[n]$ (for any $n \geq 0$) is in the *monotone grid class of M* , denoted $\text{Grid}(M)$, if we can place the points labelled from 1 to n from bottom to top into a rectangular grid that has as many rows and columns as the matrix M , and reading the labels from left to right will yield π , subject to the following two conditions: No two points are placed on the same horizontal or vertical line. Moreover, for each cell (x, y) in the grid, if $M_{x,y} = 0$, then the cell contains no points, if $M_{x,y} = +1$, then the points in this cell are increasing, and if $M_{x,y} = -1$, then the points in this cell are decreasing. This definition is illustrated in the top part of Figure 6.8. Based on this, we define $\text{Grid}_n(M) := \text{Grid}(M) \cap S_n$. It is an open problem whether $\text{Grid}(M)$ is characterized by finitely many forbidden patterns for any M (cf. [4]).

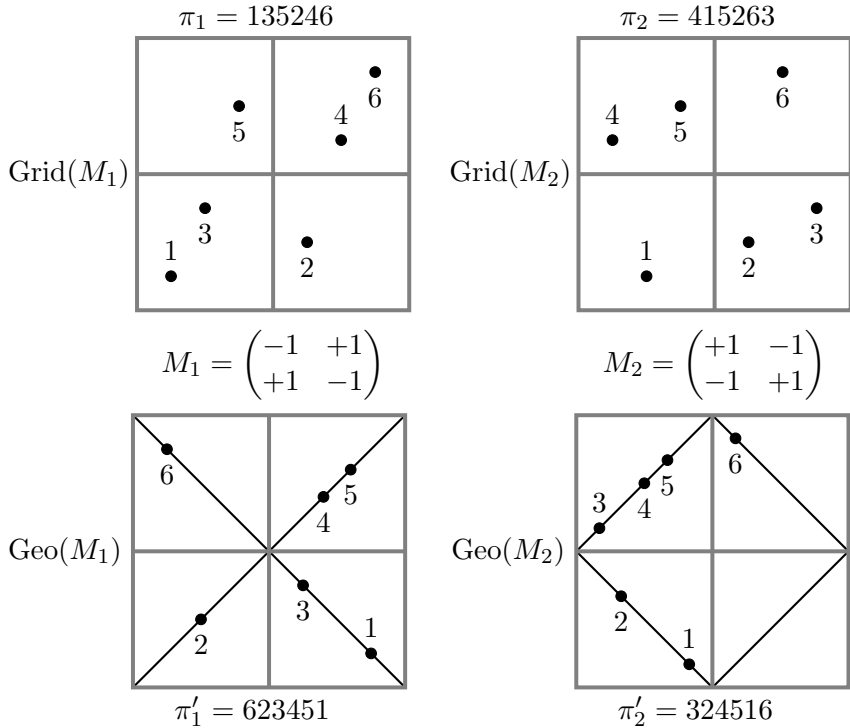


Figure 6.8: Top: Illustration of monotone grid classes. Bottom: Illustration of geometric grid classes, with X-shaped permutations on the left, and O-shaped permutations on the right. Observe that $\pi_1 \in \text{Grid}(M_1)$, but $\pi_1 \notin \text{Geo}(M_1)$, as π_1 contains the pattern 2413. Similarly, we have $\pi_2 \in \text{Grid}(M_2)$, but $\pi_2 \notin \text{Geo}(M_2)$, as π_2 contains the pattern 1423.

The second type of permutations we shall discuss in this section are geometric grid classes, introduced by Albert, Atkinson, Bouvel, Ruskuc, and Vatter [5]. They are defined using a matrix M with entries from $\{0, +1, -1\}$ as before. A permutation π of $[n]$ is in the *geometric grid class of M* , denoted $\text{Geo}(M)$, if it can be drawn in a rectangular grid as described before, with the slightly strengthened conditions that if $M_{x,y} = +1$, then the points in the cell (x, y) lie on the increasing diagonal line through this cell, and if $M_{x,y} = -1$, then the points in the cell (x, y) lie on the decreasing diagonal line through this cell. This definition is illustrated in the bottom part of Figure 6.8. Similarly to before, we define $\text{Geo}_n(M) := \text{Geo}(M) \cap S_n$. We clearly have $\text{Geo}(M) \subseteq \text{Grid}(M)$ and $\text{Geo}_n(M) \subseteq \text{Grid}_n(M)$.

Unlike for monotone grid classes, it was shown in [5] that any geometric grid class $\text{Geo}(M)$ is characterized by finitely many forbidden patterns, i.e.,

$$\text{Geo}_n(M) = S_n(\tau_1 \wedge \cdots \wedge \tau_\ell)$$

for a suitable set of patterns τ_1, \dots, τ_ℓ and for all $n \geq 0$. For instance, X-shaped permutations studied in [169, 69] are exactly the permutations in $S_n(2143 \wedge 2413 \wedge 3142 \wedge 3412)$. However, the argument given in [5] for the existence of τ_1, \dots, τ_ℓ is non-constructive, so there is no procedure known to compute these patterns from the matrix M .

Nevertheless, our next theorem provides an easily verifiable sufficient condition for deciding whether $\text{Grid}_n(M)$ and $\text{Geo}_n(M)$ are zigzag languages, based only on two particular entries of M .

Theorem 6.16. *Let M be a matrix with -1 in the top-left corner and $+1$ in the top-right corner. Then $\text{Grid}_n(M)$, $n \geq 0$, and $\text{Geo}_n(M)$, $n \geq 0$, are both hereditary sequences of zigzag languages. Consequently, all of these languages can be generated by Algorithm J.*

From the two monotone and geometric grid classes shown in Figure 6.8, only the left two satisfy the conditions of the theorem.

Proof. We only prove the theorem for monotone grid classes $\text{Grid}_n(M)$. The argument for geometric grid classes $\text{Geo}_n(M)$ is analogous.

We argue by induction on n . Note that $\text{Grid}_0(M) = S_0 = \{\varepsilon\}$ is a zigzag language by definition, so the induction basis is clear. For the induction step let $n \geq 1$. We first show that if $\pi \in \text{Grid}_{n-1}(M)$, then $c_1(\pi), c_n(\pi) \in \text{Grid}_n(M)$. For this argument we use the assumption that the top-left entry of M is -1 , and the top-right entry of M is $+1$, i.e., π can be drawn into a grid so that the points in the top-left cell are decreasing, and the points in the top-right cell are increasing. It follows that we can draw $c_1(\pi)$ on the same grid, by extending the drawing of π so that the new point n is placed to the top-left of all other points. Similarly, we can draw $c_n(\pi)$ on the same grid, by extending the drawing of π so that the new point n is placed to the top-right of all other points.

To complete the induction step, we now show that if $\pi \in \text{Grid}_n(M)$, then $p(\pi) \in \text{Grid}_{n-1}(M)$. As $\pi \in \text{Grid}_n(M)$, we can draw π into a grid respecting the monotonicity conditions described by M . Clearly, removing any entry from π , in particular the largest one, maintains this property, i.e., we can draw $p(\pi)$ on the same grid, showing that $p(\pi) \in \text{Grid}_{n-1}(M)$. This completes the proof. \square

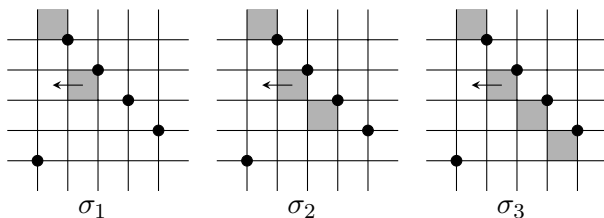
6.4 Limitations of our approach

Let us also briefly motivate the definition of tame permutation patterns given in Section 6.1.2. On the one hand, for a fixed pattern τ , it is certainly reasonable to require that *all* of the sets $S_n(\tau)$, $n \geq 0$, are zigzag languages. This is because for *any* classical pattern $\tau \in S_k$, the set $S_n(\tau) = S_n$ for $n < k$ is trivially a zigzag language, but a very uninteresting one. On the other hand, now that we are con-

cerned with an infinite sequence of sets $S_n(\tau)$, $n \geq 0$, the requirement for the sequence to be hereditary is equally reasonable, as we shall see by considering the problems that arise if we drop this requirement. For this consider the non-tame barred pattern $\tau = 132\bar{4}$. We have $132 \notin S_3(\tau)$, i.e., the permutation 132 contains the pattern, whereas $c_4(132) = 1324 \in S_4(\tau)$ avoids it. Consequently, if we were interested in the language $L_4 = S_4(\tau)$, then the corresponding set $L_3 := p(L_4)$ would be $L_3 = S_3$, and not $S_3(\tau) = S_3 \setminus \{132\}$. In general, when considering $S_n(\tau)$ for a particular value of n , then the sets $L_{i-1} := p(L_i)$ for $i = n, n-1, \dots, 1$ are not necessarily characterized by avoiding the pattern τ . In fact, it is not clear whether they are characterized by any kind of pattern-avoidance. In terms of the tree representation of zigzag languages, if the sequence $S_n(\tau)$, $n \geq 0$, is not hereditary, then different values of n correspond to different tree prunings. In the example from before, the node 132 is pruned from the tree for $S_3(\tau)$, but it is not pruned from the tree for $S_4(\tau)$. In contrast to that, in a hereditary sequence, all sets $S_n(\tau)$, $n \geq 0$, arise from pruning the infinite rooted tree of permutations in a way that is consistent for all n . Summarizing, for patterns τ for which $S_n(\tau)$ is not hereditary, our proof of Theorem 5.1 breaks seriously. Not only that, Algorithm J in general fails to generate $S_n(\tau)$. For instance, it fails to generate $S_4(\tau)$, $\tau = 132\bar{4}$, when initialized with $\text{id}_4 = 1234$, while it succeeds to generate $S_4(\tau')$ for the non-tame barred pattern $\tau' = \bar{4}132$.

For a classical pattern $\tau \in S_k$ that has the largest value k at the leftmost or rightmost position, we have that $S_k(\tau) = S_k \setminus \{\tau\}$ is not a zigzag language (as τ equals either $c_1(p(\tau))$ or $c_k(p(\tau))$), i.e., τ is not tame. Moreover, in general Algorithm J fails to generate $S_k(\tau)$. For instance, running Algorithm J on $S_3(321)$ gives only three permutations 123, 132, 312, and then the algorithm stops. This is admittedly a very strong limitation of our approach, as many interesting permutation patterns have the largest value at the boundary, such as 321, which gives rise to an important Catalan family $S_n(321)$.

By what we said before, the condition for tameness stated in Lemma 6.4 is not only sufficient, but also necessary. In a similar way, it can be shown that the conditions stated in Lemmas 6.5–6.9 are necessary for tameness. The situation is slightly more complicated for Theorem 6.10: Conditions (i) and (ii) of the theorem are indeed necessary. Specifically, if condition (i) is violated, then $S_k(\sigma) = S_k \setminus \{\tau\}$ is not a zigzag language, and if condition (ii) is violated, then $S_k(\sigma) \neq p(S_{k+1}(\sigma))$, i.e., the hereditary property is violated. However, conditions (iii) and (iv) are not necessary. Consider for instance the patterns $\sigma_1, \sigma_2, \sigma_3$ shown below:



They all satisfy conditions (i), (ii) and (iv), but violate condition (iii) due to the cells (2,3) and (1,3), connected by an arrow in the figures. However, the proof of Theorem 6.10 given in Section 6.1.8 can be modified to show that σ_1 is tame. The idea is to apply the exchange argument that involves a point in a match of the pattern and that is illustrated on the right hand side of Figure 6.3 twice instead of only once. This idea can be iterated, and by applying the exchange argument three or four times, respectively, one can show that σ_2 and σ_3 are tame as well. This kind of reasoning apparently leads to combinatorial chaos, depending on the relative location of points and shaded cells in the pattern, and this prevents us from being able to formulate conditions for a mesh pattern that are necessary and sufficient for tameness. This is not an issue from our point of view, because again, we do not see any interesting families of pattern-avoiding permutations that would satisfy such more complicated conditions but not the conditions stated in Theorem 6.10.

I'll simply pour paint onto the canvas, and that's it... It didn't work out. It is something specific that I want to make. So, I'm not just randomly making a mess with paint, there's usually quite a strong structure in these paintings.

—Herbert Brandl,
*Mountains in the Industrial Park and Other
 Romantic Observations: Ines Mitterer in
 Conversation with Herbert Brandl*

CHAPTER 7

Lattice congruences of the weak order

Except for Section 7.2.7, which has not been published, this chapter is based on [89], which is joint work with Torsten Mütze.

The second application of our framework is the generation of the lattice congruences of the weak order of permutations, which we will define briefly below and in more detail in Section 7.1.2.

For a permutation $\pi \in S_n$, the *inversion set* of π is the set of all decreasing pairs of values of $\pi = a_1 \cdots a_n$, formally

$$\text{inv}(\pi) := \{(a_i, a_j) \mid 1 \leq i < j \leq n \text{ and } a_i > a_j\}.$$

We consider the classical *weak order* on S_n , the poset obtained by ordering all permutations from S_n by containment of their inversion sets, i.e., $\pi < \rho$ for any two permutations π, ρ in the weak order if and only if $\text{inv}(\pi) \subseteq \text{inv}(\rho)$; see the left hand side of Figure 7.1.

Equivalently, the weak order on S_n can be obtained as the poset of regions of the *braid arrangement* of hyperplanes. Also, its Hasse diagram is the graph of the *permutahedron*.

It is well-known that the weak order forms a *lattice*, i.e., joins $\pi \vee \rho$ and meets $\pi \wedge \rho$ are well-defined. A *lattice congruence* is an equivalence relation \equiv on S_n that is compatible with taking joins and meets. Formally, if $\pi \equiv \pi'$ and $\rho \equiv \rho'$ then we also have $\pi \vee \rho \equiv \pi' \vee \rho'$ and $\pi \wedge \rho \equiv \pi' \wedge \rho'$. The *lattice quotient* S_n/\equiv is obtained by taking the equivalence classes as elements, and ordering them by $X < Y$ if and only if there is a representative $\pi \in X$ and a representative $\rho \in Y$ such that $\pi < \rho$ in the weak order; see the right hand side of Figure 7.1. The study of lattice congruences of the weak order has been developed considerably in recent years, in particular thanks to Reading's works, summarized in [143, 146, 147]. All of these results have beautiful ramifications into posets, polytopes, geometry, and combinatorics. In fact, many of these results even hold in the more general setting of arbitrary Coxeter groups and for the poset of regions of general hyperplane arrangements.

It is not hard to see that there are double-exponentially (in n) many distinct lattice congruences of the weak order on S_n , and many important lattices arise as quotients of suitable lattice congruences: the Boolean lattice, the Tamari lattice [157] (shown in Figure 7.1), type A Cambrian lattices [142, 47], permutree lattices [132], the increasing flip lattice on acyclic twists [131], and the rotation lattice on diagonal rectangulations [113, 79, 43].

Additionally, Pilaud and Santos [134] showed how to realize the cover graph of any lattice quotient S_n/\equiv as the graph of an $(n-1)$ -dimensional polytope, and they called these polytopes *quotientopes*. Their results generalize many earlier constructions of polytopes for the aforementioned special lattices [117, 90, 110, 132, 133, 113]. In particular, quotientopes generalize permutahedra, associahedra, and hypercubes. Interestingly, quotientopes are defined by a set of glid-

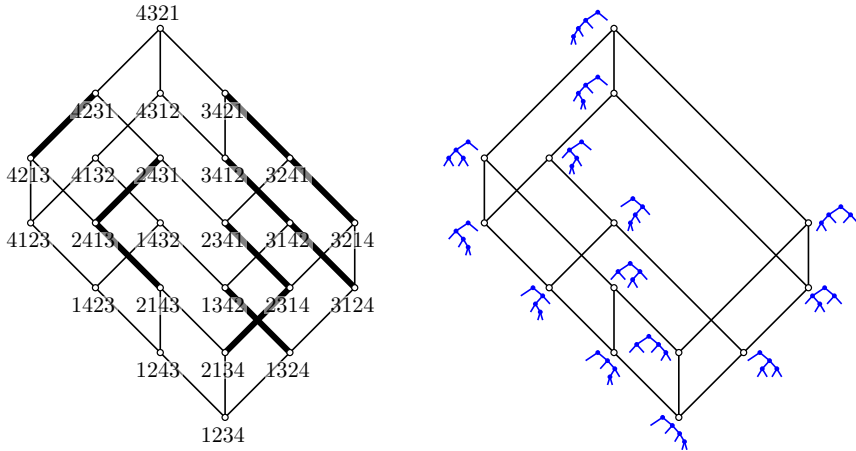


Figure 7.1: Hasse diagrams of the weak order on S_4 (left) with a lattice congruence \equiv (bold edges), and of the resulting lattice quotient S_4/\equiv (right), which is the well-known Tamari lattice (with corresponding binary trees).

ing hyperplanes that is consistent with refining the corresponding lattice congruences, i.e., moving the hyperplanes outwards corresponds to refining the equivalence classes. In particular, the permutahedron contains all other quotientopes, and the hypercube is contained in all (full-dimensional) quotientopes. Figure 7.7 shows all quotientopes for $n = 4$ ordered by refinement of the corresponding congruences, with permutahedron, associahedron, and 3-cube highlighted.

There are several long-standing open problems revolving around Hamilton paths and cycles in graphs of polytopes and other highly symmetric graphs, most prominently Barnette's and Lovász's conjectures. Barnette's conjecture [22] asserts that the graph of every simple three-dimensional polytope with an even number of edges on each face has a Hamilton cycle. Another variant of the con-

ture states that the graphs of all simple three-dimensional polytopes with face sizes at most 6, in particular all fullerenes, have a Hamilton cycle [6]. Barnette also conjectured that the graph of every simple 4-dimensional polytope has a Hamilton cycle [82, p. 1145]. Note that the simplicity of these polytopes means that their graphs are 3-regular or 4-regular, respectively. Lovász' conjecture [119] asserts that every vertex-transitive graph has a Hamilton path. A stronger form of his conjecture asserts that such graphs even have a Hamilton cycle, with five well-understood exceptions, among them the Petersen graph and the Coxeter graph.

In this chapter, we apply the general framework for exhaustive generation to the lattice quotients of the weak order on the symmetric group S_n , yielding a Hamilton path on their cover graphs (Theorem 7.12). As a consequence, since these cover graphs are also the graphs of the quotientopes introduced by Pilaud and Santos, the graph of every quotientope has a Hamilton path (Corollary 7.13); see Figure 7.7. For the permutahedron, associahedron, and hypercube, algorithmic constructions of such Hamilton paths were already known by the Steinhaus-Johnson-Trotter algorithm [162, 99], by the Lucas-Roelants van Baronaigien-Ruskey tree rotation algorithm [121] (see also [120, 96]), and by the binary reflected Gray code [81], respectively. Our results thus unify and generalize all these classical algorithms. Motivated by our Hamiltonicity results and by Barnette's and Lovász' conjectures, we also characterize which lattice congruences of the weak order on S_n yield regular or vertex-transitive quotientopes. This characterization uses arc diagrams introduced by Reading [145], and allows us to derive corresponding precise and asymptotic counting results. We also determine the minimum and maximum degrees and two-colorability of quotientopes. All of these results are summarized in Table 7.9 (theorems are referenced in the table). In those results, Catalan numbers, integer compositions and partitions, and the Erdős-Szekeres theorem make their appearance. As a last result, we formulate conditions

under which a set of pattern-avoiding permutations can be realized as a lattice congruence of the weak order on S_n (Theorem 7.47).

7.1 Generating lattice congruences of the weak order

In this section, we show how Algorithm J can be used to generate any lattice congruence of the weak order on S_n . The main results of this section are summarized in Theorem 7.12 and Corollary 7.13 below.

7.1.1 Modified zigzag languages

Before discussing lattice congruences, we first mention a slight modification of the definition of zigzag languages in Section 5.3. The following extension of this definition is necessary in order to handle the lattice congruences in full generality. Specifically, a set of permutations $L_n \subseteq S_n$ is called a *zigzag language*, if either $n = 0$ and $L_0 = \{\varepsilon\}$, or if $n \geq 1$ and $L_{n-1} := \{p(\pi) \mid \pi \in L_n\}$ is a zigzag language satisfying either one of the following conditions:

- (z) For every $\pi \in L_{n-1}$ we have $c_1(\pi) \in L_n$ and $c_n(\pi) \in L_n$.
- (\hat{z}) We have $L_n = \{c_n(\pi) \mid \pi \in L_{n-1}\}$.

The new condition (\hat{z}) expresses that L_n is obtained from L_{n-1} simply by inserting the new largest value n at the rightmost position of all permutations, i.e., the value n only ever appears to the right of $1, \dots, n-1$. In this case we will have in particular $|L_n| = |L_{n-1}|$.

Note that all results from Chapter 5 carry over straightforwardly. In particular, Theorem 5.1 still holds. Indeed, recall that in the proof of the theorem in Section 5.3.3, we defined a sequence $J(L_n)$ of all permutations from a zigzag language L_n and showed by induction that Algorithm J generates the permutations of L_n exactly in this

order. In the proof, we defined $\vec{c}(\pi)$ to be the sequence of all $c_i(\pi) \in L_n$ for $i = 1, 2, \dots, n$, starting with $c_1(\pi)$ and ending with $c_n(\pi)$, and $\overleftarrow{c}(\pi)$ to be the reverse sequence, i.e., it starts with $c_n(\pi)$ and ends with $c_1(\pi)$. We now modify the recursive definition of the sequence $J(L_n)$ as follows: If $n = 0$ then we define $J(L_0) := \varepsilon$, and if $n \geq 1$ then we consider the sequence $J(L_{n-1}) =: \pi_1, \pi_2, \dots$ and define

$$J(L_n) = \overleftarrow{c}(\pi_1), \vec{c}(\pi_2), \overleftarrow{c}(\pi_3), \vec{c}(\pi_4), \dots \quad (7.1a)$$

if condition (z) holds, and we define

$$J(L_n) = c_n(\pi_1), c_n(\pi_2), c_n(\pi_3), c_n(\pi_4), \dots \quad (7.1b)$$

if condition (\hat{z}) holds. The inductive proof then continues almost the same as in Section 5.3.3. The only additional argument is that when condition (\hat{z}) holds, the element n cannot jump at all, and hence the generation of sequence $J(L_n)$ by Algorithm J follows the inductive hypothesis for $J(L_{n-1})$.

7.1.2 Preliminaries

We begin to recall a few basic definitions for a poset $(P, <)$. An *antichain* in P is a set of pairwise incomparable elements. A subset $U \subseteq P$ is an *upset* if $x \in U$ and $x < y$ implies that $y \in U$. Similarly, $D \subseteq P$ is a *downset* if $x \in D$ and $y < x$ implies that $y \in D$. Clearly, the complement of an upset is a downset and vice versa. Moreover, the minimal elements of an upset and the maximal elements of a downset form an antichain. The *upset of an element* $x \in P$ is the upset containing exactly all y with $x < y$. Similarly, the *downset of* x is the downset containing exactly all y with $y < x$. An *interval* $X = [x, y]$ in P is the intersection of the upset of x with the downset of y , and we write $x = \min(X)$ and $y = \max(X)$.

A *cover relation* is a pair $x, y \in P$ with $x < y$ for which there is no $z \in P$ with $x < z < y$. In this case we say that y *covers* x and

we write $x \triangleleft y$. We also refer to x as a *down-neighbor* of y , and to y as an *up-neighbor* of x . Clearly, the cover relations form an acyclic directed graph with vertex set P , and this graph is referred to as the *cover graph* of P , and its edges as *cover edges*. A drawing of the cover graph with all cover edges $x \triangleleft y$ leading upwards is called a *Hasse diagram*. A poset $(P, <)$ is called a *lattice*, if for any two $x, y \in P$ there is a unique smallest element z , called the *join* $x \vee y$ of x and y , such that $z > x$ and $z > y$, and if there is a unique largest element z , called the *meet* $x \wedge y$ of x and y , satisfying $z < x$ and $z < y$. A *lattice congruence* is an equivalence relation \equiv on P such that $x \equiv x'$ and $y \equiv y'$ implies that $x \vee y \equiv x' \vee y'$ and $x \wedge y \equiv x' \wedge y'$. Given any lattice congruence \equiv , we obtain the *lattice quotient* P/\equiv (which is itself a lattice) by taking the equivalence classes as elements, and ordering them by $X < Y$ if and only if there is an $x \in X$ and a $y \in Y$ such that $x < y$ in P . Observe that the cover graph of P/\equiv is obtained from the cover graph of P by contracting all cover edges $x \triangleleft y$ with $x \equiv y$. For any $x \in P$, we let $X_P(x) = X(x)$ denote the equivalence class in P/\equiv containing x .

We will need the following two lemmata.

Lemma 7.1. *For any lattice congruence of a finite lattice, every equivalence class is an interval.*

Lemma 7.2. *Given a finite lattice $(P, <)$ and any lattice congruence \equiv , the lattice quotient P/\equiv is isomorphic to the induced subposet of P whose elements are either the minima of the equivalence classes or the maxima.*

Lemma 7.1 follows immediately from the definition of lattice congruence. Lemma 7.2 has appeared in many previous papers, see e.g. [109, 63, 45, 139]. It can be proved by showing that given two equivalence classes X and Y of \equiv and two elements $x \in X$, $y \in Y$ with $x \triangleleft y$, then we have $\min(X) < \min(Y)$ and $\max(X) < \max(Y)$.

Recall that the weak order on S_n is the order given by inclusion of inversion sets. Note that the cover relations in this poset are exactly adjacent transpositions, i.e., swaps of two entries at neighboring positions in the permutation. Observe also that the inversion set of the join $\pi \vee \rho$ of two permutations π and ρ is given by the transitive closure of $\text{inv}(\pi) \cup \text{inv}(\rho)$, and the inversion set of the meet can be computed similarly by considering the reverse permutations (which have the complementary inversion set). In the weak order on S_n , if two permutations π and ρ differ by transposing a and b , then we refer to the corresponding cover edge as an (a, b) -*edge*, and if $\pi \equiv \rho$ then we refer to it as an (a, b) -*bar*. Bars are drawn with bold edges in all our figures. The cover edges involving a fixed permutation $\pi = a_1 \cdots a_n$ can be described more precisely by considering all *ascents* of π , i.e., all pairs (a_i, a_{i+1}) with $a_i < a_{i+1}$ and all *descents* of π , i.e., all pairs (a_i, a_{i+1}) with $a_i > a_{i+1}$. Specifically, for fixed π , all cover edges $\pi < \rho$ are given by transposing the ascents of π , and all cover edges $\pi > \rho$ are given by transposing the descents of π . We let $\text{asc}(\pi)$ and $\text{desc}(\pi)$ denote the number of ascents and descents of π , respectively.

7.1.3 Combinatorics of lattice congruences of the weak order

In the following discussion of lattice congruences of the weak order, we borrow some of the terminology and notation introduced by Reading [140, 145]; see also his surveys [143, 146, 147].

