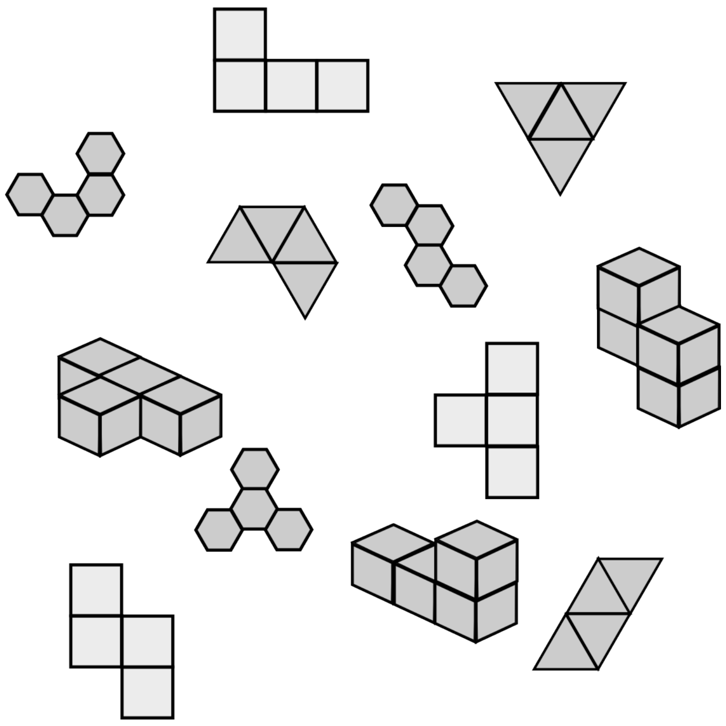
Enumeration of Lattice Animals



Gadi Aleksandrowicz

Enumeration of Lattice Animals

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Gadi Aleksandrowicz

Submitted to the Senate of
the Technion — Israel Institute of Technology
Sivan 5771 Haifa June 2011

The research thesis was done under the supervision of Prof. Gill Barequet in the Computer Science Department.

First and foremost, I wish to thank my advisor, Prof. Gill Barequet, that guided me ever since the middle of my bachelor degree to the end of my doctorate studies. Gill always surprised me with his ability to see order and patterns where I have only seen chaos, and in his creative methods for dealing with problems that inspired me throughout my studies. He was also an excellent guide for the complexities of the academic world.

I would like to thank many more lecturers that awoke in me the enthusiasm and love for Mathematics and Computer Science. Especially I would like to mention Prof. Eyal Kushilevitz that taught me several courses in complexity theory; Prof. Ron Aharoni that taught me mathematical logic; and Prof. Eli Biham that taught me cryptography. Many other lecturers taught me in an inspiring way and their door was always open for my questions. Finally, I would like to thank Prof. David Harel; although he has never taught me personally, his book "Algorithmics" was first to reveal to me the beauty of computer science, and much thanks to it I turned to study them.

I want to thank my family that always supported me, in particular in my studies, and that already at a very early age awoke in me the love of knowledge and enjoyment of study.

Finally I want to thank my partner, Adi Wolf, that accompanied me since the beginning of my Bachelor's degree, and thanks to her this was the most enjoyable period of my life so far.

The generous financial support of the Technion and the Fein and Jacobs funds is gratefully acknowledged.

Publication List

- G. Aleksandrowicz and G. Barequet. Counting d-dimensional polycubes and nonrectangular planar polyominoes. *Computing and Combinatorics,* 2006:418-427. Full version: *Int. J. of Computational Geometry and Applications*, 19(3):215–229, 2009.

- G. Aleksandrowicz and G. Barequet. Counting polycubes without the dimensionality curse. *Computing and Combinatorics,* 2008:100-109. Full version: *Discrete Mathematics,* 309(13):4576–4583, 2009.

- G. Aleksandrowicz and G. Barequet. Redelmeier's algorithm for counting lattice animals. Video review at *Symposium on Computational Geometry,* 2011:283-284

- G. Aleksandrowicz and G. Barequet. Parallel enumeration of lattice animals. *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management,* 2011:90-99

- G. Aleksandrowicz and G. Barequet. The growth rate of high-dimensional tree polycubes. *EuroComb,* 2011, to appear.

- G. Aleksandrowicz and G. Barequet. A polyomino-permutation injection and tree-like convex polyominoes (poster). *EuroComb,* 2011, to appear.

- G. Aleksandrowicz, A. Asinowski, G. Barequet and R. Barequet. Enumerating polyominoes on twisted cylinders, submitted.

- G. Aleksandrowicz, A. Asinowski and G. Barequet, Enumerating fixed tree-like convex polyominoes, submitted.

Contents

List of Figures

Abstract

Lattice animals, which can be defined as connected subgraphs of the dual graph of a given lattice, are well-known combinatorial objects, studied both in pure and recreational mathematics, and in various different fields, such a statistical physics and computational geometry. Of particular interest are lattice animals on the standard rectangular lattice (called *polyominoes* in the two dimensional case and *polycubes* for higher dimensions).

In this work, we concern ourselves with the problem of *enumerating* lattice animals: computing, or estimating, how many lattice animals exist for a given type and size. This is a difficult combinatorial problem; even for the relatively simple case of the two-dimensional rectangular lattice not much is known, and other lattices are much less studied.

We investigate the problem in several directions:

Enumeration algorithms: We describe an algorithm for enumerating directly lattice animals on every possible lattice. Using a parallel version of the algorithm we counted many types of lattice animals and found many results never given in the literature so far.

Analytical analysis: We describe formulae for several sequences of lattice animals (specifically, some of the diagonals in a table of polycubes, arranged by size and dimension).

Transfer-matrix methods: We describe a method for computing algorithmically the generating function of lattice animals on a so-called "twisted cylinder". These animals are of special interest since there are fewer of them than regular polyominoes (of any size n), thus, their growth-rate limit is a lower bound to the growth-rate of polyominoes (whose computation is one of the main problems in the field).

Bijection with permutations: We describe a novel method of identifying polyominoes with permutations, and characterizing classes of polyominoes using forbidden permutation patterns.

Chapter 1

Introduction

A *domino* consists of two squares joined together along one edge. A *polyomino* is a generalization of the concept. A polyomino of size n consists of n squares joined along their edges. Formally, a polyomino can be defined as a connected graph whose vertices are points of $\mathbb{Z}^2$, the integer lattice, and edges which are lines of unit size. A few examples for polyominoes are shown in Figure 1.1.

The concept can be generalized by considering connected sets on different kinds of lattices and geometric figures. Polyominoes in the d-dimensional orthogonal lattice (formally, connected graphs on $\mathbb{Z}^d$) are usually referred to as *polycubes*.[1] Polyominoes can be considered in the two-dimensional triangular or hexagonal lattice too; they can even be embedded in surfaces such as a cylinder or a torus; their cells can have different weights (in the standard case, an occupied cell has weight 1 and an empty cell has weight 0); and there are many more variations of the basic definition.

Polyominoes frequently appear in recreational mathematical games, especially in the context of tiling problems, for at least the past hundred years. A main example is the computer game Tetris, which features the polyominoes of size 4. Mathematical investigations of polyominoes began with the seminal work of Solomon Golomb and especially his book titled "polyominoes" [16]; the term "polyomino" itself (a play on the words "poly" and "domino") was invented by Golomb.

The term *Lattice animals* is also used (mainly in the physics literature) to refer to

[1]Term coined by Lunnon [24].

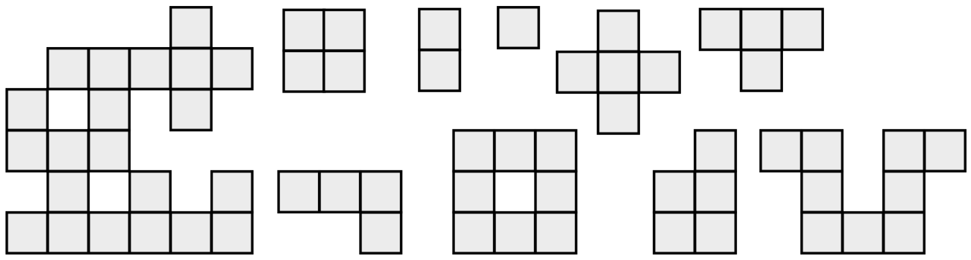
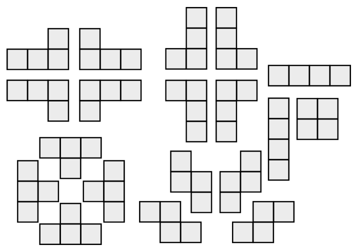


Figure 1.1: A sample of polyominoes


(a) 19 fixed

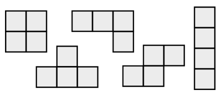

(b) 5 free

Figure 1.2: Polyominoes of size 4

polyominoes, polycubes and any other type of connected subsets of a given lattice. In this work we use the term "Lattice animals" interchangeably with the other relevant terms such as "polyomino".

In the mathematical literature, polyominoes are discussed in the context of two types of combinatorial problems: enumeration and tiling. *Enumeration* deals with determining the number of polyominoes corresponding to a certain parameter n (usually their size but other choices are possible as well, e.g., their perimeter). In most cases a simple closed formula is yet unknown, and so other results are sought: a generating function, an estimation of their growth rate as a function of n, an efficient counting algorithm, etc. *Tiling* deals with the question of whether or not a specified region can be entirely covered by polyominoes from a given set without any two polyminoes overlapping. Efficient algorithms to decide whether such a tiling exists and finding it if it does (usually for specific classes of polyominoes or regions) or proofs of their nonexistance are the main goal of research in this area.

1.1 Basic Enumeration - Overview

When considering polyominoes simply as subsets of $\mathbb{Z}^2$, there are infinitely many polyominoes of any size. Therefore, polyominoes are usually considered identical if they only differ by their relative location, i.e., two polyominoes are identical if one of them can be translated into the other. The family of *fixed* polyominoes is the set of equivalence classes of polyominoes under this relation. The class of *free* polyominoes is obtained from fixed polyominoes by considering two polyominoes identical if they can be rotated and/or reflected into one another. Hence, there are 19 fixed polyominoes of size 4, but only 5 free polyominoes of the same size (as shown in Figure 1.2).

The number of fixed polyominoes of size n (containing n squares) is usually denoted in the literature by $A(n)$. More generally, let us denote by $A_d(n)$ the number of d-dimensional polycubes of size n (so that $A(n) = A_2(n)$). We have $A_2(1) = 1$, $A_2(2) = 2, A_2(3) = 6$, $A_2(4) = 19$, and so on. The function $A_2(n)$ is the "holy grail" of polyominoes enumeration research. So far no closed-form formula, nor a generating function, has been found for $A_2(n)$. However, Klarner [22] showed that the limit (also called *Klarner's constant*) $\lambda_2 := \lim_{n\to\infty} \sqrt[n]{A_2(n)}$ exists. The proof will be described in Section 3.1

Madras [29] proved much later that the growth-rate sequence $A_2(n+1)/A_2(n)$ also converges (as n approaches infinity) and so its limit is equal to λ_2. It is estimated, but not proved, that $A_2(n) = C\lambda_2^n n^{-1}(1+o(1))$ for some constant C, and that $\lambda_2 \approx 4.06 \pm 0.02$ [21]. The best-known proven lower [4] and upper [23] bounds on λ_2 are 3.9801 and 4.6496.

Another aspect of enumeration is finding efficient algorithms for computing $A_2(n)$ exactly for specific values of n. For this purpose, only algorithms whose running time is exponential in n are known today—$A(56)$ is the largest known value. Numerical results can still provide insights as to the behavior of $A_2(n)$.

All of the above applies not only to fixed polyominoes, but also to many classes of polyominoes restricted in some way (for example, convex polyominoes) and for generalizations of polyominoes (d-dimensional polycubes, polyominoes on a triangular or hexagonal lattice, etc.).

Here is a brief summary of the previous attempts to count three-dimensional fixed polycubes:

- Lunnon [26] analyzed in 1972 three-dimensional polycubes by considering symmetry groups, and computed (manually!) $A_3(n)$ up to $n = 6$. Values of $A_3(n)$ up to $n = 12$ can be derived from a subsequent work of Lunnon [27] in 1975 (see below).

- Sykes et al. [40] used in 1976 a method proposed by Martin [31] in order to derive and analyze series expansions on a three-dimensional lattice, but did not compute new values of $A_3(n)$.

- Gaunt, Sykes, and Ruskin [15] listed $A_3(n)$ up to $n = 13$ (with a slight error in $A_3(13)$).

- Gong [17] computed, in a series of attempts (in 1992, 1997, and 2004) $A_3(n)$ up to $n = 9$, 15, and 16, respectively.

- However, Flammenkamp [38] computed $A_3(17)$ already in 1999.

- Nevertheless, the correct values of $A_3(n)$ up to $n = 17$ could already be derived from data provided by Madras et al. in 1990 [30].[2]

In higher dimensions, the first work on counting polycubes that we are aware of is that of Lunnon [27], in which he counted polycubes that could fit into restricted boxes. (In fact, Lunnon counted *proper* polycubes, that is, polycubes that cannot be embedded in lower-dimensional lattices, but the numbers of all polycubes can easily be deduced from the numbers of proper polycubes; we return to this subject in Section 3.2.) Lunnon computed values up to $A_4(11)$, $A_5(9)$, and $A_6(8)$ (with slight errors in $A_6(7)$ and $A_6(8)$). Gaunt, Sykes, and Ruskin [15] provided values up to $A_4(11)$, $A_5(10)$, $A_6(9)$, and $A_7(9)$. Gaunt [14] provided values up to $A_8(9)$ and $A_9(9)$.

[2]To do this, one needs to consider all the coefficients $c_{n,e,u}$ of terms of the form $x^n b^e \lambda^u$ in [30, App. B, pp. 5346–5349], ignore all coefficients with $u > 0$, and sum up the rest. That is, $A_3(n) = \sum_e c_{n,e,0}$. Note that the symbol λ used here is **not** the asymptotic growth rate of polyominoes.

1.2 Enumeration algorithms

The naive method for counting polyominoes is as follows: Start with a set containing a single polyomino consisting of one square. Then repeat the following: For each polyomino in the set, expand it in all possible ways by attaching a new square to an existing one, and count the results. If we consider polyominoes as graphs, then we are merely performing a breadth-first search of the graph. The problem with this method is that many polyominoes are generated in more than one way; each polyomino is generated a number of times proportional to its size. Even if we ignore the cost of detecting duplicates (using, e.g., a hash table) still a lot of work is spent on the generation of the duplicate polyominoes.

The first major improvement over the naive method was Redelmeier's algorithm [37], which is also based on a systematic construction of all polyominoes from smaller ones, but does it in a smart way which ensures that each polyomino is generated exactly once. One important feature of the algorithm is that it can be generalized to counting many other types of polyominoes—in Section 1.3 we describe the algorithm, and in chapter 2 we describe its generalizations in detail.

The current state-of-the-art enumeration algorithm for polyominoes was proposed by Jensen [20]. By using a parallel implementation, it was used for counting all fixed polyominoes up to size 56. It utilized a different approach than Redelmeier's algorithm which enables it to enumerate all polyominoes without generating them. It can be seen as a transfer-matrix algorithm. The main difficulty with the algorithm arises from the large amount of memory it consumes, which prevents most attempts at generalization; the algorithm relies heavily on the two-dimensional geometry of polyominoes. Already in three-dimensions the amount of data needed to be stored is too large, thus, the algorithm cannot be run effectively for three or higher dimensions. Up to date, only the generalized version of Redelmeier's algorithm can be used for higher dimensions. In Section 5 we describe a similar transfer-matrix algorithm which deals with a slightly different case (of polyominoes on a twisted cylinder).

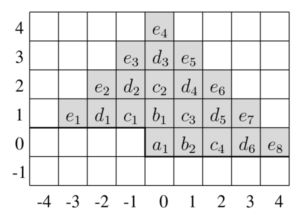
1.3 Redelmeier's algorithm

In this section we briefly describe Redelmeier's algorithm for counting two-dimensional polyominoes. The reader is referred to the original paper [37] for the full details.

Redelmeier's algorithm is a procedure for connected-subgraph counting, where the underlying graph is induced by a square lattice. Since translated copies of a fixed polyomino are considered identical, one must decide upon a canonical form. Redelmeier's choice was to fix the leftmost square of the bottom row of a polyomino at the origin, that is, at the square (0,0). (Note that coordinates are associated with squares and not with their corners.) Thus, he needed to count the number of edge-connected sets of squares (that contain the origin) in

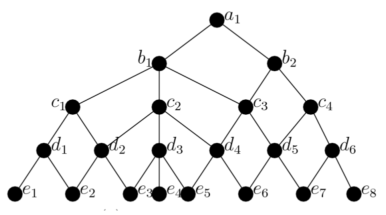
$$\{(x, y) \mid (y > 0) \text{ or } (y = 0 \text{ and } x \geqslant 0)\}.$$

The squares in this set are located above the thick line in Figure 1.3(a). The shaded



e_4
e_3 d_3 e_5
e_2 d_2 c_2 d_4 e_6
e_1 d_1 c_1 b_1 c_3 d_5 e_7
a_1 b_2 c_4 d_6 e_8
4 3 2 1 0 -1
-4 -3 -2 -1 0 1 2 3 4

(a) Reachable cells in polyominoes



a_1
b_1 b_2
c_1 c_2 c_3 c_4
d_1 d_2 d_3 d_4 d_5 d_6
e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8

(b) Corresponding graph

Figure 1.3: Polyominoes as subgraphs of a specific base graph

area in this figure consists of all the *reachable* cells, that is, possible locations of squares of polyominoes of order 5. Counting these sets of squares amounts to counting all the connected subgraphs of the graph shown in Figure 1.3(b), that contain the vertex a_1. The algorithm [37] is shown in Figure 1.4.

```
Initialize the parent to be the empty polyomino, and the untried
set to contain only the origin. The following steps are repeated
until the untried set is exhausted.

  1. Remove an arbitrary element from the untried set.
  2. Place a cell at this point.
  3. Count this new polyomino.
  4. If the size is less than $n$:
     (a) Add new neighbors to the untried set.
     (b) Call this algorithm recursively with the new parent being
         the current polyomino, and the new untried set being a
         copy of the current one.
     (c) Remove the new neighbors from the untried set.
  5. Remove newest cell.
```

Figure 1.4: Redelmeier's algorithm [37, p. 196]

Step 4(a) deserves some attention. By "new neighbors" we mean only neighbors of the new cell c that was added in Step 2, which *were not neighbors* of any polyomino cells before c was added. This ensures that we will not count the same polyomino more than once (we prove this in the more general setting of our algorithm in Section 2.2).

This sequential subgraph-counting algorithm can be applied to any graph (which, indeed, is exactly what we do in the next sections), and it has the property that it never produces the same subgraph twice.

Chapter 2

Generalizations of Redelmeier's Algorithm

2.1 A Simple Generalization

In this section we describe a simple generalization of Redelmeier's algorithm to higher dimensions, which originally appeared in [1].

We reorganize the algorithm in two stages. In the first stage we compute the cell-adjacency graph and mark the canonical cell. In the second (main) step of the algorithm we apply the original procedure, which is nothing else but counting the connected subgraphs that contained the canonical cell. As noted above, the subgraph-counting method does not depend in any way on the structure of the graph. Thus, in order to generalize the algorithm to counting d-dimensional polycubes, we only need to rewrite the first stage so that it would compute the respective neighborhood graph in the appropriate dimension (and up to the size of the sought-after polycubes). Then, we invoke the same subgraph-counting procedure.

Let us briefly review the computation of the lattice graph in d dimensions. Denote the coordinates of a cell as a vector $x = (x_1, x_2, \ldots, x_d)$, and regard it as a number in some base $t \in \mathbb{N}$ with d digits. We mapped the lattice cell x to the integer number $\Gamma(x) = \sum_{k=1}^{d} x_k t^{k-1}$. The number t is chosen large enough so that no two cells are mapped to the same number. The minimum possible choice of t is obviously the size of the range of coordinates attainable by reachable cells of the polycubes, that is, $t = 2n - 2$, where n is the polycube size. A clear benefit of this representation is the ability to compute neighboring cells efficiently: The images under $\Gamma(\cdot)$ of the immediate neighbors of a cell x along the kth direction were $\Gamma(x) \pm t^{k-1}$.

Originally, the canonical cell was fixed as $(0, \ldots, 0)$. Except in the dth direction, the original range of reachable coordinates was $[-(n-2), \ldots, n-1]$, so it was shifted by $n-2$ in order to have only nonzero coordinates. In the dth direction the range of reachable coordinates is $[0, \ldots, n-1]$, and so no shift was needed. Thus, the canonical cell was shifted to $\overline{0} = (n-2, \ldots, n-2, 0)$. It is easy to verify that a cell $x = (x_1, \ldots, x_d)$ is reachable if and only if $x \geqslant \overline{0}$ lexicographically, and reachable cells are in the d-dimensional box defined by $(0, \ldots, 0)$ and $x_M = (2n-3, \ldots, 2n-3, n-1)$.

```
ALGORITHM GraphOrthodD(int $n$, int $d$)
  begin
    1. $o := \Gamma((n-2,\dots,n-2,0))$; $M := (2n-2)^{d-1}n-1$;
    2. for $i = o,\dots,M$ do
      2.1 $b := 1$; $counter := 0$;
      2.2 for $j = 1,\dots,d$ do
        2.2.1 if $i+b \geqslant o$ then do
          2.2.1.1 neighbors[$i$][$counter$] := $i+b$;
          2.2.1.2 $counter$ := $counter$+1;
        end if
        2.2.2 if $i-b \geqslant o$ then do
          2.2.2.1 neighbors[$i$][$counter$] := $i-b$;
          2.2.2.2 $counter$ := $counter$+1;
        end if
        2.2.3 $b := b(2n-2)$;
      end for
      2.3 neighbors_num[$i$] := $counter$;
    end for
  end GraphOrthodD
```

Figure 2.1: Computing the graph for a d-dimensional orthogonal lattice

We have

$$M = \Gamma(x_M) = \sum_{k=1}^{d-1}(2n-3)(2n-2)^{k-1} + (n-1)(2n-2)^{d-1} = (2n-2)^{d-1}n-1,$$

and so, the function $\Gamma(\cdot)$ maps reachable cells to numbers in the range 0 to M.[1]

Figure 2.1 shows the algorithm for building the d-dimensional graph.