It is clear from the definition of lattice congruence, that if certain permutations are equivalent, this also forces other permutations to be equivalent. These relations on the cover edges are expressed by *forcing constraints*. The two forcing constraints that are relevant for us are shown in Figure 7.2. We refer to them as type i and type ii constraints, shown on the left and right of the figure, respectively. A type i constraint involves four permutations π, ρ, π', ρ' satisfying

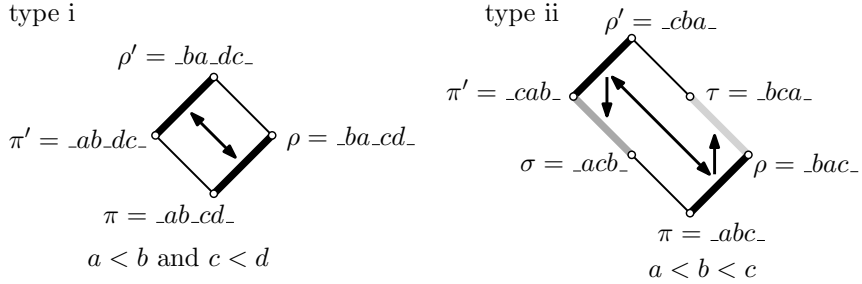


Figure 7.2: Forcing constraints in a lattice congruence of the weak order. Bold edges indicate bars, i.e., pairs of permutations that differ in an adjacent transposition and that belong to the same equivalence class.

$\pi \leq \rho \leq \rho'$ and $\pi \leq \pi' \leq \rho'$ that differ in adjacent transpositions of two values a, b or two values c, d with $a < b$ and $c < d$, as shown in the figure. This constraint expresses that $\pi \equiv \rho$ if and only if $\pi' \equiv \rho'$, i.e., either both (a, b) -edges (π, ρ) and (π', ρ') are bars or both are non-bars. A type ii constraint involves six permutations $\pi, \rho, \pi', \rho', \sigma, \tau$ satisfying $\pi \leq \rho \leq \tau \leq \rho'$ and $\pi \leq \sigma \leq \pi' \leq \rho'$ that differ in three adjacent values a, b, c with $a < b < c$, as shown in the figure. This constraint expresses that $\pi \equiv \rho$ if and only if $\pi' \equiv \rho'$, and moreover these conditions imply $\sigma \equiv \pi'$ and $\tau \equiv \rho$ (but not the converse), i.e., the first two (a, b) -edges are both either bars or non-bars, and in the first case they also force the latter two (a, c) -edges to be bars. Note that both constraints follow immediately from the definition of lattice congruence, and that they are meant to capture also the symmetric situation obtained by reversing all permutations involved in Figure 7.2.

We now consider maximal sets of cover edges that are either all bars or all non-bars in any lattice congruence. Given an (a, b) -bar, then type i constraints allow us to reorder the values to the left or right of a and b in the corresponding permutations arbitrarily. Moreover,

given an (a, b) -bar, then type ii constraints allow us to move any value that is larger or smaller than a and b to the left or right of them. Consequently, a maximal set of mutually forcing bars is characterized by the pair (a, b) , and by the values that are strictly between a and b and to the left of them. This motivates the following definition: Given a triple (a, b, L) with $1 \leq a < b \leq n$ and $L \subseteq]a, b[$, the *fence* $f(a, b, L)$ is the set of all (a, b) -edges, where the values in L are to the left of a and b in the corresponding permutations, the values in $]a, b[\setminus L$ are to the right of a and b , and the position of the remaining values $[n] \setminus [a, b]$ is arbitrary. Note that the edges of any fence form a matching in the cover graph. For instance, for $n = 4$ the fence $f(2, 4, \{3\})$ contains the $(2, 4)$ -edges $(3241, 3421)$, $(1324, 1342)$, and $(3124, 3142)$ that are mutually forcing bars; see Figure 7.3. In the figure, we visualize fences by an *arc diagram*, which consists of a vertical sequence of n points labeled $1, \dots, n$ from bottom to top, and for every fence $f(a, b, L)$ there is an arc joining the a th and b th point, with the points in L left of the arc, and the points in $]a, b[\setminus L$ right of the arc. We let

$$F_n := \{f(a, b, L) \mid 1 \leq a < b \leq n \text{ and } L \subseteq]a, b[\}$$

denote the set of all fences.

The (non-mutual) forcing constraints between fences induced by type ii constraints yield a partial order on F_n , called the *forcing order*. Specifically, two fences $f(a, b, L)$ and $f(c, d, M)$ satisfy $f(a, b, L) \prec f(c, d, M)$ in the forcing order, if $a \leq c < d \leq b$, $(a, b) \neq (c, d)$, and $M = L \cap]c, d[$. Note that two such fences form a cover relation in the forcing order if and only if $(c, d) = (a + 1, b)$ or $(c, d) = (a, b - 1)$. Consequently, every non-maximal fence $f(a, b, L)$ is covered by two other fences, and every non-minimal fence $f(a, b, L)$ covers two fences if either $a = 0$ or $b = n$, and four fences if $0 < a < b < n$. The interpretation is that if $f(a, b, L) \prec f(c, d, M)$, then the bars of the fence $f(c, d, M)$ force the bars of the fence $f(a, b, L)$, i.e., forcing goes downward in the forcing order. For example, we

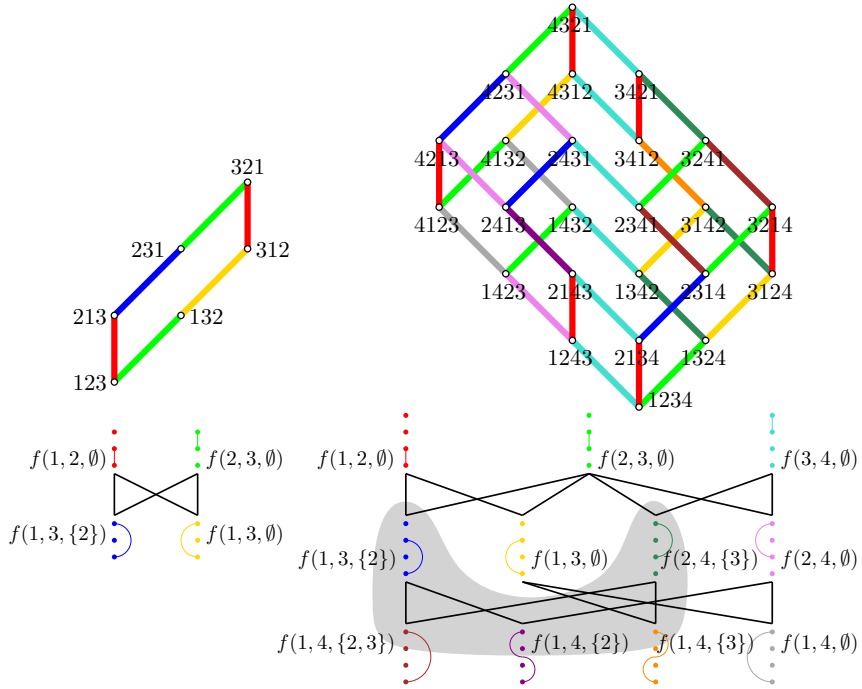


Figure 7.3: Illustration of fences and the forcing order for $n = 3$ (left) and $n = 4$ (right). Cover edges of the same fence are drawn in the same color. The highlighted region shows a downset in the forcing order, corresponding to the lattice quotient in Figure 7.1.

have $f(1, 4, \{2, 3\}) \prec f(2, 4, \{3\})$, i.e., the three bars $(3241, 3421)$, $(1324, 1342)$, and $(3124, 3142)$ from before force the two bars $(2314, 2341)$ and $(3214, 3241)$.

Theorem 7.3 ([146, Section 10-5]). *For every lattice congruence \equiv of the weak order on S_n , there is a subset of fences $F_{\equiv} \subseteq F_n$ such that in each equivalence class of \equiv , all cover edges are a bar from a fence in F_{\equiv} , and all other cover edges are not in any fence from F_{\equiv} .*

Moreover, F_{\equiv} is a downset of the forcing order \prec and the map $\equiv \mapsto F_{\equiv}$ is a bijection between the lattice congruences of the weak order on S_n and the downsets of the forcing order \prec .

From now on we use F_{\equiv} as the set of fences corresponding to a lattice congruence \equiv given by Theorem 7.3. The downset F_{\equiv} describes exactly all the cover edges that are contracted to obtain the lattice quotient S_n/\equiv . Equivalently, the upset $F_n \setminus F_{\equiv}$ describes all cover edges that are not contracted in the quotient.

In the dual setting of hyperplane arrangements considered in [140, 134], the dual of a fence is called a shard. Moreover, these authors represent a lattice congruence \equiv not by the set of fences F_{\equiv} that contains all cover edges that are contracted to obtain the lattice quotient S_n/\equiv , but by the set $F_n \setminus F_{\equiv}$ of cover edges that are not contracted in the quotient. The latter representation allows describing each equivalence class by a non-crossing arc diagram that contains only arcs corresponding to fences from $F_n \setminus F_{\equiv}$ [145]. On the other hand, our representation makes the characterization of congruences with regular and vertex-transitive quotient graphs in Section 7.2 somewhat more natural.

We may order all downsets of the forcing order by inclusion, yielding another lattice; see Figure 7.4. By Theorem 7.3, this corresponds to ordering all lattice congruences of the weak order on S_n by refinement. The finest lattice congruence \equiv does not use any fences $F_{\equiv} = \emptyset$, and corresponds to the set of all permutations S_n , and the coarsest lattice congruence \equiv uses all fences $F_{\equiv} = F_n$, and corresponds to contracting all permutations into a single equivalence class.

7.1.4 Restrictions, rails, ladders, and projections

Given a lattice congruence \equiv of the weak order on S_n , the *restriction of \equiv* , denoted \equiv^* , is the relation on S_{n-1} induced by all permu-

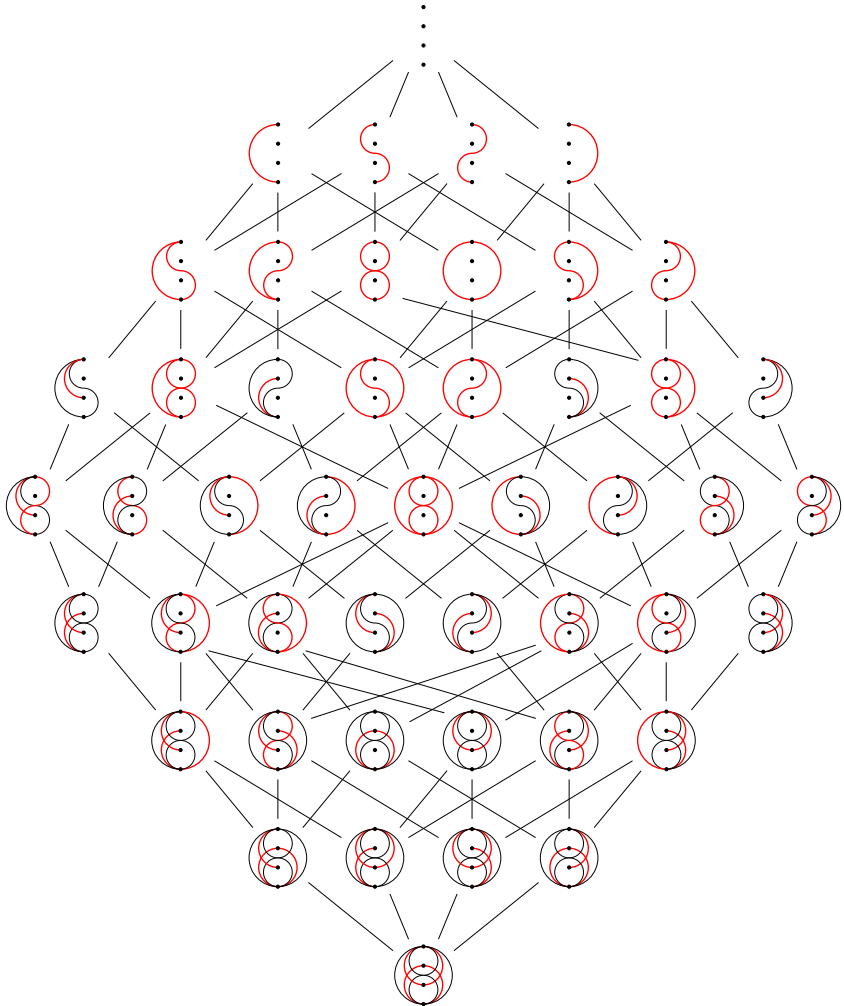


Figure 7.4: Lattice of congruences of the weak order on S_n for $n = 4$, represented by downsets of the forcing order. Each downset of fences is represented by an arc diagram containing all the corresponding arcs, where the arcs corresponding to maximal fences of the downset are highlighted. The figure shows only downsets containing no non-essential fences $f(a, a+1, \emptyset)$, $a \in [n-1]$, as otherwise the congruence is equivalent to a lower-dimensional one (see Lemma 7.16 below).

tations that have the largest value n at the last position, i.e., it is the set of all pairs (π, ρ) with $\pi, \rho \in S_{n-1}$ for which $c_n(\pi) \equiv c_n(\rho)$.

Lemma 7.4. *For every lattice congruence \equiv of the weak order on S_n , the restriction \equiv^* is a lattice congruence on S_{n-1} .*

Proof. Clearly, for any two permutations $\pi, \rho \in S_{n-1}$ we have

$$c_n(\pi) \vee c_n(\rho) = c_n(\pi \vee \rho) \quad \text{and} \quad c_n(\pi) \wedge c_n(\rho) = c_n(\pi \wedge \rho). \quad (7.2)$$

Now consider four permutations $\pi, \pi', \rho, \rho' \in S_{n-1}$ satisfying $\pi \equiv^* \pi'$ and $\rho \equiv^* \rho'$. From the definition of restriction, we have $c_n(\pi) \equiv c_n(\pi')$ and $c_n(\rho) \equiv c_n(\rho')$. Applying the definition of lattice congruence to \equiv , we obtain that $c_n(\pi) \vee c_n(\rho) \equiv c_n(\pi') \vee c_n(\rho')$ and $c_n(\pi) \wedge c_n(\rho) \equiv c_n(\pi') \wedge c_n(\rho')$. Applying (7.2) to these relations yields $c_n(\pi \vee \rho) \equiv c_n(\pi' \vee \rho')$ and $c_n(\pi \wedge \rho) \equiv c_n(\pi' \wedge \rho')$, from which we obtain $\pi \vee \rho \equiv^* \pi' \vee \rho'$ and $\pi \wedge \rho \equiv^* \pi' \wedge \rho'$ with the definition of restriction. This proves the lemma. \square

The following definitions are illustrated in Figure 7.5. Recall that for any permutation $\pi \in S_{n-1}$ and for any $1 \leq i \leq n$, the permutation $c_i(n)$ is obtained from π by inserting the largest value n at position i . Given any permutation $\pi \in S_{n-1}$, we refer to the cover edges $c_n(\pi) \triangleleft c_{n-1}(\pi) \triangleleft \cdots \triangleleft c_1(\pi)$ in S_n as the *rail* $r(\pi)$. Given two permutations $\pi, \rho \in S_{n-1}$ with $\pi \triangleleft \rho$, we refer to the cover edges of the weak order induced by the permutations on the rails of π and ρ as the *ladder* $\ell(\pi, \rho)$. Let k and $k+1$ be the positions in which π and ρ differ. Note that the ladder $\ell(\pi, \rho)$ has exactly all cover edges of the rails, plus the cover edges $c_i(\pi) \triangleleft c_i(\rho)$ for all $1 \leq i \leq n$ except for $i = k+1$, which are referred to as the *stairs* of the ladder. We see that the cover graph of the weak order on S_n has the following recursive structure: It is the union of all ladders $\ell(\pi, \rho)$ obtained from all cover edges $\pi \triangleleft \rho$ with $\pi, \rho \in S_{n-1}$.

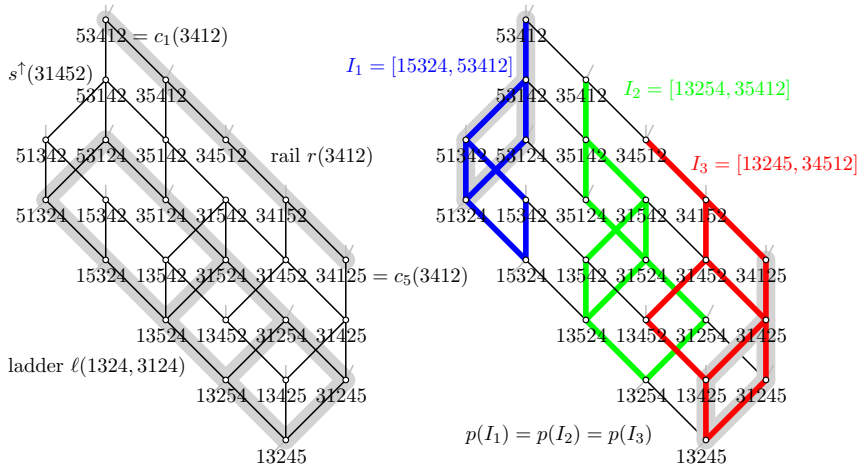


Figure 7.5: Illustration of rails, ladders, and projections. The figure shows only a subset of permutations from S_5 . The equivalence classes on the right are shown as intervals.

Lemma 7.5. *For every lattice congruence \equiv of the weak order on S_n , the following three statements are equivalent:*

- (i) $\text{id}_n \equiv c_{n-1}(\text{id}_{n-1})$, i.e., the identity permutation and the one obtained from it by transposing the last two entries form a bar.
- (ii) There is a permutation $\pi \in S_{n-1}$ such that for all $1 \leq i < n$ we have $c_i(\pi) \equiv c_{i+1}(\pi)$, i.e., the rail $r(\pi)$ consists entirely of bars.
- (iii) For all permutations $\pi \in S_{n-1}$ and all $1 \leq i < n$ we have $c_i(\pi) \equiv c_{i+1}(\pi)$, i.e., all rails $r(\pi)$ consist entirely of bars.

Proof. Clearly, (iii) implies (ii) and (iii) implies (i), so it suffices to prove that (ii) implies (iii) and that (i) implies (iii). We prove this by showing that if there is an $(n-1, n)$ -bar in S_n , then (iii) follows. If there is an $(n-1, n)$ -bar, this means that the fence $f(n-1, n, \emptyset)$ is in F_{\equiv} . However, as F_{\equiv} is a downset in the forcing order (recall Theorem 7.3), it follows that all fences $f(a, n, L)$ with

$1 \leq a \leq n-1$ and an arbitrary subset $L \subseteq]a, n[$ are also in F_{\equiv} . For any $\pi = a_1 \cdots a_{n-1} \in S_{n-1}$, the i th edge along the rail $r(\pi) = c_n(\pi) \triangleleft c_{n-1}(\pi) \triangleleft \cdots \triangleleft c_1(\pi)$ is an (a_{n-i}, n) -edge, so it is a bar regardless of the values of $a_1 \cdots a_{n-i}$. \square

Combining Lemmata 7.1 and 7.5 yields the following lemma.

Lemma 7.6. *Let \equiv be an equivalence relation of the weak order on S_n with $\text{id}_n \not\equiv c_{n-1}(\text{id}_{n-1})$. Then for every rail $r(\pi)$, $\pi \in S_{n-1}$, and every equivalence class $X \in S_n/\equiv$ we have that $X \cap r(\pi)$ is an interval of $r(\pi)$. Moreover, there are two distinct equivalence classes X and Y containing the first and last permutation of the rail, i.e., $c_n(\pi) \in X$ and $c_1(\pi) \in Y$.*

Recall that for $\pi \in S_n$, the permutation $p(\pi) \in S_{n-1}$ is obtained by removing the largest value n from π . Given a set of permutations $X \subseteq S_n$, we refer to $p(X) := \{p(\pi) \mid \pi \in X\}$ as the *projection* of X . This definition and the following crucial lemma are illustrated in Figure 7.5.

Lemma 7.7. *For every lattice congruence \equiv of the weak order on S_n and every equivalence class X of \equiv , we have that the projection $p(X)$ is an equivalence class of the restriction \equiv^* . In particular, any two equivalence classes X, Y of S_n/\equiv either have the same projection $p(X) = p(Y)$ or disjoint projections $p(X) \cap p(Y) = \emptyset$.*

The proof of this lemma essentially proceeds by repeatedly applying the forcing constraints shown in Figure 7.2 along ladders. However, we do not apply these constraints directly, but using the fences captured by Theorem 7.3.

Proof. For any $n \geq 1$ and any permutation $\pi \in S_n$, we let $N(\pi)$ denote the set of all permutations that differ from π in an adjacent transposition, i.e., all neighbors in the cover graph of S_n . Now consider a fixed lattice congruence \equiv on S_n , fix an equivalence class X

of \equiv and some permutation $\pi \in X$, and consider its projection $\pi' := p(\pi) \in S_{n-1}$. The lemma is a consequence of the following two statements:

- (i) For every $\rho \in N(\pi) \subseteq S_n$ with $\pi \equiv \rho$ we have that $p(\pi) \equiv^* p(\rho)$.
- (ii) For every $\rho' \in N(\pi') \subseteq S_{n-1}$ with $\pi' \equiv^* \rho'$ there is a $\rho \in N(\pi)$ with $\pi \equiv \rho$ and $p(\rho) = \rho'$, or there is a $\sigma \in N(\pi)$ and a $\rho \in N(\sigma)$ with $\pi \equiv \sigma \equiv \rho$ and $p(\pi) = p(\sigma) = \pi'$ and $p(\rho) = \rho'$.

In words, (i) asserts that the projection of any bar incident to π is a bar incident to π' in the restriction, and (ii) asserts that for any bar incident to π' in the restriction, there are one or two consecutive bars starting at π whose projection is this bar.

We begin proving (i). Let $\rho \in N(\pi) \subseteq S_n$ with $\pi \equiv \rho$. If π and ρ are endpoints of an (a, n) -bar for some $a < n$ (i.e., this bar is part of the rail $r(\pi')$), then we have $p(\pi) = p(\rho)$, so trivially $p(\pi) \equiv^* p(\rho)$. Otherwise π and ρ are endpoints of some (a, b) -bar for $a < b < n$ (i.e., this bar is a stair of some ladder), so the fence $f(a, b, L)$ is in F_{\equiv} , where L is the set of all values from $]a, b[$ left of a and b in π and ρ . By the definition of a fence, it follows that $c_n(p(\pi)) \equiv c_n(p(\rho))$, i.e., the permutations obtained from π and ρ by moving the largest value n to the rightmost position are equivalent. By the definition of restriction, we obtain that $p(\pi) \equiv^* p(\rho)$, as claimed.

We now prove (ii). Let $\rho' \in N(\pi') \subseteq S_{n-1}$ with $\pi' \equiv^* \rho'$. Clearly, π' and ρ' are endpoints of some (a, b) -bar in \equiv^* for $a < b \leq n-1$. By the definition of restriction, it follows that $c_n(\pi') \equiv c_n(\rho')$, so $f(a, b, L)$ is a fence in F_{\equiv} , where L is the set of all values from $]a, b[$ left of a and b in π' and ρ' . In the following we assume that $\pi' \leq \rho'$, i.e., π' contains the ascent (a, b) , and ρ' contains the descent (b, a) . Let i be such that $\pi = c_i(\pi')$, and let k be the position of b in π' . We now distinguish two cases. If $i \neq k$, then $\pi = c_i(\pi') \leq c_i(\rho')$ is a cover edge in S_n (it is a stair of the ladder $\ell(\pi', \rho')$), and since it is contained in the fence $f(a, b, L)$, we have $\pi = c_i(\pi') \equiv c_i(\rho')$,

i.e., this cover edge is indeed a bar. This means we can take $\rho := c_i(\rho') \in N(\pi)$, which satisfies $p(\rho) = \rho'$ by definition. On the other hand, if $i = k$, then $c_i(\pi')$ and $c_i(\rho')$ are *not* endpoints of a cover edge (this is the missing stair in the ladder $\ell(\pi', \rho')$). However, we may take $\sigma := c_{i-1}(\pi') \in N(\pi)$ and $\rho := c_{i-1}(\rho') \in N(\sigma)$ (note that $i = k \geq 2$), and then $\pi \prec \sigma$ is an (a, n) -edge, and $\sigma \prec \rho$ is an (a, b) -edge. As $f(a, b, L)$ is a fence in F_{\equiv} , the forcing order implies that $f(a, n, L')$ is also a fence, where L' is defined as the set of all values from $]a, n[$ left of a and n in π and σ . Consequently, we have $\pi \equiv \sigma \equiv \rho$, and moreover $p(\pi) = p(\sigma) = \pi'$ and $p(\rho) = \rho'$ by the definition of σ and ρ , i.e., these two cover edges are indeed bars. In the remaining subcase $\pi' \succ \rho'$ we can take $\rho := c_i(\rho') \in N(\pi)$ if $i \neq k$, and $\sigma := c_{i+1}(\pi')$ and $\rho := c_{i+1}(\rho')$ if $i = k$, and argue similarly to before.

This proves the lemma. □

We state the following two lemmata for further reference. The first lemma is an immediate consequence of Lemma 7.7. For any lattice congruence \equiv of the weak order on S_n and any fence $f(a, b, L)$ in F_{\equiv} with $b < n$, we let $f^*(a, b, L)$ denote the fence formed by the union of all (a, b) -edges in the weak order on S_{n-1} obtained by removing the largest value n from all permutations of $f(a, b, L)$.

Lemma 7.8. *For every lattice congruence \equiv of the weak order on S_n , its restriction \equiv^* satisfies $F_{\equiv^*} = \{f^*(a, b, L) \mid f(a, b, L) \in F_{\equiv} \text{ and } b < n\}$.*

Rephrased in terms of arc diagrams, Lemma 7.8 asserts that the arc diagram of the restriction \equiv^* is obtained from the arc diagram of \equiv simply by removing the highest point labeled n , and by deleting all arcs incident to it.

Lemma 7.9. *For every lattice congruence \equiv of the weak order*

on S_n and any equivalence class $X \in S_n/\equiv$, consider its minimum $\pi := \min(X)$ and maximum $\rho := \max(X)$. Then their projections $p(\pi)$ and $p(\rho)$ are the minimum and maximum of the equivalence class $p(X)$ of the restriction \equiv^* .

Proof. Suppose for the sake of contradiction that the maximum of $p(X)$ is not $p(\rho)$, but another permutation $\sigma \in S_{n-1}$. As $\sigma \in p(X)$, we obtain from Lemma 7.7 that $c_i(\sigma) \in X$ for some $1 \leq i \leq n$. As σ is the unique maximum of $p(X)$ (recall Lemma 7.1), there exist two entries a, b with $a < b$ that are inverted in σ , i.e., b appears before a in σ , but not in $p(\rho)$. As inserting n into a permutation does not change the relative order of a and b , the entries a, b are also inverted in $c_i(\sigma)$, but not in ρ . However, by the definition of the weak order on S_n , this means that $c_i(\sigma) \not\leq \rho$, contradicting the fact that ρ is the maximum of X . A similar argument shows that $p(\pi)$ is the minimum of $p(X)$. \square

7.1.5 Jumping through lattice congruences

For any lattice congruence \equiv of the weak order on S_n , a set of representatives for the equivalence classes S_n/\equiv is a subset $R_n \subseteq S_n$ such that for every equivalence class $X \in S_n/\equiv$, exactly one permutation is contained in R_n , i.e., $|X \cap R_n| = 1$. Recall that $X(\pi)$, $\pi \in S_n$, denotes the equivalence class from S_n/\equiv containing π . A meaningful definition of ‘generating the lattice congruence’ is to generate a set of representatives for its equivalence classes. We also require that any two successive representatives form a cover relation in the lattice quotient S_n/\equiv . This is what we achieve with the help of Algorithm J.

We recursively define such a set of representatives R_n as follows; see Figure 7.6: If $n = 0$ then $R_0 := \{\varepsilon\}$, and if $n \geq 1$ then we first compute the representatives R_{n-1} for the restriction \equiv^* to S_{n-1} , and we then distinguish two cases: If $\text{id}_n \not\equiv c_{n-1}(\text{id}_{n-1})$, then we consider every representative $\pi \in R_{n-1}$, the corresponding rail $r(\pi)$ in S_n ,

and from every equivalence class $X \in S_n/\equiv$ with $X \cap r(\pi) \neq \emptyset$ we pick exactly one permutation from $X \cap r(\pi)$. In particular, we always pick $c_1(\pi)$ and $c_n(\pi)$, which is possible by Lemma 7.6, yielding a set R_π . We then take the union of those permutations,

$$R_n := \bigcup_{\pi \in R_{n-1}} R_\pi. \quad (7.3a)$$

On the other hand, if $\text{id}_n \equiv c_{n-1}(\text{id}_{n-1})$ we define

$$R_n := \{c_n(\pi) \mid \pi \in R_{n-1}\}. \quad (7.3b)$$

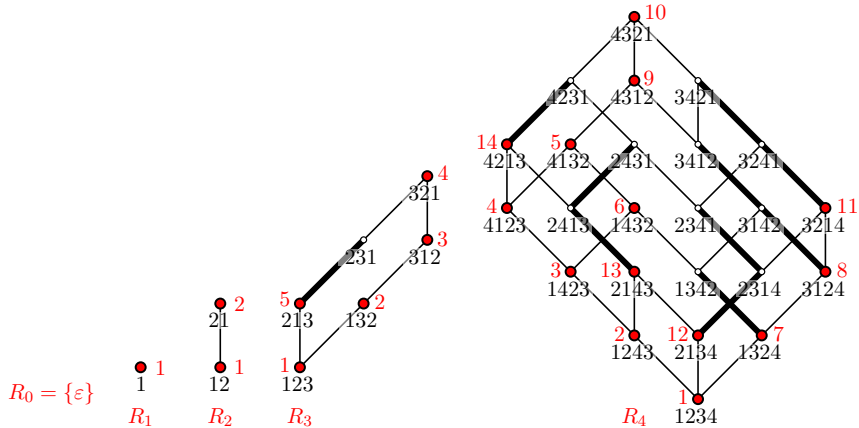


Figure 7.6: Illustration of the representatives and the jumping order for the lattice congruence shown in Figure 7.1. The filled dots are the permutations in the sets R_n , and the small numbers next to them indicate the ordering in the sequences $J(R_n)$ defined in (7.1).

Lemma 7.10. *For every lattice congruence \equiv of the weak order on S_n , the set $R_n \subseteq S_n$ defined in (7.3) is indeed a set of representatives for S_n/\equiv . Moreover, R_n is a zigzag language satisfying condition (z) if (7.3a) holds, and condition (\hat{z}) if (7.3b) holds.*

Proof. We argue by induction on n . The statement clearly holds for $n = 0$. For the induction step, suppose that R_{n-1} is a set of representatives for the equivalence classes of S_{n-1}/\equiv^* , and that R_{n-1} is a zigzag language. If $\text{id}_n \not\equiv c_{n-1}(\text{id}_{n-1})$, we obtain from Lemma 7.7 that for every equivalence class X of S_n/\equiv , the projection $p(X)$ is an equivalence class of the restriction \equiv^* . Therefore, we know by induction that R_{n-1} contains a unique representative $\pi \in S_{n-1}$ for $p(X)$, so by our choice of R_π we indeed have $|X \cap R_\pi| = 1$, and moreover R_n as defined in (7.3a) satisfies $|X \cap R_n| = 1$. Furthermore, as we chose R_π to contain $c_1(\pi)$ and $c_n(\pi)$ for all $\pi \in S_{n-1}$, we obtain that R_n is a zigzag language satisfying condition (z) in the definition. On the other hand, if $\text{id}_n \equiv c_{n-1}(\text{id}_{n-1})$, then we obtain from Lemma 7.5 and Lemma 7.7 that every equivalence class X of S_n/\equiv satisfies $X = \{c_1(\pi), \dots, c_n(\pi) \mid \pi \in p(X)\}$, showing that R_n as defined in (7.3b) is indeed a set of representatives for S_n/\equiv . Moreover, in this case R_n is a zigzag language satisfying condition (\hat{z}) in the definition. This completes the proof. \square

Lemma 7.11. *Running Algorithm J with input $L_n := R_n$, where R_n is the set of representatives of a lattice congruence \equiv defined in (7.3), then for any two permutations $\pi, \rho \in R_n$ that are visited consecutively, $X(\pi)$ and $X(\rho)$ form a cover relation in the quotient S_n/\equiv .*

Proof. Let R_n be a set of representatives of a lattice congruence \equiv defined in (7.3), and consider the set $L_n := R_n$, which is a zigzag language by Lemma 7.10. If (7.3a) holds, then by Lemma 7.10 the set R_n satisfies condition (z), so the permutations of $L_n = R_n$ are generated in the sequence $J(L_n)$ defined in (7.1a). Observe that all permutations in $\overleftarrow{c}(\pi_k)$ or $\overrightarrow{c}(\pi_k)$, $\pi_k \in R_{n-1} \subseteq S_{n-1}$, lie on the rail $r(\pi_k)$. If $\pi, \rho \in R_n$ are visited consecutively and lie on the same rail, i.e., $\pi = c_i(\pi_k)$ and $\rho = c_j(\pi_k)$ with $1 \leq i < j \leq n$, then there

is an integer s with $i \leq s < j$ such that

$$\pi = c_i(\pi_k) \equiv \dots c_s(\pi_k) \not\equiv c_{s+1}(\pi_k) \equiv c_{s+2}(\pi_k) \equiv \dots \equiv c_j(\pi_k) = \rho,$$

so $X(\pi)$ and $X(\rho)$ form a cover relation in the quotient S_n/\equiv . Moreover, when transitioning from the last permutation of $\overleftarrow{c}(\pi_k)$ to the first permutation of $\overrightarrow{c}(\pi_{k+1})$, or from the last permutation of $\overrightarrow{c}(\pi_{k+1})$ to the first permutation of $\overleftarrow{c}(\pi_{k+2})$, then we move from $c_n(\pi_k)$ to $c_n(\pi_{k+1})$, or from $c_1(\pi_{k+1})$ to $c_1(\pi_{k+2})$, respectively. Consequently, as π_k and π_{k+1} , and also π_{k+1} and π_{k+2} form a cover relation in the weak order on S_{n-1} by induction, we obtain that any two consecutive permutations π, ρ in $J(L_n)$ form a cover relation in the weak order on S_n .

On the other hand, if (7.3b) holds, then by Lemma 7.10 the set R_n satisfies condition (z), so the permutations of $L_n = R_n$ are generated in the sequence $J(L_n)$ defined in (7.1b). In this case, the claim follows immediately by induction. \square

Combining Lemmata 7.10 and 7.11 yields the following theorem.

Theorem 7.12. *For every lattice congruence \equiv of the weak order on S_n , let $R_n \subseteq S_n$ be the set of representatives defined in (7.3). Then Algorithm J generates a sequence $J(R_n) = \pi_1, \pi_2, \dots$ of all permutations from R_n such that $X(\pi_1), X(\pi_2), \dots$ is a Hamilton path in the cover graph of the lattice quotient S_n/\equiv .*

For every lattice congruence \equiv , Pilaud and Santos [134, Corollary 10] defined a polytope, called the *quotientope* for \equiv , whose graph is exactly the cover graph of the lattice quotient S_n/\equiv . These polytopes generalize many known polytopes, such as hypercubes, associahedra, permutahedra etc. The following result is an immediate corollary of Theorem 7.12, and it is illustrated in Figure 7.7.

Corollary 7.13. *For every lattice congruence \equiv of the weak order on S_n , Algorithm J generates a Hamilton path on the graph of the corresponding quotientope.*

Remark 7.14. *Observe that in the definition (7.3a), whenever we encounter an equivalence class $X \in S_n/\equiv$ with $|X \cap r(\pi)| \geq 2$ and $c_1(p(\pi)), c_n(p(\pi)) \notin X$, then we have freedom to pick an arbitrary permutation from $X \cap r(\pi)$ for the set of representatives R_π . By imposing a total order on S_n (e.g., lexicographic order), we can make these choices unique, and this will make the resulting sets of representatives consistent across the entire lattice of congruences ordered by refinement. Specifically, given two equivalence relations \equiv and \equiv' where \equiv is a refinement of \equiv' , computing the representatives R_n and R'_n according to this rule will result in $R_n \supseteq R'_n$. However, the resulting jump ordering $J(R_n)$ may not be a subsequence of $J(R'_n)$, as argued in Remark 5.3. This consistent choice of representative permutations is illustrated in Figure 7.8.*

Remark 7.15. *In Lemma 5.4, we showed that if each of the zigzag languages R_k , $2 \leq k \leq n-1$, has even cardinality, then the ordering of permutations $J(R_n)$ defined by Algorithm J is cyclic. Consequently, if for a given lattice congruence, the number of equivalence classes of each restriction to S_k , $2 \leq k \leq n-1$, is even, then Algorithm J generates a Hamilton cycle on the graph of the corresponding quotientope (the converse does not hold in general, but under the additional assumption $|R_2| < |R_3| < \dots < |R_{n-1}|$). This happens for instance for the permutahedron and for the hypercube, but not for the associahedron, even though the associahedron is known to admit a Hamilton cycle [120, 96]. We are not aware if this condition on the parity of the number of equivalence classes of a lattice congruence can be characterized more easily (e.g., via the arc diagram of the congruence). We will come back to the question about Hamilton cycles in Section 7.4.*

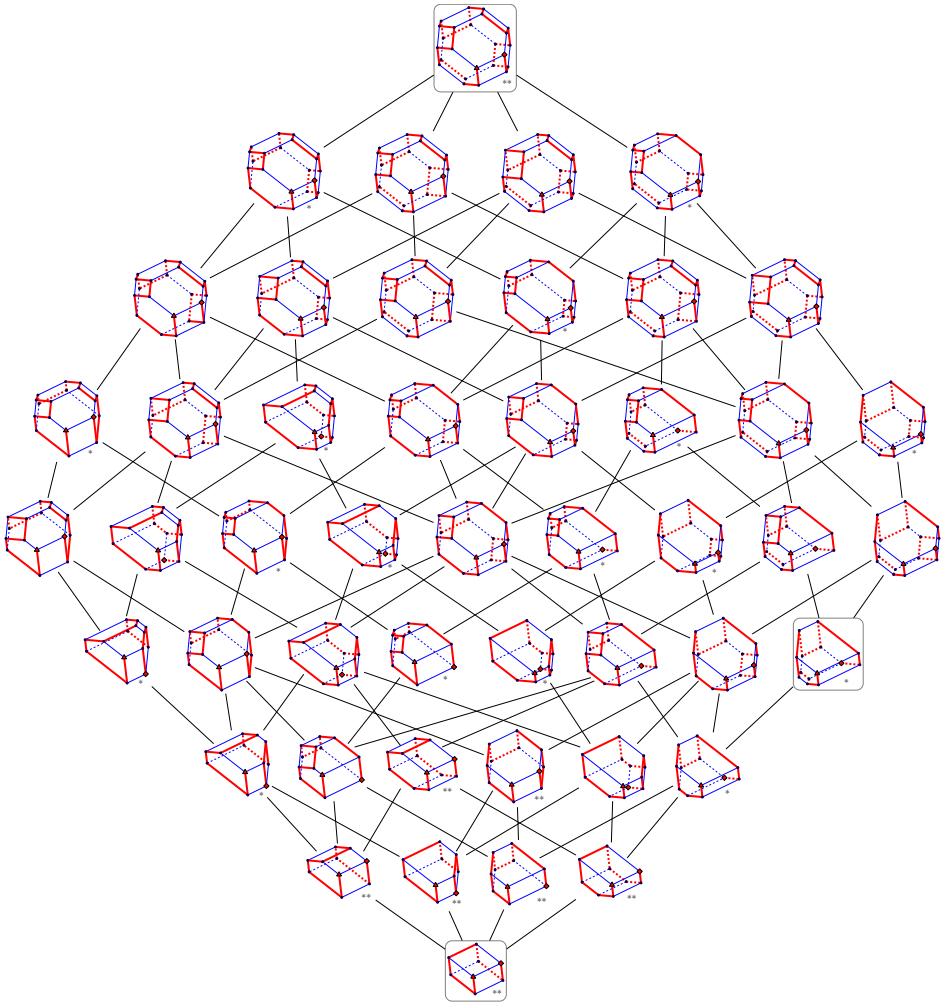


Figure 7.7: Lattice congruences of the weak order on S_4 , ordered by refinement and realized as polytopes, where only the full-dimensional polytopes are shown. The polytopes are arranged in the same way as in Figure 7.4. The figure shows the Hamilton path on each quotientope computed by Algorithm J, with the start and end vertex indicated by a triangle and diamond, respectively. Permutohedron (top), associahedron (one of four isomorphic variants; middle right) and 3-cube (bottom) are highlighted. The graphs marked with * are regular, and those marked with ** are vertex-transitive.

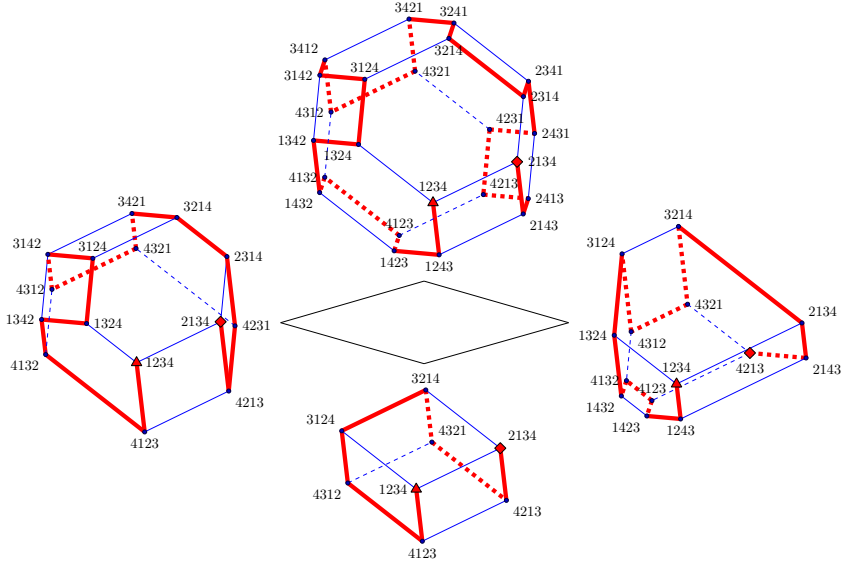


Figure 7.8: Four quotientopes from Figure 7.7 ordered as a diamond. These are the permutahedron (top), the associahedron (right), the 3-cube (bottom), and some other polytope (left). The figure illustrates the consistent choice of representative permutations for the congruence classes, i.e., permutations for lower quotientopes are subsets of permutations for the higher ones.

7.2 Regular, vertex-transitive, and bipartite lattice quotients

In this section, we characterize regular and vertex-transitive quotientopes combinatorially via their arc diagrams, which in particular allows us to count them. We may either consider these objects in terms of the equivalence classes of the lattice congruence, or in terms of the cover graph of the resulting lattice quotient. As several congruences may give the same cover graph, the latter distinction is

coarser, yielding fewer distinct objects. Overall, we obtain six different classes of objects, and Table 7.9 summarizes our results for each of them. The table provides the exact counts for small values of n , various exact and asymptotic counting formulas, as well as references to the theorems where they are established. In the table, we encounter various familiar counting sequences, namely the squared Catalan numbers, and weighted integer compositions and partitions. We also establish the precise minimum and maximum degrees for those graph classes, and in the latter result the famous Erdős-Szekeres theorem makes its appearance.

7.2.1 Preliminaries

We let \mathcal{C}_n denote the set of all lattice congruences of the weak order on S_n . Throughout this section, we will denote lattice congruences by capital Latin letters such as $R \in \mathcal{S}_n$, and whenever we consider two permutations π, ρ in the same equivalence class of S_n/R , we write $\pi \equiv_R \rho$ or simply $\pi \equiv \rho$, if R is clear from the context. Recall from Theorem 7.3 that every lattice congruence $R \in \mathcal{C}_n$ corresponds to a downset $F_R \subseteq F_n$ of fences in the forcing order, and that such a downset can be represented by its arc diagram, which contains exactly one arc for each fence from F_R . The *reduced arc diagram* contains only the arcs that correspond to maximal elements in the downset F_R , i.e., to fences that are pairwise incomparable in the forcing order. Every fence not of the form $f(a, a+1, \emptyset)$, $a \in [n-1]$, is referred to as *essential*, and we let $F_n^* \subseteq F_n$ denote the set of all essential fences. We refer to any lattice congruence R with $F_R \subseteq F_n^*$ as *essential*, and we let $\mathcal{C}_n^* \subseteq \mathcal{C}_n$ denote the set of all essential lattice congruences. Note that by this definition, the arc diagrams of essential lattice congruences do not contain any arcs that connect consecutive points a and $a+1$, $a \in [n-1]$.

We refer to the underlying undirected graph of the cover graph of any lattice quotient S_n/R , $R \in \mathcal{C}_n$, as a *quotient graph* Q_R , and we

Table 7.9: Number of different classes of quotient graphs that arise from essential lattice congruences and their minimum and maximum degrees. In this table, C_n denotes the n th Catalan number, $c_{n,k}$ denotes the number of integer compositions of n with exactly k many 2s, and t_n denotes the number of 2s in all integer partitions of n . The last column contains references to the corresponding sequence numbers in the OEIS [129].

Description (Ref. and OEIS)		$n = 2$	3	4	5	6	7	General formulas/ bounds
quotient graphs (Thm. 7.17, A330039)	$ \mathcal{Q}_n $	1	4	47	3.322	11.396	0.000	$[2^{2^{n-2}}, 2^{2^n-2n}]$
regular (Cor. 7.26, A001246)	$ \mathcal{R}_n $	1	4	25	196	1.764	17.424	$= C_{n-1}^2$ $= 16^{n(1+o(1))}$
vertex-transitive (Cor. 7.33, A052528)	$ \mathcal{V}_n $	1	4	8	22	52	132	$= \sum_{k \geq 0} 3^k c_{n-1,k}$ $= 2.48...^{n(1+o(1))}$
bipartite (Cor. 7.43, A000108)	$ \mathcal{B}_n $	1	2	5	14	42	132	$= C_{n-1}$ $= 4^{n(1+o(1))}$
non-isomorphic (Thm. 7.18, A330040)	$ \mathcal{Q}'_n $	1	3	19	748	2.027	3.309	$\geq 2^n - 2n + 1$
non-iso. regular (A330042)	$ \mathcal{R}'_n $	1	3	10	51	335	2.909	?
non-iso. vertex-tr. (Cor. 7.35, A024786)	$ \mathcal{V}'_n $	1	3	4	8	11	19	$= t_{n+1}$ $= e^{\pi \sqrt{2n/3(1+o(1))}}$
non-iso. bipartite	$ \mathcal{B}'_n $	1	2	4	9	21	55	?
minimum degree (Thm. 7.24)		1	2	3	4	5	6	$= n - 1$
maximum degree (Thm. 7.27, A123663)		1	2	4	5	7	8	$= 2n - \lceil 2\sqrt{n} \rceil$

define $\mathcal{Q}_n := \{Q_R \mid R \in \mathcal{C}_n^*\}$.

All 47 essential lattice congruences \mathcal{C}_n^* for $n = 4$ are shown in Figure 7.4, ordered by refinement of the congruences and represented by their arc diagrams, where the arcs of the reduced diagrams are highlighted. Recall from the previous section that for every essential lattice congruence $R \in \mathcal{C}_n^*$, Pilaud and Santos [134, Corollary 10] defined an $(n - 1)$ -dimensional polytope, called the *quotientope* of R , whose graph is exactly the quotient graph Q_R . These polytopes are shown in Figure 7.7, where the regular and vertex-transitive graphs are marked with * and **, respectively.

The following lemma justifies that in our definition of \mathcal{C}_n^* , we exclude fences that are not essential. The reason is that including them results in a dimension collapse, i.e., the resulting lattice quotient is isomorphic to some quotient of smaller dimension; see Figure 7.10.

Given two posets $(P, <_P)$ and $(Q, <_Q)$, the *Cartesian product* is the poset $(P \times Q, <)$ with $(p, q) < (p', q')$ if and only if $p <_P p'$ and $q <_Q q'$. For any set of fences $F \subseteq F_n$ and any interval $[s, t]$, $1 \leq s \leq t \leq n$, we define $F|_{[s, t]} := \{f(a, b, L) \in F \mid s \leq a < b \leq t\}$, i.e., we select all fences from F that lie entirely in this interval. Moreover, for any integer s we define $F + s := \{f(a + s, b + s, L + s) \mid f(a, b, L) \in F\}$ with $L + s := \{x + s \mid x \in L\}$, i.e., we shift all fences by s .

Lemma 7.16. *Let $R \in \mathcal{C}_{n+1}$ be a lattice congruence with a non-essential fence $f(s, s + 1, \emptyset) \in F_R$, and define lattice congruences $A \in \mathcal{C}_s$ and $B \in \mathcal{C}_{n+1-s}$ by $F_A = F_R|_{[1, s]}$ and $F_B = F_R|_{[s+1, n+1]} - s$. Moreover, let $R' \in \mathcal{C}_n$ be the lattice congruence given by*

$$F_{R'} = F_A \cup (F_B + (s - 1)) \cup D, \quad (7.4)$$

where D is the downset of the fences $f(s - 1, s + 1, \emptyset)$ and $f(s - 1, s + 1, \{s\})$ in the forcing order for S_n . Then S_{n+1}/R and S_n/R' are both isomorphic to the Cartesian product of S_s/A and S_{n+1-s}/B . In particular, the lattice quotients S_{n+1}/R and S_n/R' are isomorphic.

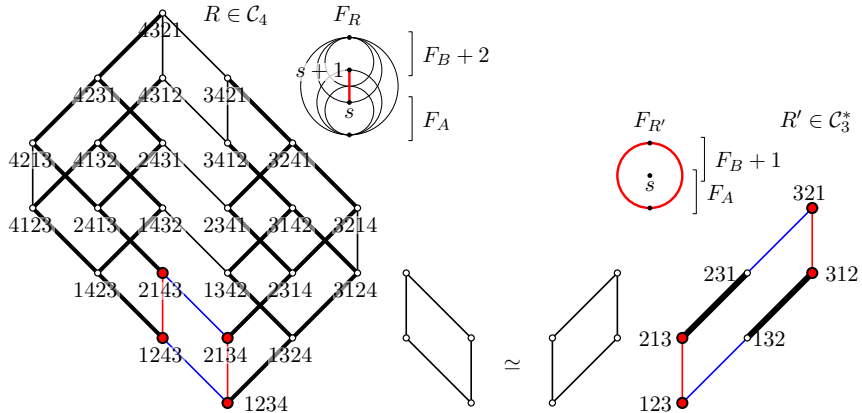


Figure 7.10: Illustration of Lemma 7.16. The left hand side shows the lattice congruence from \mathcal{C}_4 given by the downset of the non-essential fence $f(2, 3, \emptyset)$. The right hand side shows the lattice congruence from \mathcal{C}_3^* given by the downset of the essential fences $\{f(1, 3, \emptyset), f(1, 3, \{2\})\}$. Both lattice quotients are isomorphic to the Cartesian product of S_2 and S_2 , whose cover graph is a 4-cycle.

Proof. Consider two equivalence classes X and Y of R , and two permutations $\pi \in X$ and $\rho \in Y$ that differ in an adjacent transposition of two entries a and b . As F_R contains the fence $f(s, s+1, \emptyset)$, the definition of forcing order implies that F_R also contains all fences $f(c, d, L)$ for all $c \in [1, s]$, $d \in [s+1, n+1]$ and $L \subseteq]c, d[$. This means there are permutations $\pi_0 \in X$ and $\rho_0 \in Y$, such that in π_0 and ρ_0 all entries from $[1, s]$ appear before all entries from $[s+1, n+1]$, and π_0 and ρ_0 differ in an adjacent transposition of a and b , and either $a, b \in [1, s]$ or $a, b \in [s+1, n+1]$. We can reach π_0 and ρ_0 from π and ρ , respectively, by moving down within the equivalence classes X or Y towards permutations with fewer inversions, repeatedly swapping any entry from $[1, s]$ that is to the right of any entry from $[s+1, n+1]$. It follows that every

cover relation of S_{n+1}/R has a corresponding cover relation in the Cartesian product of S_s/A and S_{n+1-s}/B .

Consider two equivalence classes X and Y of R' , and two permutations $\pi \in X$ and $\rho \in Y$ that differ in an adjacent transposition of two entries a and b . By the definition (7.4), the set $F_{R'}$ contain the fences $f(s-1, s+1, \emptyset)$, $f(s-1, s+1, \{s\})$, and all fences in their downset of the forcing order for S_n , so the definition of forcing order yields that $F_{R'}$ also contains all fences $f(c, d, L)$ for all $c \in [1, s-1]$, $d \in [s+1, n]$ and $L \subseteq]c, d[$. It follows that either $a, b \in [1, s]$ or $a, b \in [s, n]$. In the first case, there are permutations $\pi_0 \in X$ and $\rho_0 \in Y$, such that in π_0 and ρ_0 all entries from $[1, s]$ appear at consecutive positions, surrounded by all entries from $[s+1, n]$, and π_0 and ρ_0 differ in an adjacent transposition of a and b . In the second case, there are permutations $\pi^0 \in X$ and $\rho^0 \in Y$, such that in π^0 and ρ^0 all entries from $[s, n]$ appear at consecutive positions, surrounded by all entries from $[1, s-1]$, and π^0 and ρ^0 differ in an adjacent transposition of a and b . Moreover, as $\pi, \pi_0, \pi^0 \in X$ and $\rho, \rho_0, \rho^0 \in Y$, we obtain that every cover relation of S_n/R' has a corresponding cover relation in the Cartesian product of S_s/A and S_{n+1-s}/B . \square

Given any lattice congruence $R \in \mathcal{C}_n$ for which F_R contains non-essential fences, we may repeatedly apply Lemma 7.16 to eliminate them, until we arrive at a lattice congruence $R' \in \mathcal{C}_m^*$, $m < n$, with an isomorphic quotient graph $Q_{R'} \simeq Q_R$.

7.2.2 Exact counts for small dimensions

With computer help, we determined the number of essential lattice congruences, or equivalently, the number of quotient graphs, for $2 \leq n \leq 6$. The results are shown in Table 7.9. We also computed the sets $\mathcal{R}_n, \mathcal{V}_n \subseteq \mathcal{Q}_n$, and $\mathcal{B}_n \subseteq \mathcal{Q}_n$ of all regular, vertex-transitive, and bipartite quotient graphs, respectively, for $2 \leq n \leq 7$, with the help of Theorems 7.25 and 7.42.