The cell-neighborhood relations are kept in the array `neighbors`$[x][y]$, where x is the identity number of the current cell and y is a serial number in the range $(0,\dots,2d-1)$ (all possible directions in d dimensions). The subgraph-counting step starts from the cell with identity $\overline{0}$.

The cell-adjacency graph contains, then, at most $(2n-2)^{d-1}n$ nodes. Actually, only roughly one half of these nodes, that is, about $2^{d-2}(n-1)^{d-1}n$, are really reachable (see Figure 1.3(a) for an illustration). In two dimensions, this number (n^2) is negligible for polyominoes whose counting is time-wise possible. For example, Redelmeier counted polyominoes of up to size 24. However, in higher dimensions a "dimensionality curse" appears. The size of the graph, and not the number of distinct polycubes, becomes the algorithm's bottleneck. For example, in order to count polycubes of size 7 in 7 dimensions, we need to maintain a graph with about 10^7 nodes. Recall that each node has $2d$ neighbors—14, in the last example. This means that even for modest values of d and n the algorithm would need hundreds of megabytes of memory, let alone for higher values, for which computing $A_d(n)$ becomes infeasible.

[1]In fact, x_M itself is not reachable. The reachable cell with the largest image under Γ is $y = (n-2,\dots,n-2,n-1)$ (the top cell of a "stick" aligned with the dth direction). We have $\Gamma(y) = (t^{d+1} - 2t^{d-1} - t + 2)/(2(t-1))$, where $t = 2n-2$.

2.2 Eliminating the Computation of the Graph

The next step in the generalization of Redelmeier's algorithm, first described in [2], is to eliminate the need to hold the entire graph in memory throughout the computation. Clearly, the distribution of cells of different statuses is extremely uneven: At most n cells are (at a time) in the current polycube, at most $2nd$ cells are (at a time) in the untried set (since each of the n used cells has at most $2d$ free neighbors), and all the other cells are free (unoccupied). Since the lattice graph, in any dimension, has a well-defined structure, it is not necessary to create it in advance. Instead, one could create the graph locally "on demand." Given a cell, one could compute its neighbors on-line and always in the same order. In principle, instead of maintaining a huge vertex set (lattice cells) of the graph, and keeping the status of each cell (occupied, in the untried set, or free), we could keep only the first two small sets of cells, and deduce that a cell is free from the fact that it does not belong to any of these two sets.

The major potential obstacle is that, in principle, we need to distinguish between free cells that, in the course of the algorithm, were already part of some counted polycubes, and are thus "done with," and cells that were not yet part of any polycube. It turns out that this differentiation is redundant, and we can manage without keeping this information for each free cell.

To show this, let us formally prove the correctness of the algorithm.[2] We need to prove that even though the graph contains plenty of cycles, no subgraph is obtained twice with different orders of nodes. It will then become clear that no information about the nodes of the underlying graph should be maintained in the course of the algorithm. From the definition of the algorithm, it is clear that every polycube will be counted at least once. This is because a polycube is a *connected* set of cells that includes the origin, and so any exhaustive procedure of adding cells by connectivity will reach any polycube. The more delicate issue is to prove that every polycube is counted only once.

Theorem 1 *Redelmeier's algorithm counts every connected subgraph (polycube) that contains the origin exactly once.*

Proof: Denote a polycube of size n as an ordered set $(a_1, \ldots, a_n)$, where the cells a_i are ordered according to their arrival at the polycube in the course of the algorithm. (Obviously, $a_1 = \overline{0}$ in all polycubes.) Assume for contradiction that the same polycube P is generated twice, i.e., there are two ordered sets $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, generated by the algorithm, which differ only in their permutations of cells. Let A be the set that was generated first, and k (for $2 \leqslant k \leqslant n$) be the smallest index for which $a_i \neq b_i$. (It is clear from the definition of the algorithm that the same ordered set cannot be generated twice.)

Let us focus on the recursion subtree whose root is the polycube $P' = (a_1, \ldots, a_{k-1})$. We can assume by induction (on the level of recursion) that P' is generated only once in the course of the algorithm. First, a_k is moved from the untried set to the polycube.

[2]Redelmeier [37, p. 195] only gave a brief description of the course of the algorithm. The reader is also referred to the discussion in [32, §2].

Then, in a lower subtree of the recursion, *all* the sets whose prefix is $(a_1, \ldots, a_{k-1}, a_k)$ are explored. Following this, a_k is removed from the polycube. Possibly, other cells are moved (one at a time) from the untried set to the polycube, and subtrees of the recursion are performed. Finally, b_k is moved from the untried set to the polycube, forming the set $(a_1, \ldots, a_{k-1}, b_k)$, and the recursion goes down again. Our goal is to prove that in this subtree of the recursion, the cell a_k cannot be added to the polycube, and, hence, P cannot be generated again.

To this aim we need a few observations:

1. When P' is generated the first time, both a_k and b_k are in the untried set. This is because a recursion subtree fully exhausts all the polycubes whose prefix is the one at the top of that subtree. Thus, A and B must be generated at the same subtree, and for this to happen, the untried set must contain both a_k and b_k when P' is generated.

2. Executing a subtree of the recursion does not change the untried set at the top of the subtree; down the recursion, cells are added and removed to the untried set in a symmetric way.[3]

3. By definition, a cell is in the untried set if it is an immediate neighbor of the current polycube. A cell is added to this set only when it becomes a new neighbor; later in the course of the algorithm, the cell cannot remain in the untried set if it is no longer a neighbor of the polycube. This follows from the order of the algorithm: once a cell enters the untried set, this set is fully exhausted before the recursion goes up and shrinks the polycube at the top of this subtree.

From these observations we deduce that both a_k and b_k are neighbors of P'. (Otherwise, they could not be added to the untried set when P' was first created; in addition, since P' is created only once, a_k and b_k are added to the untried set at the same time.) After the termination of the recursion subtree, at the top of which $(a_1, \ldots, a_{k-1}, a_k)$ is, a_k is removed from the polycube (but not returned to the untried set), and later b_k is added to the polyomino. At this time (and down this subtree), a_k *cannot* be added to the polycube, because it cannot be added to the untried set. This follows from the fact that a_k cannot become a *new* neighbor of the polycube, since it is already a neighbor of it. (Recall the definition of "new neighbors.")

This contradicts the assumption that $(b_1, \ldots, b_n)$ contains a_k as a cell, and the claim follows. □

The above proof implies that after a_k is removed from the current polycube, it will never again be added to the untried set (and hence neither to the polycube) as long as the recursion continues with P' as its root. However, when the recursion is unfolded, and the current polycube is shrunk so that it is a subset of P' plus some other cells, it is certainly possible that a_k will be added again to the untried set and later to the polycube. Nevertheless, no future polycube will have P' as a subset.

[3]In his actual implementation of the algorithm, Redelmeier [37, p. 197] took advantage of this fact to avoid the creation of copies of the untried set in the recursive calls; instead, he used only one expanding and shrinking stack for maintaining all versions of this set.

The above proof also implies that we do not need to store any information about the free cells. This is because in order to avoid counting the same polyomino twice, all that is required is the current polyomino and its neighboring cells. To maintain this information we only need to be able to explore the graph locally "on demand," and for each neighbor c' of a cell c which is moved from the untried set to the polycube, determine whether or not it is now a *new* neighbor. The first task is easy to accomplish, since the structure of the orthogonal lattice is well defined, and so is the definition of the mapping $\Gamma(\cdot)$. The second task is also feasible: Such a cell c' is a new neighbor if and only if its only neighboring cell in the current polycube is c. This is easy to determine using the polycube's set of cells.

2.3 Complexity Analysis

For the analysis we assume that $n > d$, otherwise the polycubes cannot really span all the d dimensions. In d dimensions, there are $\Theta((2n)^d)$ distinct reachable cells, so the amount of memory (and time) needed to store (and process) the identity of a single cell is $\Theta(d \log n)$. (Note that in other studies this factor was considered as $\Theta(1)$ or $\Theta(d)$.)

We maintain two data structures: The set of currently-occupied cells (the current polycube) and the set of untried cells (neighbors of the polycube). The size of the current polyomino is at most n, while the size of the untried set is at most $2dn-2(n-1) = 2((d-1)n+1)$. That is, the amount of cells stored by the algorithm is $\Theta(dn)$. For this we need $\Theta(d^2 n \log n)$ space. For a fixed value of d, this is $\Theta(n \log n)$.

Since the new version of the algorithm does not compute the lattice graph, the size of the latter does not directly affect the algorithm's running time. The important factor is the number of operations (additions and deletions of cells) on the untried set. This number is proportional to $A_d(n)$, the number of counted polycubes. (More precisely, the number of these operations is proportional to the total number of d-dimensional polycubes of *all* sizes up to n, but the polycubes of size n outnumber all the smaller polycubes.)

We maintain the current polycube as a balanced binary tree (sorted by $\Gamma(\cdot)$ values), and since it always contains up to n cells, each search, addition, or deletion requires $O(\log n)$ steps (where each step requires $O(d \log n)$ time, as explained above). The untried set should be a queue-like structure which supports two types of operations: (1) Adding new cells at the rear and removing cells (to be added to the current polycube) from the front. (2) Searching for a node, so that a cell will not be added twice to the set. An appropriate tree structure (or two trees with cross references) can do the job. Since the size of the untried set is $O(dn)$, each operation on the set requires $O(\log d + \log n) = O(\log n)$ steps.

When a cell c is moved from the untried set to the polycube, its $2d$ neighbors are computed, and each such neighbor c' is tested to see if it is a *new* neighbor. Recall that c' is a new neighbor if it is free and among all its own $2d$ neighbors, *only* c belongs to the current polycube. Hence, each such test can be done in $O(d \log n)$ steps by using the polycube structure, for a total of $O(d^2 \log n)$ steps for all the neighbors of c. This can be done more efficiently if we keep a secondary set of cells that were removed from the polycube but are still its neighbors, thus, they cannot become "new neighbors" as long as they maintain this property. The size of this set is comparable to that of the untried set, that is, $\Theta(dn)$. For each cell c in this secondary set we maintain the count of its neighbors that belong to the polycube, cells due to which the cell c cannot become a new neighbor. When a cell is removed from the polycube, not only is it put in the secondary set and its count is initialized appropriately, but also the "neighbor counts" of all its neighbors (which are in this set) are decreased by 1. When the neighbor count of a cell reaches 0, the cell is removed from this secondary set, thus, it may again become a new neighbor. With this set we can test in only $O(\log n)$ steps whether or not a cell c' is a new neighbor of the polycube. If it is not new, then it is not added to the untried set. (A byproduct of this is that the data structure implementing the untried set does not have to support the search operation, since we will never attempt to add to it a cell which is already there.)

Let us now sum up the amount of work (in steps) performed in each operation. Recall that a single operation on any of the lists requires $O(\log n)$ steps. Removing a cell c from the polycube takes $O(\log n)$ steps. We also add c to the secondary set in $O(\log n)$ steps, and initialize its count in $O(d \log n)$ steps by checking which of its $2d$ neighbors belong to the polycube. In addition, we need to update the counts of the $O(d)$ neighbors of c in the secondary list. For each such neighbor we compute its id in $O(1)$ steps, search for it in the list in $O(\log n)$ steps, update its count (if it is in the list) in $O(1)$ steps, and remove it from the list (if necessary) in $O(\log n)$ steps. This amounts to $O(\log n)$ steps per neighbor, for a total of $O(d \log n)$ steps for all neighbors of c.

Moving a cell c from the untried set to the polycube takes $O(\log n)$ steps. In addition, we need to compute which cells, neighboring to c, become new neighbors of the polycube. For each such neighbor we compute its id in $O(1)$ steps, and search for it both in the polycube and in the secondary list in $O(\log n)$ steps. Then, we either add it to the untried set (if it is neither in the polycube nor in the secondary set) in $O(\log n)$ steps, or update its count (if it is only in the secondary set) in $O(1)$ steps. This amounts to $O(\log n)$ steps per neighbor, for a total of $O(d \log n)$ steps for all neighbors of c.

Overall, each basic operation of the algorithm (adding or removing a cell) requires $O(d \log n)$ steps. Since the algorithm performs $A_d(n)$ operations, and each step requires $O(d \log n)$ time, the total running time is $O(A_d(n) d^2 \log^2 n)$. In fact, we implemented the three sets as one hash table. Practically, this reduces only the polynomial term, while the major factor, $A_d(n)$, is exponential in n, where the base of the exponent is an unknown constant that depends solely on d (e.g., $\sim$4.06 for $d = 2$).

2.4 Parallelization

Our algorithm can be parallelized efficiently, and so it can be run simultaneously on an unlimited number of different computers. This is done in a fashion similar to the one used by Mertens and Lautenbacher [33] in their parallelization of the two-dimensional version of Redelmeier's algorithm (for counting polyominoes in the plane). The parallelism relies crucially on the fact that the execution of any subtree of the recursion does not change the contents of the data structures at the top of the subtree, and so the computation can be distributed between many processors in such a way that only one of them counts the number of animals in some subtree, while all the other processors skip it altogether and count animals in other subtrees.

In practice, the main procedure recurses only until some level $k < n$. Regardless of the value of n, this induces $A(k)$ subtrees. (For the various types of $A(k)$ counted) These subtrees can be assigned independently to the processors which we have at hand. The results for all levels m (where $1 \leqslant m \leqslant n$, and, in particular, $m = n$) are collected from all the processors, and their summation yields the values of $A(m)$.

The parallelism is perfect in the sense that the split of the computation into parallel tasks can be done at any level $1 \leqslant k \leqslant n$. Setting the level k is based on the desired complexity of a task. Note that for a fixed value of k, the complexities of different subtrees whose roots are at level k may differ significantly. Obviously, setting $k = 1$ results in a single task, in which case the algorithm runs sequentially. Also note that all tasks involve running the entire algorithm up to level k, and recursing only into the designated subtrees. In principle we could avoid this by transferring over the Internet the initial and target configurations (i.e., contents of the data structures that monitor the recursive algorithm), effectively allowing a "warm restart" of the algorithm at any task. In practice, our experiments showed that with our choices of k, the time wasted on running the entire algorithm up to level k in each task was absolutely negligible.

In some cases, care should be taken to differentiate between animals that should be counted and those that should not be counted. For example, when we count only convex polycubes, or proper polycubes (polycubes that span *all* dimensions,[4] or any other specialized type of animals, intermediate animal-like structures do not match the definition of the sought animals, but we cannot skip their creation during the course of the algorithm since subsequent animals may well be of the desired type.

Our actual implementation of the parallel version of Redelmeier's algorithm is based on the "server-client" model. The server was designed to count at the same time several types of animals, hence, it maintains a list of the currently running counters. Each counter is characterized by the type of lattice, the maximum animal size, the value k that controls the parallelization, etc. One of these counters is the *active* counter, corresponding to the lattice on which animals are counted at the present time. Each counter keeps a list of counting tasks, that is, subtrees or ranges of subtrees of the enumeration tree. A task is characterized by the counter type and by the range of k-level subtrees to be processed. Any counter also keeps an indication, for each of its tasks, whether or not the task was already executed, and if yes, what its counting results are.

When a new client introduces itself to the server (by communicating over the Internet), the latter responds with a request to carry out one of the yet-unhandled tasks of the currently active counter. The client then invokes a local copy of the polyomino-counting program, performing the entire computation up to level k, and recursing to the subtrees only in the requested range. At the end of the computation, the client forwards the counting results to the server, again, by communicating over the Internet. Upon receiving these data, the server updates its records. Namely, the task is marked as handled, and the counting results are stored and added to the total results of the active counter. Then,

[4]For example, in the plane, "stick" (straight-line) polyominoes are not proper because they span only one dimension.

n	$A_3(n)$	Ref.	n	$A_3(n)$	Ref.
1	1		10	8,294,738	
2	3		11	60,494,549	
3	15		12	446,205,905	[27]
4	86		13	3,322,769,321	
5	534		14	24,946,773,111	
6	3,481	[26]	15	188,625,900,446	
7	23,502		16	1,435,074,454,755	
8	162,913		17	10,977,812,452,428	[38]$^{(@)}$
9	1,152,870		18	**84,384,157,287,999**	[1]

$^{@}$Attributed to A. Flammenkamp in 1999.

Table 2.1: Numbers of fixed three-dimensional polycubes

a new task is assigned to the same client, and the process continues until all tasks are handled. At this time the active counter has finished its execution and another counter, if available, becomes the active counter.

To accommodate situations in which a task is “lost,” that is, the main server never receives the task’s results from the client to which the task was assigned, the server assigns tasks to clients in a cyclic manner. When all tasks are either completed or assigned to clients, the server returns to the beginning of the task list, and for each yet uncompleted task, if a nominal amount of time has passed since its allocation to a client, the task is assigned to the next available client. In principle, this can lead to the execution of the same task by more than one client, and to the reporting of this task’s results more than once. (This happens only towards the completion of executing a counter.) In this case the server checks for contradicting results, and if such results are encountered, a warning message is sent to the operator of the server. (So far we have never experienced such a situation, indicating, so we believe, that the client code is free of bugs.)

2.5 Results for d-Dimensional Polycubes

Tables 2.1 and 2.2 show the values of $A_d(n)$ obtained in 3–8 dimensions. New values, which were tabulated here and in our papers for the first time, are shown in bold font. Values of $A_8(n)$ confirm previously-published values which relied on unproven formulae. The nonparallel version of the program computed the reported values of $A_6(\cdot)$ and $A_8(\cdot)$ in about 26 days and a little more than a week, respectively. The values of $A_7(\cdot)$ were computed by the parallel program in about a week, using a dozen computers for gathering a total of 91 days of CPU. All the old values in the table agree with previous publications, except for the values of $A_6(7)$ and $A_6(8)$ computed by Lunnon [27]. From his calculations of proper polyominoes, one can infer the values 4,038,205 and 71,976,512, respectively. However, the now-known [5] explicit formulae for those cases confirm our counts.

n	$A_4(n)$	$A_5(n)$
1	1	1
2	4	5
3	28	45
4	234	495
5	2,162	6,095
6	21,272	80,617
7	218,740	1,121,075
8	2,323,730	16,177,405
9	25,314,097	240,196,280
10	281,345,096	**3,648,115,531**
11	3,178,474,308	**56,440,473,990**
12	**36,400,646,766**	**886,696,345,225**
13	**421,693,622,520**	**14,111,836,458,890**
14	**4,933,625,049,464**	
15	**58,216,226,287,844**	

n	$A_6(n)$	$A_7(n)$	$A_8(n)$
1	1	1	1
2	6	7	8
3	66	**91**	**120**
4	901	**1,484**	**2,276**
5	13,881	**27,468**	**49,204**
6	231,008	**551,313**	**1,156,688**
7	4,057,660*	**11,710,328**	**28,831,384**
8	74,174,927*	**259,379,101**	**750,455,268**
9	**1,398,295,989**	**5,933,702,467**	
10	**27,012,396,022**	**139,272,913,892**	

*These values do not agree with the values of $A_6(7) = 4,038,205$ and $A_6(8) = 71,976,512$ that can be computed from the counts of Lunnon [27], but as shown in [5] and [28], our results are the correct ones.

Table 2.2: Numbers of fixed d-dimensional polycubes (new values in bold)

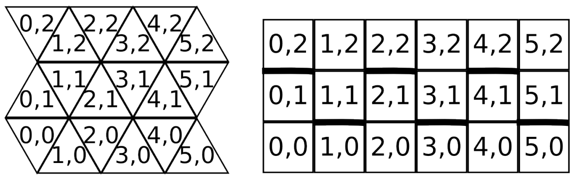


Figure 2.2: Representing a triangular lattice by a rectangular lattice (bold segments indicate removed adjacency relations)

2.6 Nonrectangular Planar Lattices

In this section we describe our extensions of the algorithm to counting planar triangular polyominoes (*polyiamonds*) and hexagonal polyominoes (*polyhexes*). For both cases Lunnon [25] considered cells with three coordinates (albeit in the plane), while we model these lattices with cells from the standard rectangular lattice with different adjacency relations.

2.6.1 Polyiamonds

For a triangular lattice we use equilateral triangles in two inverse orientations. We model a triangular lattice by a rectangular lattice with a restriction on the neighborhood relations: A cell whose sum of coordinates is even (resp., odd) has only an upper (resp., lower) neighboring cell but not a lower (resp., upper) neighbor. All cells have right and left neighbors; see Figure 2.2.

As in the orthogonal case, we choose the canonical cell to be the leftmost triangle in the lowest row. Since we have two types of triangles ("up" and "down," e.g., $(0, 1)$ and $(0, 0)$, respectively, in Figure 2.2), there are two possible types of canonical cells. Nevertheless, it turns out that it suffices to count only triangular polyominoes whose canonical cell is of one type, say, "down."

Recall that we denote by $T(n)$ the number of triangular polyominoes of size n. Let $T'(n)$ mark the number of triangular polyominoes whose canonical call is "down." Observe that $T(n) = T'(n) + T'(n - 1)$. To see this, note that when the canonical cell is "up," its only possible neighbor is its right "down" neighbor. Therefore, the number of triangular polyominoes of size n whose canonical cell is "up" is the same as the number of polyominoes of size $(n - 1)$ whose canonical cell is "down," the right neighbor of the original canonical cell.

2.6.2 Polyhexes

Hexagonal polyominoes have already been counted by a transfer-matrix method,[41] which is much faster than our subgraph-counting approach. For completeness, we also describe this case here.

We can model a hexagonal lattice by using a rectangular lattice with a special type of adjacency relationship. Namely, consider a hexagonal lattice as a rectangular lattice in which every second column is "slid down" by half a square. Every two adjacent squares

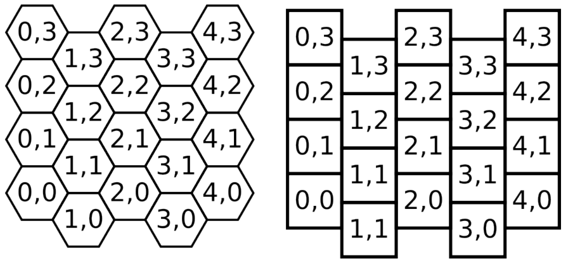


Figure 2.3: Representing a hexagonal lattice by a rectangular lattice

(even along half of a cell boundary edge) in the modified lattice are considered neighbors (see Figure 2.3). Thus, we allow the cell to have six neighbors in a hexagonal lattice instead of four as in a rectangular lattice. The two additional neighbors are diagonal—either the upper or the lower diagonal cells, depending on the parity of the x coordinate of the cell. In the example shown in Figure 2.3, the two additional neighbors of $(1, 1)$ (odd x coordinate) are $(0, 0)$ and $(2, 0)$ (the two lower diagonal cells), while the additional neighbors of $(2, 1)$ (even x coordinate) are $(1, 2)$ and $(3, 2)$ (the two upper diagonal cells).