Many of the quotient graphs from \mathcal{Q}_n are isomorphic; cf. [134, Figure 8]. This happens for instance if the corresponding arc diagrams differ only by rotation or reflection, but not only in this case; see Figure 7.11. To this end, we let \mathcal{Q}'_n denote all non-isomorphic quotient graphs from \mathcal{Q}_n , and we let \mathcal{R}'_n , \mathcal{V}'_n , and \mathcal{B}'_n be the non-isomorphic regular, vertex-transitive, and bipartite ones. The corresponding counts for small n are also shown in Table 7.9. We clearly have $\mathcal{V}_n \subseteq \mathcal{R}_n \subseteq \mathcal{Q}_n$ and $\mathcal{V}'_n \subseteq \mathcal{R}'_n \subseteq \mathcal{Q}'_n$.

7.2.3 Counting quotient graphs

The following theorem shows that there are double-exponentially many quotient graphs.

Theorem 7.17. *For all $n \geq 3$, we have $2^{2^{n-2}} \leq |\mathcal{Q}_n| \leq 2^{2^n - 2n}$.*

Proof. The number of fences $f(a, b, L) \in F_n$ with $b - a = k \in \{1, \dots, n - 1\}$ is exactly $f_k := (n - k)2^{k-1}$, as for fixed k , there are $(n - k)$ different choices for a and b , and for fixed a and b , there are 2^{k-1} many choices for $L \subseteq]a, b[$. As all fences with $k = n - 1$ are essential for $n \geq 3$ and also incomparable in the forcing order, we obtain at least $2^{f_{n-1}}$ distinct downsets. The total number of essential fences is $\sum_{k=2}^{n-1} f_k = 2^n - 2n =: s$, so there are at most 2^s distinct downsets. \square

To estimate the cardinality of \mathcal{Q}'_n , we have to factor out symmetries of the arc diagrams, i.e., horizontal and vertical reflections, which account for a factor of at most 4. However, isomorphic graphs also arise from arc diagrams that do not only differ by those symmetries; see Figure 7.11. In particular, we have $|\mathcal{Q}_n|/|\mathcal{Q}'_n| > 4$ for $n = 5$ and $n = 6$; see Table 7.9.

This difference in the growth rates can partially be explained by arc diagrams that induce a graph product structure. I.e., if we have an arc diagram with two arcs corresponding to the fences $f(s - 1, s +$

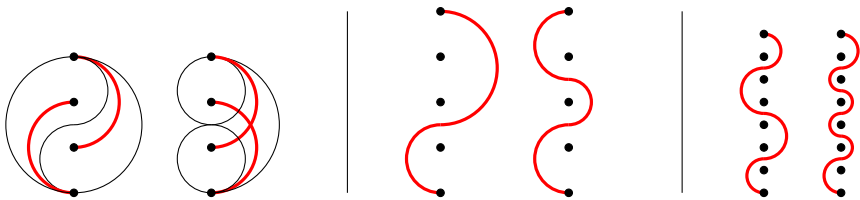


Figure 7.11: Three pairs of lattice congruences from \mathcal{C}_4^* (left), \mathcal{C}_5^* (middle), and \mathcal{C}_8^* (right), with distinct arc diagrams but isomorphic quotient graphs.

$1, \emptyset$) and $f(s-1, s+1, \{s\})$, then by Lemma 7.16 the two parts of the arc diagram separated by these two fences can be mirrored independently, or modified as described by Figure 7.11, yielding the same resulting quotient graph. Such operations clearly yield many more than 4 symmetries. We cannot fully explain this, but we provide the following lower bound.

Theorem 7.18. *For all $n \geq 3$, we have $|\mathcal{Q}'_n| \geq 2^n - 2n + 1$.*

Proof. We argued before that the total number of essential fences in the forcing order is $2^n - 2n =: s$. This implies that the lattice of congruences ordered by refinement (see Figure 7.4) contains a chain $R_0, \dots, R_s \in \mathcal{C}_n^*$ of size $s+1$, where R_0 is the maximal element and R_s is the minimal element, and along this chain we have $|F_{R_i}| = i$ for $i = 0, \dots, s$. Consequently, the number of vertices of the quotient graphs Q_{R_i} , $i = 0, \dots, s$, forms a strictly decreasing sequence, starting with $n!$ and ending with 2^{n-1} . This is because whenever an additional fence is added, the equivalence classes grow, and so the quotient graph shrinks. In particular, all those quotient graphs are non-isomorphic, proving that $|\mathcal{Q}'_n| \geq s+1$. \square

7.2.4 Regular quotient graphs

It turns out that the regular quotient graphs \mathcal{R}_n can be characterized and counted precisely via their arc diagrams. Specifically, we say that an arc is *simple* if it does not connect two consecutive points and if it does not cross the vertical line. Also, we say that a reduced arc diagram is *simple* if it contains only simple arcs. Note that the fence $f(a, b, L)$ corresponding to a simple arc either satisfies $L = \emptyset$ or $L =]a, b[$. For example, in Figure 7.11, the leftmost two reduced arc diagrams are simple, whereas the others are not. In Theorem 7.25 below, we establish that a quotient graph is regular if and only if the corresponding reduced arc diagram is simple. This yields a closed counting formula involving the squared Catalan numbers; see Corollary 7.26.

The first lemma allows us to compute degrees of the quotient graph by considering only the minima and maxima of equivalence classes.

Lemma 7.19. *Let X be an equivalence class of a lattice congruence $R \in \mathcal{C}_n$. Consider all descents in $\pi := \min(X)$ and all permutations π'_1, \dots, π'_d obtained from π by transposing one of them. Also, consider all ascents in $\rho := \max(X)$ and all permutations ρ'_1, \dots, ρ'_a obtained from ρ by transposing one of them. Then the down-neighbors of X in the quotient graph Q_R are $X(\pi'_1), \dots, X(\pi'_d)$, and they are all distinct, and the up-neighbors of X in the quotient graph are $X(\rho'_1), \dots, X(\rho'_a)$, and they are all distinct. In particular, the degree of X in the quotient graph is the number of descents of $\min(X)$ plus the number of ascents of $\max(X)$.*

Proof. From Lemma 7.1 it follows that for any lattice congruence, the down-neighbors of the minimum of an equivalence class X all belong to distinct equivalence classes, and the up-neighbors of the maximum of X all belong to distinct equivalence classes. Recall that in the weak order on S_n , the down-neighbors of a vertex are reached by adjacent transpositions of descents, and the up-neighbors

are reached by adjacent transpositions of ascents. From this the statement follows with the help of Lemma 7.2. \square

The next lemma helps us to compute the maximum of an equivalence class quickly. It is an immediate consequence of the definition of forcing order.

Lemma 7.20. *Consider a lattice congruence $R \in \mathcal{C}_n$ and a permutation π with an ascent (a, b) . Let A be a substring ending with a of entries of π of size at most a , and B a substring starting with b of entries that are of size at least b , i.e., we have $\pi = L A B R$ for some substrings L, R . If the permutation ρ obtained by transposing the pair (a, b) is in the same equivalence class as π , i.e., $\pi \equiv \rho$, then they are also in the same equivalence class as the permutation obtained by swapping the entire substrings A and B , i.e., $\pi \equiv \rho \equiv L B A R$.*

There is a corresponding version of Lemma 7.20 for swapping substrings around a descent (b, a) , to quickly compute the minimum of an equivalence class, but we omit stating this symmetric variant explicitly here.

We first rule out non-simple arc diagrams as candidates for giving a regular quotient graph.

Lemma 7.21. *If an essential lattice congruence $R \in \mathcal{C}_n^*$ has a non-simple arc in its reduced arc diagram, then the quotient graph Q_R is not regular.*

Proof. As R is essential, F_R does not contain any fence of the form $f(s, s+1, \emptyset)$, $s \in [n-1]$. Consequently, the equivalence class containing the identity permutation id_n does not contain any other permutations and so has degree $n-1$ in Q_R by Lemma 7.19. In the following we identify an equivalence class X whose degree in Q_R is n , which proves that Q_R is not a regular graph. This part of the proof is illustrated in Figure 7.12.

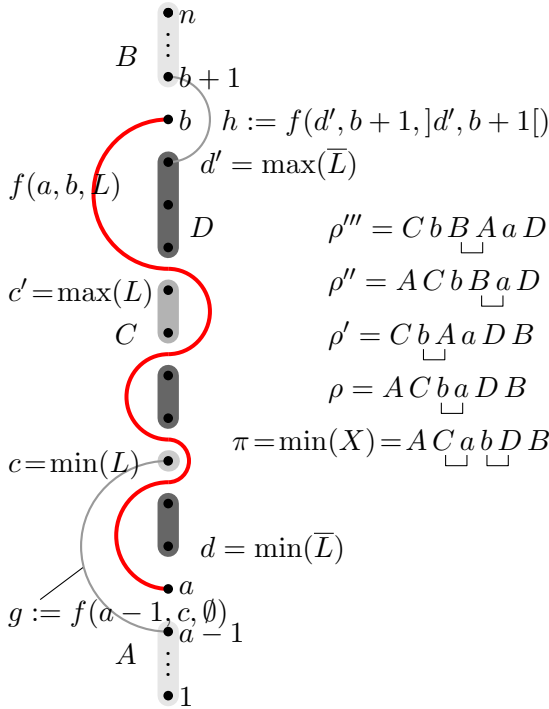


Figure 7.12: Illustration of the proof of Lemma 7.21. Descents in the permutations are marked by square brackets. Consider a non-simple arc in the reduced arc diagram of R , and consider the corresponding fence $f(a, b, L) \in F_R$, $a < b$. We define $\bar{L} :=]a, b[\setminus L$. The assumption that the arc is not simple means that L and \bar{L} are both non-empty. We define $c := \min(L)$, $c' := \max(L)$, $d := \min(\bar{L})$, and $d' := \max(\bar{L})$, and we write C and D for the increasing sequences of numbers in the sets L and \bar{L} , respectively. We also define the sequences $A := (1, \dots, a-1)$ and $B := (b+1, \dots, n)$. Now consider the equivalence class $X \in S_n/R$ which contains the permutations $\pi := A C a b D B$ and $\rho := A C b a D B$.

Clearly, π and ρ differ in an adjacent transposition of the entries a and b , and we have $\pi \equiv \rho$ due to the fence $f(a, b, L) \in F_R$. We

first prove that $\pi = \min(X)$. By Lemma 7.19, we need to check the descents of π , and there are exactly two of them, namely (c', a) and (b, d) . None of them can be transposed to reach a permutation in X , as neither the fence $f(a, c',]a, c'[\cap L)$, nor the fence $f(d, b,]d, b[\cap L)$ is in F_R , as both are above $f(a, b, L)$ in the forcing order, and if one of them was in F_R , then the arc corresponding to $f(a, b, L)$ would not be in the reduced arc diagram. This proves that π is the minimum of X .

Now consider the permutation ρ . It has only one descent (b, a) , and so $n - 1$ ascents. The ascents (c', b) and (a, d) in ρ cannot be transposed to reach a permutation in X , as neither the fence $f(c', b,]c', b[\cap L)$ nor the fence $f(a, d,]a, d[\cap L)$ is in F_R , as both are above $f(a, b, L)$ in the forcing order. Similarly, the ascents that lie entirely within C or D cannot be transposed, as for any such ascent (r, s) , the corresponding fence $f(r, s,]r, s[\cap L)$ is above $f(a, b, L)$ in the forcing order. Moreover, none of the ascents $(s, s + 1)$ that lie entirely within A or B can be transposed, as F_R is essential by assumption and so contains none of the fences $f(s, s + 1, \emptyset)$, $s \in [n - 1]$. It remains to consider the ascents $(a - 1, c)$ and $(d', b + 1)$. They can possibly be transposed to reach a permutation in X , but only if the fence $g := f(a - 1, c, \emptyset)$ or the fence $h := f(d', b + 1,]d', b + 1[)$ is in F_R , which may or may not be the case. If $g \notin F_R$ and $h \notin F_R$, then we have $\rho = \max(X)$, and so $\text{desc}(\max(X)) = 1$. If $g \in F_R$ and $h \notin F_R$, then Lemma 7.20 shows that $\rho' := C b A a D B = \max(X)$, and again we get $\text{desc}(\max(X)) = 1$, as the only descent in ρ' is $(b, 1)$. If $g \notin F_R$ and $h \in F_R$, then Lemma 7.20 shows that $\rho'' := A C b B a D = \max(X)$, and again we get $\text{desc}(\max(X)) = 1$, as the only descent in ρ'' is (n, a) . If $g \in F_R$ and $h \in F_R$, then Lemma 7.20 shows that $\rho''' := C b B A a D = \max(X)$, and again we get $\text{desc}(\max(X)) = 1$, as the only descent in ρ''' is $(n, 1)$.

We have shown that $\text{desc}(\min(X)) = 2$ and $\text{desc}(\max(X)) = 1$, and so $\text{asc}(\max(X)) = n - 2$. Therefore, by Lemma 7.19, the degree

of X in Q_R is $\text{desc}(\min(X)) + \text{asc}(\max(X)) = 2 + (n - 2) = n$, which shows that Q_R is not regular. This completes the proof. \square

We now aim to prove that a simple reduced arc diagram implies an $(n - 1)$ -regular quotient graph. For this we need two auxiliary lemmata.

Lemma 7.22. *Consider a lattice congruence $R \in \mathcal{C}_n$ and an equivalence class X such that $\min(X)$ has n at the rightmost position, and $\max(X) = \cdots c n d \cdots$, where $c, d \in [n - 1]$. Then (c, d) is a descent in $p(\max(X)) \in S_{n-1}$. Similarly, suppose that $\max(X)$ has n at the leftmost position, and $\min(X) = \cdots a n b \cdots$, where $a, b \in [n - 1]$. Then (a, b) is an ascent in $p(\min(X)) \in S_{n-1}$.*

Proof. Consider an equivalence class X such that $\pi := \min(X)$ has n at the rightmost position, and $\rho := \max(X) = \cdots c n d \cdots$, where $c, d \in [n - 1]$. Let $\pi' := p(\pi) \in S_{n-1}$ and $\rho' := p(\rho) \in S_{n-1}$. By Lemma 7.9, π' and ρ' are the minimum and maximum of the equivalence class $p(X)$ of the restriction R^* , and as $\pi = c_n(\pi') = \pi' n$, we also have that $\sigma := c_n(\rho') = \rho' n \in X$. Note that ρ is obtained from σ by moving n to the left until the entry c is directly left of it. In particular, the down-neighbor τ of ρ obtained by transposing n and d satisfies $\tau \equiv \rho$ (recall Lemma 7.1). By Lemma 7.20, it follows that (c, d) must be a descent in τ , as otherwise, the up-neighbor τ' of ρ obtained by transposing c and n would also satisfy $\tau' \equiv \rho$, contradicting the fact that ρ is the maximum of X . Consequently (c, d) is also a descent in $\sigma = \rho' n$ and in $\rho' = p(\max(X))$. The proof of the second part of the lemma is analogous. \square

For any permutation $\pi \in S_n$ and any integers $a, b \in [n]$ with $a < b$, we let $L(\pi, a, b)$ denote the set of all entries of π that are to the left of a , and whose values are in the interval $[a, b]$. For example, for $\pi = 817632459$ we have $L(\pi, 1, 3) = \emptyset$, $L(\pi, 2, 5) = \{3\}$, and $L(\pi, 3, 7) = \{6\}$.

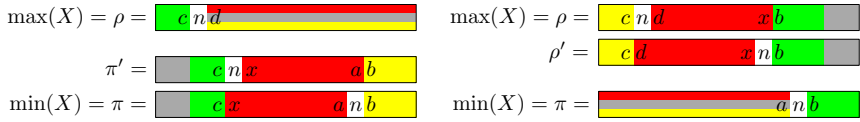


Figure 7.13: Illustration of Lemma 7.23.

Lemma 7.23. *Consider a lattice congruence $R \in \mathcal{C}_n$ and an equivalence class X such that $\pi := \min(X) = \cdots a n b \cdots$ and $\rho := \max(X) = \cdots c n d \cdots$, where $a, b, c, d \in [n-1]$. Let π' be the last permutation in X obtained from π by moving n to the left, and let ρ' be the last permutation in X obtained from ρ by moving n to the right. Then we have the following:*

- (i) *the entry left of n in π' is c ;*
- (ii) *the entry right of n in ρ' is b ;*
- (iii) *if $c \neq a$, then c is to the left of a in π , and we have $c, b > x$ for all x between c and n in π , in particular $a < b$;*
- (iv) *if $b \neq d$, then b is to the right of d in ρ , and we have $c, b > x$ for all x between n and b in ρ , in particular $c > d$;*
- (v) *for any entry x between c and n in π , the fence $f(x, n, L(\pi, x, n))$ is in F_R ;*
- (vi) *for any entry x between n and b in ρ , the fence $f(x, n, L(\rho, x, n))$ is in F_R ;*
- (vii) *the fences $f(b, n, L(\pi, b, n))$ and $f(x, c, L(\pi, x, c))$, where x is the entry right of c in π , are not in F_R ;*
- (viii) *the fences $f(c, n, L(\rho, c, n))$ and $f(x, b, L(\rho, x, b))$, where x is the entry left of b in ρ , are not in F_R .*

Proof. We only need to prove (i), (iii), (v) and (vii), as the other four statements are symmetric.

We first prove (i). Suppose for the sake of contradiction that this is not the case and that there is another entry $e \neq c$ left of n in π' . Note that e must be to the right of c in π' and π , as otherwise (n, c)

would be an inversion in π' , but not in ρ , contradicting the fact that ρ is the maximum of X . We let $\pi'' \notin X$ be the permutation obtained from π' by transposing e and n , i.e., $\text{inv}(\pi'') = \text{inv}(\pi') \cup \{(n, e)\}$. It follows that $(n, e) \notin \text{inv}(\rho)$, as otherwise π'' would be contained in the interval $[\pi, \rho] = X$. This means that e is left of n and left of c in ρ . This implies that $c < e$, as otherwise (c, e) would be an inversion in π' , but not in ρ .

We now move up in the weak order from π' to ρ , creating a sequence of permutations that all contain the ascent (e, n) , as follows: Starting from π' , we repeatedly choose an arbitrary ascent that we can transpose to stay inside X . First note that (n, x) , $x \in [n - 1]$, is never an ascent, so n never moves to the right. Also (e, n) can never be transposed, as $(n, e) \notin \text{inv}(\rho)$. Whenever we encounter a transposition that involves the entry e , then it must be a transposition of the form $(e', e) \rightarrow (e, e')$ with $e' < e$ (otherwise the inversion set would shrink), and we then immediately also perform the transposition $(e', n) \rightarrow (n, e')$, keeping e and n next to each other, and reaching a permutation in X by Lemma 7.20. As $(c, e) \notin \text{inv}(\pi')$ and $(c, e) \in \text{inv}(\rho)$, we must encounter a step in which c and e are transposed. However, by the same reasoning, we can then also transpose c and n to reach another permutation in X , a contradiction to $(n, c) \notin \text{inv}(\rho)$. This completes the proof of (i).

We now prove (iii). The fact that c is left of a in π is an immediate consequence of (i). Note that π' is obtained from π by moving n to the left until n is directly right of c , and n cannot move further as $(n, c) \notin \text{inv}(\rho)$. Conversely, π is obtained from π' by moving n to the right until n is directly left of b , and n cannot move further as π is the minimum of X . In particular, the entry n can be moved across the largest entry e between c and n in π . Lemma 7.20 therefore shows that $b, c > e$, as otherwise n could move to the left of c in π' or to the right of b in π .

We now prove (v). This follows immediately by considering all per-

mutations encountered in X between π to π' , by moving n to the left.

It remains to prove (vii). It is clear that $f(b, n, L(\pi, b, n))$ is not in F_R , as we could otherwise transpose b and n in π , reaching a down-neighbor of π in X . It is also clear that $f(x, c, L(\pi, x, c))$ is not in F_R , as we know that $c > x$ from (iii), and so we could transpose c and x , reaching another down-neighbor of π in X . \square

With these lemmata in hand, we are now ready to establish a tight lower bound for the minimum degree of quotient graphs Q_n .

Theorem 7.24. *For every essential lattice congruence $R \in \mathcal{C}_n^*$, the minimum degree of the quotient graph Q_R is $n - 1$.*

Pilaud and Santos proved in [134] that for every essential lattice congruence, Q_R is the graph of an $(n-1)$ -dimensional polytope. This in particular implies Theorem 7.24. Nevertheless, in this chapter we provide a purely combinatorial proof of the theorem, with the goal of later improving the estimates in the proof when proving Theorem 7.25.

Proof. We prove the theorem by induction on n . For $n = 1$ there is only the trivial lattice congruence for the weak order on $S_n = \{1\}$, and the corresponding quotient graph is an isolated vertex which indeed has minimum degree $n - 1 = 0$. This settles the base case for the induction.

Let $n \geq 2$ and consider an essential lattice congruence $R \in \mathcal{C}_n^*$. The equivalence class X that contains the identity permutation id_n , contains no other permutations by the assumption that R is essential, and so the degree of $X = \{\text{id}_n\}$ in Q_R is exactly $n - 1$ by Lemma 7.19. By Lemma 7.19, it therefore suffices to show that for every equivalence class X of R and its minimum $\pi := \min(X)$ and maximum $\rho := \max(X)$, we have that $\text{desc}(\pi) + \text{asc}(\rho) \geq n - 1$.

For this consider the position of the entry n in both π and ρ . By the

assumption that R is essential, we know that $f(n-1, n, \emptyset) \notin F_R$, which implies $\text{id}_n \not\equiv c_{n-1}(\text{id}_{n-1})$. Applying Lemma 7.6 shows that n cannot be simultaneously at the rightmost position of π and at the leftmost position of ρ . Consequently, we are in one of five possible cases:

- (a) both π and ρ have n at the rightmost position.
- (b) both π and ρ have n at the leftmost position.
- (c) π has n at the rightmost position, and $\rho = \cdots c n d \cdots$, where $c, d \in [n-1]$.
- (d) ρ has n at the leftmost position, and $\pi = \cdots a n b \cdots$, where $a, b \in [n-1]$.
- (e) $\pi = \cdots a n b \cdots$ and $\rho = \cdots c n d \cdots$, where $a, b, c, d \in [n-1]$.

Cases (c) and (d) are exactly the ones discussed in Lemma 7.22, and case (e) is exactly the one discussed in Lemma 7.23. We only prove (a), (c) and (e), as the proof of (b) is analogous to (a), and the proof of (d) is analogous to (c).

By Lemma 7.9, Lemma 7.19, and by induction we know that

$$\text{desc}(p(\pi)) + \text{asc}(p(\rho)) \geq n - 2. \quad (7.5)$$

First consider case (a) above. As n is at the rightmost position in π and ρ , we have

$$\text{desc}(\pi) + \text{asc}(\rho) = \text{desc}(p(\pi)) + \text{asc}(p(\rho)) + 1, \quad (7.6)$$

where the $+1$ comes from the ascent involving n in ρ . Combining (7.5) and (7.6) yields

$$\text{desc}(\pi) + \text{asc}(\rho) \geq (n - 2) + 1 = n - 1, \quad (7.7)$$

with equality if and only if (7.5) holds with equality.

Now consider case (c) above. As n is at the rightmost position in π and n is between c and d in ρ , we have

$$\text{desc}(\pi) + \text{asc}(\rho) = \text{desc}(p(\pi)) + \text{asc}(p(\rho)) + 1 - \text{asc}(cd), \quad (7.8)$$

where the $+1$ comes from the ascent (c, n) in ρ . By Lemma 7.22, (c, d) is a descent, so $\text{asc}(cd) = 0$, and hence combining (7.5) and (7.8) yields

$$\text{desc}(\pi) + \text{asc}(\rho) \geq (n - 2) + 1 = n - 1, \quad (7.9)$$

with equality if and only if (7.5) holds with equality.

Now consider case (e) above. In this case we have

$$\text{desc}(\pi) + \text{asc}(\rho) = \text{desc}(p(\pi)) + 1 - \text{desc}(ab) + \text{asc}(p(\rho)) + 1 - \text{asc}(cd), \quad (7.10)$$

where the $+1$ s come from the descent (n, b) in π and the ascent (c, n) in ρ , respectively. We first consider the subcase that $c = a$ and $b = d$. In this case $(a, b) = (c, d)$ is either a descent or an ascent, so in any case $\text{desc}(ab) + \text{asc}(cd) = 1$, and hence combining (7.5) and (7.10) yields

$$\text{desc}(\pi) + \text{asc}(\rho) \geq (n - 2) + 1 = n - 1, \quad (7.11)$$

with equality if and only if (7.5) holds with equality.

We now consider the subcase that $c = a$ and $b \neq d$. From Lemma 7.23 (iv) we obtain that $c > d$, i.e., $\text{asc}(cd) = 0$, and hence combining (7.5) and (7.10) yields

$$\text{desc}(\pi) + \text{asc}(\rho) \geq (n - 2) + 2 - \text{desc}(ab) \geq n - 1, \quad (7.12)$$

with equality if and only if (7.5) holds with equality and $\text{desc}(ab) = 1$.

The subcase $c \neq a$ and $b = d$ is similar, and yields

$$\text{desc}(\pi) + \text{asc}(\rho) \geq (n - 2) + 2 - \text{asc}(cd) \geq n - 1, \quad (7.13)$$

with equality if and only if (7.5) holds with equality and $\text{asc}(cd) = 1$.

It remains to consider the subcase $c \neq a$ and $b \neq d$. From Lemma 7.23 (iii) and (iv) we know that $a < b$ and $c > d$, i.e., $\text{desc}(ab) = 0$ and $\text{asc}(cd) = 0$, and hence combining (7.5) and (7.10) yields

$$\text{desc}(\pi) + \text{asc}(\rho) \geq (n - 2) + 2 = n > n - 1. \quad (7.14)$$

This completes the proof of the theorem. \square

We are now in position to prove the main result of this section, a characterization of regular quotient graphs via their arc diagram.

Theorem 7.25. *The regular quotient graphs \mathcal{R}_n are obtained from exactly those lattice congruences \mathcal{C}_n^* that have a simple reduced arc diagram.*

After the original paper [89] was published, we learned that Theorem 7.25 can be derived from Theorem 1.13 in [57], which characterizes regular quotient graphs in terms of the minimal sets of join-irreducible elements that need to be contracted to generate the congruence.

Proof. By Lemma 7.21, if the reduced arc diagram of a lattice congruence $R \in \mathcal{C}_n^*$ is not simple, then the quotient graph Q_R is not regular. In the following we will prove the converse, that if the reduced arc diagram of R is simple, then the quotient graph Q_R is $(n - 1)$ -regular. We argue by induction on n , using that by Lemma 7.8, the arc diagram of the restriction of a lattice congruence is obtained by removing the highest point labeled n and all arcs incident with it. In particular, removing the highest point of a simple reduced arc diagram produces another simple reduced arc diagram.

For the induction proof we closely follow the proof of Theorem 7.24 given before, and show that all inequalities in that proof are actually tight if we add the assumption of a simple reduced arc diagram.

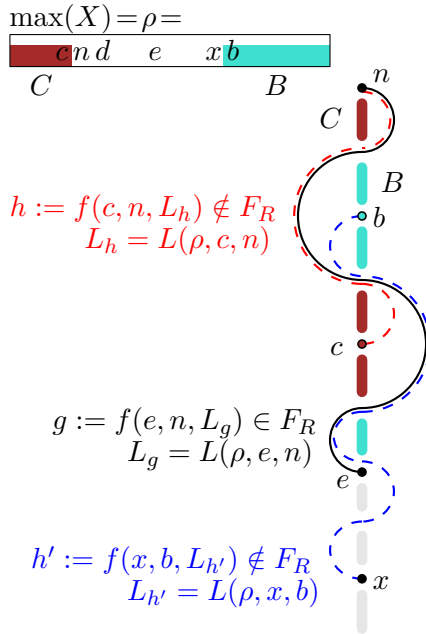


Figure 7.14: Illustration of the proof of Theorem 7.25.

So consider a lattice congruence $R \in \mathcal{C}_n^*$ with a simple reduced arc diagram, an arbitrary equivalence class X of R , and let $\pi := \min(X)$ and $\rho := \max(X)$. We aim to prove that $\text{desc}(\pi) + \text{asc}(\rho) = n - 1$, assuming by induction that (7.5) holds with equality, i.e., we have

$$\text{desc}(p(\pi)) + \text{asc}(p(\rho)) = n - 2. \quad (7.15)$$

We now consider the same cases (a)–(e) as in the proof of Theorem 7.24. The cases (a) and (b) are easy, as (7.7) holds with equality by (7.15). Similarly, the cases (c) and (d) are easy, as (7.9) holds with equality by (7.15). It remains to consider the case (e). The subcase $c = a$ and $b = d$ is again easy, as (7.11) holds with equality because by (7.15). We consider the remaining three subcases of (e)

and show the following:

- (e1) If $c = a$ and $b \neq d$ we have that $\text{desc}(ab) = 1$. From this it follows that (7.12) holds with equality by (7.15).
- (e2) If $c \neq a$ and $b = d$ we have that $\text{asc}(cd) = 1$. From this it follows that (7.13) holds with equality by (7.15).
- (e3) The subcase $c \neq a$ and $b \neq d$ cannot occur (recall the strict inequality (7.14)).

In proving (e1)–(e3), we will use the assumption that the reduced arc diagram is simple. Note that claims (e1)–(e3) follow immediately from the next two claims:

- (e1') If $b \neq d$ and the arc diagram is simple, then $b < c$.
- (e2') If $c \neq a$ and the arc diagram is simple, then $c < b$.

Indeed, if $c = a$ and $b \neq d$, then (e1') gives $b < c = a$, showing that $\text{desc}(ab) = 1$, proving (e1). Similarly, if $c \neq a$ and $b = d$, then (e2') gives $c < b = d$, showing that $\text{asc}(cd) = 1$, proving (e2). Lastly, if $c \neq a$ and $b \neq d$, then (e1') and (e2') together give $b < c$ and $c < b$, a contradiction, so this case cannot occur.

We begin proving (e1'); see Figure 7.14. By Lemma 7.23 (iv), b is to the right of d in ρ . Let e be the maximum entry between n and b in ρ , and let x be the entry directly left of b . It may happen that $e = d$, or $e = x$ or both, but this is irrelevant. In fact, the entry d will not play any role in our further arguments. We clearly have $e \geq x$. Applying Lemma 7.23 (iv), we obtain that $c, b > e$. Suppose for the sake of contradiction that $b > c$. Combining the previous inequalities, we get $x \leq e < c < b < n$, i.e., we have the situation shown in Figure 7.14. From Lemma 7.23 (vi), we obtain that the fence $g := f(e, n, L_g)$ with $L_g := L(\rho, e, n)$ is in F_R . We let C denote the set of values that are strictly larger than e and not to the right of c in ρ . Similarly, we let B denote the set of values that are strictly

larger than e and not to the left of b in ρ . By these definitions and the maximal choice of e , we get $L_g = C$ and $\overline{L_g} :=]e, n[\setminus L_g = B$.

As $c \in L_g$ and $b \in \overline{L_g}$, the arc corresponding to the fence g that connects e with n has c on its left and b on its right, i.e., this arc is not simple. It follows that this arc cannot be in the reduced arc diagram of R . This means there must be another fence $g' = f(u, v,]u, v[\cap L_g)$, $e \leq u < v \leq n$, represented by a simple arc, that forces g in the forcing order. Clearly, as this arc is simple, we have that

$$]u, v[\cap L_g = \emptyset \quad \text{or} \quad]u, v[\cap L_g =]u, v[, \quad (7.16)$$

i.e., $]u, v[$ is an interval of consecutive numbers from B or C , respectively.

From Lemma 7.23 (viii), we also know that the fences $h := f(c, n, L_h)$ with $L_h := L(\rho, c, n)$ and $h' := f(x, b, L_{h'})$ with $L_{h'} := L(\rho, x, b)$ are *not* in F_R . Observe that $L_h =]c, n[\cap L_g$ and $L_{h'} \cap]e, b[= L_g \cap]e, b[$, i.e., the arcs corresponding to the fences h and h' pass to the left and right of the points in the intervals $]c, n[$ or $]e, b[$, respectively, exactly in the same way as the arc corresponding to the fence g ; see Figure 7.14. It follows that the interval $[u, v]$ cannot be contained in the interval $[e, b]$, as otherwise g' would force h' in the forcing order, and we know that $h' \notin F_R$. Similarly, it follows that the interval $[u, v]$ cannot be contained in the interval $[c, n]$, as otherwise g' would force h in the forcing order, and we know that $h \notin F_R$. We conclude that $u < c$ and $v > b$. This however, would mean that c is contained in the interval $]u, v[\cap L_g$, but b is not (as $b \notin L_g$), so none of the two conditions in (7.16) can hold, which means that the fence g' cannot exist. (In other words, the arc corresponding to g' would also have to be non-simple so that g' could force g .) We arrive at a contradiction to the assumption $b > c$. This completes the proof of (e1').

The proof of (e2') is analogous to the proof of (e1'), and uses

Lemma 7.23 (iii) instead of (iv), (v) instead of (vi), and (vii) instead of (viii). We omit the details. \square

From Theorem 7.25, we obtain the following corollary.

Corollary 7.26. *The number of regular quotient graphs \mathcal{R}_n is $|\mathcal{R}_n| = C_{n-1}^2$, where C_n is the n th Catalan number $C_n := \frac{1}{n+1} \binom{2n}{n}$.*

Proof. By Theorem 7.25, we need to count simple reduced arc diagrams on n points. Clearly, arcs passing to the left of the points are independent from arcs passing to the right of the points, so the result is proved by showing that diagrams where all arcs pass on the same side are counted by the Catalan numbers C_{n-1} . Note that the arcs are all simple, so no arc connects two consecutive points. Circular arcs in such a diagram are non-nesting, by the assumption that the diagram is reduced, as nested arcs correspond to fences that are comparable in the forcing order. A bijection between such non-nesting circular arc diagrams on n points and Dyck paths with $2(n-1)$ steps is illustrated in Figure 7.15. Here, a *Dyck path* with $2n$ steps refers to a series of steps going up from 0 to $2n$, where at every step, we go left or right by one unit. Further, the startpoint 0 and the endpoint $2n$ are on a vertical line, and the path cannot cross this line, i.e., it has no step on one side of the line (but it can touch the line). It has been shown that the distinct Dyck paths with $2n$ steps are counted by the Catalan number C_n [49]. The lemma then follows. \square

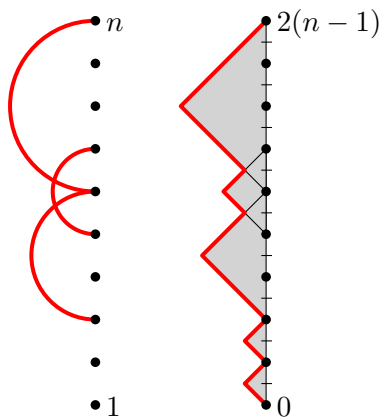


Figure 7.15: Bijection between non-nesting arc diagrams on n points and Dyck paths with $2(n-1)$ steps.

7.2.5 Maximum degree

The next theorem establishes an exact formula for the maximum degrees of quotient graphs.

Theorem 7.27. *For every lattice congruence $R \in \mathcal{C}_n$, the maximum degree of the quotient graph Q_R is at most $2n - \lceil 2\sqrt{n} \rceil$. Moreover, there is a lattice congruence with a vertex of this degree.*

For proving Theorem 7.27, we need the following variant of the famous Erdős-Szekeres theorem.

Lemma 7.28. *Consider a sequence of distinct integers of length n , and let r and s be the length of the longest monotonically increasing and decreasing subsequences, respectively. Then we have $r + s \geq \lceil 2\sqrt{n} \rceil$.*

The Erdős-Szekeres theorem is usually stated in the slightly weaker form that one of r or s is at least $\lceil \sqrt{n} \rceil$. The proof of our lemma follows Seidenberg's proof [153] (see also [156]).

Proof. Let x_1, \dots, x_n be the sequence we consider. For $i = 1, \dots, n$, let a_i and b_i be the lengths of the longest increasing or decreasing subsequences ending with x_i . Note that for $1 \leq i < j \leq n$ we either have $x_i < x_j$, and then we know that $a_i < a_j$, or we have $x_i > x_j$, and then we know that $b_i < b_j$. Consequently, all pairs (a_i, b_i) must be distinct, and we have $1 \leq a_i \leq r$ and $1 \leq b_i \leq s$, implying that $n \leq rs$. From the arithmetic/geometric mean inequality we obtain $r + s = 2(r + s)/2 \geq 2\sqrt{rs} \geq 2\sqrt{n}$. As r and s must be integers, this implies the lower bound $r + s \geq \lceil 2\sqrt{n} \rceil$. \square

Proof of Theorem 7.27. Consider any permutation ρ in the weak order on S_n , and consider another permutation $\pi < \rho$ in its downset. Consider the longest monotonically decreasing subsequence of ρ , and let r denote its length. Note that ρ has at most $n - 1 - (r - 1) = n - r$

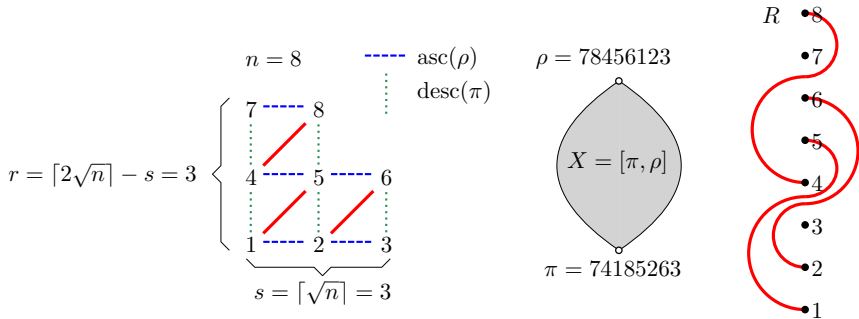


Figure 7.16: Illustration of the proof of Theorem 7.27. The right part of the figure shows the reduced arc diagram of the lattice congruence R .

ascents, regardless of the values between the elements of the subsequence. Similarly, consider the longest monotonically increasing subsequence of ρ , and let s denote its length. Observe that the elements of this subsequence appear in the same relative order in π , so π has at most $n - 1 - (s - 1) = n - s$ descents, regardless of the values between the elements of the subsequence. Overall, we have $\text{desc}(\pi) + \text{asc}(\rho) \leq 2n - (r + s)$. Applying Lemma 7.19 and Lemma 7.28 completes the proof of the upper bound in the theorem.

It remains to construct a lattice congruence $R \in \mathcal{C}_n$ that has an equivalence class X with $\text{desc}(\min(X)) + \text{asc}(\max(X)) = 2n - \lfloor 2\sqrt{n} \rfloor$. This construction is illustrated in Figure 7.16. Fill the numbers $1, 2, \dots, n$ into a table with $s := \lceil \sqrt{n} \rceil$ columns, row by row from bottom to top, and from left to right in each row. The topmost row may not be filled completely. It can be checked that the number of rows r of the table is $r = \lfloor 2\sqrt{n} \rfloor - s$. Now consider the permutation π obtained by reading the columns of the table from left to right, and from top to bottom in each column. It satisfies $\text{desc}(\pi) = (n - 1) - (s - 1) = n - s$. Also consider the permutation ρ obtained by reading the rows of the table from top to bottom, and from left

to right in each row. It satisfies $\text{asc}(\rho) = (n - 1) - (r - 1) = n - r$. We now construct a lattice congruence R that has an equivalence class X with $\min(X) = \pi$ and $\max(X) = \rho$. From Lemma 7.19, we then obtain that the degree of X in the quotient graph Q_R is $\text{desc}(\pi) + \text{asc}(\rho) = 2n - (r + s) = 2n - \lceil 2\sqrt{n} \rceil$.

We construct R by specifying a set of fences in the forcing order, and then take the downset of all those fences as F_R . The fences are constructed as follows: For each pair of numbers a and b in our table where b is one row above and one column to the right of a , we let L be the set of all numbers left of b in the same row as b , and we add the fence $f(a, b, L)$. Now F_R is obtained by taking the downset of all those fences in the forcing order. Observe that as our initial fences $f(a, b, L)$ all satisfy $b - a = s + 1$, all fences $f(a, b, L)$ in F_R satisfy $b - a \geq s + 1$. Using the definition of fences and Theorem 7.3, it can be verified directly that ρ and π belong to the same equivalence class. To see that π is the minimal element of its equivalence class, note that all descents of π have difference s , so none of these fences is in R , meaning that none of the edges leading to a down-neighbor of π is a bar. Similarly, to see that ρ is the maximal element of its equivalence class, note that all ascents of ρ have difference 1, so none of these fences is in R , meaning that none of the edges leading to an up-neighbor of ρ is a bar. \square

7.2.6 Vertex-transitive quotient graphs

It turns out that all vertex-transitive quotient graphs \mathcal{V}_n and \mathcal{V}'_n can be characterized and counted precisely via weighted integer compositions and partitions, respectively; see Theorems 7.32 and 7.34 and Corollaries 7.33 and 7.35 below. As $\mathcal{V}_n \subseteq \mathcal{R}_n$, by Theorem 7.25 we only need to consider simple reduced arc diagrams as candidates for vertex-transitive quotient graphs. However, as we shall see, we will have to impose further restrictions on the diagram. Specifically, we refer to an arc corresponding to a fence $f(a, b, L)$ with $L = \emptyset$ as a

left arc, and with $L =]a, b[$ as a *right arc*. Also, we say that an arc connecting two points $s - 1$ and $s + 1$, $s \in [2, n - 1]$, is *short*. Moreover, we say that the reduced arc diagram is *empty*, if it contains no arcs. We will see that all reduced arc diagrams that yield vertex-transitive graphs are suitable concatenations of smaller diagrams that are either empty or contain only short left or right arcs.

The *Cartesian product* of two graphs $G = (V, E)$ and $H = (W, F)$, denoted $G \square H$, is the graph with vertex set $V \times W$ and edges connecting (v, w) with (v', w') whenever $v = v'$ and (w, w') is an edge in F , or $w = w'$ and (v, v') is an edge in E . We write $G \simeq H$ if G and H are isomorphic graphs. We say that a graph is *prime* if it is not a Cartesian product of two graphs with fewer vertices each. The following lemma captures a few simple observations that we will need later.

Lemma 7.29 ([97, page 29+Corollary 4.16+Theorem 4.19]). *The following statements hold for arbitrary connected graphs G, G', H, H' :*

- (i) *We have $G \square H \simeq H \square G$.*
- (ii) *If $G \square H \simeq G' \square H'$ and both H and H' are prime, then we have $G \simeq G'$ and $H \simeq H'$, or $G \simeq H'$ and $H \simeq G'$.*
- (iii) *G and H are vertex-transitive, if and only if $G \square H$ is vertex-transitive.*

Consider a lattice congruence $R \in \mathcal{C}_n^*$ such that F_R contains two fences $f(s - 1, s + 1, \emptyset)$ and $f(s - 1, s + 1, \{s\})$ for some $s \in [2, n - 1]$. Note that any essential fence of the form $f(a, b, L)$, with $a \in [1, s]$, $b \in [s, n]$ and $L \subseteq]a, b[$ is in the downset of one of the these two fences in the forcing order. In other words, the reduced arc diagram of R contains no arc from a point in $[1, s]$ to a point in $[s, n]$, except the short left arc and short right arc that connect the points $s - 1$ and $s + 1$. Moreover, by Lemma 7.16, the quotient graph Q_R is obtained as the Cartesian product of the quotient graphs of the two lattice congruences $A \in \mathcal{C}_s$ and $B \in \mathcal{C}_{n+1-s}$ whose reduced arc

diagrams contain exactly the arcs of the reduced arc diagram of R restricted to the intervals $[1, s]$ and $[s, n]$, respectively. We say that in the reduced arc diagram of R , the short left arc and right arc that connect the points $s - 1$ and $s + 1$ form a *loop centered at s* , and we say that the reduced arc diagram of R is the *product* of the reduced arc diagrams of A and B . In this way, the product of two reduced arc diagrams is obtained by gluing together their endpoints, and placing a loop centered at the gluing point.

With slight abuse of notation, we use S_n to also denote the cover graph of the weak order on S_n . The 5-cycle C_5 is obtained as the quotient graph for the lattice congruence $R \in \mathcal{C}_3^*$ given either by $F_R = \{f(1, 3, \emptyset)\}$, or by $F_R = \{f(1, 3, \{2\})\}$. Clearly, both S_n and C_5 are vertex-transitive, and the reduced arc diagram of the former has n points and is empty, and the reduced arc diagram of the latter has 3 points and either one short left arc or one short right arc that connects the first with the third point. In the following we argue that all vertex-transitive quotient graphs \mathcal{V}_n have arc diagrams that are products of these two basic diagrams. We first rule out any other arc diagrams as candidates for giving a vertex-transitive quotient graph.

Recall that the quotient graph Q_R has as vertices all equivalence classes of R , and an edge between any two classes X and Y that contain a pair of permutations differing in an adjacent transposition.

Lemma 7.30. *Let $n \geq 4$, and let $R \in \mathcal{C}_n^*$ be a lattice congruence whose reduced arc diagram is simple and has no loops. If the reduced arc diagram is not empty, then Q_R is not vertex-transitive.*

Proof. Given a permutation $\pi \in S_n$ and four distinct entries $a, b, c, d \in [n]$ with $a < b$ and $c < d$ such that π is incident with an (a, b) -edge and a (c, d) -edge in the cover graph of the weak order on S_n , then π forms a 4-cycle in this graph, given by all four permutations obtained from π by transposing a with b , and c with d in all

possible ways. We denote this 4-cycle by $C(\pi, (a, b), (c, d))$. Similarly, given π and three distinct entries $a, b, c \in [n]$ with $a < b < c$ such that π is incident with an (x, y) -edge and an (x, z) -edge, where $\{x, y, z\} = \{a, b, c\}$, then π forms a 6-cycle in the cover graph, given by all six permutations obtained from π by permutating a, b, c in all possible ways. We denote this 6-cycle by $C(\pi, (a, b, c))$. Let L denote the set of all entries to the left of all of a, b, c in π . The 6-cycle $C(\pi, (a, b, c))$ has two edges belonging to the fence $f(a, b, L \cap]a, b[)$ and two edges belonging to the fence $f(b, c, L \cap]b, c[)$, and we abbreviate these edge sets by E_{12} and E_{23} , respectively. It also has exactly one edge belonging to the fence $f(a, c, L \cap]a, c[)$ and one edge belonging to the fence $f(a, c, L \cap]a, c[\cup \{b\})$, and we abbreviate these edge sets by $E_{13\emptyset}$ and E_{132} , respectively. These edge sets of the 4-cycles and 6-cycles mentioned before capture how the type i and type ii forcing constraints (recall Figure 7.2) act on those cycles. In the following arguments, we have to distinguish carefully between cycles in the weak order on S_n , and cycles in the quotient graph Q_R . In particular, a 6-cycle in the weak order may result in a 6-, 5-, or 4-cycle in Q_R , or collapse to a single edge or vertex in Q_R , depending on which of the four aforementioned edges are bars.

By Theorem 7.25, all vertices in the quotient graph Q_R have degree $n - 1$. The strategy of our proof is to consider two particular vertices in the graph, and each of the $\binom{n-1}{2}$ pairs of edges incident with each of those vertices. Every such pair of edges defines a 4-, 5-, or 6-cycle in Q_R containing these two edges. In the corresponding quotientope, these cycles bound the 2-dimensional faces incident to that vertex. We will show that the number of 4-cycles and $\{5, 6\}$ -cycles incident to the two vertices is different, implying that the graph is not vertex-transitive. One of the two vertices we consider is the equivalence class that contains only the identity permutation id_n . The $n - 1$ edges incident with it are $(i, i + 1)$ -edges for $i = 1, \dots, n - 1$ (recall Lemma 7.19 and that R is assumed to be essential). There are $n - 2$ pairs of an $(i, i + 1)$ -edge and an

$(i+1, i+2)$ -edge, and every such pair of edges defines either a 5-cycle or a 6-cycle in Q_R : Indeed, the edges in E_{12} and E_{23} of the 6-cycle $C(\text{id}_n, (i, i+1, i+2))$ are not bars, as R is essential. Moreover, by the assumption that the diagram of R contains no loops, at most one of the fences $f(i, i+2, \emptyset)$ or $f(i, i+2, \{i+1\})$ is in F_R , so at most one of the edges in $E_{13\emptyset}$ or E_{132} of the 6-cycle is a bar. It follows that the corresponding cycle in Q_R is a 5-cycle or a 6-cycle. The remaining $\binom{n-1}{2} - (n-2) = \binom{n-2}{2}$ pairs of an $(i, i+1)$ -edge and a $(j, j+1)$ -edge, $j > i+1$, incident with id_n all form a 4-cycle in Q_R : Indeed, none of the edges of the 4-cycle $C(\text{id}_n, (i, i+1), (j, j+1))$ are bars, as R is essential.

In the remainder of this proof we identify any arc in the diagram of R with the fence in the downset F_R of the forcing order that it represents. An arc being in the diagram means that the corresponding fence is contracted, i.e., its edges are bars, meaning that the permutations that are the endpoints of such a bar are in the same equivalence class. Conversely, an arc not being in the diagram means that the corresponding fence is not contracted, i.e., its edges are not bars.

As the reduced arc diagram of R is not empty, we consider the arc $f(a, b, L)$ incident to the highest point. As all arcs are simple, we may assume by symmetry that it is a left arc, i.e., $L = \emptyset$, and if there is also a right arc incident to this point, then we may assume that the left arc is at least as long as the right arc.

- (i) The left arc $f(a, b, \emptyset)$ is in the diagram.
- (ii) The endpoints of all arcs are below or at point b .
- (iii) If there is a right arc ending at point b , then its starting point a' satisfies $a' \geq a$.
- (iv) No two arcs in the diagram are nested, by the assumption that the diagram is reduced, as nested arcs correspond to fences that are comparable in the forcing order. In particular, no left arc $f(a', b, \emptyset)$, $a' > a$, is in the diagram.

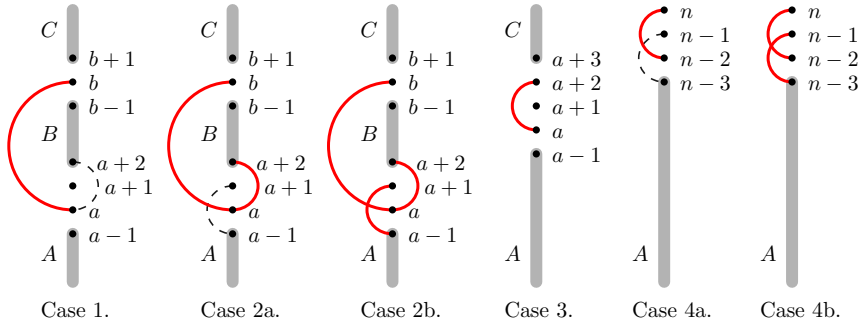


Figure 7.17: Case distinctions in the proof of Lemma 7.30. Arcs in the diagram are drawn with solid lines, arcs that are *not* in the diagram are indicated by dashed lines.

- (v) All arcs are simple, in particular, no arc connects two consecutive points, as R is assumed to be essential.

We define the sequences $A := (1, \dots, a-1)$, $B := (a+2, \dots, b-1)$, and $C := (b+1, \dots, n)$. The various cases considered in the following proof are illustrated in Figure 7.17.

Case 1: $b-a \geq 3$ and the short right arc $f(a, a+2, \{a+1\})$ is not in the diagram of R . Consider the equivalence class X_1 containing the permutation $\pi_1 := Aab(a+1)BC$. It has exactly one descent, namely $(b, a+1)$, and as the left arc $f(a+1, b, \emptyset)$ is not in the diagram by (iv), we obtain that π_1 is the minimum of X_1 (recall Lemma 7.19). Consider the permutation $\rho_1 := bAa(a+1)BC$, obtained from π_1 by transposing the substring Aa with b , and so by (i) and Lemma 7.20, ρ_1 is also contained in X_1 . Moreover, all $n-2$ ascents in ρ_1 are $(i, i+1)$, for $i \in [n-1] \setminus \{b-1, b\}$, and $(b-1, b+1)$ (if $b < n$), and so by (ii) and (v), ρ_1 is the maximum of X_1 (recall Lemma 7.19).

For the moment we assume that $b < n-1$. As Table 7.18 shows, there are two edges incident with id_n that are labelled with a trans-

Table 7.18: Summary of arguments in case 1 in the proof of Lemma 7.30. Edges that are not bars are marked with (i)–(v), referencing the argument for why that arc is not in the diagram of R . The edge marked with (*) is not a bar, and the argument is given after the table. Edges marked with ? are irrelevant for our arguments. The 6-cycles marked with [1] are only valid if $b < n$, and those marked with [2] are only valid if $b < n - 1$.

$$\rho_1 = \max(X_1) = b A a (a + 1) B C$$

$$\pi_1 = \min(X_1) = A a b (a + 1) B C$$

edges inc. only with id_n ($b - 1, b$) ($b, b + 1$) [1]	edges inc. only with X_1 ($a + 1, b$) ($b - 1, b + 1$) [1]	
6-cycles inc. only with id_n $C(\text{id}_n, (b - 2, b - 1, b))$ $C(\text{id}_n, (b - 1, b, b + 1))$ [1] $C(\text{id}_n, (b, b + 1, b + 2))$ [2]	6-cycles inc. only with X_1 $C(\pi_1, (a, a + 1, b))$ $C(\pi_1, (a + 1, a + 2, b))$ $C(\pi_1, (b - 2, b - 1, b + 1))$ [1] $C(\pi_1, (b - 1, b + 1, b + 2))$ [2]	$E_{12} E_{23} E_{13\emptyset} E_{132}$ (v) (iv) ? (*) (v) (iv) (iv) ? (v) (ii) ? (ii) (ii) (v) ? (ii)

position that does not appear at any edge incident with X_1 . Conversely, there are two edges incident with X_1 that are labelled with a transposition that does not appear at any edge incident with id_n . Together with the other edges incident with X_1 , we obtain three pairs of edges incident only with id_n that define a 6-cycle in the weak order on S_n , and four pairs of edges incident only with X_1 that define a 6-cycle. All the latter 6-cycles are $\{5, 6\}$ -cycles in the quotient graph Q_R , showing that the number of $\{5, 6\}$ -cycles incident with a vertex of Q_R is by one higher for X_1 than for id_n . The argument that at most one edge from each 6-cycle is a bar, is given at the bottom right of the table, separately for each of the various sets of edges on each cycle. E.g., for the 6-cycle $C(\pi_1, (a, a + 1, b))$, the two edges in E_{12} belong to the fence $f(a, a + 1, \emptyset)$, and the corresponding arc is not in the diagram by (v). It remains to argue

about case (*) in the table, i.e., the arc $f(a, b, \{a + 1\})$. This arc is non-simple, and so it is not in the reduced diagram. Moreover, in the forcing order it can only be forced by a simple arc $f(a', b, \emptyset)$ with $a' \geq a + 1$, which is impossible by (iv), or by the simple short right arc $f(a, a + 2, \{a + 1\})$, which is impossible by the extra assumption we imposed at the beginning of case 1. It follows that the arc $f(a, b, \{a + 1\})$ is not in the diagram of R .