In the hexagonal case we need a sharper definition for the canonical cell. While it is still defined as the leftmost cell in the lowest row, the meaning of a "row" is now restricted only to hexagons that are aligned horizontally. In other words, in our example, $(1, 0)$ is not in the same row as $(2, 0)$. Consequently, every row in a rectangular lattice becomes two rows in a hexagonal lattice—an upper and a lower hexagonal row. We fix the canonical cell to always be in the upper hexagonal row. While building the graph, we do not include cells that are below this upper row. Thus, if $(2, 0)$ is the canonical cell, then $(3, 0)$ is not reachable and is, therefore, not included in the graph.

2.7 Leapers

A (a, b)-*leaper* is a lattice animal lying on the two-dimensional orthogonal lattice, in which the neighbors of cell (x, y) are *defined* to be cells $(x \pm a, y \pm b)$ and $(x \pm b, y \pm a)$. Under this definition, regular polyominoes are (1,0)-leapers. It is not difficult also to see that (1,1)-leapers are again equivalent to regular polyominoes, and likewise for (a, a)-leapers for any a. Hence, we focus on the case of (a, b)-leapers with $a > b > 0$.

Since for such a (a, b)-leaper lattice, each cell has eight neighbors in four different possible "dimensions", such leapers can be thought of as approximations for 4-dimensional polycubes (as is well apparent from the empirical results as well). The term *polyknights* is usually used for $(2, 1)$-leapers for obvious reasons, and *polyzebra* is used for $(3, 2)$-leapers.

Although every pair (a, b) gives rise to a different type of a leaper, the counting results are quite similar for most pairs, since differences arise only in leapers big enough to contain self-intersections of different manners. For example, the number of polyknights and polyzebras up to size 5 are the same, and are equal to the number of 4-dimensional polycubes; for $n > 5$ the number of polyknights differ, but the number of polyzebras remains the same as the number of polycubes up to $n = 9$. The current counts have

n	Polyknights (A030444)	Polyzebras (A093877)
1	1	1
2	4	4
3	28	28
4	234	234
5	2,162	2,162
6	20,972	21,272
7	209,608	218,740
8	2,135,572	2,323,730
9	22,049,959	25,314,097
10	229,939,414	281,299,736
11	2,416,816,416	3,176,220,308
12	**25,569,786,440**	**36,332,751,862**

Table 2.3: Numbers of polyknights and polyzebras

computed the number of polyknights (sequence A030444 in [38]) and polyzebras (sequence A093877 in [38]) up to $n = 11$; using our method we have obtained one additional value for each sequence.

Chapter 3

Analysis of the Growth Constants

3.1 Klarner's Constant

We give a short description of Klarner's proof that $\lambda_2 := \lim_{n\to\infty} \sqrt[n]{A_2(n)}$ exists.

Klarner's main observation is that $A_2(n)$ is *log-concave*, i.e., it satisfies the relation $A_2(n) \cdot A_2(m) \leqslant A_2(n+m)$. This follows from a so-called *concatenation argument*: every polyomino A of size n can be "glued" to any polyomino B of size m by placing the rightmost cell in the topmost row of A exactly to the left of the leftmost cell in the lowest row of B. Since A and B touch but do not overlap, the result is a valid polyomino of size $n + m$. Different choices of A and B lead to different concatenated polyominoes of size $n + m$. However, not all polyominoes of size $n + m$ can be represented as a concatenation of two polyominoes of sizes n and m. This yields the wanted inequality.

Taking logarithms, we see that the sequence $a(n) = \ln A_2(n)$ is superadditive: $a(n) + a(m) \leqslant a(n + m)$. A well-known lemma of Fekete [35] shows that if the sequence $a(n)$ is superadditive, then the limit $\lim_{n\to\infty} \frac{a(n)}{n}$ exists (but may be ∞). Hence, $\lim_{n\to\infty} \frac{a(n)}{n} = \lim_{n\to\infty} \ln(\sqrt[n]{A_2(n)})$ exists, so $\lim_{n\to\infty} \sqrt[n]{A_2(n)}$ exists as well.

It remains to show that $\lim_{n\to\infty} \sqrt[n]{A_2(n)} < \infty$, by giving some upper bound on the growth of $A_2(n)$. This can be done in several ways, the one yielding the best upper bound was given by Klarner and Rivest [23]. We present here the method given by Klarner in his original paper [22] with credit to Eden [12].

We show how a binary string of length $3n - 1$ can be used to construct an arbitrary polyomino. Begin constructing the polyomino by placing one square on the grid, and assume that this square is the leftmost square in the bottom row of the polyomino. Hence, only two of its neighboring squares (the right and the upper) can be added to the polyomino. Read one bit from the string for each of them, where 0 denotes not adding a square, and 1 denotes adding a cell. Now proceed to one of the added cells; it has only three valid neighbors, since one neighboring square is the one we arrived from. Read three bits from the string and proceed as before.

It is not difficult to see that every polyomino can be generated by some string (although not all strings lead to valid polyominoes, and some polyominoes are generated by many strings). Moreover, the number of 1's in the string is the total number of cells added to the polyomino except the first cell. Hence we are only interested in binary strings of length

$3n-1$ containing *exactly* $n-1$ 1's, and so $A_2(n) \leqslant \sqrt[n]{\binom{3n-1}{n-1}} \leqslant 6.75$.

The same method can be applied in d dimensions, for a fixed value of d, and for any lattice where each cell has a constant number of neighbors. For a lattice in which each cell has k neighbors, a lattice animal can be encoded by binary strings of length $(k-1)n-1$ with exactly $n-1$ 1's, and so we obtain as an upper bound on the nth power of the growth-rate limit the term $\binom{(k-1)n-1}{n-1} \leqslant \frac{1}{k-1}\left[\frac{(k-1)^{k-1}}{(k-2)^{k-2}}\right]^n$.

Since $\frac{(k-1)^{k-1}}{(k-2)^{k-2}} = (k-1)\left[1+\frac{1}{k-2}\right]^{k-2} < (k-1)e$, we see that $(k-1)e$ is an upper bound on the growth-rate limit for any lattice in which every cell has k neighbors. In particular, for animals in a d-dimensional orthogonal lattice we have the bound $(2d-1)e$.

3.2 Proper Polycubes

3.2.1 Lunnon's formula

A d-dimensional polycube is *proper* if it spans all the d dimensions (and thus, cannot be embedded in $d-1$ dimensions). We denote by $\mathrm{DX}(n,d)$ the number of proper polycubes of size n in d dimensions. A main interest in DX stems from the fact that the number of (non-proper) polycubes can be easily computed using the formula

$$A_d(n) = \sum_{i=0}^{d} \binom{d}{i} \mathrm{DX}(n,i),$$

given originally by Lunnon [27]. The formula is proved by noting that every proper i-dimensional polycube can be embedded in the d-dimensional space in exactly $\binom{d}{i}$ different ways (according to the choice of dimensions for the polycube to occupy). Also, if $n \leqslant d$ then the polycube simply cannot occupy all the dimensions (since a polycube of size n can occupy at most $n-1$ dimensions), and so $\mathrm{DX}(n,d) = 0$ in this case. Following [5], we rewrite the above formula as

$$A_d(n) = \sum_{i=0}^{\min(n-1,d)} \binom{d}{i} \mathrm{DX}(n,i).$$

Lunnon's formula has been widely used in the statistical-physics literature on lattice animals, and is called a "partition formula," where the partition is usually according to a few more parameters (attributes of the polycubes). The earliest references for this, that we are aware of, are from the 1960s.

3.2.2 Counting proper polycubes

As mentioned above, our counts of $A_6(7)$ and $A_6(8)$ did not match those of Lunnon [27]. Since in that paper, Lunnon actually computed $\mathrm{DX}(n,d)$, the discrepancies were found by computing the expected values of $A(n)$ according to Lunnon's formula. In order to trace the source of the discrepancies, we modified our program to also count proper polycubes.

To this aim, we should have maintained a "dimension status" that should have counted, for each dimension $1 \leqslant i \leqslant d$, how many cells of the current polycube have the ith coordinate different than $\overline{0}$, the cell at the origin. Each addition or deletion of a cell to/from the polycube should have been accompanied by updating the appropriate counter in the dimension-status record. To achieve this, we needed an efficient implementation of $\Gamma^{-1}(\cdot)$, a function that maps numbers (cell ids) back to their original source cells with d coordinates.

However, for efficiency of the program, we wanted to avoid altogether the use of such a function $\Gamma^{-1}(\cdot)$. When we add a cell c to the polycube, instead of increasing by 1 the counters of all its dimensions relative to the origin (which are many, and not simple to identify), we increase only the counter of its dimension relative to the cell c' due to which it was previously put in the untried set (which is unique, and is known while c is added). This specific counter is decreased by 1 when cell c is removed from the polycube. During the entire period in which c belongs to the polycube, this dimension of the polycube is used. Thus, although the counter of dimension i does not really count the number of cells that occupy the ith dimension, it is easy to see that it is still positive whenever there is any cell that does so. Obviously, the cell at the origin does not occupy any dimension. In conclusion, a polycube is proper if and only if all the d counters in the dimension-status record are greater than zero.

3.2.3 Diagonals

As mentioned before, for $n \leqslant d$ we have $\mathrm{DX}(n,d) = 0$. Hence, in a matrix recording the values of DX the upper triangle and the main diagonal would be 0. This gives rise to the question whether a pattern can be found in the nonzero diagonals, i.e., in the sequences $\mathrm{DX}(n, n-k)$, where $k < n$ is a parameter corresponding to the ordinal number of the diagonal. Obviously, if a simple formula is found for $\mathrm{DX}(n, n-k)$ for every k, this will yield a simple formula for $A_d(n)$ (using Lunnon's formula). A detailed analysis of diagonals for proper polycubes is given in [5]; we give the basic results here.

For the first diagonal, a simple formula is indeed well-known:

Theorem 2 $\mathrm{DX}(n, n-1) = 2^{n-1}n^{n-3}$.

This formula is usually justified by the fact that these minimal proper polycubes are in one-to-one correspondence with Cayley trees. Many works mention it either implicitly of explicitly; see, e.g., [15, Eq. (2.3)] and [34, Eq. (2.9)].

We supply a proof here as well. We begin with the following lemma, which gives the number of trees that correspond to our polycubes:

Lemma 3 *The number of trees on n vertices, with directed edges labeled by $1, 2, \ldots, n-1$, is $2^{n-1}n^{n-3}$.*

Proof: The number of trees on n vertices labeled by the numbers $0, 1, 2, \ldots, n-1$ is known to be n^{n-2} (this is the so-called Cayley's formula). Each of the $n-1$ edges of the tree can be directed in two possible orientations. Hence, there are $2^{n-1}n^{n-2}$ labeled trees with directed edges.

For every edge-labeled tree, choose one of the n possible vertices and label it by 0. Now "shift" the label on each edge to the vertex in the direction *away* from vertex 0. This gives rise to a tree with labeled vertices and each edge-labeled tree gives rise to exactly n such trees, for each of the possible choices of the 0 vertex. This shows that there are $2^{n-1}n^{n-3}$ directed edge-labeled trees. □

We now show a bijection between directed edge-labeled trees and proper polycubes, thus proving the theorem.

Proof of Theorem 2: Given a polycube P, we take the adjacency graph of its cells, i.e., the vertices are the cells of P and edges correspond to pairs of face-adjoint cells. Every pair of vertices connected by an edge represents cells that differ only in one coordinate i. Thus, we label the edge by i and orient it so as to point from the cell with the smaller ith coordinate to the cell with the larger one. Obviously, this graph is connected since the polycube is.

It remains to show that the graph is indeed a tree. Consider a spanning tree of the graph, and note that all $n-1$ possible labels must be present on its $n-1$ edges, otherwise the original polycube is not proper.

If we fix an arbitrary node of the spanning tree as its root, and assume it to be at the origin of coordinates (i.e., all its coordinates are 0), then with every vertex we can associate the list of dimensions whose coordinate is nonzero. For every vertex, this list contains all the labels of the edges on the path leading to it from the root. (Note that the same label cannot appear twice with different signs, canceling out the coordinate, since then not all of the $n-1$ labels would appear).

Now suppose that a and b are two vertices in the graph that do not share an edge in the spanning tree. Then, their distance in the spanning tree is at least 2, and, hence, they differ by at least two coordinates (the coordinates that uniquely appear as edge labels on the path connecting them). Therefore, they cannot be incident to the same edge in the graph, since this would imply that they differ by one coordinate only, showing that the graph obtained from the polycube is identical to the spanning tree.

On the other hand, given a directed edge-labeled tree, we can construct the corresponding polycube by picking an arbitrary root and traversing the tree from it, constructing the polycube according to the edge labels. It is easy to see that this reverses the process of transforming a polycube into a tree. □

A somewhat more elaborate treatment yields the formula of the second diagonal:

Theorem 4 [5, Theorem 6] $\mathrm{DX}(n, n-2) = 2^{n-3}n^{n-5}(n-2)(2n^2-6n+9)$.

3.3 Tree Polyominoes

3.3.1 Introduction

A *tree* polyomino is a polyomino whose dual (cell adjacency) graph is a tree. Tree structures are dealt with extensively in [7]. Tree polyominoes on a hexagonal lattice were enumerated in [6, 18].

Similarly to Lunnon's notation [27], we denote by $\mathrm{DT}(n, d)$ (resp., $\mathrm{CT}(n, d)$) the number of proper (resp., all) tree polycubes of size n in d dimensions.

Computing $\mathrm{DT}(n,d)$ and $\mathrm{CT}(n,d)$, using our generalization of Redelmeier's algorithm, is quite straightforward. For each cell in the lattice neighboring the current animal, we maintain a reference count for the number of its neighbors in the animal itself. Whenever we add or remove a cell from the animal, we update its neighbors accordingly. When a new cell is chosen from the untried set in order to be added to the current animal, we check its reference count. If it is greater than 1, we remove it from the untried set without using it at all. This prevents any cycles from forming in our animal. Hence, we end up with a tree animal. No tree animal is missed by this method, since adding a cell to a valid animal, where the new cell has two or more neighbors in the animal, ensures that the animal will contain a cycle as long as the new cell is part of it. Hence, we discard from the untried set only cells whose addition to the animal will prevent it from being a tree. For a fixed value of n, we define $\mathrm{CT}_n(d) = \mathrm{CT}(n,d)$ as a function of one variable d.

Our goal is to estimate the growth rate constant λ_d^T for d-dimensional tree polycubes for large values of d. As in [5], we take the following strategy:
1. Compute $\mathrm{DT}(n,n-1)$ and $\mathrm{DT}(n,n-2)$;
2. Compute the general terms for the leading coefficients of $\mathrm{CT}_n(d)$; and
3. Compute the beginning of a long polynomial division $\mathrm{CT}_{n+1}(d)/\mathrm{CT}_n(d)$.

3.3.2 Diagonal formulae

Theorem 5 $\mathrm{DT}(n,n-1) = 2^{n-1}n^{n-3}$.

Proof: Since proper polycubes of size n in $n-1$ dimensions cannot have cycles, that is, all such polycubes are trees, $\mathrm{DT}(n,n-1)$ is the total number of proper d-dimensional polycubes of size n. This number is well-known to be $2^{n-1}n^{n-3}$ (see Theorem 2). □

We now give a formula for the second diagonal, the tree-animal equivalent of Theorem 4. The proof follows closely the proof given in [5].

Theorem 6 $\mathrm{DT}(n,n-2) = 2^{n-3}n^{n-5}(n-2)(2n^2-7n+12)$.

For the proof we use the following lemma:

Lemma 7 [5, Lemma 4] *The number of ordered sequences of $k \geqslant 1$ undirected rooted trees with a total of $n-k$ edges and n vertices, and distinct edge labels $1,\dots,n-k$, is $n^{n-k-1}k$.*

By "rooted" we mean that each tree in the forest has a distinguished marked vertex.
Proof: The proof in [5] uses known facts about the number of labeled trees with given degrees; we give an alternate proof which uses elementary facts from the theory of exponential generating functions (EGFs). For a general description of this method we refer the reader to [13, Chapter 2].

Denote by $f_k(n)$ the number of ordered sequences of k rooted trees with n vertices and $n-k$ distinct edge labels, and denote by $T_k(z)$ the following EGF: $T_k(z) = \sum_{n=k}^{\infty} \frac{f_k(n)}{(n-k)!} z^{n-k}$. (That is, the sequence $f_k(z)$ without the initial 0's in its beginning. This reduces the technical complexity in what follows.)

Note that we have $f_1(n) = n^{n-2}$ since we have the following bijection between rooted trees with n vertices and edge labels, and nonrooted trees with n labeled vertices, as follows: Given a vertex-labeled tree, mark the vertex labeled by n as the root and erase its label; for each other vertex, "push" its label to the (unique) adjacent edge in the direction of the root. The result follows from the fact that the number of nonrooted, vertex-labeled trees is n^{n-2} by Cayley's formula.

The generating function for the number of *rooted* trees with labeled *vertices* is well known; this is the so-called "tree function" $T(z)$ which satisfies the equation $T(z) = ze^{T(z)}$. Since a rooted tree can be obtained from a nonrooted tree by an arbitrary choice of a root, there are n^{n-1} such trees, and so $T(z) = \sum_{n=1}^{\infty} \frac{n^{n-1}}{n!} z^n$.

Now note that $T_1(z) = \sum_{n=1}^{\infty} \frac{n^{n-2}}{(n-1)!} z^{n-1} = \frac{1}{z} \sum_{n=1}^{\infty} \frac{n^{n-1}}{n!} z^n = \frac{1}{z} T(z)$. Since a sequence of k edge-labeled rooted trees is a label product of k copies of the set of rooted edge-labeled trees, we have that $T_k(n)$ is the product of k copies of $T_1(z)$, hence
$T_k(z) = (T_1(z))^k = \frac{1}{z^k} T^k(z)$. Therefore, we have $\frac{f_k(n)}{(n-k)!} = [z^{n-k}]T_k(z) = [z^n]T^k(z)$.

We now use the Lagrange inversion theorem, which states that if $z = T/\phi(T)$ for some generating function T and a function ϕ, then we have $[z^n]T^k(z) = \frac{k}{n}[w^{n-k}]\phi(w)^n$. Since the tree function satisfies the equation $T(z) = ze^{T(z)}$, we can use the inversion theorem with $\phi(w) = e^w$, hence $\frac{f_k(n)}{(n-k)!} = [z^n]T(z)^k = \frac{k}{n}[w^{n-k}]e^{nw} = \frac{k}{n} \cdot \frac{n^{n-k}}{(n-k)!} = \frac{n^{n-k-1}k}{(n-k)!}$, which yields $f_k(n) = n^{n-k-1}k$, as required. □

We can now prove Theorem 6.

Proof of Theorem 6: First note that every tree-polycube gives rise to a unique spanning tree whose vertices are the cells of the polycube and edges connect two adjacent vertices (i.e., cells whose coordinates differ by 1 in exactly one coordinate). The tree-structure of the polycube ensures that there is only one such spanning tree.

Given the spanning tree, we can label and orient its edges as follows: if (u, v) is an edge, then u, v are cells differing in exactly one coordinate i. Assume that u is the vertex closer to the root of the tree; label the edge by i if u and v differ in the ith coordinate, and direct the edge from u to v if the ith coordinate of u is the smaller one, otherwise direct the edge from v to u. Since the polycube is proper, every i in the possible coordinate range $(1, \ldots, n-2)$ must appear, and since there are $n-1$ edges, exactly one label appears twice.

Thus, we wish to count the number of trees with n vertices and directed, labeled edges, such that every label is in the range $1, \ldots, n-2$ and appears at least once, and subtract from that number the number of such trees not corresponding to a tree polycube.

From Cayley's formula we know that there are n^{n-2} labeled trees with n vertices, and after directing the edges we have exactly $2^{n-1}n^{n-2}$ trees (note that we do not require all the directed edges to point to or from a specific vertex). We now show that every directed tree with n vertices and edges labeled with *distinct* labels from the range $1, 2, \ldots, n-1$ corresponds to exactly n such directed trees with labeled vertices, and so their number is $2^{n-1}n^{n-3}$.

Given a directed tree with labeled edges, choose a vertex in the graph and label it 0. Label the rest of the vertices in the graph by $1, \ldots, n-1$ in the following way: each vertex v is marked with the label of its adjacent edge leading to the vertex 0. Since this process is invertible, we see that each edge-labeled graph gives rise to exactly n distinct vertex-labeled graphs and conclude that the number of edge-labeled graphs is $2^{n-1}n^{n-3}$.

Now choose a label i in the range $1, \ldots, n-2$, and change the label $n-1$ to i, thus producing a tree of the wanted form: a directed tree with labels in the range $1, \ldots, n-2$, with each label appearing at least once (and one label appearing twice). Note that most trees are generated *twice* in this manner, but not all; in case the tree is symmetric with respect to the edges i and $n-1$, the result is generated only once. However, all such symmetric trees will be removed in the next step of the proof (since we will show that they do not represent tree-polycubes) and so all the remaining trees will be double-counted, hence we will divide the final result by 2. For now the number of trees obtained is $(n-2)2^{n-1}n^{n-3}$.

We now perform two "corrections" on our set, removing trees that do not represent tree polycubes. First consider trees whose corresponding polycube contains overlapping cubes. Since in our polycube almost every cube occupies a new dimension, the overlap must arise from the two edges labeled with i. Moreover, these two edges must be connected to the same vertex, since different vertices in the tree differ by at least one coordinate not equal to i (since the i-edges were not used yet).

We conclude that the only possible ways in which overlap arises is in trees containing a path of the form $\bullet \xleftarrow{i} \bullet \xrightarrow{i} \bullet$ or the form $\bullet \xrightarrow{i} \bullet \xleftarrow{i} \bullet$.

We count the number of such trees and subtract it from the total number of trees. Note that we count exactly the trees that are symmetric with respect to the i-edges, thus fixing the problem pointed at earlier.

Given a path of one of the forms above, we can make it a tree by attaching three rooted trees labeled with a total of $n-3$ labels to its three vertices (placing the roots of the trees on the vertices). By Lemma 7, there are $3n^{n-4}$ sequences of 3 *undirected* trees. Directing the edges, we get $3 \cdot 2^{n-3}n^{n-4}$ trees. Since there are $(n-2)$ choices of i, and for each i there are two possible paths (as shown above), the final result is $3(n-2)2^{n-2}n^{n-4}$.

The next correction removes from the remaining trees those that give rise to a legal polycube (i.e., without overlaps) but not a *tree* polyomino. Since our polycube is small and must occupy all dimensions, the only possible way for this to occur is if the original spanning tree contains a path of the form $\bullet \xleftarrow{i} \bullet \xleftarrow{k} \bullet \xrightarrow{i} \bullet$ or the form $\bullet \xrightarrow{i} \bullet \xleftarrow{k} \bullet \xleftarrow{i} \bullet$.

We count such spanning trees in a way similar to that of the first correction. The number of sequences of four trees with labeled edges is $4n^{n-5}$. Multiplying by the number of possible edge orientations, we have $4 \cdot 2^{n-4}n^{n-5}$ trees. For each of the 2 forms above we also need to orient the k-edge in one of the two possible directions (different orientations yield different spanning trees), so we multiply by 4; also, we have $(n-2)$ choices of i and $(n-3)$ choices of k (given i). Therefore, the total number of trees counted in this manner is $4(n-2)(n-3)(4n^{n-5})$.

We conclude that the final count is:

$$\frac{(n-2)2^{n-1}n^{n-3} - 3 \cdot (n-2) \cdot 2^{n-2}n^{n-4} - 4(n-2)(n-3)(4n^{n-5})}{2}$$
$$= 2^{n-3}n^{n-5}(n-2)(2n^2 - 7n + 12)$$

. □

For comparison we now repeat the formulas for the second diagonal for standard and tree polyominoes:

- $\mathrm{DX}(n, n-2) = 2^{n-3}n^{n-5}(n-2)(2n^2 - 6n + 9)$.
- $\mathrm{DT}(n, n-2) = 2^{n-3}n^{n-5}(n-2)(2n^2 - 7n + 12)$.

3.3.3 Leading coefficients of polynomials for fixed-size polycubes

As in [5, Theorem. 8], it is easy to prove that for any fixed value of n, the function $\mathrm{CT}_n(d)$ is a polynomial in d of degree $n-1$. The proof is based on a variant of Lunnon's formula [5, 27] adapted to tree polycubes: $\mathrm{CT}(n,d) = \sum_{i=0}^{\min(n-1,d)} \binom{d}{i}\mathrm{DT}(n,i)$. One can also compute the leading coefficients of $\mathrm{CT}_n(d)$. Let $b_{n,0}$ and $b_{n,1}$ the coefficients of d^{n-1} and d^{n-2}, resp., in $\mathrm{CT}_n(d)$.

Theorem 8

- $b_{n,0} = 2^{n-1}n^{n-3}/(n-1)!$
- $b_{n,1} = -2^{n-3}n^{n-5}(7n-12)/(n-3)!$.

Proof: By collecting terms from Lunnon's formula, we find that

$$\begin{aligned}
b_{n,0} &= \mathrm{DT}(n,n-1)/(n-1)! = 2^{n-1}n^{n-3}/(n-1)! \\
b_{n,1} &= \mathrm{DT}(n,n-1)\frac{\sum_{i=0}^{n-2}(-i)}{(n-1)!} + \mathrm{DT}(n,n-2)\frac{1}{(n-2)!} \\
&= 2^{n-1}n^{n-3}\frac{-1}{2(n-3)!} + 2^{n-3}n^{n-5}(n-2)(2n^2-7n+12)\frac{1}{(n-2)!} \\
&= -2^{n-3}n^{n-5}(7n-12)/(n-3)!
\end{aligned}$$

□

3.3.4 Growth rate

Similarly to Theorems 9–11 in [5], one can prove rigorously that $\lambda_d^T = 2ed - o(d)$ as d tends to infinity. We start by noting that $\lambda_d^T < (2d-1)e = 2ed - O(1)$ for every d; this bound was established in Section 3.1 as a generalization of Eden's upper bound method.

Proving the lower bound $\lim_{d\to\infty} \frac{\lambda_d^T}{d} \geqslant 2e$ will finish the proof. More explicitly, we want to show that $\lim_{d\to\infty}\lim_{n\to\infty} \frac{\sqrt[n]{\mathrm{CT}_n(d)}}{d} \geqslant 2e$.

Intuitively, our argument is as follows:
$\lim_{n\to\infty} \frac{\sqrt[n]{\mathrm{CT}_n(d)}}{d} = \lim_{n\to\infty} \frac{\sqrt[n-1]{\mathrm{CT}_n(d)}}{d} = \lim_{n\to\infty} \sqrt[n-1]{\frac{\mathrm{CT}_n(d)}{d^{n-1}}}$.

And since, as mentioned above, $\mathrm{CT}_n(d)$ is a polynomial in d of degree $n-1$ we can write $\mathrm{CT}_n(d) = b_{n,0}d^{n-1} + o(d^{n-1})$.

Hence, it remains to compute the limit $\lim_{d\to\infty}\lim_{n\to\infty} \sqrt[n-1]{b_{n,0} + \frac{o(d^{n-1})}{d^{n-1}}}$. If the two limits could be exchanged, the result would immediately follow:

$$\lim_{n\to\infty}\lim_{d\to\infty} \sqrt[n-1]{b_{n,0} + \frac{o(d^{n-1})}{d^{n-1}}} = \lim_{n\to\infty} \sqrt[n-1]{b_{n,0}} = 2e,$$

where the last transition follows from Stirling's formula. However, it was never proven that such exchange of limits is permissible here.

In order to give a rigorous argument we follow the method of [5, Theorem 9]. Fix some $\varepsilon > 0$; we show that there exists d_0 such that for all $d > d_0$ we have $\lambda_d^T \geqslant (2-\varepsilon)ed$.

First, pick some n_0 (independent of d) such that for every $n > n_0$ we have $\sqrt[n-1]{b_{n,0}} \geqslant (2-\varepsilon/2)e$.

For this specific n_0 we can see, as above, that $\lim_{d\to\infty} \frac{\sqrt[n_0-1]{\mathrm{CT}_{n_0(d)}}}{d} = \sqrt[n_0-1]{b_{n_0,0}}$. In particular, there exists d_0, dependent on n_0, such that for all $d > d_0$ we have

$$\sqrt[n_0-1]{\mathrm{CT}_{n_0}(d)} \geqslant (2-\varepsilon)ed.$$

Now fix $d > d_0$. Note that the sequence $\sqrt[n-1]{\mathrm{CT}_n(d)}$ (considered as a sequence in n) is increasing (this can be shown using a standard concatenation argument). From this it follows that $\lambda_d^T = \lim_{n\to\infty} \sqrt[n-1]{\mathrm{CT}_n(d)} \geqslant \sqrt[n_0-1]{\mathrm{CT}_{n_0}(d)} \geqslant (2-\varepsilon)ed$, as required.

We can also provide a more accurate (but non rigorous) estimate of λ_d^T.

As shown above, $\mathrm{CT}_n(d) = b_{n,0}d^{n-1} + b_{n,1}d^{n-2} + \cdots$. Considering n fixed, we write $\mathrm{CT}_{n+1}(d)/\mathrm{CT}_n(d) = f_1(n)d + f_2(n) + O(1/d)$. A straightforward long polynomial division procedure shows that

$$f_1(n) = 2((n+1)/n)^{n-2}$$

and

$$f_2(n) = -(n-1)(n+1)^{n-4}(7n^3 + 21n^2 - 22n - 24)/(2n^n).$$

It is now easy to verify that $\lim_{n\to\infty} f_1(n) = 2e$ and $\lim_{n\to\infty} f_2(n) = -3.5e$. Hence, we conjecture that $\lambda_d^T = (2d-3.5)e + O(1/d)$, and thus λ_d^T tends to $(2d-3.5)e$ as $d \to \infty$. Note that λ_d, the growth rate of *all* d-dimensional polycubes (including nontrees), is estimated at $(2d-3)e + O(1/d)$ [5].

Chapter 4

Polyominoes and Pattern-Avoiding Permutations

4.1 Introduction

An active field in enumerative combinatorics is the analysis and enumeration of *pattern-avoiding permutations*. We start by describing the basic definitions. Let τ be a permutation of $[k]$, and let $\pi = (b_1\ b_2\ \ldots\ b_n)$ be a permutation of $[n]$. We say that π *contains* τ as a pattern if there exist indices $1 \leqslant i_1 < i_2 < \ldots < i_k \leqslant n$ such that the subpermutation of π, $(b_{i_1}\ b_{i_2}\ \ldots\ b_{i_k})$, is order-isomorphic to τ. Otherwise, we say that π *avoids* τ. (To distinguish between permutations and patterns we will omit the brackets in the specification of the pattern.)

Given a set of patterns, the goal is to enumerate the permutations avoiding those patterns (and generating them directly if possible). The hardness of the problem highly depends on the defining set of patterns; for some sets the problem is trivial; for others, it is relatively easy (e.g., there is a recursion formula or an efficient generation algorithm). Other cases are very difficult, and as of now there is no general method which deals with arbitrary sets of patterns. However, there is already a rich theory and many general enumeration methods which cover many cases. Hence, it seems very fruitful to find connections between pattern-avoiding permutations and classes of polyominoes, with the hope that understanding one family of objects will shad light of the other family of objects. For further details regarding the theory of permutation patterns see [36].

For $n = 1, 2, 3$ we have $A(n) = |S_n|$, and for $n = 4$ we have $A(4) = 19$ but $|S_4| = 24$. Hence, it seems natural to investigate sets of five patterns over four symbols. This is true not only for $A(n)$ but also for many other classes of polyominoes containing all the polyominoes up to size 4; an example, discussed in section 4.4, is the set of convex polyominoes (the first non-convex polyomino is the "horseshoe" polyomino of size 5).

A simple way to obtain a permutation from a polyomino is as follows: Define two methods of ordering the cells of a polyomino of size n. Define a permutation $\sigma \in S_n$ where $\sigma(i) = j$, if the ith cell according to the first ordering is numbered j according to the second ordering. This way we obviously generate a permutation out of every polyomino; the challenge is in choosing the numbering methods such that the resulting function from polyominoes to permutations is injective (i.e., the same permutation cannot arise from two different polyominoes).

In Section 4.2 we present a specific injective numbering method which gives rise to the description of some classes of polyominoes using pattern-avoidance. After presenting the method, we elaborate on some of the classes which can be described using it.

4.2 A Bijection between Polyominoes and Permutations

4.2.1 Terminology and notation

Consider the standard orthogonal lattice $\mathbb{Z}^2$ with squares (cells) labeled by pairs of integral coordinates (x, y). A *prepolyomino* is a (possibly disconnected) set of cells on the lattice, considered up to translation (and so a polyomino is an edge-connected prepolyomino).

We define two lexicographical orders of the cells of $\mathbb{Z}^2$, to be denoted by $\leqslant_1$ and $\leqslant_2$:

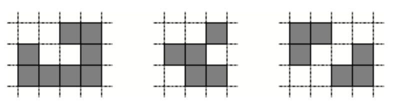
$$\begin{aligned}(a,b) \leqslant_1 (c,d) &\quad\Leftrightarrow\quad a < c \text{ or } (a = c \text{ and } b \leqslant d), \quad \text{and} \\ (a,b) \leqslant_2 (c,d) &\quad\Leftrightarrow\quad b < d \text{ or } (b = d \text{ and } a \geqslant c).\end{aligned}$$

Let $a, b \in \mathbb{Z}$ be two lattice coordinates. A *column* in $\mathbb{Z}^2$ is a set of the form $\{a\} \times \mathbb{Z}$. Similarly, a *row* in $\mathbb{Z}^2$ is a set of the form $\mathbb{Z} \times \{b\}$. A *skew column* is a set of the form $(\{a\} \times [b, +\infty)) \cup (\{a+1\} \times (-\infty, b-1])$. (That is, two opposite "half columns," where the upper half is shifted to the left by one cell relative to the lower half.) Similarly, a *skew row* is a set of the form $((-\infty, a] \times \{b\}) \cup ([a+1, +\infty) \times \{b+1\})$. (That is, two opposite "half rows," where the right half is shifted upward by one cell relative to the left half.) Alternatively, a skew column may be defined as the set $\{(c,d) \in \mathcal{G} : \ (a,b) \leqslant_1 (c,d) \leqslant_1 (a+1, b-1)\}$, and a skew row as the set $\{(c,d) \in \mathcal{G} : \ (a,b) \leqslant_2 (c,d) \leqslant_2 (a+1, b+1)\}$.

For uniqueness of representation of fixed polyominoes, we need to choose one representative of each set of polyominoes that are identical under translation. To this aim we use the same solution proposed in Section 1.3 and *anchor* every polyomino P so that the left-most cell in its bottom row is identified with some lattice cell, usually, but not necessarily, $(0,0)$.[1] A (skew) column or (skew) row of the lattice is said to be *empty* (with respect to P) if it does not contain cells of P. An empty (skew) column or (skew) row *separates* P if there are lattice cells on both sides of the (skew) column or (skew) row, which belong to P. A prepolyomino P is called *reduced* if there is no (skew) column or (skew) row which is empty with respect to P and separates it. Loosely speaking, a prepolyomino P is nonreduced if one can delete from $\mathbb{Z}^2$ one (or more) empty (skew) column(s), and/or one (or more) empty (skew) row(s), and then to glue the lattice portions appropriately so that the two lexicographical relations defined above between cells of P will not be affected. Figure 4.1 shows a few examples of reduced and nonreduced prepolyominoes. In nonreduced prepolyominoes, empty separating (skew) columns and rows are shown by the light shading.

Note that the property of being reduced is well defined since it is invariant under translations. On the other hand, it is not invariant under reflections (see the right polyominoes

[1]For ease of exposition, we deviate from this convention in the proof of Theorem 10.


(a) Reduced

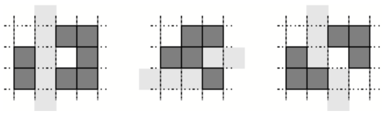
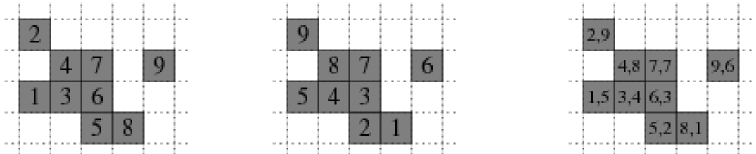

(b) Nonreduced

Figure 4.1: Examples of reduced and nonreduced prepolyominoes and polyominoes



(a) P: $\leqslant_1$ labeling (b) P: $\leqslant_2$ labeling (c) P: double labeling

$\tilde{\varphi}(P) = (5\ 9\ 4\ 8\ 2\ 3\ 7\ 1\ 6)$

Figure 4.2: The definition of $\tilde{\varphi}$

in Figs. 4.1(a,b)). This happens since the notion of reduced prepolyominoes relies on the specific lexicographical orders that we use.

Observation 9 *All polyominoes are reduced.*

The opposite is not true: A reduced prepolyomino is not necessarily even vertex-connected, as is demonstrated by the rightmost polyomino in Figure 4.1(a).

Let us denote the set of prepolyominoes by Π, the set of reduced prepolyominoes by $\mathcal{R}$, and the set of polyominoes by $\mathcal{P}$. The set of prepolyominoes (resp., reduced prepolyominoes, polyominoes) of size n will be denoted by Π_n (resp., $\mathcal{R}_n$, $\mathcal{P}_n$). As usual, the set of permutations of $[n]$ will be denoted by S_n, and let $S = \bigcup_{n\in\mathbb{N}} S_n$.

4.2.2 Representing polyominoes by permutations

We define a function $\tilde{\varphi} : \Pi \to S$ as follows. Let $P \in \Pi_n$ be a prepolyomino. Label the cells of P by $\{1, 2, \ldots, n\}$ in two ways: according to the $\leqslant_1$ and the $\leqslant_2$ order. For all $1 \leqslant i \leqslant n$, let a_i be the label in the $\leqslant_2$ order of the cell that has label i in the $\leqslant_1$ order, and set $\tilde{\varphi}(P) = (a_1\ a_2\ \ldots\ a_n)$. We shall also mark the two indices in which each cell is labeled by the pair $\langle i, a_i\rangle$.[2] See Figure 4.2 for an illustration.

The function $\tilde{\varphi}$ has an interesting graphical interpretation. Refer to Figure 4.3. Let P be a polyomino in Π. Replace each cell of P by its center point so that adjacent cells of P become adjacent nodes of the animal in the dual lattice. Rotate the dual lattice slightly in the clockwise direction without altering the x and y orders between cells of different columns and rows, respectively. This way, one obtains a set of points in which no two points are on the same horizontal or vertical line. Now rearrange the points of this

[2]We write $\langle i, j\rangle$ to distinguish this labeling of a cell (of a specific polyomino) from the standard labeling (i, j) of a cell of the lattice.

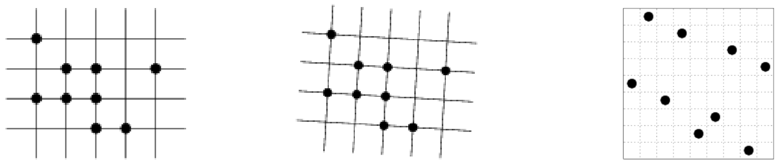

(a) P as an animal (b) P slightly rotated (c) The graph of $\tilde{\varphi}(P)$

Figure 4.3: A graphical interpretation of $\tilde{\varphi}$

set, preserving the horizontal and vertical orders between them, so that they constitute a permutation graph. This permutation is precisely $\tilde{\varphi}(P)$.

Note that the function $\tilde{\varphi}$ is not one-to-one. Consider, for example, the prepolyomino P shown in Figure 4.2 and the prepolyomino P' obtained from P by moving the cell labeled 9 in the $\leqslant_1$ labeling to its left neighboring cell. It is easy to verify that $\tilde{\varphi}(P) = \tilde{\varphi}(P') = (5\ 9\ 4\ 8\ 2\ 3\ 7\ 1\ 6)$. This obviously happens due to the fact that P is not reduced. Indeed, the described change is equivalent to deleting an empty separating skew column and gluing the respective portions of the prepolyomino. As we remarked above, this does not affect the $\leqslant_1$ and $\leqslant_2$ orders of cells of the prepolyomino. However, as we prove below, the function $\varphi : \mathcal{R} \to S$, the restriction of $\tilde{\varphi}$ to *reduced* prepolyominoes, *is* one-to-one. Moreover, it is an onto function, and, thus, it is a bijection.

Theorem 10 *The function $\varphi : \mathcal{R} \to S$ is a bijection.*

Proof: For a cell $C \in P$, once P is anchored in $\mathbb{Z}^2$, the x and y coordinates of C will be denoted by C^x and C^y, respectively.

(i) φ **is one-to-one.** Let P and Q be two reduced prepolyominoes in $\mathcal{R}_n$ such that $\varphi(P) = \varphi(Q)$. In particular, this means that cells $C \in P$ and $D \in Q$ have the same label in the $\leqslant_1$ order if and only if they have the same label in the $\leqslant_2$ order.

For $i = 1, 2, \ldots, n$, denote by C_i the cell of P labeled by i in the $\leqslant_1$ order, and denote by D_i the cell of Q labeled by i in the $\leqslant_1$ order. Anchor P and Q in the lattice $\mathcal{G}$ so that C_1 and D_1 coincide. We shall prove by induction on α that for all $\alpha = 1, 2, \ldots, n$, C_α and D_α also coincide, that is, lie in the same cell of $\mathcal{G}$.

For $\alpha = 1$ there is nothing to prove. Thus, we assume that $\alpha \geqslant 1$ and that C_α and D_α coincide, and our goal is to prove that $C_{\alpha+1}$ and $D_{\alpha+1}$ also coincide. Denote by β the label of C_α and of D_α in the $\leqslant_2$ order. Denote by γ the label of $C_{\alpha+1}$ and of $D_{\alpha+1}$ in the $\leqslant_2$ order. To summarize, the double labeling of C_α and of D_α is $\langle \alpha, \beta \rangle$, and the double labeling of $C_{\alpha+1}$ and of $D_{\alpha+1}$ is $\langle \alpha + 1, \gamma \rangle$.

Assume without loss of generality that C_α and D_α occupy the lattice cell $(0, 0)$. We claim that there are two possible situations:

1. Either $C_{\alpha+1}$ and $D_{\alpha+1}$ lie in $\{0\} \times [1, +\infty)$; or
2. $C_{\alpha+1}$ and $D_{\alpha+1}$ lie in $\{1\} \times (-\infty, 0]$.

Indeed, if $C^x_{\alpha+1} < 0$, or if $C^x_{\alpha+1} = 0$ and $C^y_{\alpha+1} < 0$, then $C_{\alpha+1} <_1 C_\alpha$, which is a contradiction (recall that α and $\alpha + 1$ are the labels of these cells in the $\leqslant_1$ order). If $C^x_{\alpha+1} > 1$, or if $C^x_{\alpha+1} = 1$ and $C^y_{\alpha+1} > 0$, then the skew column $(\{0\} \times [1, +\infty)) \cup (\{1\} \times (-\infty, 0])$ is empty and it separates P, which is impossible since P is a reduced prepolyomino. The same reasoning applies to $D_{\alpha+1}$. Finally, if either $C_{\alpha+1}$ or $D_{\alpha+1}$ lies in $\{0\} \times [1, +\infty)$, then $\gamma > \beta$; and if either of them lies in $\{1\} \times (-\infty, 0]$, then $\gamma < \beta$, which is also impossible. Therefore, it is not the case that one of the cells $C_{\alpha+1}$ and $D_{\alpha+1}$ lies in $\{0\} \times [1, +\infty)$ and the other lies in $\{1\} \times (-\infty, 0]$. Hence, only the two cases listed above are possible.