In the cases $b = n - 1$ and $b = n$ one or two of the 6-cycles in the first two columns of Table 7.18 are invalid, but the remaining 6-cycles still result in a surplus of $\{5, 6\}$ -cycles incident with X_1 .

Case 2: $b - a \geq 3$ and the short right arc $f(a, a + 2, \{a + 1\})$ is in the diagram of R . Consider the equivalence class X_2 containing the permutation $\pi_2 := A(a + 1)abBC$. This permutation has two descents, $(a + 1, a)$ and $(b, a + 2)$, and it can be checked that $\pi_2 = \min(X_2)$.

Subcase 2a: We now additionally assume that if $a > 1$, then the short left arc $f(a - 1, a + 1, \emptyset)$ is not in the diagram of R . Using this assumption, it can be checked that $\rho_2 := \max(X_2) = A(a + 1)bBCa$. As before, we now consider all transpositions that appear as edge labels at only either id_n or X_2 , and we consider the resulting pairs of transpositions that define a 6-cycle in the weak order on S_n , yielding the 6-cycles shown in Table 7.19, where some of them exist only under the extra conditions on a and b stated in the table.

Unlike in case 1, where we argued that there are *more* $\{5, 6\}$ -cycles incident with X_1 than with id_n , in case 2 we argue that there are *fewer* $\{5, 6\}$ -cycles incident with X_2 than with id_n . For this we consider the two 6-cycles marked with (*) and (**) in the table, and argue that each of them is contracted to a 4-cycle in the quotient graph Q_R . Indeed, for the 6-cycle $C(\rho_2, (a - 1, a + 1, b))$, the edges in E_{12} are not bars by the assumption that the short left arc $f(a - 1, a + 1, \emptyset)$ is not in the diagram of R , the edges in E_{23} are not bars

Table 7.19: Summary of arguments in case 2a. Some 6-cycles are only valid under the following extra conditions: [1] $b < n$, [2] $b < n - 1$, [3] $b > a + 3$, [4] $a > 1$, [5] $a > 2$.

$$\rho_2 = \max(X_2) = A(a+1)bBCa$$

$$\pi_2 = \min(X_2) = A(a+1)abBC$$

edges inc. only with id_n	edges inc. only with X
$(a-1, a)$ [4]	$(a-1, a+1)$ [4]
$(a+1, a+2)$	$(a+1, b)$
$(b-1, b)$ [3]	$(a+2, b)$ [3]
$(b, b+1)$ [1]	$(b-1, b+1)$ [1]
6-cycles inc. only with id_n	6-cycles inc. only with X
$C(\text{id}_n, (a-2, a-1, a))$ [5]	$C(\pi_2, (a-2, a-1, a+1))$ [5]
$C(\text{id}_n, (a, a+1, a+2))$	$C(\rho_2, (a-1, a+1, b))$ [4] (*)
$C(\text{id}_n, (a+1, a+2, a+3))$ [3]	$C(\pi_2, (a, a+1, b))$ (**)
$C(\text{id}_n, (b-2, b-1, b))$ [3]	$C(\rho_2, (a+1, a+2, b))$ [3]
$C(\text{id}_n, (b-1, b, b+1))$ [1]+[3]	$C(\pi_2, (a+2, a+3, b))$ [3]
$C(\text{id}_n, (b, b+1, b+2))$ [2]	$C(\pi_2, (b-2, b-1, b+1))$ [1]+[3]
	$C(\pi_2, (b-1, b+1, b+2))$ [2]

by (iv), the edge in $E_{13\emptyset}$ is a bar, as the left arc $f(a-1, b, \emptyset)$ is forced by the left arc $f(a, b, \emptyset)$ in the forcing order, and the edge in E_{132} is a bar, as the arc $f(a-1, b, \{a+1\})$ is forced by the short right arc $f(a, a+2, \{a+1\})$. For the 6-cycle $C(\pi_2, (a, a+1, b))$, the edges in E_{12} are not bars by (v), the edges in E_{23} are not bars by (iv), the edge in $E_{13\emptyset}$ is a bar by (i), and the edge in E_{132} is a bar, as the arc $f(a, b, \{a+1\})$ is forced by the short right arc $f(a, a+2, \{a+1\})$.

As one can check from the table, the deficiency of $\{5, 6\}$ -cycles incident with X_2 compared to id_n continues to hold even when some of the 6-cycles in Table 7.19 are invalid as a consequence of some or all of the conditions [1]–[5] being violated, as the cycle marked with (**) is always valid.

Subcase 2b: We now assume that $a > 1$ and that the short left arc $f(a-1, a+1, \emptyset)$ is in the diagram of R . Using this assumption, it can be checked that $\rho'_2 := \min(X_2) = (a+1) b B C A a$. Proceeding similarly to before, we obtain a table that differs from Table 7.19 exactly by omitting all lines marked [4] or [5], i.e., we obtain the same conclusion that there is a deficiency of $\{5, 6\}$ -cycles incident with X_2 compared to id_n .

Case 3: $a < n-2$ and $b = a+2$. In this case we reconsider the equivalence class X_1 defined in case 1, yielding the following simplified Table 7.20.

Table 7.20: Summary of arguments in case 3. The 6-cycle marked with [1] is only valid if $a < n-3$.

$\rho_1 = \max(X_1) = (a+2) A a (a+1) C$ $\pi_1 = \min(X_1) = A a (a+2) (a+1) C$	
edges inc. only with id_n ($a+2, a+3$)	edges inc. only with X_1 ($a+1, a+3$)
6-cycles inc. only with id_n $C(\text{id}_n, (a+2, a+3, a+4))$	6-cycles inc. only with X_1 $C(\rho_1, (a, a+1, a+3))$ (*) $C(\pi_1, (a+1, a+3, a+4))$ [1]

The cycle $C(\text{id}_n, (a+2, a+3, a+4))$ is also a 6-cycle incident with id_n in the quotient graph Q_R by (ii) and (v). We now show that the 6-cycle $C(\rho_1, (a, a+1, a+3))$ marked with (*) is a 5-cycle incident with X_1 in Q_R , which proves that X_1 is incident with one more 5-cycle than id_n . Indeed, the edges in E_{12} of the marked cycle are not bars by (v), the edges in E_{23} are not bars by (ii), the edge in E_{132} is not a bar, as the right arc $f(a, a+3, \{a+1, a+2\})$ is not in the diagram of R by (ii), and none of the short right arcs $f(a, a+2, \{a+1\})$ or $f(a+1, a+3, \{a+2\})$ that might force it is in the diagram by the assumption that the diagram has no loops, or by (ii), respectively. Furthermore, the edge in $E_{13\emptyset}$ is a bar, as the arc $f(a, a+3, \{a+2\})$

is forced by the short left arc $f(a, a+2, \emptyset)$, which is in the diagram by (i).

Case 4: $a = n-2$ and $b = a+2 = n$. In this case the short right arc $f(n-2, n, \{n-1\})$ is not in the diagram of R , by the assumption that the diagram contains no loops.

Subcase 4a: We now additionally assume that the short left arc $f(n-3, n-1, \emptyset)$ is not in the diagram of R . In this subcase, we consider two equivalence classes X_4 and Y_4 that are distinct from id_n . The first equivalence class X_4 is the one containing the permutation $\pi_4 := A n (n-1) (n-2)$, and one can check that $\pi_4 = \min(X_4)$ and that $\rho_4 := \max(X_4) = n A (n-1) (n-2)$. The second equivalence class Y_4 contains only a single permutation $\sigma_4 = \min(Y_4) = \max(Y_4) = A (n-1) n (n-2)$. There is only a single 6-cycle incident with only either X_4 or Y_4 , namely $C(\rho_4, (n-3, n-2, n-1))$, and one can argue that it is a $\{5, 6\}$ -cycle in the quotient graph Q_R , implying that there are more $\{5, 6\}$ -cycles incident with X_4 than with Y_4 in Q_R ; see Table 7.21.

Table 7.21: Summary of arguments in case 4a.

$\rho_4 = \max(X_4) = n A (n-1) (n-2)$	$\sigma_4 = \min(Y_4) = \max(Y_4) = A (n-1) n (n-2)$
$\pi_4 = \min(X_4) = A n (n-1) (n-2)$	
edges inc. only with X_4	edges inc. only with Y_4
$(n-2, n-1)$	$(n-2, n)$
6-cycles inc. only with X_4	6-cycles inc. only with Y_4
$C(\rho_4, (n-3, n-2, n-1))$	

Subcase 4b: We now assume that the short left arc $f(n-3, n-1, \emptyset)$ is in the diagram of R . We consider the equivalence class X'_4 that contains the permutation $\pi'_4 := A (n-1) (n-2) n$. One can check that $\pi'_4 = \min(X'_4)$ and that $\rho'_4 := \max(X'_4) = (n-1) A (n-2) n$. There is only a single 6-cycle incident with only either id_n or X'_4 ,

namely $C(\rho'_4, (n-3, n-2, n))$; see Table 7.22. We now argue that this is a 5-cycle in the quotient graph Q_R , implying that there are more 5-cycles incident with X'_4 than with id_n in Q_R . Indeed, the edges in E_{12} are not bars by (v), the edges in E_{23} are not bars, as the short right arc $f(n-2, n, \{n-1\})$ is not in the diagram by the assumption that it has no loops. Moreover, the edge in $E_{13\emptyset}$ is a bar, as the arc $f(n-3, n, \{n-1\})$ is forced by the short left arc $f(n-3, n-1, \emptyset)$. Finally, the edge in E_{132} is not a bar, as the right arc $f(n-3, n, \{n-2, n-1\})$ is not in the reduced diagram by (iii), and the two short right arcs $f(n-3, n-1, \{n-2\})$ and $f(n-2, n, \{n-1\})$ that may force it in the forcing order are not in the diagram by the assumption of loop-freeness.

Table 7.22: Summary of arguments in case 4b.

$\rho'_4 = \max(X'_4) = (n-1) A (n-2) n$	
$\pi'_4 = \min(X'_4) = A (n-1) (n-2) n$	
edges inc. only with id_n $(n-1, n)$	edges inc. only with X'_4 $(n-2, n)$
6-cycles inc. only with id_n	6-cycles inc. only with X'_4 $C(\rho'_4, (n-3, n-2, n))$

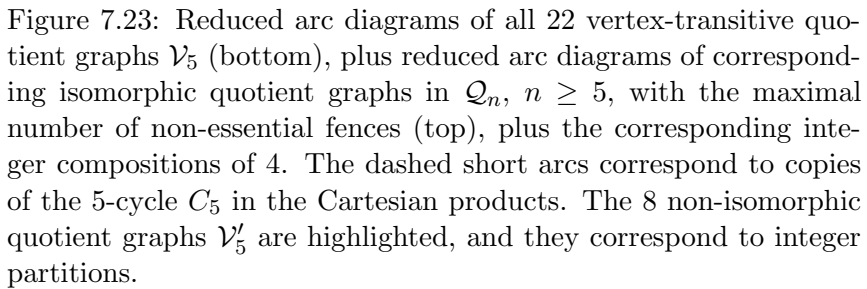
This completes the proof of the lemma. \square

With Lemma 7.30 in hand, we are now ready to characterize vertex-transitive quotient graphs via their arc diagram.

Lemma 7.31. *For $n \geq 2$, every vertex-transitive quotient graph from \mathcal{V}_n is a Cartesian product with factors from the set of graphs*

$$\mathcal{P} := \{S_2, S_3, S_4, \dots\} \cup \{C_5\}. \quad (7.17)$$

The corresponding reduced arc diagrams are products of empty diagrams on at least 2 points, and of diagrams on 3 points that have either a short left arc or a short right arc.



Proof. Consider the reduced arc diagram of a lattice congruence $R \in \mathcal{V}_n$. By Lemma 7.16, for any loop in the diagram centered at some point $s \in [2, n-1]$, we may split the diagram into two diagrams on the intervals $[1, s]$ and $[s, n]$, and Q_R is the Cartesian product of the quotient graphs of the two lattice congruences defined by the reduced arc diagrams on the two intervals. We repeat this elimination of loops exhaustively, yielding a factorization of $Q_R \simeq Q_{A_1} \square \cdots \square Q_{A_p}$ such that the reduced arc diagram of every lattice congruence $A_i \in \mathcal{C}_{n_i}^*$, $n_i \geq 2$, has no loops. As Q_R is vertex-transitive, Lemma 7.29 (iii) yields that all factors Q_{A_i} must be vertex-transitive. Therefore, by Lemma 7.30, for any factor with $n_i \geq 4$, we know that the arc diagram of A_i must be empty, i.e., we have $Q_{A_i} = S_{n_i}$. For any factor with $n_i = 3$, there are exactly three essential lattice congruences yielding a vertex-transitive quotient graph, given by an arc diagram on 3 points that is either empty, or that has a short left arc or a short right arc, and the corresponding graphs are $Q_{A_i} = S_3$ in the first case, and $Q_{A_i} = C_5$ in the latter two cases. For any factor with $n_i = 2$, there is exactly one essential lattice congruence, represented by an empty arc diagram on 2 points, i.e., we have $Q_{A_i} = S_2$. This proves the lemma. \square

Given an integer n , a *composition* of n is a way to write n as a sum of positive integers a_1, \dots, a_p , i.e., $n = a_1 + \cdots + a_p$. A *partition* of n is a composition of n where the summands are sorted decreasingly, i.e., $a_1 \geq \cdots \geq a_p$.

Theorem 7.32. *For every $n \geq 2$ and every integer composition $a_1 + \cdots + a_p$ of $n-1$ with exactly k many 2s, there are 3^k vertex-transitive quotient graphs in \mathcal{V}_n , and these graphs are isomorphic to the Cartesian products $G_1 \square \cdots \square G_p$, where $G_i = S_{a_i+1}$ if $a_i \neq 2$ and $G_i \in \{S_3, C_5\}$ if $a_i = 2$ for all $i = 1, \dots, p$. The corresponding reduced arc diagrams are products of empty diagrams on a_i+1 points if $a_i \neq 2$, and of diagrams on 3 points that are either empty, or have*

one short left arc, or one short right arc that connects the first and third point if $a_i = 2$. All of these graphs are distinct, and every graph in \mathcal{V}_n arises in this way.

Proof. The proof is illustrated in Figure 7.23. We consider an integer composition $a_1 + \cdots + a_p$ of $n - 1$ with exactly k many 2s. For each summand $a_i \neq 2$, we consider the empty arc diagram on $a_i + 1$ points, and for each summand $a_i = 2$, we consider an arc diagram on 3 points that is either empty, or has one short left arc, or one short right arc. As the latter case happens k times, we have 3^k choices. Consider the lattice congruences A_1, \dots, A_p defined by these arc diagrams, and consider the lattice congruence $R \in \mathcal{C}_n^*$ whose reduced diagram is the product of these diagrams. By Lemma 7.16, we have that $Q_R \simeq Q_{A_1} \square \cdots \square Q_{A_p}$. Moreover, if $a_i \neq 2$, then we have $Q_{A_i} = S_{a_i+1}$, and if $a_i = 2$, then we have $Q_{A_i} \in \{S_3, C_5\}$, i.e., all factors in this product are vertex-transitive. Applying Lemma 7.29 (iii), we see that Q_R is vertex-transitive as well. Clearly, all arc diagrams constructed in this way from integer compositions are distinct, yielding distinct graphs in \mathcal{V}_n . By Lemma 7.31, every graph in \mathcal{V}_n arises from such a composition. \square

The following corollary is an immediate consequence of Theorem 7.32.

Corollary 7.33. *Let $c_{n,k}$ denote the number of integer compositions of n with exactly k many 2s. For $n \geq 2$, we have $|\mathcal{V}_n| = \sum_{k \geq 0} 3^k c_{n-1,k}$.*

Define $a_n := \sum_{k \geq 0} 3^k c_{n-1,k}$ for $n \geq 2$ and $b_n := a_{n+1}$ for $n \geq 1$. The sequence b_n is OEIS sequence A052528, and the first few terms are 1, 4, 8, 22, 52, 132, 324, 808, 2000. This sequence also has a linear recurrence, namely $b_0 = b_1 = 1$ and $b_n = 2b_{n-2} + \sum_{0 \leq i \leq n-1} b_i$ for $n \geq 2$. The generating function is $(1-x)/(2x^3 - 2x^2 - 2x + 1)$, so the asymptotic growth of b_n and a_n is $(1/x_0)^n$, where x_0 is the smallest positive root of $2x^3 - 2x^2 - 2x + 1$, numerically $x_0 \approx 0.403032$ and

$1/x_0 \approx 2.481194$.

Theorem 7.34. *For every $n \geq 2$ and every integer partition $a_1 + \cdots + a_p$ of $n - 1$ with exactly k many 2s, there are $k + 1$ vertex-transitive quotient graphs in \mathcal{V}'_n , and these graphs are the Cartesian products $G_1 \square \cdots \square G_p$, where $G_i = S_{a_i+1}$ if $a_i \neq 2$ and $G_i \in \{S_3, C_5\}$ if $a_i = 2$ for all $i = 1, \dots, p$. The corresponding reduced arc diagrams are products of empty diagrams on $a_i + 1$ points if $a_i \neq 2$, and of k diagrams on 3 points, exactly $j \in \{0, \dots, k\}$ of which are empty and $k - j$ of which have one short left arc that connects the first and third point, if $a_i = 2$. All of these graphs are non-isomorphic, and every graph in \mathcal{V}'_n arises in this way.*

The 8 non-isomorphic vertex-transitive graphs for $n = 5$ are highlighted in Figure 7.23.

Proof. By Theorem 7.32, our task is to consider all integer compositions $a_1 + \cdots + a_p$ of $n - 1$, and among the corresponding Cartesian products $G_1 \square \cdots \square G_p$, select those which are non-isomorphic graphs. By Lemma 7.29 (i), reordering of the factors of any two Cartesian products yields isomorphic graphs, and as all graphs in the set \mathcal{P} defined in (7.17) are prime, Lemma 7.29 (ii) shows that these reordering operations are the only ones yielding isomorphic graphs. Consequently, the non-isomorphic quotient graphs can be identified with integer partitions, obtained by sorting the summands of a composition decreasingly, and for a partition with k many 2s, we may choose $j \in \{0, \dots, k\}$ factors that are 6-cycles S_3 , and the remaining $k - j$ factors as 5-cycles C_5 . \square

Corollary 7.35. *Let t_n denote the number of 2s in all integer partitions of n . For $n \geq 2$, we have $|\mathcal{V}'_n| = t_{n+1}$.*

By Corollary 7.35, the number of vertex-transitive quotient graphs for $n = 2, \dots, 10$ is $t_n = 1, 3, 4, 8, 11, 19, 26, 41, 56$, respectively,

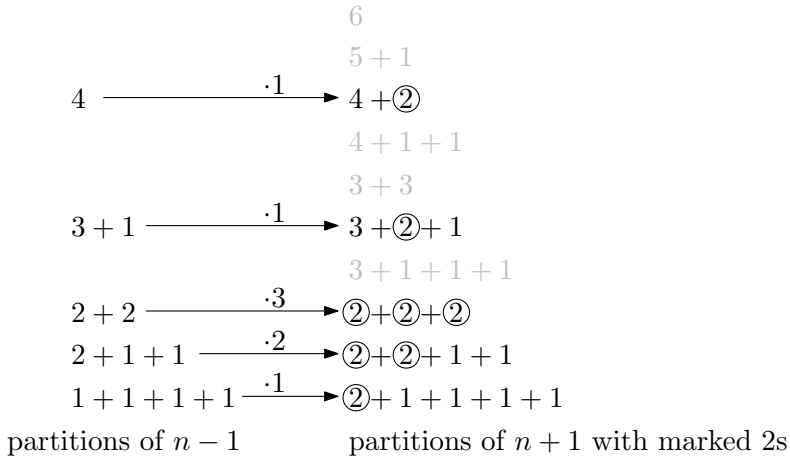


Figure 7.24: Illustration of the proof of Corollary 7.35 for $n = 5$.

which is OEIS sequence A024786. It can be shown that $t_n = e^{\pi\sqrt{2n/3}(1+o(1))}$.

Proof. By Theorem 7.34, there are exactly $\sum_{k \geq 0} (k+1)p_{n-1,k}$ non-isomorphic vertex-transitive quotient graphs \mathcal{V}'_n , where $p_{n,k}$ denotes the number of integer partitions of n with exactly k many 2s. It remains to show that this sum equals t_{n+1} . This argument is illustrated in Figure 7.24. Given any integer partition of $n - 1$ with exactly k many 2s, there are $(k + 1)$ ways to insert another marked 2 into this partition, yielding a partition of $n + 1$ with a marked 2. As all partitions of $n + 1$ with a marked 2 arise in this way, this corresponds exactly to counting the number of 2s in all integer partitions of $n + 1$. \square

Remark 7.36. *The lattice congruences that yield vertex-transitive quotient graphs described in Theorem 7.32 are precisely the δ -permutree congruences from [132] for decorations $\delta \in \{\oplus, \otimes, \ominus, \boxtimes\}^n$ such that any \otimes or \ominus is preceded and followed by \boxtimes .*

7.2.7 Bipartite quotient graphs

Similar to the regular and vertex-transitive quotient graphs, the bipartite quotient graphs can be characterized and counted precisely via their arc diagrams. Specifically, for $a, b \in [n]$ with $a \leq b - 2$, we define a set of arcs *complete with respect to a and b* as the set of all possible arcs connecting a and b , and we denote it by $\text{COMPLETE}(a, b)$. The set of corresponding fences to these arcs is also called complete with respect to a and b . Note that this set contains all 2^{b-a-1} fences that have the form $f(a, b, L)$ where $L \subseteq]a, b[$. In Theorem 7.42 below, we establish that a quotient graph is bipartite if and only if the corresponding reduced arc diagram contains only complete sets of arcs. In other words, the maximal elements of the corresponding downset F_R is a union of complete sets of fences with respect to a_i and b_i , where the $[a_i, b_i]$'s are non-nesting intervals. It follows that the bipartite quotient graphs can be counted by the Catalan numbers; see Corollary 7.43. We also give a count of the number of vertices in a bipartite quotient graph in Lemma 7.44 below. Together with Remark 7.15, this implies that all bipartite quotient graphs have a Hamilton cycle; see Corollary 7.45.

We start with a simple observation, as follows.

Lemma 7.37. *Let $R \in \mathcal{C}_n^*$ be a lattice congruence whose reduced arc diagram contains only complete sets of arcs. Then for any $a, b \in [n]$, if an (a, b) -edge is a bar, then all (a, b) -edges are bars. Moreover, the (a, b) -edges are bars, if and only if $a \leq c < d \leq b$, where $\text{COMPLETE}(c, d)$ is a complete set of arcs in the reduced arc diagram of R .*

Proof. Since an (a, b) -edge is a bar, some fence $f(a, b, L)$ is contained in F_R . If the reduced arc diagram contains $\text{COMPLETE}(a, b)$, then the lemma statement directly follows. Otherwise, it contains an arc corresponding to a fence $f(c, d, L')$ that is above $f(a, b, L)$ in the forcing order (i.e., $a \leq c < d \leq b$ and $L' = L \cap]c, d[$). Since

the reduced arc diagram only contains complete sets of arcs, it also contains $f(c, d, L'' \cap]c, d[)$ for any $L'' \in]a, b[$. Since the reduced arc diagram does not contain $\text{COMPLETE}(a, b)$, we conclude that $f(a, b, L'') \in F_R$ for any $L'' \in]a, b[$. In other words, all (a, b) -edges are bars. Further, it is easy to see that the second statement in the lemma also follows the arguments above. \square

We call a poset $(P, <)$ is *graded of rank k* if all maximal chains in the lattice have the same length k . In that case, there exists a unique rank function $\rho : P \rightarrow [0, k]$ such that $\rho(y) = \rho(x) + 1$ if y covers x in P , and we call $\rho(x)$ the *rank* of x . In the following lemma, we show the gradedness of lattice quotients corresponding to reduced arc diagrams that contain only complete sets of arcs.

Lemma 7.38. *Let $R \in \mathcal{C}_n^*$ be an essential lattice congruence whose reduced arc diagram contains only complete sets of arcs with respect to $(a_1, b_1), \dots, (a_\ell, b_\ell)$. Assume $b_1 < \dots < b_\ell$. Further, define $a_0 = 0, b_0 = 1$, and $b_{\ell+1} = n+1$. Then S_n/R is graded with rank k , where*

$$k := \frac{n(n-1)}{2} - \sum_{j=0}^{\ell} a_j(b_{j+1} - b_j).$$

Proof. Since R is essential, the minimum of S_n/R contains only id_n , and the maximum contains only $\text{di}_n := n(n-1)\dots 1$. Every maximal chain of S_n/R has these two as their endpoints. We can view such a maximal chain as a maximal chain in the weak order where the edges corresponding to bars are contracted. Along one of the maximal chains from id_n to di_n in the weak order, when we traverse an (a, b) -edge for $a < b$, we add (b, a) to the inversion set. Since $\text{inv}(\text{id}_n) = \emptyset$ and $\text{inv}(\text{di}_n) = \{(i, j) \mid n \geq i > j \geq 1\}$, we conclude that for any $1 \leq a < b \leq n$, there is exactly one (a, b) -edge in every maximal chain. By Lemma 7.37, for any such a, b , since the (a, b) -edges are either all bars or all non-bars, we conclude that

all maximal chains of S_n/R have the same length. Hence, S_n/R is graded of rank k , for some k .

We now compute k . As argued above, k is exactly the number of pairs (a, b) , $1 \leq a < b \leq n$, such the (a, b) -edges are not bars, which, by Lemma 7.37, are exactly those where there is no $i \in [\ell]$ such that $a \leq a_i < b_i \leq b$. For $j \in [0, \ell]$ and any $b \in [b_j, b_{j+1} - 1]$, it is easy to see that (a, b) satisfies the above condition if and only if $a_j < a < b$. Therefore, the desired count is

$$\begin{aligned} k &= \sum_{j=0}^{\ell} \sum_{b=b_j}^{b_{j+1}-1} (b - a_j - 1) = \sum_{b=1}^n (b - 1) - \sum_{j=0}^{\ell} a_j (b_{j+1} - b_j) \\ &= \frac{n(n-1)}{2} - \sum_{j=0}^{\ell} a_j (b_{j+1} - b_j) \end{aligned}$$

This completes the proof of the lemma. \square

Remark 7.39. For each $1 \leq a < b \leq n$, the number of (a, b) -edges in the weak order is $(n-1)!$. From the proof of Lemma 7.38 above, we can see that k is the number of the pairs (a, b) such that the (a, b) -edges are not bars. Therefore the number of edges in Q_R is $k(n-1)!$.

We now show the sufficient condition for bipartite quotient graphs.

Corollary 7.40. Let $R \in \mathcal{C}_n^*$ be a lattice congruence whose reduced arc diagram contains only complete sets of arcs. Then Q_R is bipartite.