Case 1: $C_{\alpha+1}$ and $D_{\alpha+1}$ lie in $\{0\} \times [1, +\infty)$. Assume that $C_{\alpha+1}$ and $D_{\alpha+1}$ lie in the lattice cells $(0, k)$ and $(0, \ell)$, respectively. We need to show that $k = \ell$.

Assume for contradiction that $k \neq \ell$, and consider the $\leqslant_2$ order. For $i = 1, 2, \ldots, n$, denote by E_i the cell of P labeled by i in the $\leqslant_2$ order, and denote by F_i the cell of Q labeled by i in the $\leqslant_2$ order. For example, we have $C_\alpha = E_\beta$, $D_\alpha = F_\beta$, $C_{\alpha+1} = E_\gamma$, and $D_{\alpha+1} = F_\gamma$. Let δ be the smallest number such that $\beta < \delta \leqslant \gamma$ and the cells $E_\delta \in P$ and $F_\delta \in Q$ lie in different lattice rows. (Such an index δ exists since at least γ satisfies this condition.) Then, by our assumption, $E_{\delta-1}$ and $F_{\delta-1}$ lie in the same row, say, in the jth row, $E_{\delta-1}$ being in the lattice cell (a, j) and $F_{\delta-1}$ being in the lattice cell (b, j). Assume that $E_{\delta-1}$ and $F_{\delta-1}$ are labeled by ζ in the $\leqslant_1$ order, and that E_δ and F_δ are labeled by η in the $\leqslant_1$ order. Now, similarly to the reasoning applied above to the $\leqslant_1$ order, we obtain that E_δ belongs to the region $((-\infty, a-1] \times \{j\}) \cup ([a, +\infty) \times \{j+1\})$, and F_δ belongs to the region $((-\infty, b-1] \times \{j\}) \cup ([b, +\infty) \times \{j+1\})$, otherwise there exists an empty skew row that separates P. Moreover, it is not the case that E_δ lies in $(-\infty, a-1] \times \{j\}$ and F_δ lies in $[b, +\infty) \times \{j+1\}$, since the first assumption implies that $\eta < \zeta$, whereas the second assumption implies $\eta > \zeta$, which is a contradiction. In a similar manner, it is not the case that E_δ lies in $[a, +\infty) \times \{j+1\}$ and F_δ lies in to $(-\infty, b-1] \times \{j\}$. Therefore,

1. Either E_δ lies in $(-\infty, a-1] \times \{j\}$ and F_δ lies in to $(-\infty, b-1] \times \{j\}$; or
2. E_δ lies in $[a, +\infty) \times \{j+1\}$ and F_δ lies in $[b, +\infty) \times \{j+1\}$.

In both cases, E_δ and F_δ lie in the same row, which contradicts the definition of δ. Hence, $k = \ell$, and, thus, $C_{\alpha+1}$ and $D_{\alpha+1}$ coincide, as claimed.

Case 2: $C_{\alpha+1}$ and $D_{\alpha+1}$ lie in $\{1\} \times (-\infty, 0]$. The proof in this case is similar to that in the previous case and is, therefore, omitted.

(ii) φ **is onto.** Let $\pi \in S_n$. We construct $P \in \mathcal{R}_n$ such that $\varphi(P) = \pi$ as follows.

Let $A_1 \cup A_2 \cup \ldots \cup A_k$ be the partition of $[n]$ into naturally-ordered maximal raises of π, that is, each A_i has the form $\{a, a+1, a+2, \ldots, b\}$, where
$\pi(a) < \pi(a+1) < \pi(a+2) < \ldots < \pi(b)$, $\pi(a) < \pi(a-1)$ (unless $a = 0$), and $\pi(b) > \pi(b+1)$ (unless $b = n$), and $\min\{c : \; c \in A_{i+1}\} = \max\{d : \; d \in A_i\} + 1$ for $i = 1, 2, \ldots, k-1$.

Similarly, let $B_1 \cup B_2 \cup \ldots \cup B_\ell$ be the partition of $[n]$ into naturally-ordered maximal descending left-right runs of π, that is, each B_i has the form $\{a, a+1, a+2, \ldots, b\}$ when $\pi^{-1}(a) > \pi^{-1}(a+1) > \pi^{-1}(a+2) > \ldots > \pi^{-1}(b)$, $\pi^{-1}(a) > \pi^{-1}(a-1)$ (unless $a = 0$), and $\pi^{-1}(b) < \pi^{-1}(b+1)$ (unless $b = n$),
and $\min\{c : \; c \in B_{i+1}\} = \max\{d : \; d \in B_i\} + 1$ for $i = 1, 2, \ldots, \ell - 1$.

Construct a prepolyomino P anchored in $\mathbb{Z}^2$. It will use only the columns $1, 2, \ldots, k$ and only the rows $1, 2, \ldots, \ell$) according to the following rule: *A lattice cell (x, y) contains a cell of P if and only if there is $i \in [n]$ such that $i \in A_x$ and $\pi(i) \in B_y$.* This rule defines a prepolyomino of size n.

First, for each i there are unique coordinates x, y such that $i \in A_x$ and $\pi(i) \in B_y$. Second, it is impossible that for $i_1 \neq i_2$ we have $i_1, i_2 \in A_x$ and $\pi(i_1), \pi(i_2) \in B_y$, since if $i_1 < i_2$ and $i_1, i_2 \in A_x$, then $\pi(i_1) < \pi(i_2)$, which contradicts the assumption that $\pi(i_1), \pi(i_2) \in B_y$.

Next, we prove that P is a reduced prepolyomino. P has no empty separating columns and rows since for every $1 \leqslant x \leqslant k$ there exists $i \in [n]$ such that $i \in A_x$, and for every $1 \leqslant y \leqslant \ell$ there exists $j \in [n]$ such that $\pi(j) \in B_y$. Suppose that P has a separating empty skew column $(\{x\} \times [y, +\infty)) \cup (\{x+1\} \times (-\infty, y-1])$. Let (x, z) be the highest cell in the column x occupied by a cell of P, and let $(x+1, z')$ be the lowest cell in the column $x+1$ occupied by a cell of P. (By the above assumption, these columns are not empty.) Let i, j be two indices for which $i \in A_x, \pi(i) \in B_z$, and $j \in A_{x+1}, \pi(j) \in B_{z'}$. Then, it must be that i is the maximum index in A_x and j is the minimum index in A_{x+1}, and, hence, $j = i+1$, which implies that $\pi(j) < \pi(i)$. On the other hand, since $\pi(i) \in B_z$, $\pi(j) \in B_{z'}$, and $z \leqslant y - 1 < y \leqslant z'$, we have $\pi(i) < \pi(j)$, which is a contradiction.

Finally, we prove that $\varphi(P) = \pi$. For $i \in [n]$, assume that $i \in A_x, \pi(i) \in B_y$, and denote by C_i the cell of P that lies in the lattice cell (x, y). It is easy to see that C_i is labeled by i in the $\leqslant_1$ order of P: It is bigger than all $C_{i'}$ such that $i' \in A_{x'}$ with $x' < x$, as well as of all $C_{i''}$ such that $i'' \in A_x$ with $i'' < i$. Similarly, C_i is labeled by $\pi(i)$ in the $\leqslant_2$ order of the cells of P. To summarize, the double labeling of C_i is $\langle i, \pi(i) \rangle$. Therefore, $\varphi(P) = \pi$.

□

Remark Graphically, in order to reconstruct the reduced prepolyomino $P = \varphi^{-1}(\pi)$ from the permutation π, we consider the graph of π and draw vertical lines which separate maximal raises, and horizontal lines which separate maximal descending runs. Among rectangles obtained in this way, we shade those that contain a point of the graph of π (note that each rectangle contains at most one such point). Finally, we convert all the rectangles into unit squares. The resulting reduced prepolyomino consisting of the shaded squares is P. See Figure 4.4 for an illustration.

Theorem 10 readily implies the following.

Corollary 11 *There are exactly $n!$ reduced prepolyominoes of size n.*

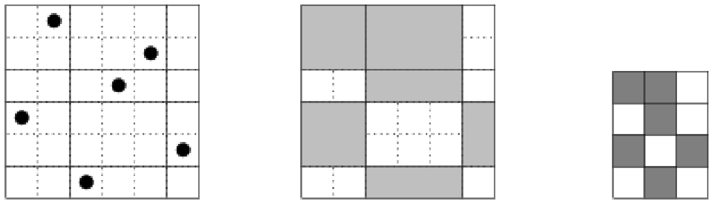

(a) Permutation graph (b) Shaded regions (c) Prepolyomino

Figure 4.4: Reconstructing a prepolyomino P from its image $\varphi(P)$

Let $\psi_n : \mathcal{P}_n \to S_n$ be the restriction of φ (or $\tilde{\varphi}$) to $\mathcal{P}_n$ (polyominoes of size n), and let $\psi = \bigcup_{n\in\mathbb{N}} \psi_n$. The following corollary follows immediately from Theorem 10.

Corollary 12 *The function ψ is one-to-one.*

However, for $n \geqslant 3$ there are reduced prepolyominoes which are not polyominoes, and, thus, ψ_n is not an onto function.

4.3 From Classical Patterns to Bivincular Patterns

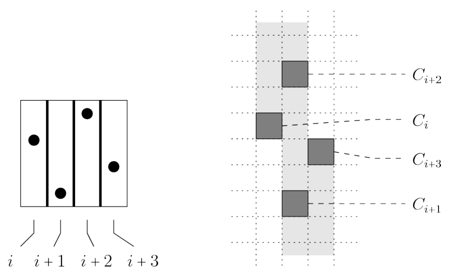
If it were possible to characterize the image of ψ by means of forbidden patterns, that is, to map all polyominoes to permutations that avoid some set of classical patterns, we could hope, using methods from the field of permutation patterns, to improve the existing enumeration estimates for $|\mathcal{P}_n|$. However, the following proposition shows that unfortunately such a characterization does not exist.

Proposition 13 *For any permutation τ, there exists a polyomino $P \in \mathcal{P}$ such that $\psi(P)$ contains τ as a pattern.*

Proof: Let τ be a permutation of $[n]$. Let P be the full $n \times n$ square, and anchor it in $\mathbb{Z}^2$ so that it occupies the square $[1, n] \times [1, n]$. For each $1 \leqslant i \leqslant n$, denote by C_i the cell of P that lies in the cell whose coordinates are $(i, \tau(i))$. Assume that these cells of P have labels $i_1, i_2, \ldots, i_n$ in the $\leqslant_1$ order. Then, it is easy to verify that elements in positions $i_1, i_2, \ldots, i_n$ in $\psi(P)$ form the pattern τ. $\square$

Hence, we cannot hope to classify all polyominoes in terms of permutations avoiding a specific set of patterns. However, we can restrict ourselves to certain families of polyominoes. In addition, we can enhance our vocabulary of describing forbidden patterns. One such generalization is the so-called *bivincular* patterns, introduced recently in [10]. A *bivincular pattern* is an ordered triple $T = \langle \tau, A, B \rangle$, where $\tau = (a_1\ a_2\ \ldots\ a_k)$ is a permutation of $[k]$, and A and B are two subsets of $[k-1]$. A permutation $\pi = (b_1\ b_2\ \ldots\ b_n)$ contains T if

- There exist indices $1 \leqslant i_1 < i_2 < \ldots < i_k \leqslant n$ such that the subpermutation of π, $(b_{i_1}\ b_{i_2}\ \ldots\ b_{i_k})$, is order-isomorphic to τ;
- For every $a \in A$ we have $i_{a+1} = i_a + 1$; and



(a) The pattern T (b) Cells of P

Figure 4.5: For a polyomino $P \in \mathcal{P}$, $\psi(P)$ avoids $T = \langle 3142, \{1,2,3\}, \varnothing\rangle$

- For every $b \in B$ we have $\pi(i_{b+1}) = \pi(i_b) + 1$.

Otherwise, π avoids T.

Let us explain the meaning of a bivincular pattern. If π contains a classical pattern τ, this means that some points in the graph of π are order-isomorphic to the graph of τ, with no restriction on whether or not they are adjacent (column-wise of row-wise). In contrast, if π contains a bivincular pattern $\langle \tau, A, B\rangle$, then it is required that some of this points belong to adjacent columns or rows in the graph of π. These requirements are indicated in the sets A and B, for columns and rows, respectively.

Obviously, for $A = B = \varnothing$, we get classical patterns; in this case we shall continue writing, for simplicity, τ rather than $\langle \tau, \varnothing, \varnothing\rangle$. For $B = \varnothing$ we get generalized permutation patterns in a so-called *dash notation* [3]. For example, the bivincular pattern $\langle 213, \{2\}, \varnothing\rangle$ is specified as 2-13 in the dash notation.

In contrast with Preposition 13, there exist bivincular patterns that do not occur in permutations corresponding to polyominoes.

Proposition 14 *For any polyomino $P \in \mathcal{P}$, the permutation $\psi(P)$ avoids the bivincular pattern $T = \langle 3142, \{1,2,3\}, \varnothing\rangle$.*[3]

Proof: Refer to Figure 4.5. Assume for contradiction that there is polyomino $P \in \mathcal{P}$, such that the permutation $\pi = \psi(P)$ has the pattern T. That is, there is an index i, $1 \leqslant i \leqslant n-3$, such that $\pi(i+1) < \pi(i+3) < \pi(i) < \pi(i+2)$. For each $j \in [n]$, let C_j be the cell of P whose double labeling is $\langle j, \pi(j)\rangle$. Hence, C_{i+1} and C_{i+2} lie in the same column, C_i lies in the preceding column, and C_{i+3} lies in the following column. Let $(a,b), (a+1,c), (a+1,d), (a+2,e)$ be the lattice cells in which $C_i, C_{i+1}, C_{i+2}, C_{i+3}$ lie, respectively. Therefore we have $b < c \leqslant d < e$. Evidently, the areas $\{a\} \times [d+1, +\infty]$, $\{a+1\} \times [-\infty, b-1]$, $\{a+1\} \times [b+1, e-1]$, $\{a+1\} \times [e+1, +\infty]$, and $\{a+2\} \times [-\infty, c-1]$ are empty. This means that the region
$(\{a\} \times [e, +\infty]) \cup (\{a+1\} \times [b+1, e-1]) \cup (\{a+2\} \times [-\infty, b])$ separates P into two non-empty parts, which contradicts the fact that P is a polyomino. □

[3]In dash notation, this is the pattern 3142, without any dashes.

Remark It is quite simple to generalize Proposition 13 and show that for any bivincular pattern T of the form $\langle \tau, A, \varnothing \rangle$, with $|A| < 3$, or of the form $\langle \tau, \varnothing, B \rangle$, with $|B| < 3$, there exists a polyomino $P \in \mathcal{P}$ such that the permutation $\psi(P)$ contains the pattern T.

4.4 Convex Polyominoes

A prepolyomino P is *convex* if any vertical or horizontal line crosses P in at most one continuous sequence of cells. Formally, P is convex if the following conditions hold:

1. If $(a, d), (a, f) \in P$ with $d < f$, then for each e such that $d < e < f$, we have $(a, e) \in P$; and

2. If $(a, d), (c, d) \in P$ with $a < c$, then for each b such that $a < b < c$, we have $(b, d) \in P$.

If the first (resp., second) condition holds, P is vertically (resp., horizontally) convex. Thus, a prepolyomino is convex if and only if it is both horizontally and vertically convex.

Proposition 15 *A convex reduced prepolyomino is also a polyomino.*

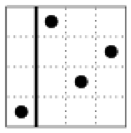
Proof: Since P is convex, each column of P is of the form $\{a\} \times [b, c]$ (where $b \leqslant c$). We shall prove that any two adjacent non-empty columns of P are edge-connected, and, hence, P is a polyomino. Assume that P is anchored in the lattice, that its intersection with the lattice column a is $\{a\} \times [b, c]$ (where $b \leqslant c$), and that its intersection with the lattice column $a + 1$ is $\{a + 1\} \times [d, e]$ (where $d \leqslant e$). Our goal is to show that $d \leqslant c$ and $e \geqslant b$.

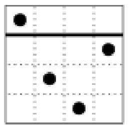
If $d > c$, then $(\{a\} \times [c + 1, +\infty)) \cup (\{a + 1\} \times (-\infty, c])$ is an empty skew-column that separates P—a contradiction. Assume, then, that $e < b$. The cell $(a + 1, b)$ is empty by our assumption. Therefore, the half-row $[a + 2, +\infty) \times \{b\}$ is empty since P is (horizontally) convex. Hence, the half-row $(-\infty, a - 1] \times \{b - 1\}$ is not empty, otherwise $((-\infty, a] \times \{b - 1\}) \cup ([a + 1, +\infty) \times \{b\})$ would be an empty skew-row separating P. Repeating this reasoning inductively, we find that for every f, $e < f \leqslant b$, $[a + 2, +\infty) \times \{f\}$ is empty while $(-\infty, a - 1] \times \{f - 1\}$ is not. Eventually, we get that $(-\infty, a - 1] \times \{e\}$ is not empty. However, we have that $(a, e) \notin P$ and $(a + 1, e) \in P$—a contradiction to P being (horizontally) convex.

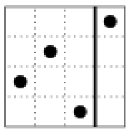
Therefore, $d \leqslant c$ and $e \geqslant b$, hence, P is edge-connected, that is, P is a polyomino. □

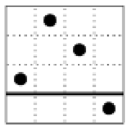
Remark If a prepolyomino P is only vertically convex (or only horizontally convex), then it is not necessarily a polyomino. For counterexamples, consider $\varphi^{-1}(4\ 2\ 1\ 3)$ and $\varphi^{-1}(1\ 4\ 2\ 3)$.

Theorem 16 *The image of the family of convex polyominoes under ψ is precisely the class of ($\langle 1423, \{1\}, \varnothing \rangle$, $\langle 4213, \varnothing, \{3\} \rangle$, $\langle 2314, \{3\}, \varnothing \rangle$, $\langle 2431, \varnothing, \{1\} \rangle$, $\langle 3142, \{2\}, \{2\} \rangle$)-avoiding permutations.*


(a) $\langle 1423, \{1\}, \varnothing \rangle$


(b) $\langle 4213, \varnothing, \{3\} \rangle$


(c) $\langle 2314, \{3\}, \varnothing \rangle$


(d) $\langle 2431, \varnothing, \{1\} \rangle$

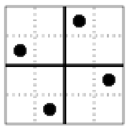

(e) $\langle 3142, \{2\}, \{2\} \rangle$

Figure 4.6: Bivincular patterns avoided by permutations that correspond to convex polyominoes

Figure 4.6 shows the patterns mentioned in the theorem above. Bold lines separate columns or rows which are required to be adjacent. Notice that the first four patterns are rotationally equivalent.

Proof: Denote by $\mathcal{F}$ the class of permutations that avoid the bivincular patterns indicated in the statement of the theorem. Let $P \in \mathcal{R}$. We shall prove that P is a convex polyomino if and only if $\pi = \varphi(P) \in \mathcal{F}$. Denote the cell of P, whose double labeling is $\langle i, \pi(i) \rangle$, by C_i.

1. Let P be a convex polyomino anchored in $\mathbb{Z}^2$. Our goal is to show that π avoids the five patterns indicated in the statement of the theorem. The five proofs are similar, and so we shall prove in detail only that π avoids the first pattern. Assume for contradiction that π has the pattern $\langle 1423, \{1\}, \varnothing \rangle$. That is, there exist indices $1 \leqslant i < j < k < \ell \leqslant m$ such that $\pi(i) < \pi(k) < \pi(\ell) < \pi(j)$ and $j = i + 1$. Then, i and j belong to the same raise of π, and therefore, the cells C_i and C_j lie in the same column of P, C_j being above C_i. Furthermore, since P is convex, C_j lies immediately above C_i. Assume, without loss of generality, that C_i lies in the lattice cell $(0, p)$, and so C_j lies in the lattice cell $(0, p + 1)$. Now assume that C_k and C_ℓ lie in the rows r and s, respectively. Since $j < \ell$ and $\pi(j) > \pi(\ell)$, we have that $s \leqslant p + 1$. Since $k < \ell$, we have that $\pi(k)$ and $\pi(\ell)$ belong to different descending left-right runs, and, therefore, $r < s$. Similarly, since $i < k$, we have $p < r$. To summarize, $p < r < s \leqslant p + 1$, which is impossible.

2. Let P be a nonconvex reduced prepolyomino anchored in $\mathbb{Z}^2$. Suppose that P is not vertically convex. Then, there exist cells $C_i, C_{i+1} \in P$ such that, without loss of generality, C_i (resp., C_j) lies in the lattice cell $(0, p)$ (resp., $(0, q)$), where $q - p > 1$, and the region $\{0\} \times [p+1, q-1]$ is empty. If for $r_1 \neq r_2$, $p < r_1, r_2 \leqslant q$, the two half-rows $[1, +\infty) \times \{r_t\}$ $(t = 1, 2)$ are not empty, then π has the pattern $\langle 1423, \{1\}, \varnothing \rangle$. Similarly, if for $s_1 \neq s_2$, $p \leqslant s_1, s_2 < q$, the two half-rows $(-\infty, -1] \times \{s_t\}$ $(t = 1, 2)$

are not empty, then π has the pattern $\langle 2314, \{3\}, \varnothing\rangle$. Otherwise, if $[1, +\infty) \times \{r\}$ is not empty for at most one value of r, $p < r \leqslant q$, and $(-\infty, -1] \times \{r\}$ is not empty for at most one value of s, $p \leqslant s < q$, then $r = s = p + 1 = q - 1$ (otherwise there is an empty skew row that separates P), in which case π has the pattern $\langle 3142, \{2\}, \{2\}\rangle$. Similarly, one can prove that if P is not vertically convex, then the corresponding permutation has either $\langle 4213, \varnothing, \{3\}\rangle$, $\langle 2431, \varnothing, \{1\}\rangle$, or $\langle 3142, \{2\}, \{2\}\rangle$ as a pattern.

$\square$

Remark The conditions (1) π *avoids* $T = \langle 3142, \{2\}, \{2\}\rangle$; (2) π *avoids* $T' = \langle 3142, \{2\}, \varnothing\rangle$ (31-42 in the dash notation); and (3) π *avoids* $T'' = \langle 3142, \varnothing, \{2\}\rangle$; are equivalent. Therefore, one can replace in Theorem 16 the pattern T by either (the formally weaker) T' or T''.

The generating function for the enumeration sequence can be found in [9, 11]. The first few values of the sequence are $(1, 2, 6, 19, 59, 176, 502, 1374, 3630, 9312, \ldots)$ (sequence A067675 in [38]).

4.5 Tree Convex Polyominoes

In this section we enumerate $\mathbf{H}$, the family of tree *convex* polyominoes, defined as:

Definition 17 *A* tree *convex polyomino is a convex polyomino that does not contain a* 2×2 *block.*

(This might not be the most "natural" definition but it is best suited for our purposes.)

We demonstrate how the injection ψ from polyominoes to permutations can be used for this purpose. The key idea for enumerating tree convex polyominoes is classifying them as permutations that avoid specific patterns.

Theorem 18 *The image of* $\mathbf{H}$ *the function* ψ *is precisely*

$$\mathcal{H} = S(\,1423,\ 4213,\ 2314,\ 2431,\ \langle 3142, \{2\}, \{2\}\rangle,\ 2413\,).$$

Proof: Let us denote

$$\mathcal{H}' = S(\langle 1423, \{1\}, \varnothing\rangle, \langle 4213, \varnothing, \{3\}\rangle, \langle 2314, \{3\}, \varnothing\rangle, \langle 2431, \varnothing, \{1\}\rangle, \langle 3142, \{2\}, \{2\}\rangle, 2413).$$

We shall see that in fact $\mathcal{H}' = \mathcal{H}$. However, since H' is clearly a subclass of $\mathcal{G}$ from Theorem 16, it will be more natural and easy to prove that $\psi(\mathbf{H}) = \mathcal{H}'$.

Let $P \in \mathcal{P}$ be a polyomino.[4] We shall first prove that P is a tree convex polyomino if and only if $\rho = \psi(P) \in \mathcal{H}'$. Then, we shall prove that $\mathcal{H}' = \mathcal{H}$. Denote the cell of P, whose double labeling is $\langle i, \pi(i)\rangle$, by C_i.

[4]Note that we could write here $P \in \mathcal{R}$, and the entire proof would hold for "tree convex reduced prepolyominoes." Thus, the function ψ may be replaced by φ throughout the discussion.

1. Let P be a tree convex polyomino anchored in $\mathbb{Z}^2$, and let $\rho = \psi(P)$. Since P is convex, ρ avoids the first five patterns in the definition of $\mathcal{H}'$. It remains to show that ρ avoids the pattern 2413. Assume for contradiction that ρ has the pattern 2413, that is, there are indices $1 \leqslant i < j < k < \ell \leqslant n$ such that $\rho(k) < \rho(i) < \rho(\ell) < \rho(j)$. Consider C_i, C_j, C_k, and C_ℓ. Assume that these cells of P lie, respectively, in the cells (a, e), (b, f), (c, g), and (d, h) of $\mathbb{Z}^2$. Then, we must have $a \leqslant b < c \leqslant d$ and $g \leqslant e < h \leqslant f$. Consider the half-column $\{b\} \times (-\infty, e]$. Due to the convexity of P, there is a continuous path (a sequence of edge-connected neighboring cells of P) from C_1 to C_3, which contains a lattice cell that belongs this half-column. Therefore, since $(b, f) \in P$, we also have $(b, e) \in P$. Similar arguments also show that $(b, h), (c, e), (c, h) \in P$. Now, the convexity of P implies that for all r, s, such that $b \leqslant r \leqslant c$ and $e \leqslant s \leqslant h$, the lattice cell (r, s) belongs to P. In particular, $(b, e), (b, e+1), (b+1, e), (b+1, e+1)$ belong to P. Thus, P contains a 2×2 block—a contradiction.

2. Conversely, assume that P is not convex tree. If it is not convex, then, by Theorem 16, $\rho = \psi(P)$ contains one of the first five patterns in the definition of $\mathcal{H}'$. Assume now that P contains a 2×2 block, the cells C_i, C_j, C_k, C_ℓ which lie, respectively, in the lattice cells $(a, e), (a, e+1), (a+1, e), (a+1, e+1)$. Then, we have $1 \leqslant i < j < k < \ell \leqslant n$ and $\rho(k) < \rho(i) < \rho(\ell) < \rho(j)$, and, thus, ρ must have the pattern 2413.

3. It remains to prove that $\mathcal{H}'$ is identical to $\mathcal{H}$, the class stated in the theorem. Let $\rho \in \mathcal{H}'$; our aim is to prove that it avoids the patterns 1423, 4213, 2314, and 2431. The four proofs are similar (notice that the graphs of these permutations are rotationally equivalent), and so we shall prove in detail only that ρ avoids 1423. Assume for contradiction that there are indices $1 \leqslant i < j < k < \ell \leqslant n$ such that $\rho(i) < \rho(k) < \rho(\ell) < \rho(j)$. Let i' be the maximum number such that $i' < j$ and $\rho(i') < \rho(\ell)$. (Such an index i' exists since at least i satisfies this condition.) Then, $\rho(i'+1) > \rho(\ell)$. It cannot be the case that $\rho(k) < \rho(i') < \rho(\ell)$, otherwise i', j, k, ℓ form the forbidden pattern 2413. Therefore, $\rho(i') < \rho(k)$. However, in this case $i', i'+1, k, \ell$ form the pattern $\langle 1423, \{1\}, \varnothing \rangle$, and the claim follows.[5]

□

Our strategy for enumerating $\mathbf{H}$, the family of tree convex polyominoes, is to partition it into two disjoint families, and count each subfamily separately. The first family, $\mathbf{X}$, consists of so-called "positive pseudocrosses" (see below), and the second family, $\mathbf{Y}$, consists of all tree convex polyominoes which are not (or do not contain) positive pseudocrosses.

A *positive pseudocross* is either

- A *horizontal positive pseudocross*: a polyomino of the form

$$([a, d] \times \{f\}) \cup (\{b\} \times [e, f-1]) \cup (\{c\} \times [f+1, h]),$$

[5]The reader may notice that all the forbidden classical patterns can be obtained at once by analyzing directly the structure of convex polyominoes. For clarity of presentation, we prefer to first prove the claim for generalized patterns, and then analyze some of the latter in order to replace them by classical patterns.

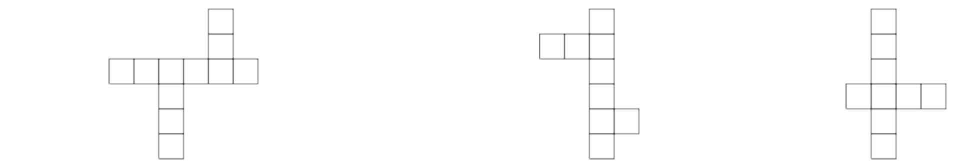

(a) a horizontal pseudocross (b) a vertical pseudocross (c) a cross

Figure 4.7: Positive pseudocrosses

where $a < b < c < d$ and $e < f < h$;

- A *vertical positive pseudocross*: a polyomino of the form

$$(\{b\} \times [e,h]) \cup ([a,b-1] \times \{g\}) \cup ([b+1,d] \times \{f\}),$$

where $a < b < d$ and $e < f < g < h$; or

- A *cross*: a polyomino of the form

$$([a,d] \times \{f\}) \cup (\{b\} \times [e,h]),$$

where $a < b < d$ and $e < f < h$.

Figure 4.7 shows a few positive pseudocrosses. (*Negative pseudocrosses* are defined analogously, but are irrelevant here. Note that the set of positive pseudocrosses (which are not crosses) is not invariant under reflection about axis-parallel lines: A positive pseudocross is reflected to a negative pseudocross, and vice versa.)

Observation 19 *A convex tree polyomino P contains a positive pseudocross if and only if P itself is a positive pseudocross.*

The definition of positive pseudocrosses is motivated by the following fact.

Theorem 20 *The image under ψ of $\mathbf{Y}$, the family of tree convex polyominoes which are* not *positive pseudocrosses, is precisely*

$$\mathcal{Y} = S(\,1423,\ 4213,\ 2314,\ 2431,\ 3142,\ 2413\,).$$

Proof: In view of Theorem 18 and Observation 19, it suffices to show that a tree convex polyomino P is a pseudocross if and only if $\psi(P)$ contains the pattern 3142.

The "only if" direction is immediate from the definitions of φ (recall that ϕ is its restriction) and of pseudocrosses.

For the "if" direction, assume that $\psi(P)$ contains the pattern 3142. Then, P contains four cells C_i, C_j, C_k, C_ℓ which lie, respectively, in the lattice cells $(a,e), (b,f), (c,g), (d,h)$, such that $a < b \leqslant c < d$ and $e < f \leqslant g < h$.

If both $b < c$ and $f < g$, then, by reasoning very similar to that in the proof of Theorem 18 (part 1), we obtain that P contains a 2×2 block. Assume, then, that $b < c$

and $f = g$. Since P is convex, it contains $[a, d] \times \{f\}$, $\{b\} \times [e, f-1]$, and $\{c\} \times [f+1, h]$. Thus, P contains a positive (horizontal) pseudocross; therefore, by Observation 19, P is a pseudocross.

Similarly, if $b = c$ and $f < g$, then P is a vertical pseudocross, and if $b = c$ and $f = g$, then P is a cross. $\square$

Recall the definition of a *separable permutation*. A non-empty permutation π is separable either if it is the unique permutation of $\{1\}$, or if the graph of π can be split into two non-empty blocks B_1 and B_2, which are themselves separable in a recursive manner. In the latter case either all the points in B_1 are to the "south-west" of all the points of B_2 (i.e., π has an *ascending structure*), or all the points in B_1 are to the "north-west" of all the points of B_2 (i.e., π has a *descending structure*).

Let π be a separable permutation of $[n]$ (where $n \geqslant 2$) with an ascending structure. Then, π can be split in a unique way into blocks $C_1, C_2, \ldots, C_k$, such that for $i = 1, 2, \ldots, k-1$, all the points in C_i are to the "south-west" of all the points of C_{i+1}, and each block C_i, $i = 1, 2, \ldots, k$, either consists of one point or has a descending structure. We call the sequence $(C_1, C_2, \ldots, C_k)$ the *first-level decomposition* of the structure of π. A similar definition refers to permutations with a descending structure: in this case all the points in C_i are to the "north-west" of all the points of C_{i+1}, and each block C_i either consists of one point or has an ascending structure.

Separable permutations are known to be precisely $(2413, 3142)$-avoiding permutations [8]. Since $\mathcal{Y}$ avoids, inter alia, the patterns 2413 and 3142, it is a subclass of the class of separable permutations.

We prove the following two claims on the structure of permutations which belong to $\mathcal{Y}$ and of permutations which belong to $\mathcal{X} = \psi(\mathbf{X})$.

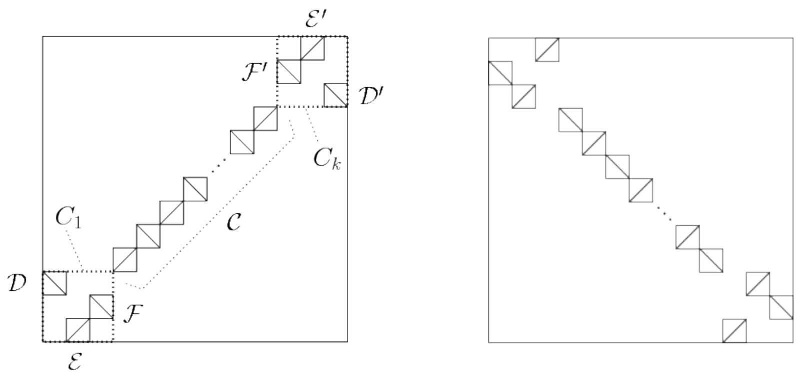
Proposition 21 *Consider $\pi \in \mathcal{Y}$, and assume that π has an ascending structure. Let $(C_1, C_2, \ldots, C_k)$ be the first-level decomposition of the structure of π. Then:*

1. *Each of the blocks $C_2, C_3, \ldots, C_{k-1}$ consists of either one point or several points in descending position;*

2. *If C_1 contains more than one point, then the first-level decomposition of its (descending) structure is $(D_1, D_2, \ldots, D_\ell)$, where $D_1, D_2, \ldots, D_{\ell-1}$ contain one point, and the first-level decomposition of the (ascending) structure of D_ℓ (if it contains more than one point) is $(E_1, E_2, \ldots, E_m)$, where $E_1, E_2, \ldots, E_{m-1}$ contain one point, and E_m (if it contains more than one point) consists of points in descending position.*

3. *Similarly, if C_k contains more than one point, then the first-level decomposition of its (descending) structure is $(D'_1, D'_2, \ldots, D'_{\ell'})$, where $D'_2, \ldots, D'_{\ell'-1}, D'_{\ell'}$ contain one point, and the first-level decomposition of the (ascending) structure of D'_1 (if it contains more than one point) is $(E'_1, E'_2, \ldots, E'_{m'})$, where $E'_2, \ldots, E'_{m'-1}, E'_{m'}$ contain one point, and E'_1 (if it contains more than one point) consists of points in descending position.*

A similar claim holds if π has a descending structure.



(a) π has an ascending structure (b) π has a descending structure

Figure 4.8: Permutations corresponding to tree convex polyominoes which do not contain a positive pseudocross (all blocks may be empty)

Figure 4.8(a) shows schematically a permutation $\pi \in \mathcal{Y}$ with an ascending structure. In the figure, ⧄ denotes a block consisting of points in ascending position, and ⧅ denotes a block consisting of points in descending position. If a few consecutive blocks C_is consist of one point, then their union may be seen as an ascending block ⧄. Therefore, $\mathcal{C} := C_2 \cup \ldots \cup C_{k-1}$ consists of alternating ⧄ and ⧅ blocks. Similarly, $\mathcal{D} := D_1 \cup \ldots \cup D_{\ell-1}$ and $\mathcal{D}' := D'_2 \cup \ldots \cup D'_{\ell'}$ are ⧅; $\mathcal{E} := E_1 \cup \ldots \cup E_{m-1}$ and $\mathcal{E}' := E'_2 \cup \ldots \cup E'_{m'}$ are ⧄; finally, $\mathcal{F} := E_m$ and $\mathcal{F}' := E'_1$ are ⧅. Figure 4.8(b) illustrates schematically a permutation in $\mathcal{Y}$ with an ascending structure. This diagram may be obtained from that in Figure 4.8(a) by a 90° rotation.

Proof: Let π be a $(1423, 4213, 2314, 2431, 3142, 2413)$-avoiding permutation of $[n]$, where $n \geqslant 2$, and assume that it has an ascending structure.

1. Consider a block C_i, where $2 \leqslant i \leqslant k-1$. Assume that C_i contains two points P and Q, such that Q is to the north-east of P. Since C_i has a descending structure, either it contains a point R which lies to the north-west of both P and Q, or it contains a point R' which lies to the south-east of both P and Q. However, in the first case the points S, R, P, Q, where S is any point of C_{i-1}, form the forbidden pattern 1423, while in the second case the points P, Q, R', S', where S' is any point of C_{i+1}, form the forbidden pattern 2314.

2. The reasoning here is very similar to that of the previous item. First, if for some $1 \leqslant i \leqslant \ell - 1$, D_i (whose structure is ascending unless it consists of one point) contains two points in ascending position, then these two points, together with any point of D_ℓ and any point of C_2, form the forbidden pattern 2314.

   Next, if for some $1 \leqslant i \leqslant m-1$, E_i (whose structure is descending unless it consists of one point) contains two points in descending position, then these two points, together with any point of $D_{\ell-1}$ and any point of E_m, form the forbidden pattern 4213.

   Finally, if E_m contains two points P and Q in ascending position, then either these

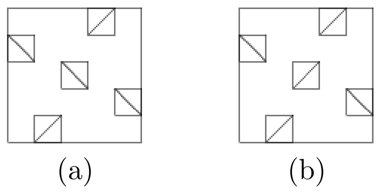


Figure 4.9: Permutations corresponding to tree convex polyominoes which contain a positive pseudocross

two points form the forbidden pattern 1423 together with a point of E_1 and a point which lies in E_m to the north-west of both P and Q; or they form the forbidden pattern 2314 together with a point of C_2 and a point which lies in E_m to the south-east of both P and Q.

3. The proof is essentially identical to that of the previous item (notice the symmetry in Figure 4.8).

The proof for permutations π with a descending structure (refer to Figure 4.8(b)) is identical.

On the other hand, it is easy to verify that any permutation, whose structure is as in Figure 4.8, avoids all of the patterns 1423, 4213, 2314, 2431, 3142, and 2413. □

Proposition 22 *All permutations in $\mathcal{X}$ have one of the structures shown in Figure 4.9 (all blocks are non-empty).*

This follows directly from the definitions of ψ and of pseudocrosses; therefore, the proof is omitted. If a polyomino P is a horizontal pseudocross, then the central block in $\psi(P)$ is ⧅; if P is a vertical pseudocross, then the central block is ⧄; and if P is a cross, then the central block consists of one point.

Remark One can also consider all possible ways to construct tree convex polyominoes, inspect to which permutations they are mapped by $\tilde{\varphi}$, using the graphical interpretation of this function (see Figure 4.3), and observe that these are precisely the permutations with the structures described in Propositions 21 and 22.

Considering the structures of $\mathcal{Y}$ and $\mathcal{X}$, we find the generating function for the enumerating sequence of $\mathcal{H} = \mathcal{Y} \cup \mathcal{X}$, and, therefore, of $\mathbf{H} = \mathbf{Y} \cup \mathbf{X}$.

Proposition 23 *The generating function for the enumerating sequence of $\mathcal{H}$ is*

$$x + 2 \cdot \left(\left(\frac{x}{1-2x} \right) \cdot \left(\left(\frac{x}{1-x} \right)^3 + 1 \right)^2 + \left(\frac{x}{1-x} \right)^6 - \left(\frac{x}{1-x} \right) \right) + \frac{(1+x)x^5}{(1-x)^5}. \quad (4.1)$$

Proof: Consider first the structure of a permutation in $\mathcal{Y}$ with an ascending structure (refer to Figure 4.8(a)). The term $\frac{x}{1-2x}$ counts the number of ways to insert the part denoted by $\mathcal{C}$. If C_1 consists of more than one point and is not just ⧅, then $\mathcal{D}, \mathcal{E}, \mathcal{F}$ are non-empty, and the term $\left(\frac{x}{1-x}\right)^3$ counts the number of ways to insert these blocks. The addition of 1 is to account for the case in which C_1 consists of more than one point or has the form ⧅. The expression is squared since the same is true for the blocks $\mathcal{D}', \mathcal{E}', \mathcal{F}'$. The term $\left(\frac{x}{1-x}\right)^6$ counts the number of such permutations in which $\mathcal{C}$ is empty, but $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, $\mathcal{D}'$, $\mathcal{E}'$, and $\mathcal{F}'$ are non-empty. The term $\frac{x}{1-x}$ is deducted to avoid the counting of the permutation $(n\ n-1\ \ldots\ 2\ 1)$, which in fact has a descending separation structure. The expression described by now is multiplied by 2 in order to count permutations with a descending separation as well. The summand x is added since the unique permutation of $\{1\}$ was undercounted. The expression described thus far counts the number of permutations in $\mathcal{Y}$.

The term $\frac{(1+x)x^5}{(1-x)^5}$ counts the number of permutations in $\mathcal{X}$: $\frac{x^5}{(1-x)^5}$ counts permutations as in Figure 4.9(a), while $\frac{x^6}{(1-x)^5}$ counts permutations as in Figure 4.9(b), excluding those that have just one point in the central block and, thus, have been already counted. □

The expression in Eq. (4.1) simplifies to

$$\frac{x(1-4x+8x^2-6x^3+4x^4)}{(1-x)^4(1-2x)}.$$

Since this is a rational function, the enumerating sequence satisfies a linear recurrence [39, p. 202, Th. 4.1.1] whose characteristic equation is $(x-2)(x-1)^4 = 0$, that is,

$$a_n = 6a_{n-1} - 14a_{n-2} + 16a_{n-3} - 9a_{n-4} + 2a_{n-5}.$$

The first numbers in the sequence (for $n \geqslant 1$) are

$$(1, 2, 6, 18, 51, 134, 328, 758, 1677, 3594, 7530, 15530, 31687, 64190, 129420, \ldots)$$

We solve the recursion using the initial values and obtain that the nth value of the sequence is given by the formula

$$a_n = 2^{n+2} - \frac{n^3 - n^2 + 10n + 4}{2}. \tag{4.2}$$

Notice that the asymptotic growth rate, 2, was expected since the principal contribution in the generating function is that of the term $\frac{x}{1-2x}$.

Chapter 5

Polyominoes on a Twisted Cylinder

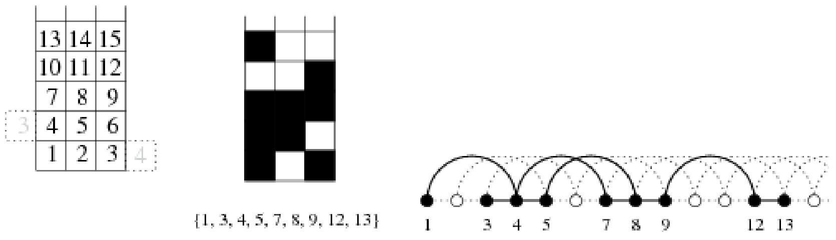
5.1 Introduction

The notion of polyominoes on a *twisted cylinder* was introduced in [4]. For a fixed natural number w, the twisted cylinder of *width*[1] w is the surface obtained from the Euclidean plane by identifying all pairs of points of the form (x, y), $(x - kw, y + k)$, for $k \in \mathbb{Z}$. Pictorially, we can imagine this as taking the stripe $[0, w] \times \mathbb{R}$ and gluing, for all y, (w, y) with $(0, y + 1)$ (see Figure 5.1(a)).