Proof. By Lemma 7.38, Q_R is graded with rank k . Let ρ be the rank function. We color each equivalence class $X \in S_n/R$ with the color $\rho(X) \pmod{2}$. Since a cover edge connects two equivalence classes of different ranks, no two equivalence classes with the same color are adjacent in Q_R . Hence, Q_R is bipartite, as claimed. \square

The following lemma shows that the condition above is also necessary.

Lemma 7.41. *Let $R \in \mathcal{C}_n^*$ be a lattice congruence whose reduced arc diagram contains an arc which is not in a complete set of arcs. Then Q_R is not bipartite.*

Proof. Let $a, c \in [n]$ with $a < c$ such that the reduced arc diagram of R contains an arc connecting a and c but not $\text{COMPLETE}(a, c)$. Further, we choose a, c such that $c - a$ is minimized. We choose $L, M \subseteq]a, c[$ that satisfy

- (i) one is obtained from the other by removing an element b , and
- (ii) the arc $f(a, c, L)$ is in the reduced arc diagram while the arc $f(a, c, M)$ is not.

Note that (i) implies that L and M form a cover relation in the partial order by inclusion of the subsets of $]a, c[$. A choice of L and M that satisfy (i) and (ii) is then possible, because otherwise, the cover graph of the above partial order contains at least two connected components, one corresponding to the subsets L where $f(a, c, L)$ is in the reduced arc diagram of R and another corresponding to the subsets L where $f(a, c, L)$ is not.

We assume that $L = M \cup \{b\}$; the case where $M = L \cup \{b\}$ is analogous. We consider two cases.

Case 1: $f(a, c, M) \notin F_R$. Let π be the permutation that contains abc as a substring, where the entries in M are on the left of a , and the entries in $]a, c[\setminus M$ are on the right of a . Consider the 6-cycle obtained from π by permutating a, b, c in all possible ways. Due to the forcing constraint as illustrated in Figure 7.2, since $f(a, c, M) \notin F_R$, we can conclude that the edge corresponding to the fence $f(a, c, L)$ is the only bar in this cycle. Hence, the above 6-cycle results in a 5-cycle in Q_R . Consequently, Q_R is not bipartite.

Case 2: $f(a, c, M) \in F_R$. $f(a, c, M)$ then must be forced by an arc $f(d, e, N)$ in the reduced arc diagram of R , where $a \leq d < e \leq c$. Because of the minimality of the difference $c - a$ in the choice of a, c , we conclude that the reduced arc diagram must have $\text{COMPLETE}(d, e)$. In particular, it contains the arc $f(d, e, L \cap]d, e[)$. However, this arc forces $f(a, c, L)$, which contradicts the assumption that $f(a, c, L)$ is in the reduced arc diagram. Hence, this case cannot occur.

The lemma then follows. \square

Corollary 7.40 and Lemma 7.41 straightforwardly imply the following theorem.

Theorem 7.42. *The bipartite quotient graphs \mathcal{B}_n are obtained from exactly those lattice congruences R in \mathcal{C}_n^* that contains only complete sets of arcs in the reduced arc diagram.*

From Theorem 7.42, we obtain the following corollary.

Corollary 7.43. *The number of bipartite quotient graph \mathcal{B}_n is $|\mathcal{B}_n| = C_{n-1}$, where C_n is the n th Catalan number $C_n = \frac{1}{n+1} \binom{2n}{n}$.*

Proof. Using Theorem 7.42, we can count the number of bipartite quotient graphs by counting the number of reduced arc diagrams that have only complete set of arcs. A complete set of arc with respect to two points a, b with $a, b \in [n]$ and $a < b$ can be represented by the interval $[a, b]$. As the lattice congruence is essential, we must have $b - a \leq 2$. Further, the intervals representing the complete sets of arcs should be non-nesting, by the assumption that the diagram is reduced. Therefore, we need to count the sets of non-nesting intervals of length at least 2 between 1 and n . This is exactly the Catalan number C_{n-1} , as discussed in the proof of Corollary 7.26. \square

In Remark 7.15, we state that Algorithm J can generate a Hamilton cycle for a quotient graph, if the number of equivalence classes of each restriction to S_k , for $2 \leq k \leq n-1$, is even. We show that this condition is met for bipartite quotient graphs, and hence all these graphs have a Hamilton cycle. In particular, we can count the number of vertices in the bipartite quotient graphs in the following lemma.

Lemma 7.44. *For $n \geq 2$, let $R \in \mathcal{C}_n^*$ be a lattice congruence whose reduced arc diagrams contain only complete sets of arcs with respect to $(a_1, b_1), \dots, (a_\ell, b_\ell)$ for some $\ell \geq 0$. Assume $b_1 < \dots < b_\ell$. Further, define $a_0 = 0, b_0 = 1$ and $b_{\ell+1} = n+1$. Then the number of vertices of Q_R is*

$$\prod_{i=0}^{\ell} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!}.$$

Proof. We prove this by induction on n . In the base case for $n = 2$, Q_R is an edge and $\ell = 0$. The number of vertices is 2, which agrees with the formula above.

For the inductive step $n > 2$, we count the number of vertices of Q_R by counting the minima of the corresponding equivalence classes. Consider the restriction R^* of R to S_{n-1} . By Lemma 7.9, we can obtain the minima in S_n/R by adding the symbol n to each minimum in S_{n-1}/R^* , such that in the resulting permutation π , no descents are bars (and hence we cannot transpose any descent to obtain an equivalent permutation that is below π in the weak order). By Lemma 7.37, this means that we can add n in front of every symbol x , where for all $i \in [\ell]$, $x \leq a_i < b_i \leq n$ does not hold. In other words, $x \in [a_\ell + 1, n]$. Therefore, for every minimum in S_{n-1}/R^* , we can obtain $n - a_\ell = b_{\ell+1} - a_\ell - 1$ minima in S_n/R , and no two minima obtained this way are the same. Since the number of min-

ima in S_n/R is also the number of vertices in Q_R , we have

$$|V(Q_R)| = (b_{\ell+1} - a_\ell - 1)|V(Q_{R^*})|. \quad (7.18)$$

If $b_\ell = n$, the reduced arc diagram of R^* has the complete sets of arcs with respect to $(a_1, b_1), \dots, (a_{\ell-1}, b_{\ell-1})$. By the inductive hypothesis, Q_{R^*} has $\prod_{i=0}^{\ell-1} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!}$. Then combining with (7.18) and noting that $b_\ell = n = b_{\ell+1} - 1$, we have

$$V(Q_R) = (b_{\ell+1} - a_\ell - 1) \prod_{i=0}^{\ell-1} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!} = \prod_{i=0}^{\ell} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!}.$$

If $b_\ell \neq n$, the reduced arc diagram of R^* has the complete sets of arcs with respect to $(a_1, b_1), \dots, (a_\ell, b_\ell)$. By the inductive hypothesis, Q_{R^*} has $\frac{(n - a_\ell - 1)!}{(b_\ell - a_\ell - 1)!} \prod_{i=0}^{\ell-1} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!}$. Combining with (7.18), we have

$$\begin{aligned} V(Q_R) &= (b_{\ell+1} - a_\ell - 1) \frac{(n - a_\ell - 1)!}{(b_\ell - a_\ell - 1)!} \prod_{i=0}^{\ell-1} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!} \\ &= \prod_{i=0}^{\ell} \frac{(b_{i+1} - a_i - 1)!}{(b_i - a_i - 1)!}. \end{aligned}$$

In either case, we obtain the claimed count, and hence complete the proof. \square

Corollary 7.45. *Each bipartite quotient graph in \mathcal{B}_n for $n \geq 3$ has a Hamilton cycle.*

Proof. We first argue that all bipartite quotient graphs have an even number of vertices. If the arc diagram of R is empty, the number of vertices in Q_R is $n!$, which is even. Otherwise, let b be the smallest number such that the reduced arc diagram of R contains $\text{COMPLETE}(a, b)$ for some a . Since $a \leq b - 2$, $b \geq 3$. By Lemma 7.44,

the number of vertices is divisible by $(b-1)!$, which is even. Hence, all bipartite quotient graphs have an even number of vertices.

For a bipartite quotient graph Q_R in \mathcal{B}_n for $n \geq 3$, by Lemma 7.42, it is easy to see that its restrictions to S_k , for $2 \leq k \leq n-1$ are also bipartite. Hence, the number of equivalence classes in each of these restrictions are even. By Remark 7.15, Algorithm J can then generate a Hamilton cycle for Q_R . The corollary then follows. \square

7.3 Pattern-avoiding permutations and lattice congruences

In Chapter 6, we investigated how Algorithm J can be used to generate different classes of pattern-avoiding permutations. In this section, we briefly comment on the relation between pattern-avoiding permutations and lattice congruences.

Recall the following theorem in Chapter 6:

Theorem 6.3 (restated). *Let F be an arbitrary propositional formula consisting of logical ANDs \wedge , ORs \vee , and tame patterns as variables, then $S_n(F)$, $n \geq 0$, is a hereditary sequence of zigzag languages. Consequently, all of these languages can be generated by Algorithm J.*

Given Theorems 6.3 and 7.12, it is natural to ask: What is the relation between pattern-avoiding permutations and lattice congruences? Can every lattice congruence of the weak order on S_n be realized by an avoidance set of tame patterns? Conversely, does every tame permutation pattern give rise to a lattice congruence? As we discuss next, the answer to the latter two questions is “no” in general, so pattern-avoiding permutations and lattice congruences are essentially different concepts, except in a few special cases, captured by Theorem 7.47 below, and demonstrated by some relevant

examples listed after the theorem.

Firstly, it is clear that every lattice congruence of the weak order on S_n can be described by an avoidance set of patterns of size n , by avoiding all except one permutation from each equivalence class. Note that in general we will not be able to improve on this approach considerably, as the number of lattice congruences grows double-exponentially with n (recall Theorem 7.17), so exponentially many avoidance patterns are needed to describe most lattice congruences. Moreover, the avoidance patterns resulting from this method will in general not be tame. For instance, consider the lattice congruence shown in Figure 7.1, and consider the equivalence class $\{2134, 2314, 2341\}$. It contains two patterns that are not tame, so at least one of them has to be avoided following this approach (even though we know that this particular congruence could be described more compactly by avoiding the tame pattern 231).

Conversely, consider the tame pattern $\tau = 2413$, and suppose we want to realize all pattern-avoiding permutations in S_5 as a lattice congruence. Now consider the permutation $\pi = 25314$, which contains the pattern τ . This means π must be in the same equivalence class with at least one of the four permutations $(\pi_1, \pi_2, \pi_3, \pi_4) = (52314, 23514, 25134, 25341)$ that are obtained from π by adjacent transpositions (recall Lemma 7.1). This means we have to use at least one of the fences $f(2, 5, \emptyset)$, $f(3, 5, \emptyset)$, $f(1, 3, \{2\})$, or $f(1, 4, \{2, 3\})$, respectively, in the forcing order. However, this forces the pairs of permutations $(25341, 52341)$, $(23541, 25341)$, $(52134, 52314)$, or $(52314, 52341)$, respectively, to also be in the same equivalence class (separately for each pair). As none of those permutations contains the pattern τ , we get a contradiction.

We say that a set of vincular patterns P is *well-behaved*, if each pattern $\tau \in P$ of size k can be written as $\tau = A \underline{k1} B$ where AB is a permutation of $\{2, \dots, k-1\}$, and moreover any vincular pattern obtained by permuting the entries within A and within B is

also in P . Note that if A is nonempty, then $\tau = A \underline{k1} B$ is a tame pattern. Here are some examples of well-behaved sets of vincular patterns (cf. 6.1): $P_1 = \{231\}$, $P_2 = \{2413\}$, $P'_2 = \{3412\}$, $P_3 = \{35124, 35142\}$, $P'_3 = \{24513, 42513\}$. The set P_1 is a special case of the family of well-behaved sets of patterns $Q_k := \{\tau \underline{k1} \mid \tau \text{ permutation of } \{2, \dots, k-1\}\}$ for $k \geq 3$ (note that $P_1 = Q_3$). Clearly, this property is preserved under taking unions, so $P_2 \cup P'_2$, $P_3 \cup P'_3$, or $P_1 \cup P_3$ are also well-behaved sets of patterns.

Theorem 7.47. *Let P be a well-behaved set of vincular patterns. For every $\tau = a_1 \cdots a_k \in P$, consider the position i of the largest value k in τ , i.e., $a_i a_{i+1} = \underline{k1}$, and consider the rewriting rule*

$$\begin{aligned} & _x_1_ \cdots _x_{i-1_} x_i x_{i+1} _x_{i+2} \cdots _x_k _ \\ \equiv & _x_1_ \cdots _x_{i-1_} x_{i+1} x_i _x_{i+2} \cdots _x_k _, \end{aligned} \quad (7.19)$$

where the values x_1, \dots, x_k appear in the same relative order as in τ , and which therefore transposes the largest value x_i and the smallest value x_{i+1} in this subpermutation. Combined for all $\tau \in P$, these rewriting rules define a lattice congruence of the weak order on S_n for any $n \geq 1$. Moreover, every equivalence class contains exactly one permutation that avoids every $\tau \in P$, which is the minimum of its equivalence class.

Theorem 7.47 follows from [141, Theorem 9.3]. Here we provide a short independent proof.

Proof. We first show that the set of all pairs of permutations $\pi \succ \rho$ in the weak order on S_n that match one of the rewriting rules (7.19) given by the patterns $\tau \in P$ form a downset of fences in the forcing order.

So suppose we are given $\pi \succ \rho$ in S_n of the form

$$\begin{aligned} \pi &= _x_1_ \cdots _x_{i-1_} x_i x_{i+1} _x_{i+2} \cdots _x_k _, \\ \rho &= _x_1_ \cdots _x_{i-1_} x_{i+1} x_i _x_{i+2} \cdots _x_k _, \end{aligned}$$

where the values x_1, \dots, x_k appear in the same relative order as in one of the patterns $\tau \in P$. Then we have $\pi \equiv \rho$ by (7.19). Furthermore, the cover edge $\rho \triangleleft \pi$ is contained in the fence $f(x_{i+1}, x_i, L)$, where L is the set of all values between x_{i+1} and x_i that appear to the left of x_i and x_{i+1} in π and ρ (in particular, $\{x_1, \dots, x_{i-1}\} \subseteq L$). Let F_{\equiv} be the union of all those fences, taken over all choices of π and ρ and all patterns $\tau \in P$.

Consider a fence $f(a, b, M)$ such that $f(a, b, M) \prec f(x_{i+1}, x_i, L)$, i.e., we have $a \leq x_{i+1}$, $b \geq x_i$, $L = M \cap]x_{i+1}, x_i[$, in particular $x_1, \dots, x_{i-1} \in M$ and $x_{i+2}, \dots, x_k \notin M$. Consider the two permutations $\pi' \succ \rho'$ defined by

$$\begin{aligned}\pi' &:= x_1 \dots x_{i-1} A b a x_{i+2} \dots x_k B, \\ \rho' &:= x_1 \dots x_{i-1} A a b x_{i+2} \dots x_k B,\end{aligned}$$

where A and B are the increasing sequences of the elements in $M \setminus \{x_1, \dots, x_{i-1}\}$ and $[n] \setminus (M \cup \{x_{i+2}, \dots, x_k, a, b\})$, respectively. Then the edge $\rho' \triangleleft \pi'$ is in $f(a, b, M)$. Furthermore, as $a \leq x_{i+1} = \min\{x_1, \dots, x_k\}$ and $b \geq x_i = \max\{x_1, \dots, x_k\}$, the subpermutation $x_1 \dots x_{i-1} b a x_{i+2} \dots x_k$ of π' is a match of the pattern τ and hence $\pi' \equiv \rho'$ by the corresponding rewriting rule, which shows that $f(a, b, M) \in F_{\equiv}$. It follows that F_{\equiv} is a downset of fences in the forcing order, i.e., F_{\equiv} defines a lattice congruence of the weak order on S_n (recall Theorem 7.3).

It remains to show that any two permutations connected by an edge in one of the fences of F_{\equiv} are related by one of our rewriting rules. For this consider a fence $f(a, b, L)$ from F_{\equiv} . By the definition above, there exists a sequence (x_1, \dots, x_k) of numbers from $[n]$ and a pattern $\tau = a_1 \dots a_k \in P$ with the vincular pair at position $(i, i+1)$, i.e., $a_i a_{i+1} = \underline{k1}$, such that x_1, \dots, x_k appear in the same relative order as in τ , $a = x_{i+1}$, $b = x_i$, $x_1, \dots, x_{i-1} \in L$, and $x_{i+2}, \dots, x_k \notin L$. Hence, for any edge $\rho \triangleleft \pi$ in $f(a, b, L)$, we have that $(x_{i+1}, x_i) = (a, b)$ is the pair transposed along this edge, and the values x_1, \dots, x_{i-1}

appear to the left of this pair (not necessarily in this order), and the values x_{i+1}, \dots, x_k appear to the right of this pair (not necessarily in this order), in both ρ and π . As P is well-behaved, P contains all the patterns obtained from τ by permuting a_1, \dots, a_{i-1} to the left of a_i and a_{i+2}, \dots, a_k to the right of a_{i+1} , implying that π matches a pattern from P , and hence there is a rewriting rule witnessing that $\pi \equiv \rho$.

The last part of the theorem follows by interpreting the rewriting rules as a downward orientation of all bars of the lattice congruence \equiv . Note here that the rule (7.19) removes the inversion (x_{i+1}, x_i) , so each equivalence class forms a directed acyclic graph, and the unique sink in it, which must be the minimum, is the permutation that avoids all patterns in P . \square

Applying Theorem 7.47 to the well-behaved sets of patterns mentioned before, $P_1 = \{231\}$ yields the Tamari lattice via the sylvester congruence $_b_ca_ \equiv _b_ac_$ where $a < b < c$, with 231-avoiding permutations as the minima of the equivalence classes; see Figure 7.1 and note that $S_n(231) = S_n(\underline{231})$. More generally, for the set of patterns Q_k defined before, we obtain the increasing flip lattice on acyclic twists studied in [131]. Moreover, $P_2 \cup P'_2 = \{2413, 3412\}$ yields the rotation lattice on diagonal rectangulations [113, 79, 43], with twisted Baxter permutations as minima. Lastly, $P_3 \cup P'_3 = \{35124, 35142, 24513, 42513\}$ yields the lattice on generic rectangulations [144], with 2-clumped permutations as minima.

7.4 Open questions

We conclude with the following open questions:

- We showed that every quotient graph has a Hamilton path, and it is natural to ask whether it also has Hamilton *cycle* for $n \geq 3$; recall Remark 7.15. We conjecture that this is the case, and

we verified this conjecture with computer help for $n \leq 5$. The proof technique described in [96] for the associahedron might be applicable for larger classes of quotientopes.

- Given our results in Table 7.9, it seems challenging to characterize the non-isomorphic quotient graphs \mathcal{Q}'_n by their arc diagrams, and to count them; recall the examples from Figure 7.11. In particular, we wonder whether the sequence $|\mathcal{Q}'_n|$ grows more than exponentially with n , possibly even double-exponentially like $|\mathcal{Q}_n|$?
- It would also be interesting to provide a lower bound on the number of non-isomorphic regular graphs \mathcal{R}'_n that improves upon the trivial bound $|\mathcal{R}'_n| \geq |\mathcal{V}'_n|$, which comes from number partitions. In the proof of Theorem 7.34, we can replace any factor S_{a_i+1} , $a_i \geq 3$, coming from a partition with a part a_i by a prime regular quotient graph. E.g., for $n = 4$ there are 7 non-isomorphic prime regular graphs (10 regular, 4 of which vertex-transitive, 3 of which are products). This technique could be improved, if we produce non-isomorphic prime regular graphs for larger part sizes a_i . To this end, it seems worthwhile to study the set $\mathcal{P}'_n \subseteq \mathcal{R}'_n$ of non-isomorphic prime regular graphs. Probably most arc diagrams with a single simple arc are prime graphs, which would show that $|\mathcal{P}'_n| \geq \Theta(n)$, and this would improve the lower bound for $|\mathcal{R}'_n|$.
- Similarly, the count of non-isomorphic bipartite graphs \mathcal{B}'_n is still open. The numbers of edges and vertices in a bipartite graph (Remark 7.39 and Lemma 7.44) can provide a lower bound. It is also unclear how to characterize quotient graphs beyond two-colorability or whether all quotient graphs are colorable with a constant number of colors. The latter seems difficult, since this question is still open even for associahedra. The best known upper bound for associahedra is $\mathcal{O}(\log(n))$ number of colors [25].
- Lastly, it would also be interesting to investigate which of our results extend to lattice quotients of the weak order on finite Coxeter groups other than the symmetric group S_n (see [143, 146]).

Bibliography

- [1] S. Aaronson. The computational expressiveness of a model train set: A paperlet. <https://scottaaronson.blog/?p=5402>, 2021. Accessed: 6 July 2022.
- [2] 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. doi:10.1016/j.dam.2006.03.018.
- [3] O. Aichholzer, M. K. Chiu, H. P. Hoang, Y. Maus, B. Vogtenhuber, and A. Weinberger. Gioan’s theorem for complete bipartite graphs. In *Abstracts 38th European Workshop on Computational Geometry*, pages 31:1–31:6, 2022.
- [4] M. Albert and R. Brignall. 2×2 monotone grid classes are finitely based. *Discrete Math. Theor. Comput. Sci.*, 18(2):Paper No. 1, 10, 2016.
- [5] M. H. Albert, M. D. Atkinson, M. Bouvel, N. Ruškuc, and V. Vatter. Geometric grid classes of permutations. *Trans. Amer. Math. Soc.*, 365(11):5859–5881, 2013. doi:10.1090/S0002-9947-2013-05804-7.

- [6] R. E. L. Aldred, S. Bau, D. A. Holton, and B. D. McKay. Nonhamiltonian 3-connected cubic planar graphs. *SIAM J. Discrete Math.*, 13(1):25–32, 2000. doi:10.1137/S0895480198348665.
- [7] J. Ani, E. D. Demaine, D. Hendrickson, and J. Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. In P. Mutzel, M. S. Rahman, and Slamin, editors, *WALCOM: Algorithms and Computation*, pages 187–198, Cham, 2022. Springer International Publishing.
- [8] 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.
- [9] M. D. Atkinson. Permutations which are the union of an increasing and a decreasing subsequence. *Electron. J. Combin.*, 5:Research paper 6, 13 pp., 1998. URL: http://www.combinatorics.org/Volume_5/Abstracts/v5i1r6.html.
- [10] D. Auger, P. Coucheney, and L. Duhazé. Polynomial Time Algorithm for ARRIVAL on Tree-Like Multigraphs. In S. Szeider, R. Ganian, and A. Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/opus/volltexte/2022/16810>, doi:10.4230/LIPIcs.MFCS.2022.12.
- [11] S. Avgustinovich, S. Kitaev, V. N. Potapov, and V. Vajnovszki. Gray coding cubic planar maps. *Theoret. Comput. Sci.*, 616:59–69, 2016. doi:10.1016/j.tcs.2015.12.013.

-
- [12] S. Avgustinovich, S. Kitaev, and A. Valyuzhenich. Avoidance of boxed mesh patterns on permutations. *Discrete Appl. Math.*, 161(1-2):43–51, 2013. doi:10.1016/j.dam.2012.08.015.
 - [13] 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 (Grimmetz). doi:10.1016/0166-218X(95)00026-N.
 - [14] D. Avis and D. A. Hoang. On reconfiguration graph of independent sets under token sliding. *arXiv preprint arXiv:2203.16861*, 2022.
 - [15] D. Avis and M. Newborn. On pop-stacks in series. *Utilitas Math.*, 19:129–140, 1981.
 - [16] E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the Mahonian statistics. *Sém. Lothar. Combin.*, 44:Art. B44b, 18, 2000.
 - [17] S. Bacchelli, E. Barucci, E. Grazzini, and E. Pergola. Exhaustive generation of combinatorial objects by ECO. *Acta Inform.*, 40(8):585–602, 2004. doi:10.1007/s00236-004-0139-x.
 - [18] E. Barucci, A. Del Lungo, E. Pergola, and R. Pinzani. ECO: a methodology for the enumeration of combinatorial objects. *J. Differ. Equations Appl.*, 5(4-5):435–490, 1999. doi:10.1080/10236199908808200.
 - [19] J.-L. Baril. Efficient generating algorithm for permutations with a fixed number of excedances. *Pure Math. Appl. (P.U.M.A.)*, 19(2-3):61–69, 2008.

- [20] J.-L. Baril. More restrictive Gray codes for some classes of pattern avoiding permutations. *Inform. Process. Lett.*, 109(14):799–804, 2009. doi:10.1016/j.ipl.2009.03.025.
- [21] J.-L. Baril. Classical sequences revisited with permutations avoiding dotted pattern. *Electron. J. Combin.*, 18(1):Paper 178, 18 pp., 2011.
- [22] D. W. Barnette. Conjecture 5. In *Recent progress in combinatorics. Proceedings of the Third Waterloo Conference on Combinatorics, May 1968. Edited by W. T. Tutte*, pages xiv+347. Academic Press, New York-London, 1969.
- [23] J. J. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA J. Comput.*, 4(4):387–411, 1992. doi:10.1287/ijoc.4.4.387.
- [24] A. Bernini, E. Grazzini, E. Pergola, and R. Pinzani. A general exhaustive generation algorithm for Gray structures. *Acta Inform.*, 44(5):361–376, 2007. doi:10.1007/s00236-007-0053-0.
- [25] L. A. Berry, B. Reed, A. Scott, and D. R. Wood. A logarithmic bound for the chromatic number of the associahedron. *arXiv preprint arXiv:1811.08972*, 2018.
- [26] S. Bhyravarapu, T. A. Hartmann, H. P. Hoang, S. Kalyanasundaram, and I. V. Reddy. Conflict-free coloring: Graphs of bounded clique width and intersection graphs, 2021. URL: <https://arxiv.org/abs/2105.08693>, doi:10.48550/ARXIV.2105.08693.
- [27] S. Billey. Permutation patterns for k -vexillary permutations, 2013. <https://sites.math.washington.edu/~billey/papers/k.vex.html>.

- [28] S. Billey and B. Pawlowski. Permutation patterns, Stanley symmetric functions, and generalized Specht modules. *J. Combin. Theory Ser. A*, 127:85–120, 2014. doi:10.1016/j.jcta.2014.05.003.
- [29] A. Biniarz, K. Jain, A. Lubiw, Z. Masárová, T. Miltzow, D. Mondal, A. M. Naredla, J. Tkadlec, and A. Turcotte. Token swapping on trees. *arXiv preprint arXiv:1903.06981*, 2019.
- [30] 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. doi:10.1145/360336.360343.
- [31] M. Bóna. Exact enumeration of 1342-avoiding permutations: a close link with labeled trees and planar maps. *J. Combin. Theory Ser. A*, 80(2):257–272, 1997. doi:10.1006/jcta.1997.2800.
- [32] M. Bóna. *Combinatorics of permutations*. Discrete Mathematics and its Applications (Boca Raton). CRC Press, Boca Raton, FL, second edition, 2012. With a foreword by Richard Stanley. doi:10.1201/b12210.
- [33] M. Bonamy and N. Bousquet. Token sliding on chordal graphs. In *Graph-theoretic concepts in computer science*, volume 10520 of *Lecture Notes in Comput. Sci.*, pages 127–139. Springer, Cham, 2017. URL: https://doi.org/10.1007/978-3-319-68705-6_10, doi:10.1007/978-3-319-68705-6_10.
- [34] B. Bond and L. Levine. Abelian networks I. Foundations and examples. *SIAM J. Discrete Math.*, 30(2):856–874, 2016. doi:10.1137/15M1030984.