A polyomino on the twisted cylinder is a finite edge-connected set of unit integer cells on this cylinder. Two polyominoes obtained from each other by translation are considered identical. In order to have a unique representation, we shall assume without loss of generality that the cell $[0, 1] \times [0, 1]$ is the leftmost cell belonging to all polyominoes. We shall label this cell by 1, and all other cells in the cylinder to the right of 1 so that the cell i will be the right neighbor of the cell $i - 1$. Therefore, the twisted cylinder of width w may be seen as a host graph (dual of the lattice graph) whose vertices are labeled by $\mathbb{N}$, and there is an edge $\{i, j\}$ if $|i - j| = 1$ (a horizontal edge) or $|i - j| = w$ (a vertical edge). Alternatively, we have a half-infinite circulant graph with $\aleph_0$ vertices, in which each vertex $i \in \mathbb{N}$ is connected to vertices $i \pm 1$ and $i \pm w$ (unless the attempted connection is to a non-natural number). In this interpretation, a polyomino P may be regarded as a finite subset of $\mathbb{N}$ such that $1 \in P$ and the corresponding induced subgraph (in the mentioned-above host graph) is connected. Figures 5.1(b,c) show a sample polyomino on the twisted cylinder of width 3.

Denote the number of fixed polyominoes on a twisted cylinder of width w by $A^{(w)}(n)$. It is easy to observe that for any fixed values of w and n, such that $n > w$, we have $A^{(w)}(n) < A(n)$. This follows from the fact that one can easily embed any polyomino on a twisted cylinder in the regular plane, but the opposite operation is not always possible due to self overlaps of the polyominoes. Furthermore, it is also true [4] that for any fixed value of w, the limit $\lambda^{(w)} := \lim_{n\to\infty} A^{(w)}(n+1)/A^{(w)}(n)$ exists and that $\lambda^{(1)} \leqslant \lambda^{(2)} \leqslant \lambda^{(3)} \leqslant \ldots$.

[1]The term 'width,' and not 'perimeter,' is used for consistency with [19] and [4].



(a) Cylinder (b) Polyomino (c) Subgraph of the host graph

Figure 5.1: A polyomino on the twisted cylinder of width 3

We now show that $\lim_{w\to\infty} \lambda^{(w)} = \lambda$. First note that $A^{(w)}(w) = A(w)$ since every polyomino of size w in the plane can be embedded in the twisted cylinder of width w. Hence,

$$(1)\ \lim_{w\to\infty} \sqrt[w]{A^{(w)}(w)} = \lim_{w\to\infty} \sqrt[w]{A(w)} = \lambda$$

Using a concatenation argument (similar to the one shown in subsection 1.1), one can see that $A^{(w)}(n) \cdot A^{(w)}(m) \leqslant A^{(w)}(m+n)$, and in particular $(A^{(w)}(n))^2 \leqslant A^{(w)}(2n)$. Hence, $A^{(w)}(n) \leqslant \sqrt{A^{(w)}(2n)}$ and so $\sqrt[n]{A^{(w)}(n)} \leqslant \sqrt[2n]{A^{(w)}(2n)}$. This implies that $\sqrt[n]{A^{(w)}(n)} < \lambda^{(w)}$ for every n, since for any given n, $\sqrt[n\cdot 2^i]{A^{(w)}(n\cdot 2^i)}$ is a rising subsequence converging to $\lambda^{(w)}$, and so it must lie entirely below it.

Now, given $\varepsilon > 0$, choose W_0 such that for all $w > W_0$ we have $\sqrt[w]{A(w)} > \lambda - \varepsilon$ (this is always possible due to equation 1). Take any $w > W_0$; now we have that $\lambda - \varepsilon < \sqrt[w]{A(w)} = \sqrt[w]{A^{(w)}(w)} < \lambda^{(w)}$. Hence, $\lambda^{(w)} \to \lambda$, as required.

It is shown in [4] that $\lambda^{(w)}$ is the only positive eigenvalue of a huge matrix (the size of each of which dimensions is proportional to the wth Motzkin number), and much effort is spent in this work for computing numerically this eigenvalue for as large as possible values of w.

The currently best lower bound on λ is achieved by computing $\lambda^{(22)} \approx 3.9801$. Finding explicit formulae for the numbers of polyominoes on twisted cylinders of different widths, and understanding the relations between these formulae, might shed light on the real value of λ.

5.2 Transfer Matrix

Polyominoes on twisted cylinders are relatively easy to enumerate due to a limit on the size of their boundary; more specifically, when constructing them square by square, their boundary (the set of cells whose neighbors have not been reached by the construction method yet) can be limited to the width of the cylinder. This can be used to give a transfer-matrix algorithm for their enumeration.

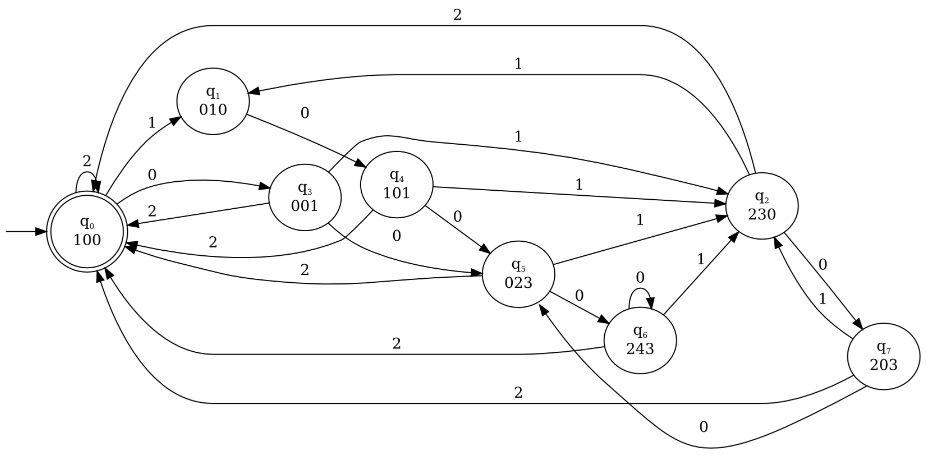
In this section we show how this process can be described using a finite-state automaton, showing that the set of all polyominoes on a twisted cylinder is equivalent to a regular language and giving rise to an efficient transfer-matrix algorithm for enumerating words in this language, and for computing the generating function for its enumerating sequence.

In particular, this shows that for every width w, the generating function for the number of polyominoes on the twisted cylinder of width w can be computed in an algorithmic manner, is a rational function, and the number of such polyominoes satisfies a linear recurrence relation with constant coefficients. However, for large values of w, both the generating function and the recurrence formula become very large (the growth of their complexity is exponential in w since it corresponds to the wth Motzkin number).

In [4], a *transfer matrix* is used in order to encode the growth of polyominoes on the twisted cylinder of width w.[2] The first cell is always "occupied." Then, the next cell is made either "occupied" or "empty," and the process continues ad infinitum. The *signature* of a polyomino P characterizes to which connected components the last w cells of P (including both occupied and empty cells) belong. Naturally, polyominoes with more than one connected component are invalid, but they can become valid later if enough cells are added so as to make the entire polyomino connected. The addition of an empty (resp., occupied) cell to a polyomino P (possibly disconnected) of size n and with a signature s, produces a polyomino of size n (resp., $n + 1$) with signature s' (resp., s''). The key idea is that one does not need to keep in memory all possible polyominoes, but only all possible signatures. The identity of s, and whether the new cell is empty or occupied, determines unambiguously the size and signature of the new polyomino. This is precisely the information encoded in the transfer matrix.

Putting it differently, the transfer matrix encodes a finite automaton defined as follows. The states of the automaton are all possible signatures. The initial state corresponds to the signature that encodes w empty cells. Each state has up to w outgoing edges, labeled '0' through '$w - 1$'. Being at state s, and upon reading the input character k (for $0 \leqslant k < w$), the automaton switches to the new state s' that corresponds to concatenating to a polyomino with signature s an occupied cell and then k more empty cells. (Naturally, concatenating w empty cells to any polyomino P will "terminate" it since then any further occupied cells will be disconnected from P.) This guarantees that a sequence of n input characters will correspond to a polyomino of size precisely n. The only accepting state of the automaton, s_a, corresponds to the signature that encodes an occupied cell followed by $w - 1$ empty cells. (Any valid polyomino can be "normalized" by concatenating to it empty cells as needed.) The connectedness of polyominoes with the signature that corresponds to s_a is guaranteed by the edges of the automaton. Any edge, that manifests the split of the polyomino into components that can never be connected again, leads to a "dead-end" state. (Such edges are omitted.) Our goal is, thus, to count all words over the alphabet $\{0,1,\ldots,w - 1\}$ that are recognized by the automaton, that is, correspond to valid (connected) polyominoes. Figure 5.2(a) shows the automaton that corresponds to the twisted cylinder of width 3. Note that states are named in the notation of [4, 19]. For simplicity of exposition, the initial state was "united" with the accepting state '100'. Actually, the initial state should have been named '000' (the empty polyomino) and have the same outgoing edges as state '100', and no incoming edges. Also, a "dead end" state should be present as well, but this modification does not change the outcome.

[2]Jensen [19] invented this method in order to *count* polyominoes on regular (*non*-twisted cylinders), and his implementation did not actually use the matrix per se. He used a system of "transfer rules" and did not look for any enumerating formula.



(a) Automaton

$$\begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}$$

(b) Transfer matrix

Figure 5.2: An automaton implementing the building of polyominoes on the twisted cylinder of width 3, and the corresponding transfer matrix

We now follow closely the method described by Stanley [39, §4.7]. The transfer matrix A that describes this automaton is of size $k \times k$, where k is the number of different signatures. In fact, k is equal to the $(w+1)$st Motzkin number [4]. The entry A_{ij} contains the number of edges leading from state s_i to state s_j. (In our case, entries can only be 0 or 1.) Denote by f_i the generating function for the number of words accepted by the automaton if s_i is the initial state, and by $\overline{f}$ the vector of all these generating functions. Since the number of accepting paths of length n starting at s_i is identical to the number of accepting paths of length $n-1$ starting at states reachable from s_i by one step, we have the relation $f_i = \sum_j A_{ij} x f_j + \delta_i$, where $\delta_i = 1$ if $s_i = s_a$ and 0 otherwise. Therefore, we have $\overline{f} = xA\overline{f} + \overline{v}$ ($xA = Ax$ since this is a commutative ring), where $\overline{v}$ is a binary vector containing 1 in the entries corresponding to accepting states and 0 elsewhere. By solving for $\overline{f}$, we obtain that $\overline{f} = (I - xA)^{-1}\overline{v}$. Our case is simple in the sense that we have only one accepting state and that the initial and accepting states are the same. By using Cramer's rule, one can easily see that the generating function of s_a is given by $\det(B)/\det(M)$, where $M = I - xA$ and B is obtained from M by erasing the row and column corresponding to s_a (in general one has to erase the row corresponding to the initial state and column corresponding to the accepting state).

Figure 5.2(b) shows the transfer matrix that corresponds to the growth of polyominoes on the twisted cylinder of width 3. The method described above yields the generating function $(x^3 + x^2 + x - 1)/(2x^3 + x^2 + 2x - 1)$, which implies immediately [39, p. 202, Th. 4.1.1] that the recurrence formula in this case is $a_n = 2a_{n-1} + a_{n-2} + 2a_{n-3}$.[3]

The same analysis can be applied algorithmically to any w (and indeed we have written a short program that does exactly this) but for values of $w \geqslant 7$ the resulting generating function is too large to be considered practical. We present here our results for twisted cylinders of widths between 4 and 6 (The meaning of a_n should be understood from the context; we omit from the notation the dependence on the width of the cylinder.)

For width 4, the obtained generating function is

$$\frac{x^7 - 4x^6 + 2x^5 - 3x^3 + 5x^2 - 4x + 1}{2x^7 - 6x^6 + 2x^5 + 2x^4 - 7x^3 + 8x^2 - 5x + 1},$$

and the minimal linear recurrence is

$$a_n = 5a_{n-1} - 8a_{n-2} + 7a_{n-3} - 2a_{n-4} - 2a_{n-5} + 6a_{n-6} - 2a_{n-7},$$

with $a_1 = 1$, $a_2 = 2$, $a_3 = 6$, $a_4 = 19$, $a_5 = 59$, $a_6 = 181$, and $a_7 = 555$. The growth rate (solution of the characteristic equation) is $3.0609\ldots$ (λ_4 in [4]).

For width 5, the generating function is found to be

[3]The cited theorem simply says that the recurrence formula is manifested in the denominator of the generating function. Actually, the recurrence can be obtained directly from the transfer matrix (once its minimal polynomial is known), but we find it beneficial to also have the generating function at hand.

$$\frac{\begin{array}{c}(x^{18}+x^{17}+7x^{16}-3x^{15}-17x^{14}-13x^{13}+13x^{12}+8x^{11}\\ -10x^{10}-12x^9-10x^8-16x^7+3x^6-2x^4+5x^3-x^2-3x+1)\end{array}}{\begin{array}{c}(2x^{18}+14x^{16}-x^{15}-23x^{14}-22x^{13}+16x^{12}+14x^{11}\\ -10x^{10}-18x^9-9x^8-25x^7-x^6-5x^4+6x^3+x^2-4x+1)\end{array}}.$$

The minimal linear recurrence is

$$\begin{aligned} a_n &= 4a_{n-1}-a_{n-2}-6a_{n-3}+5a_{n-4}+a_{n-6}+25a_{n-7}+9a_{n-8}+18a_{n-9}+10a_{n-10} \\ &\quad -14a_{n-11}-16a_{n-12}+22a_{n-13}+23a_{n-14}+a_{n-15}-14a_{n-16}-2a_{n-18}, \end{aligned}$$

with $a_1 = 1$, $a_2 = 2$, $a_3 = 6$, $a_4 = 19$, $a_5 = 63$, $a_6 = 211$, $a_7 = 707$, $a_8 = 2360$, $a_9 = 7853$, $a_{10} = 26070$, $a_{11} = 86434$, $a_{12} = 286416$, $a_{13} = 948991$, $a_{14} = 3144464$, $a_{15} = 10419886$, $a_{16} = 34530671$, $a_{17} = 114435963$, and $a_{18} = 379251561$. The growth rate is $3.3141\ldots$ (λ_5 in [4]).

For width 6, the computed generating function is

$$\frac{\begin{array}{c}(x^{48}-6x^{47}-2x^{46}-3x^{45}-19x^{44}-41x^{43}+117x^{42}-109x^{41}-42x^{40}+48x^{39}\\ -450x^{38}-182x^{37}-73x^{36}-40x^{35}+819x^{34}+204x^{33}+1200x^{32}+439x^{31}-332x^{30}\\ +1477x^{29}-973x^{28}+1727x^{27}-1550x^{26}+1747x^{25}-847x^{24}+102x^{23}+305x^{22}-913x^{21}\\ +983x^{20}-1534x^{19}+1398x^{18}-1336x^{17}+937x^{16}-503x^{15}+354x^{14}-302x^{13}+148x^{12}\\ -9x^{11}+31x^{10}-90x^9+146x^8-150x^7+79x^6-22x^5+8x^4-17x^3+17x^2-7x+1)\end{array}}{\begin{array}{c}(2x^{48}-9x^{47}-7x^{46}+x^{45}-34x^{44}-103x^{43}+201x^{42}-161x^{41}-147x^{40}+245x^{39}\\ -731x^{38}-299x^{37}-120x^{36}-330x^{35}+1101x^{34}+183x^{33}+1622x^{32}+1200x^{31}-1016x^{30}\\ +2893x^{29}-1924x^{28}+2952x^{27}-2472x^{26}+2537x^{25}-793x^{24}-367x^{23}+1109x^{22}-1941x^{21}\\ +2102x^{20}-2815x^{19}+2416x^{18}-2181x^{17}+1411x^{16}-787x^{15}+550x^{14}-435x^{13}+208x^{12}\\ -46x^{11}+107x^{10}-204x^9+260x^8-217x^7+100x^6-32x^5+21x^4-30x^3+23x^2-8x+1)\end{array}}.$$

The minimal linear recurrence is

$$\begin{aligned} a_n &= 8a_{n-1}-23a_{n-2}+30a_{n-3}-21a_{n-4}+32a_{n-5}-100a_{n-6}+217a_{n-7} \\ &\quad -260a_{n-8}+204a_{n-9}-107a_{n-10}+46a_{n-11}-208a_{n-12}+435a_{n-13} \\ &\quad -550a_{n-14}+787a_{n-15}-1411a_{n-16}+2181a_{n-17}-2416a_{n-18}+2815a_{n-19} \\ &\quad -2102a_{n-20}+1941a_{n-21}-1109a_{n-22}+367a_{n-23}+793a_{n-24}-2537a_{n-25} \\ &\quad +2472a_{n-26}-2952a_{n-27}+1924a_{n-28}-2893a_{n-29}+1016a_{n-30} \\ &\quad -1200a_{n-31}-1622a_{n-32}-183a_{n-33}-1101a_{n-34}+330a_{n-35}+120a_{n-36} \\ &\quad +299a_{n-37}+731a_{n-38}-245a_{n-39}+147a_{n-40}+161a_{n-41}-201a_{n-42} \\ &\quad +103a_{n-43}+34a_{n-44}-a_{n-45}+7a_{n-46}+9a_{n-47}-2a_{n-48}, \end{aligned}$$

with $(a_i)_{i=1}^{48}$ = (1, 2, 6, 19, 63, 216, 754, 2650, 9311, 32648, 114210, 398711, 1389910, 4840947, 16852765, 58657588, 204151644, 710538469, 2473082513, 8608079201, 29963112912, 104298075975, 363053581718, 1263768927868, 4399117783874, 15313119079728, 53304212536356, 185549212643365, 645886848026350, 2248296466237295, 7826193746149272, 27242536831652344, 94829719987493235, 330096849979162682, 1149048320483048893, 3999771751380580614, 13922977784041234403, 48465093550790494424, 168704234381507799362, 587249847637486241853, 2044183333472295697123, 7115685974133683488273, 24769298361096626327975, 86220519493365324684086, 300128726780117770956748, 1044731035631558645331234, 3636649341693369874678279, 12658969612510481574745708). The growth rate is 3.4809... (λ_6 in [4]).

Note that the complexities of the recurrence formulae for polyominoes on twisted cylinders of widths 3–6, that is, 4,8,19,49, are almost identical to the dimensions of the respective transfer matrices, that is, the 4th through 7th Motzkin numbers, namely, 4,9,21,51.

5.3 Polyominoes on Twisted Cylinder of Width 3

In Section 5 we presented polyominoes on twisted cylinders and proved that for cylinder width $W = 3$ the number of polyominoes satisfies the recurrence relation $a_n = 2a_{n-1} + a_{n-2} + 2a_{n-3}$. We now show how this result can also be obtained using our bijection to permutations and characterization of classes of polyominoes via forbidden patterns introduced in Chapter 4.

Denote by $\mathcal{F}$ the set of (2431, 3412, 3421, 4123, 4213, 4231, 4312, 4321)-avoiding permutations, and by $\mathcal{F}_n$ the set of such permutations of $[n]$. Let us first establish the desired bijection.

Theorem 24 *For every natural n, there is a bijection between $\mathcal{A}_{3,n}$ and $\mathcal{F}_n$.*

Proof: The proof is organized as follows. First, we define recursively a set $\mathcal{B}$ of permutations, and show a recursion-preserving bijection between $\mathcal{A}_3$ and $\mathcal{B}$. Second, we prove that $\mathcal{B}$ and $\mathcal{F}$ are identical.

Step 1. For every $n \in \mathbb{N}$, we define the set of permutations $\mathcal{B}_n$ as follows. For $n = 1, 2, 3$, $\mathcal{B}_n$ is the set of all $n!$ permutations of $[n]$. For $n \geqslant 4$, $\mathcal{B}_n$ is defined recursively as follows (refer to Figure 5.3).

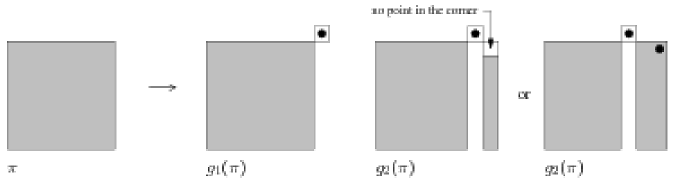
- For each $\pi \in \mathcal{B}_{n-1}$, let $g_1(\pi)$ be the permutation τ of $[n]$ defined by

  For $1 \leqslant i \leqslant n-1$, $\tau(i) = \pi(i)$; $\tau(n) = n$.

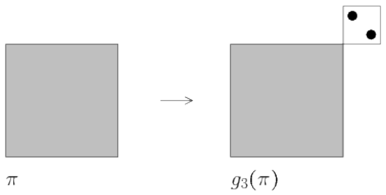
- For each $\pi \in \mathcal{B}_{n-1}$, let $g_2(\pi)$ be the permutation τ of $[n]$ defined according to whether or not $\pi(n-1) = n-1$:

  If $\pi(n-1) \neq n-1$, then
  For $1 \leqslant i \leqslant n-2$, $\tau(i) = \pi(i)$; $\tau(n-1) = n$; $\tau(n) = \pi(n-1)$;
  Else (if $\pi(n-1) = n-1$)
  For $1 \leqslant i \leqslant n-3$, $\tau(i) = \pi(i)$; $\tau(n-2) = n$; $\tau(n-1) = \pi(n-2)$; $\tau(n) = \pi(n-1) = n-1$.

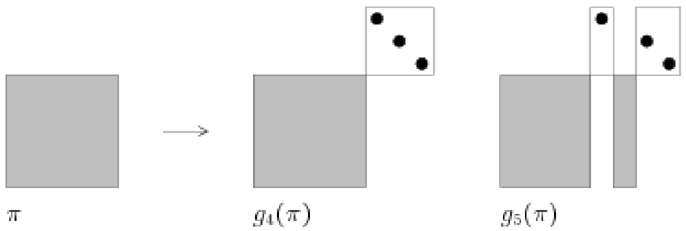
- For each $\pi \in \mathcal{B}_{n-2}$, let $g_3(\pi)$ be the permutation τ of $[n]$ defined by



(a) One-level recursion: $g_1(\pi)$ and $g_2(\pi)$



(b) Two-level recursion: $g_3(\pi)$



(c) Three-level recursion: $g_4(\pi)$ and $g_5(\pi)$

Figure 5.3: Recursive definition of $\mathcal{B}_n$

For $1 \leqslant i \leqslant n-2$, $\tau(i) = \pi(i)$; $\tau(n-1) = n$; $\tau(n) = n-1$.