- [35] P. Bose, J. F. Buss, and A. Lubiw. Pattern matching for permutations. *Inform. Process. Lett.*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.
- [36] M. Bousquet-Mélou, A. Claesson, M. Dukes, and S. Kitaev. $(2+2)$ -free posets, ascent sequences and pattern avoiding permutations. *J. Combin. Theory Ser. A*, 117(7):884–909, 2010. doi:10.1016/j.jcta.2009.12.007.
- [37] P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *Electron. J. Combin.*, 18(2):Paper 5, 14 pp., 2011.
- [38] R. Brignall, M. Engen, and V. Vatter. A counterexample regarding labelled well-quasi-ordering. *Graphs Combin.*, 34(6):1395–1409, 2018. doi:10.1007/s00373-018-1962-0.
- [39] A. Burstein and I. Lankham. Restricted patience sorting and barred pattern avoidance. In *Permutation patterns*, volume 376 of *London Math. Soc. Lecture Note Ser.*, pages 233–257. Cambridge Univ. Press, Cambridge, 2010. doi:10.1017/CB09780511902499.013.
- [40] K. Bérczi, H. P. Hoang, and L. Tóthmérész. On approximating the rank of graph divisors, 2022. doi:10.48550/ARXIV.2206.09662.
- [41] J. Cardinal and S. Felsner. Topological drawings of complete bipartite graphs. *J. Comput. Geom.*, 9(1):213–246, 2018.
- [42] J. Cardinal, A. Merino, and T. Mütze. Combinatorial generation via permutation languages. IV. Elimination trees. <https://arxiv.org/abs/2106.16204>, 2021.
- [43] 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.

- [44] C. Ceballos, F. Santos, and G. M. Ziegler. Many non-equivalent realizations of the associahedron. *Combinatorica*, 35(5):513–551, 2015. doi:10.1007/s00493-014-2959-9.
- [45] I. Chajda and V. Snášel. Congruences in ordered sets. *Math. Bohem.*, 123(1):95–100, 1998.
- [46] A. Chalcraft and M. Greene. Train sets. *Eureka*, 53:5–12, 1994.
- [47] G. Chatel and V. Pilaud. Cambrian Hopf algebras. *Adv. Math.*, 311:598–633, 2017. doi:10.1016/j.aim.2017.02.027.
- [48] J. Chen, Y. Liu, S. Lu, B. O’Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):Art. 21, 19, 2008.
- [49] K. L. Chung and W. Feller. On fluctuations in coin-tossing. *Proceedings of the National Academy of Sciences*, 35(10):605–608, 1949.
- [50] A. Claesson and S. Kitaev. Classification of bijections between 321- and 132-avoiding permutations. In *20th Annual International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2008)*, Discrete Math. Theor. Comput. Sci. Proc., AJ, pages 495–506. Assoc. Discrete Math. Theor. Comput. Sci., Nancy, 2008.
- [51] A. Claesson, S. Kitaev, and E. Steingrímsson. Decompositions and statistics for $\beta(1, 0)$ -trees and nonseparable permutations. *Adv. in Appl. Math.*, 42(3):313–328, 2009. doi:10.1016/j.aam.2008.09.001.
- [52] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.
- [53] J. Cooper, B. Doerr, J. Spencer, and G. Tardos. Deterministic random walks on the integers. *European Journal of Com-*

- binatorics*, 28(8):2072 – 2090, 2007. EuroComb '05 - Combinatorics, Graph Theory and Applications.
- [54] C. Dang, Q. Qi, and Y. Ye. Computations and complexities of Tarski's fixed points and supermodular games, 2020. [arXiv: 2005.09836](https://arxiv.org/abs/2005.09836).
- [55] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.
- [56] M. de Berg, B. M. P. Jansen, and D. Mukherjee. Independent-set reconfiguration thresholds of hereditary graph classes. *Discrete Appl. Math.*, 250:165–182, 2018. doi:10.1016/j.dam.2018.05.029.
- [57] L. Demonet, O. Iyama, N. Reading, I. Reiten, and H. Thomas. Lattice theory of torsion classes. To appear in *Trans. Amer. Math. Soc.*; preprint available at <https://arxiv.org/abs/1711.01785>, 2018.
- [58] E. Deutsch, E. Munarini, and S. Rinaldi. Skew Dyck paths. *J. Statist. Plann. Inference*, 140(8):2191–2203, 2010. doi:10.1016/j.jspi.2010.01.015.
- [59] P. Diaconis and R. Graham. *Magical mathematics*. Princeton University Press, Princeton, NJ, 2012. The mathematical ideas that animate great magic tricks, With a foreword by Martin Gardner.
- [60] P. T. Do, D. Rossin, and T. T. H. Tran. Permutations weakly avoiding barred patterns and combinatorial bijections to generalized Dyck and Motzkin paths. *Discrete Math.*, 320:40–50, 2014. doi:10.1016/j.disc.2013.12.007.
- [61] P. T. Do, T. T. H. Tran, and V. Vajnovszki. Exhaustive generation for permutations avoiding (colored)

- regular sets of patterns. *Discrete Applied Mathematics*, 2019. URL: <http://www.sciencedirect.com/science/article/pii/S0166218X19302240>, doi:10.1016/j.dam.2019.04.014.
- [62] J. Dohrau, B. Gärtner, M. Kohler, J. Matoušek, and E. Welzl. ARRIVAL: a zero-player graph game in $NP \cap coNP$. In *A journey through discrete mathematics*, pages 367–374. Springer, Cham, 2017.
- [63] G. Dorfer. Lattice-extensions by means of convex sublattices. In *Contributions to general algebra, 9 (Linz, 1994)*, pages 127–132. Hölder-Pichler-Tempsky, Vienna, 1995.
- [64] W. M. B. Dukes, M. F. Flanagan, T. Mansour, and V. Vajnovszki. Combinatorial Gray codes for classes of pattern avoiding permutations. *Theoret. Comput. Sci.*, 396(1-3):35–49, 2008. doi:10.1016/j.tcs.2007.12.002.
- [65] S. Dulucq, S. Gire, and O. Guibert. A combinatorial proof of J. West’s conjecture. *Discrete Math.*, 187(1-3):71–96, 1998. doi:10.1016/S0012-365X(98)80005-8.
- [66] S. Dulucq, S. Gire, and J. West. Permutations with forbidden subsequences and nonseparable planar maps. In *Proceedings of the 5th Conference on Formal Power Series and Algebraic Combinatorics (Florence, 1993)*, volume 153, pages 85–103, 1996. doi:10.1016/0012-365X(95)00130-0.
- [67] M. Elder. Permutations generated by a stack of depth 2 and an infinite stack in series. *Electron. J. Combin.*, 13(1):Research Paper 68, 12 pp., 2006. URL: http://www.combinatorics.org/Volume_13/Abstracts/v13i1r68.html.
- [68] S. Elizalde. Generating trees for permutations avoiding generalized patterns. *Ann. Comb.*, 11(3-4):435–458, 2007. doi:10.1007/s00026-007-0328-8.

- [69] S. Elizalde. The X-class and almost-increasing permutations. *Ann. Comb.*, 15(1):51–68, 2011. doi:10.1007/s00026-011-0082-9.
- [70] K. Etessami, C. Papadimitriou, A. Rubinstein, and M. Yannakakis. Tarski’s Theorem, Supermodular Games, and the Complexity of Equilibria. In T. Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2020.18.
- [71] J. Fearnley, M. Gairing, M. Mnich, and R. Savani. Reachability switching games. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 124, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
- [72] J. Fearnley, M. Gairing, M. Mnich, and R. Savani. Reachability switching games. *Log. Methods Comput. Sci.*, 17(2):Paper No. 10, 29, 2021.
- [73] J. Fearnley, S. Gordon, R. Mehta, and R. Savani. Unique end of potential line. *J. Comput. System Sci.*, 114:1–35, 2020.
- [74] J. Fearnley, D. Pálvölgyi, and R. Savani. A faster algorithm for finding tarski fixed points. *ACM Trans. Algorithms*, mar 2022. Just Accepted. doi:10.1145/3524044.
- [75] A. Fink, K. Mészáros, and A. S. Dizier. Zero-one Schubert polynomials. *Math. Z.*, 297(3-4):1023–1042, 2021. doi:10.1007/s00209-020-02544-2.
- [76] B. Gärtner, T. D. Hansen, P. Hubáček, K. Král, H. Mosaad, and V. Slívová. ARRIVAL: next stop in CLS. In *45th International Colloquium on Automata, Languages, and Pro-*

- gramming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 60, 13. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
- [77] B. Gärtner, S. Haslebach, and H. P. Hoang. A subexponential algorithm for ARRIVAL. In *48th International Colloquium on Automata, Languages, and Programming*, volume 198 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 69, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2021.
- [78] C. Ge. The multi-run procedure in ARRIVAL. Bachelor’s Thesis, ETH Zurich, 2021.
- [79] S. Giraudo. Algebraic and combinatorial structures on pairs of twin binary trees. *J. Algebra*, 360:115–157, 2012. doi:10.1016/j.jalgebra.2012.03.020.
- [80] I. P. Goulden and J. West. Raney paths and a combinatorial relationship between rooted nonseparable planar maps and two-stack-sortable permutations. *J. Combin. Theory Ser. A*, 75(2):220–242, 1996. doi:10.1006/jcta.1996.0074.
- [81] F. Gray. Pulse code communication, 1953. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.
- [82] B. Grünbaum. Polytopes, graphs, and complexes. *Bull. Amer. Math. Soc.*, 76:1131–1201, 1970. doi:10.1090/S0002-9904-1970-12601-5.
- [83] 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. URL: <https://0-doi-org.pugwash.lib.warwick.ac.uk/10.1090/tran/8199>, doi:10.1090/tran/8199.
- [84] E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. In *Proceedings*

- of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 1214–1225. SIAM, Philadelphia, PA, 2020.
- [85] S. Haslebacher. Restrictions on ARRIVAL. Bachelor’s Thesis, ETH Zurich, 2020.
- [86] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoret. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.
- [87] R. A. Hearn and E. D. Demaine. *Games, puzzles, and computation*. CRC Press, 2009.
- [88] H. P. Hoang, S. Lendl, and L. Wulf. Assistance and interdiction problems on interval graphs. *CoRR*, abs/2107.14550, 2021. arXiv:2107.14550.
- [89] H. P. Hoang and T. Mütze. Combinatorial generation via permutation languages. II. Lattice congruences. *Israel J. Math.*, 244(1):359–417, 2021. URL: <https://0-doi-org.pugwash.lib.warwick.ac.uk/10.1007/s11856-021-2186-1>, doi:10.1007/s11856-021-2186-1.
- [90] C. Hohlweg and C. E. M. C. Lange. Realizations of the associahedron and cyclohedron. *Discrete Comput. Geom.*, 37(4):517–543, 2007. doi:10.1007/s00454-007-1319-6.
- [91] B. Holdsworth and C. Woods. *Digital logic design*. Elsevier, 2002.
- [92] A. E. Holroyd, L. Levine, K. Mészáros, Y. Peres, J. Propp, and D. B. Wilson. *Chip-Firing and Rotor-Routing on Directed Graphs*, pages 331–364. Birkhäuser Basel, Basel, 2008. doi:10.1007/978-3-7643-8786-0_17.

- [93] A. E. Holroyd and J. Propp. Rotor walks and Markov chains. In *Algorithmic probability and combinatorics*, volume 520 of *Contemp. Math.*, pages 105–126. Amer. Math. Soc., Providence, RI, 2010.
- [94] S. Huczynska and V. Vatter. Grid classes and the Fibonacci dichotomy for restricted permutations. *Electron. J. Combin.*, 13(1):Research Paper 54, 14 pp., 2006. URL: http://www.combinatorics.org/Volume_13/Abstracts/v13i1r54.html.
- [95] B. Hujter, V. Kiss, and L. Tóthmérész. On the complexity of the chip-firing reachability problem. *Proc. Amer. Math. Soc.*, 145(8):3343–3356, 2017. doi:10.1090/proc/13498.
- [96] F. Hurtado and M. Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Comput. Geom.*, 13(3):179–188, 1999. doi:10.1016/S0925-7721(99)00016-4.
- [97] W. Imrich and S. Klavžar. *Product graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 2000. Structure and recognition, With a foreword by Peter Winkler.
- [98] M. Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003. doi:10.1007/978-3-0348-8005-3.
- [99] S. Johnson. Generation of permutations by adjacent transposition. *Math. Comp.*, 17:282–285, 1963.
- [100] M. Jurdziński. Deciding the winner in parity games is in $UP \cap co-UP$. *Information Processing Letters*, 68(3):119 – 124, 1998.

- [101] M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoret. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.
- [102] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103, 1972.
- [103] C. S. Karthik. Did the train reach its destination: the complexity of finding a witness. *Inform. Process. Lett.*, 121:17–21, 2017.
- [104] R. Kaye. A Gray code for set partitions. *Information Processing Lett.*, 5(6):171–173, 1976.
- [105] S. Kitaev. Partially ordered generalized patterns. *Discrete Math.*, 298(1-3):212–229, 2005. doi:10.1016/j.disc.2004.03.017.
- [106] S. Kitaev. *Patterns in permutations and words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg, 2011. With a foreword by Jeffrey B. Remmel. doi:10.1007/978-3-642-17333-2.
- [107] D. E. Knuth. *The Art of Computer Programming. Vol. 3*. Addison-Wesley, Reading, MA, 1998. Sorting and searching, Second edition [of MR0445948].
- [108] D. E. Knuth. *The Art of Computer Programming. Vol. 4A. Combinatorial algorithms. Part 1*. Addison-Wesley, Upper Saddle River, NJ, 2011.
- [109] M. Kolibiar. Congruence relations and direct decompositions of ordered sets. *Acta Sci. Math. (Szeged)*, 51(1-2):129–135, 1987.

- [110] C. Lange and V. Pilaud. Associahedra via spines. *Combinatorica*, 38(2):443–486, 2018. doi:10.1007/s00493-015-3248-y.
- [111] I. P. Lankham. *Patience sorting and its generalizations*. ProQuest LLC, Ann Arbor, MI, 2007. Thesis (Ph.D.)—University of California, Davis. URL: <https://www.proquest.com/docview/304901553>.
- [112] A. Lascoux and M.-P. Schützenberger. Schubert polynomials and the Littlewood-Richardson rule. *Lett. Math. Phys.*, 10(2-3):111–124, 1985. doi:10.1007/BF00398147.
- [113] S. Law and N. Reading. The Hopf algebra of diagonal rectangulations. *J. Combin. Theory Ser. A*, 119(3):788–824, 2012. doi:10.1016/j.jcta.2011.09.006.
- [114] C. W. Lee. The associahedron and triangulations of the n -gon. *European J. Combin.*, 10(6):551–560, 1989. doi:10.1016/S0195-6698(89)80072-1.
- [115] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
- [116] Y. Li and J. Sawada. Gray codes for reflectable languages. *Inform. Process. Lett.*, 109(5):296–300, 2009. doi:10.1016/j.ipl.2008.11.007.
- [117] J.-L. Loday. Realization of the Stasheff polytope. *Arch. Math. (Basel)*, 83(3):267–278, 2004. doi:10.1007/s00013-004-1026-y.
- [118] D. Lokshtanov and A. E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):Art. 7, 19, 2019. doi:10.1145/3280825.
- [119] L. Lovász. Problem 11, in Combinatorial structures and their applications. In *Proc. Calgary Internat. Conf. (Calgary, Al-*

- berta*, 1969), pages 243–246, New York, 1970. Gordon and Breach Science Publishers.
- [120] J. M. Lucas. The rotation graph of binary trees is Hamiltonian. *J. Algorithms*, 8(4):503–535, 1987. doi:10.1016/0196-6774(87)90048-4.
- [121] J. M. Lucas, D. Roelants van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366, 1993. doi:10.1006/jagm.1993.1045.
- [122] G. Manuell. A simple lower bound for arrival. *arXiv preprint arXiv:2108.06273*, 2021.
- [123] N. Matsumoto, A. Nakamoto, and S. Negami. Diagonal flips in plane graphs with triangular and quadrangular faces. *Discrete Appl. Math.*, 283:292–305, 2020. doi:10.1016/j.dam.2020.01.007.
- [124] A. Merino and T. Mütze. Combinatorial generation via permutation languages. III. Rectangulations. To appear in *Discrete Comput. Geom.*; preprint available at <https://arxiv.org/abs/2103.09333>, 2021.
- [125] T. Mütze. Combinatorial gray codes—an updated survey. *arXiv preprint arXiv:2202.01280*, 2022.
- [126] N. Nishimura. Introduction to reconfiguration. *Algorithms (Basel)*, 11(4):Paper No. 52, 25, 2018. doi:10.3390/a11040052.
- [127] J. Noonan and D. Zeilberger. The enumeration of permutations with a prescribed number of “forbidden” patterns. *Adv. in Appl. Math.*, 17(4):381–407, 1996. doi:10.1006/aama.1996.0016.

- [128] J. R. Norris. *Markov chains*, volume 2 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 1998. Reprint of 1997 original.
- [129] OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2020. <http://oeis.org>.
- [130] R. Parviainen. Wilf classification of bi-vincular permutation patterns. <https://arxiv.org/abs/0910.5103>, 2009.
- [131] V. Pilaud. Brick polytopes, lattice quotients, and Hopf algebras. *J. Combin. Theory Ser. A*, 155:418–457, 2018. doi: 10.1016/j.jcta.2017.11.014.
- [132] V. Pilaud and V. Pons. Permutrees. *Algebr. Comb.*, 1(2):173–224, 2018.
- [133] V. Pilaud and F. Santos. The brick polytope of a sorting network. *European J. Combin.*, 33(4):632–662, 2012. doi: 10.1016/j.ejc.2011.12.003.
- [134] V. Pilaud and F. Santos. Quotientopes. *Bull. Lond. Math. Soc.*, 51(3):406–420, 2019. doi:10.1112/blms.12231.
- [135] L. Pournin. The diameter of associahedra. *Adv. Math.*, 259:13–42, 2014. doi:10.1016/j.aim.2014.02.035.
- [136] V. B. Priezzhev, D. Dhar, A. Dhar, and S. Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Phys. Rev. Lett.*, 77:5079–5082, 1996.
- [137] L. K. Pudwell. *Enumeration schemes for pattern-avoiding words and permutations*. ProQuest LLC, Ann Arbor, MI, 2008. Thesis (Ph.D.)—Rutgers The State University of New Jersey - New Brunswick. URL: http://gateway.proquest.com/openurl?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&res_dat=xri:pqdiss&rft_dat=xri:pqdiss:3335550.

- [138] L. K. Pudwell. Enumeration schemes for permutations avoiding barred patterns. *Electron. J. Combin.*, 17(1):Research Paper 29, 27 pp., 2010. URL: http://www.combinatorics.org/Volume_17/Abstracts/v17i1r29.html.
- [139] N. Reading. Order dimension, strong Bruhat order and lattice properties for posets. *Order*, 19(1):73–100, 2002. doi:10.1023/A:1015287106470.
- [140] N. Reading. Lattice and order properties of the poset of regions in a hyperplane arrangement. *Algebra Universalis*, 50(2):179–205, 2003. doi:10.1007/s00012-003-1834-0.
- [141] N. Reading. Lattice congruences, fans and Hopf algebras. *J. Combin. Theory Ser. A*, 110(2):237–273, 2005. doi:10.1016/j.jcta.2004.11.001.
- [142] N. Reading. Cambrian lattices. *Adv. Math.*, 205(2):313–353, 2006. doi:10.1016/j.aim.2005.07.010.
- [143] N. Reading. From the Tamari lattice to Cambrian lattices and beyond. In *Associahedra, Tamari lattices and related structures*, volume 299 of *Prog. Math. Phys.*, pages 293–322. Birkhäuser/Springer, Basel, 2012. doi:10.1007/978-3-0348-0405-9_15.
- [144] N. Reading. Generic rectangulations. *European J. Combin.*, 33(4):610–623, 2012. doi:10.1016/j.ejc.2011.11.004.
- [145] N. Reading. Noncrossing arc diagrams and canonical join representations. *SIAM J. Discrete Math.*, 29(2):736–750, 2015. doi:10.1137/140972391.
- [146] N. Reading. Finite Coxeter groups and the weak order. In *Lattice theory: special topics and applications. Vol. 2*, pages 489–561. Birkhäuser/Springer, Cham, 2016.

- [147] N. Reading. Lattice theory of the poset of regions. In *Lattice theory: special topics and applications. Vol. 2*, pages 399–487. Birkhäuser/Springer, Cham, 2016.
- [148] T. Rokicki, H. Kociemba, M. Davidson, and J. Dethridge. The diameter of the Rubik’s cube group is twenty [reprint of mr3068558]. *SIAM Rev.*, 56(4):645–670, 2014. doi:10.1137/140973499.
- [149] G. Rote. Personal communication, 2020.
- [150] F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *J. Combin. Theory Ser. A*, 119(1):155–169, 2012. doi:10.1016/j.jcta.2011.07.005.
- [151] C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295292.
- [152] J. Sawada and A. Williams. Efficient oracles for generating binary bubble languages. *Electron. J. Combin.*, 19(1):Paper 42, 20 pp., 2012.
- [153] A. Seidenberg. A simple proof of a theorem of Erdős and Szekeres. *J. London Math. Soc.*, 34:352, 1959. doi:10.1112/jlms/s1-34.3.352.
- [154] R. Smith and V. Vatter. A stack and a pop stack in series. *Australas. J. Combin.*, 58:157–171, 2014.
- [155] Z. E. Stankova. Forbidden subsequences. *Discrete Math.*, 132(1-3):291–316, 1994. doi:10.1016/0012-365X(94)90242-9.
- [156] J. M. Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In *Discrete probability and algorithms (Minneapolis, MN, 1993)*, volume 72 of *IMA Vol. Math. Appl.*, pages 111–131. Springer, New York, 1995. doi:10.1007/978-1-4612-0801-3_9.

- [157] D. Tamari. The algebra of bracketings and their enumeration. *Nieuw Arch. Wisk. (3)*, 10:131–146, 1962.
- [158] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.
- [159] B. Tenner. Database of permutation pattern avoidance, 2018. <https://math.depaul.edu/bridget/patterns.html>.
- [160] L. Tóthmérés. Rotor-routing reachability is easy, chip-firing reachability is hard. *European J. Combin.*, 101:Paper No. 103466, 9, 2022. doi:10.1016/j.ejc.2021.103466.
- [161] L. Trevisan. Approximation algorithms for unique games. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 197–205, 2005.
- [162] H. F. Trotter. Algorithm 115: Perm. *Commun. ACM*, 5(8):434–435, 1962. URL: <http://doi.acm.org/10.1145/368637.368660>, doi:10.1145/368637.368660.
- [163] H. Úlfarsson. A unification of permutation patterns related to Schubert varieties. *Pure Math. Appl. (P.U.M.A.)*, 22(2):273–296, 2011.
- [164] V. Vajnovszki. Generating involutions, derangements, and relatives by ECO. *Discrete Math. Theor. Comput. Sci.*, 12(1):109–122, 2010.
- [165] V. Vajnovszki and R. Vernay. Restricted compositions and permutations: from old to new Gray codes. *Inform. Process. Lett.*, 111(13):650–655, 2011. doi:10.1016/j.ipl.2011.03.022.
- [166] J. van den Heuvel. The complexity of change. In *Surveys in combinatorics 2013*, volume 409 of *London Math. Soc. Lecture Note Ser.*, pages 127–160. Cambridge Univ. Press, Cambridge, 2013.

- [167] A. Vella. Pattern avoidance in permutations: linear and cyclic orders. *Electron. J. Combin.*, 9(2):Research paper 18, 43 pp., 2002/03. Permutation patterns (Otago, 2003). URL: http://www.combinatorics.org/Volume_9/Abstracts/v9i2r18.html.
- [168] U. Wagner and E. Welzl. Connectivity of triangulation flip graphs in the plane (Part I: Edge flips). In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 2823–2841. SIAM, Philadelphia, PA, 2020.
- [169] S. D. Waton. *On permutation classes defined by token passing networks, gridding matrices and pictures: three flavours of involvement*. PhD thesis, University of St Andrews, 2007.
- [170] J. West. *Permutations with forbidden subsequences and stack-sortable permutations*. ProQuest LLC, Ann Arbor, MI, 1990. Thesis (Ph.D.)—Massachusetts Institute of Technology.
- [171] 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. doi:10.1007/978-3-642-40104-6_46.
- [172] A. Woo and A. Yong. When is a Schubert variety Gorenstein? *Adv. Math.*, 207(1):205–220, 2006. doi:10.1016/j.aim.2005.11.010.
- [173] L. Xiang, K. Cheng, and K. Ushijima. Efficient generation of Gray codes for reflectable languages. In *Computational Science and Its Applications - ICCSA 2010, International Conference, Fukuoka, Japan, March 23-26, 2010, Proceedings, Part IV*, pages 418–426, 2010. doi:10.1007/978-3-642-12189-0_37.

- [174] 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. doi:10.1145/606603.606607.
- [175] D. Zeilberger. A proof of Julian West’s conjecture that the number of two-stack-sortable permutations of length n is $2(3n)!/((n+1)!(2n+1)!)$. *Discrete Math.*, 102(1):85–93, 1992. doi:10.1016/0012-365X(92)90351-F.
- [176] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.
- [177] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoret. Comput. Sci.*, 158(1-2):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

Curriculum Vitae

Hung Hoang

born 2 November 1989

citizen of Vietnam

- | | |
|-----------|--|
| 2018–2022 | Ph.D. in Theoretical Computer Science
ETH Zürich |
| 2016–2017 | M.Sc. in Management Science (Operational Research)
London School of Economics and Political Science |
| 2014–2016 | Business Analyst
Tata Capital Advisors Pte Ltd, Singapore |
| 2013–2014 | Audit Senior
Deloitte & Touche LLP, Singapore |
| 2011–2013 | Audit Associate
Deloitte & Touche LLP, Singapore |
| 2007–2011 | Bachelor of Business Administration (Accountancy)
National University of Singapore |

About the cover art

The cover art is adapted from the artwork submitted to the Intercultural Science-Art Project, started at the 8th Heidelberg Laureate Forum in 2021. My friend Đỗ Công Lý created this, after our inspiring conversation on combinatorial reconfiguration, a common theme across many problems in my research and the topic of this thesis.

The title of the artwork, "Dịch biến - Diện bích", features spoonerism, a common word play in Vietnamese where we swap the components of two words to form two new words. The first two words, "dịch biến", mean "movement and transformation", capturing this strand of research (and arguably the area of graph algorithms in general). The second two words, "diện bích", mean "wall gazing". It is commonly used in the phrase "cửu niên diện bích" (nine years of wall gazing), which refers to the tale of Bodhidharma facing the wall of a cave and meditating for nine years. I find it an interesting word play, as one part refers to change, while the other to distancing oneself from the changes in the outside world.

While the wall can represent the nine years in the tale, it also resembles the Rubik's cube, a popular object in combinatorial reconfiguration, as discussed in Chapter 1. The meditation of the monk in front of this wall can then symbolize the research in this field.