- For each $\pi \in \mathcal{B}_{n-3}$, let $g_4(\pi)$ be the permutation τ of $[n]$ defined by

  For $1 \leqslant i \leqslant n-3$, $\tau(i) = \pi(i)$; $\tau(n-2) = n$; $\tau(n-1) = n-1$; $\tau(n) = n-2$.

- For each $\pi \in \mathcal{B}_{n-3}$, let $g_5(\pi)$ be the permutation τ of $[n]$ defined by

  For $1 \leqslant i \leqslant n-4$, $\tau(i) = \pi(i)$; $\tau(n-3) = n$; $\tau(n-2) = \pi(n-3)$; $\tau(n-1) = n-1$; $\tau(n) = n-2$.

Finally, we set $\mathcal{B}_n = \{g_1(\pi), g_2(\pi) : \pi \in \mathcal{B}_{n-1}\} \cup \{g_3(\pi) : \pi \in \mathcal{B}_{n-2}\} \cup \{g_4(\pi), g_5(\pi) : \pi \in \mathcal{B}_{n-3}\}$, and $\mathcal{B} = \bigcup_{n \in \mathbb{N}} \mathcal{B}_n$. (Note that these classes are disjoint since each class is uniquely characterized by the configuration formed by points in the three upper rows.)

Step 2. We prove that for every natural n, the set $\mathcal{B}_n$ coincides with $\mathcal{F}_n$, the set of $(2431, 3412, 3421, 4123, 4213, 4231, 4312, 4321)$-avoiding permutations of $[n]$.

We begin by proving by induction on n that a permutation $\pi \in \mathcal{B}_n$ avoids all eight indicated permutations. For $n = 1, 2, 3$, the claim is clear. Assume, then, that $n \geqslant 4$. We need to verify that if π avoids all the eight permutations, then $g_1(\pi), \ldots, g_5(\pi)$ avoid them all as well. Thus, we have a total of forty statements to check. The proofs of all these statements are similar, being based on the observation that if the graph of τ is obtained from the graph of π by adding several elements (shown as points in Figure 5.3), and τ has a pattern that is avoided in π, then this pattern must involve at least one added element.[4] For lack of space we prove in detail only one such statement, that if π avoids the pattern 2431, then so does $g_5(\pi)$. All the other proofs are identical in nature.

Refer again to the right-hand side of Figure 5.3(c), and assume for contradiction that $\tau = g_5(\pi)$ contains the pattern 2431. That is, there are indices $1 \leqslant i < j < k < \ell \leqslant n$ such that $\tau(\ell) < \tau(i) < \tau(k) < \tau(j)$. In such a case we must have $j = n-3$. This is justified as follows. First, $j \in \{n-3, n-1, n\}$, otherwise we have $\tau(i), \tau(j), \tau(k), \tau(\ell) \leqslant n-3$, and, therefore, π also has the pattern 2431, which is a contradiction. Second, if $j = n-1$ or $j = n$, then the indices k and ℓ cannot exist as we assumed. Now, since $j = n-3$ and $k > \ell$, we must have $k = n-2$ or $k = n-1$. ($k \neq n$ since otherwise the index ℓ cannot exist.) However, both options lead to a contradiction. If $k = n-2$, then $\ell \in \{n-1, n\}$, which is impossible since both cases imply that $\tau(k) < \tau(\ell)$ (due to the facts that $\tau(k) = \tau(n-2) \leqslant n-3$ and $\tau(\ell) \in \{n-1, n-2\}$). Finally, if $k = n-1$, then we must have $\ell = n$, which is also impossible since this implies that $\tau(\ell) > \tau(i)$ (due to the facts that $\tau(\ell) = \tau(n) = n-2$ and $\tau(i) \leqslant n-3$).

We continue with proving by induction on n that a permutation $\tau \in \mathcal{F}_n$ is also in $\mathcal{B}_n$. For $n = 1, 2, 3$, the claim is clear. Assume, then, that $n \geqslant 4$. Our goal is to show that τ is either $g_1(\pi)$, $g_2(\pi)$, $g_3(\pi)$, $g_4(\pi)$, or $g_5(\pi)$, for some permutation π of either $[n-1]$, $[n-2]$, or $[n-3]$. In all cases, π is a subpermutation of τ, hence, it avoids the eight forbidden

[4]The graph of a permutation $\tau \in S_n$ is the set of points $\{(i, \tau(i)) : 1 \leqslant i \leqslant n\}$.

patterns. Therefore, by the induction hypothesis, $\pi \in \mathcal{B}_{n-1} \cup \mathcal{B}_{n-2} \cup \mathcal{B}_{n-3}$, which would imply that $\tau \in \mathcal{B}_n$.

To this end we analyze all possible suffixes of the permutation τ.

- $\tau(n) = n$. π is the permutation of $[n-1]$ defined by $\pi(i) = \tau(i)$ for all $1 \leqslant i \leqslant n-1$. In this case $\tau = g_1(\pi)$.
- $\tau(n-1) = n$ and $\tau(n) = n-1$. π is the permutation of $[n-2]$ defined by $\pi(i) = \tau(i)$ for all $1 \leqslant i \leqslant n-2$. In this case $\tau = g_3(\pi)$.
- $\tau(n-1) = n$ and $\tau(n) < n-1$. π is the permutation of $[n-1]$ defined by $\pi(i) = \tau(i)$ for all $1 \leqslant i \leqslant n-2$ and $\pi(n-1) = \tau(n)$. In the case $\tau = g_2(\pi)$.
- $\tau(n-2) = n$, $\tau(n-1) = n-1$, and $\tau(n) = n-2$. π is the permutation of $[n-3]$ defined by $\pi(i) = \tau(i)$ for all $1 \leqslant i \leqslant n-3$. In the case $\tau = g_4(\pi)$.
- $\tau(n-2) = n$ and $\tau(n) = n-1$. π is the permutation of $[n-1]$ defined by $\pi(i) = \tau(i)$ for all $1 \leqslant i \leqslant n-3$, $\pi(n-2) = \tau(n-1)$, and $\pi(n-1) = \tau(n)$. In the case $\tau = g_2(\pi)$.
- $\tau(n-2) = n$, $\tau(n-1) = n-1$, and $\tau(n) < n-2$. This combination is impossible since in this case τ contains the forbidden pattern 2431 (with $i = \tau^{-1}(n-2)$, $j = n-2$, $k = n-1$, and $\ell = n$).
- $\tau(n-2) = n$ and $\tau(i) = n-1$ (for $i < n-3$). This combination is also impossible. We either have $\tau(n-1) < \tau(n)$ or $\tau(n-1) > \tau(n)$. In the former case, τ contains the forbidden pattern 3412, while in the latter case, τ contains the forbidden pattern 3421 (both with $i = \tau^{-1}(n-1)$, $j = n-2$, $k = n-1$, and $\ell = n$).
- $\tau(n-3) = n$, $\tau(n-1) = n-1$, and $\tau(n) = n-2$. π is the permutation of $[n-3]$ defined by $\pi(i) = \tau(i)$ for all $1 \leqslant i \leqslant n-4$ and $\pi(n-3) = \tau(n-2)$. In the case $\tau = g_5(\pi)$.
- All other cases in which $\tau(n-3) = n$ are impossible. In order to see this, notice that all the patterns of the form $4{*}{*}{*}$, except 4132, are forbidden. Therefore, in this case we have $\tau(n-2) < \tau(n) < \tau(n-1)$. This implies that $\tau(n-1) = n-1$, otherwise τ contains the forbidden pattern 3421 (with $i = \tau^{-1}(n-1)$, $j = n-3$, $k = n-1$, and $\ell = n$). Furthermore, we necessarily have $\tau(n) = n-2$, otherwise τ contains the forbidden pattern 2431 (with $i = \tau^{-1}(n-2)$, $j = n-3$, $k = n-1$, $\ell = n$). Hence, we arrive at the previous case.
- $\tau(k) = n$ for $k < n-3$. This case is also impossible. Otherwise, applying the reasoning as in the previous paragraph twice, first for $n-1$ and n, and the for $n-2$ and $n-1$, one concludes that $\tau(n-1) = n-1$ and $\tau(n) = n-2$ simultaneously with $\tau(n-2) = n-1$ and $\tau(n-1) = n-2$. This is clearly impossible.

By combining Steps 1 and 2, we essentially establish a recursion-preserving bijection φ_n between $\mathcal{A}_{3,n}$ and $\mathcal{F}_n$. Specifically, for $n = 1, 2, 3$, φ_n is defined as is shown in Figure 5.4. For $n \geqslant 4$, a polyomino is transformed into a permutation as follows. Let $Q \in \mathcal{A}_{3,n}$ with

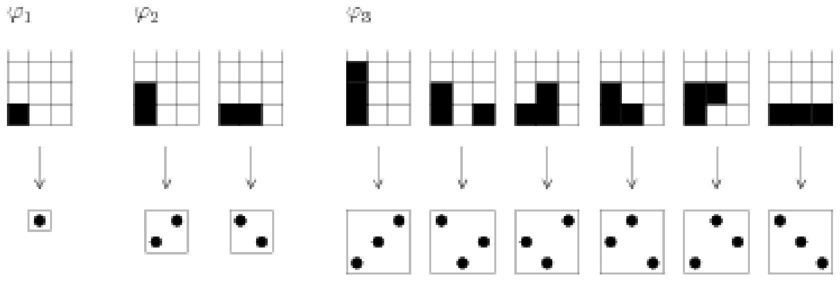


Figure 5.4: Initializing the recursion

$n \geqslant 4$. Let P be the polyomino of size m (for $n-3 \leqslant m \leqslant n-1$) for which $Q = f_i(P)$ (for some $1 \leqslant i \leqslant 5$). Let $\pi = \varphi_m(P)$. Then, $\varphi_n(Q)$ is defined as $g_i(\pi)$. □

Thus, the union $\varphi = \cup_{n \in \mathbb{N}}(\varphi_n)$ is a recursion- and size-preserving bijection between $\mathcal{A}_3$, the set of polyominoes on the twisted cylinder of width 3, and $\mathcal{F}$, the set of $(2431, 3412, 3421, 4123, 4213, 4231, 4312, 4321)$-avoiding permutations. In fact, we proved two more bijections between $\mathcal{A}_3$ and two other sets of permutations, $\mathcal{F}'$ and $\mathcal{F}''$, avoiding the patterns $(2413, 3412, 3421, 4123, 4132, 4213, 4312, 4321)$ and $(1342, 1432, 3142, 3412, 4123, 4132, 4213, 4312)$, respectively. The details of the two additional bijections are very similar to those of the bijection shown above, and are, thus, omitted. Note that the conjunction of the three bijections is an indirect proof that the three sets of permutations, $\mathcal{F}$, $\mathcal{F}'$, and $\mathcal{F}''$, are in the same Wilf class, that is, for any fixed n, there is the same number of permutations of $[n]$ avoiding either set of patterns.

The Maple package VATTER [43] (based on Zeilberger's original package WILF [42]) readily produces an efficient (polynomial-time) enumeration scheme of the polyominoes in $\mathcal{F}$. The command

$$seq := SeqS(SchemeFast(\{[2,4,3,1],[3,4,1,2],[3,4,2,1],[4,1,2,3],\\ [4,2,1,3],[4,2,3,1],[4,3,1,2],[4,3,2,1]\},3,1),20);$$

computes instantly the first 20 values of the sequence. It takes less than a minute to compute the first 500 values of this sequence. Then, feeding just a few consecutive values either to the same Maple package (by typing "`Findrec(seq,n,N,4);`") or to another mathematics software (e.g., by using the Mathematica command "`FindLinearRecurrence[{1,2,6,16,42,112,298,792,2106,5600}]`") yields the expected recurrence formula, $a_n = 2a_{n-1} + a_{n-2} + 2a_{n-3}$.

Chapter 6

Conclusions and Open Problems

In this research we have studied several different approaches, both computational and theoretical, for the enumeration problem of lattice animals. We review them briefly and suggest further directions and improvements that can be pursued in future work.

Enumeration algorithms. We have used our generalized, parallel version of Redelmeier's algorithm to count various types of lattice animals, but by no means all possible types. A natural extension of our work will be the application of the algorithm to additional types of lattice animals. The algorithm itself can be further optimized to yield even better results in some cases; some work in this direction was done in a yet unpublished paper [28] which improves our implementation and achieves better results for d-dimensional orthogonal polycubes.

Growth constants. The upper and lower bounds for λ_2 remain the major open problem in the field. An extremely promising approach for improving the lower bound is via the twisted-cylinder polyominoes described in Chapter 5. However, obtaining such bounds using the current methods requires dealing with extremely high numbers and large matrices. Although this is possible in practice for a few more values, the obtained bound might still be far from the true value of λ_2.

Another possible approach not dealt with in this research is to use a method similar to that in Section 3.2 and count "proper" polyominoes on twisted cylinder—polyominoes that can be embedded on cylinder of width w but cannot be embedded in any cylinder of a smaller width. One can derive formulas for the first few diagonals of tables of proper cylinder polyominoes as well, and this might lead to a better understanding of the limiting behavior of the growth constant for twisted cylinder, and so to a better lower bound on λ_2 itself.

The upper bound remained unchanged since 1973. Using modern computers, the method of [23] might be pushed a little further, but we believe that a whole new approach is needed to obtain a significant improvement. Such an approach is yet to be discovered, but perhaps the novel injection from polyominoes into permutations introduced in Chapter 4 will prove fruitful in this regard.

Another question is whether or not, and if yet, to which extent, the upper-bound and lower-bound methods can be generalized to other types of lattice animals. The upper bound method of [23] was adapted in [41] to hexagonal lattice animals, but we are not aware of anything else. Because it depends very much on the limited number of "far" neighbors each cell has, this method might be very difficult to adapt (if at all possible) to high-dimensional polycubes.

As for the lower-bound method, it might be useful for hexagonal and triangular lattices (although some technical problems arise when attempting to adapt it), but, again, in high dimensions the method is much harder to use for the same reason the transfer matrix counting algorithm of [19] fails—the transfer matrix is too big since there are too many possible "frontlines".

Proper polycubes and diagonal formulas. We have seen in Chapter 3 that proper polycubes are the key for understanding the behavior of high-dimensional polycubes. A very good estimate of λ_d and λ_d^T was obtained using relatively simple analysis of the diagonals in the table counting proper polycube. As more diagonal formulas are found, the bounds can be further improved. The question yet remains open if a general, simple pattern for all the diagonals exist.

Polyomino-permutations bijection. In Chapter 4 we presented a method for encoding polyominoes by permutations. This is a very natural and fruitful method, and has a very nice geometric representation, but it is by no means the only possible such method. A promising future direction of research is to consider other bijections which preserve some of the structure of the original polyominoes in a way that enables classes of polyominoes to be defined using some characterization of sets of permutations (most notably, using forbidden patterns). Our method might also characterize more classes of polyominoes other than the ones already found.

Bibliography

[1] G. Aleksandrowicz and G. Barequet. Counting d-dimensional polycubes and nonrectangular planar polyominoes. *Int. J. of Computational Geometry and Applications*, 19(3):215–229, 2009.

[2] G. Aleksandrowicz and G. Barequet. Counting polycubes without the dimensionality curse. *Discrete Mathematics*, 309(13):4576–4583, 2009.

[3] E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the mahonian. *Séminaire Lotharingien de Combinatoire*, 44, 2000.

[4] G. Barequet, M. Moffie, A. Ribó, and G. Rote. Counting polyominoes on twisted cylinders. *Integers (electroic journal)*, 6:#A22, 2006.

[5] R. Barequet, G. Barequet, and G. Rote. Formulae and growth rates of high-dimensional polycubes. *Combinatorica*, 30(3):257–275, 2010.

[6] L.W. Beineke and R.E. Pippert. On the enumeration of planar trees of hexagons. *Glasgow Mathematical Journal*, 15:131–147, 1974.

[7] F. Bergeron, G. Labelle, and P. Leroux. Combinatorial species and tree-like structures. *Encyclopedia of Mathematics and its Applications*, 67, 1997.

[8] P. Bose, J.F. Buss, and A. Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65:277–283, 1998.

[9] M. Bousquet-Mélou. A method for the enumeration of various classes of column-convex polygons. *Discrete Mathematics*, 154:1–25, 1996.

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

[11] M. Bousquet-Mélou and J.-M. Fédou. The generating function of convex polyominoes: The resolution of a q-differential system. *Discrete Mathematics*, 137:53–75, 1995.

[12] M. Eden. A two-dimensional growth process. *Proc. 4th Berkeley symp. on mathamtical statistics and probability*, 4:223–239, 1961.

[13] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. Available from: `http://algo.inria.fr/flajolet/Publications/books.html`

[14] D.S. Gaunt. The critical dimension for lattice animals. *J. of Physics A: Mathematical and General*, 13:L97–L101, 1980.

[15] D.S. Gaunt, M.F. Sykes, and H. Ruskin. Percolation processes in d-dimensions. *J. of Physics A: Mathematical and General*, 9:1899–1911, 1976.

[16] S.W. Golomb. *Polyominoes*. Scribners, New York, 1965.

[17] K. Gong. `http://kevingong.com/Polyominoes/Enumeration.html`

[18] F. Harary and R.C. Read. The enumeration of tree-like polyhexes. *Proc. Edinburgh Mathematical Society*, 17:1–13, 1970.

[19] I. Jensen. Enumerations of lattice animals and trees. *J. of Statistical Physics*, 102:865–881, 2001.

[20] . Jensen. Counting polyominoes: A parallel implementation for cluster computing. *Lecture Notes in Computer Science*, 2659:203–212, June 2003.

[21] I. Jensen and A.J. Guttmann. Statistics of lattice animals (polyominoes) and polygons. *J. of Physics A: Mathematical and General*, 33:L257–L263, 2000.

[22] D.A. Klarner. Cell growth problems. *Canadian J. of Mathematics*, 19:851–863, 1967.

[23] D.A. Klarner and R.L. Rivest. A procedure for improving the upper bound for the number of n-ominoes. *Canadian J. of Mathematics*, 25:585–602, 1973.

[24] W.F. Lunnon. Counting polyominoes. *Computers in Number Theory*, 1971.

[25] W.F. Lunnon. Counting hexagonal and triangular polyominoes. *Graph Theory and Computing*, pages 87–100, 1972.

[26] W.F. Lunnon. Symmetry of cubical and general polyominoes. *Graph Theory and Computing*, pages 101–108, 1972.

[27] W.F. Lunnon. Counting multidimensional polyominoes. *The Computer Journal*, 18:366–367, 1975.

[28] S. Luther and S. Mertens. Counting lattice animals in high dimensions. *Arxiv*, 2011. Available from: `http://arxiv.org/PS_cache/arxiv/pdf/1106/1106.1078v1.pdf`

[29] N. Madras. A pattern theorem for lattice clusters. *Annals of Combinatorics*, 3:357–384, 1999.

[30] N. Madras, C.E. Soteros, S.G. Whittington, J.L. Martin, M.F. Sykes, S. Flesia, and D.S. Gaunt. The free energy of a collapsing branched polymer. *J. of Physics A: Mathematical and General*, 23:5327–5350, 1990.

[31] J.L. Martin. *Computer techniques for evaluating lattice constants*, volume 3 of *Phase Transitions and Critical Phenomena*, pages 97–112. Academic Press, London, 1974.

[32] S. Mertens. Lattice animals: A fast enumeration algorithm and new perimeter. *J. of Statistical Physics*, 58:1095–1108, 1990.

[33] S. Mertens and M.E. Lautenbacher. Counting lattice animals: A parallel attack,. *J. of Statistical Physics*, 66:669–678, 1992.

[34] P.J. Peard and D.S. Gaunt. $1/d$-expansions for the free energy of lattice animal models of a self-interacting branched polymer. *J. Physics A: Mathematical and General*, 28:6109–6124, 1995.

[35] G. Polya and G. Szegö. *Problems and Theorems in Analysis.* Springer-Verlag, New York, 1976.

[36] Lara Pudwell. *Enumeration Schemes for Pattern-Avoiding Words and Permutations.* PhD thesis, Rutgers University, 2008.

[37] D.H. Redelmeier. Counting polyominoes: Yet another attack. *Discrete Mathematics*, 36:191–203, 1981.

[38] N.J.A. Sloane. The on-line encyclopedia of integer sequences. http://oeis.org/.

[39] R. Stanely. *Enumerative Combinatorics, vol. I,*. Cambridge University Press, 1997.

[40] M.F. Sykes, D.S. Gaunt, and M. Glen. Percolation processes in three dimensions. *J. of Physics A: Mathematical and General*, 10:1705–1712, 1976.

[41] M. Vöge and A.J. Guttmann. On the number of hexagonal polyominoes. *Theoretical Computer Science*, 307:433–453, 2003.

[42] D. Zeilberger. Enumeration schemes, and more importantly, their automatic generation. *Annals of Combinatorics*, (2):185–195, 1998.

[43] D. Zeilberger. On vince vatter's brilliant extension of doron zeilberger's enumeration schemes for counting herb wilf's classes. http://www.math.rutgers.edu/ zeilberg/mamarim/mamarimhtml/vatter.html, 2006. Available from: `http://www.math.rutgers.edu/{\mytilde}zeilberg/mamarim/mamarimhtml/vatter.html`.

הנתונים המספריים מסייעים לנו בניתוח אנליטי של קצב הגידול של חיות סריג ב-d ממדים עבור ערכים הולכים וגדלים של d. אנו מתמקדים בספירה של חיות סריג נאותות ב-d ממדים - חיות שלא ניתן לשכן ב-$d-1$ ממדים. ניתן להראות כי (עבור סוגים מסויימים של חיות סריג) מספר חיות הסריג הכולל בגודל מסויים ב-d ממדים ניתן לכתיבה כצירוף לינארי של מספר חיות השריג מאותו גודל הנאותות בכל המימדים הקטנים מ-d, כך שבעיית הספירה הכללית ניתנת לצמצום לבעיית הספירה של חיות נאותות. אם מתבוננים בטבלה ששורותיה מתאימות לגדלים של חיות נאותות, ועמודותיה למימדים, הרי שהאלכסון הראשי והאלכסונים שמעליו הם כולם אפסים (כל חיה בגודל d תמיד ניתנת לשיכון ב-$d-1$ מימדים), ולכן העניין הוא באלכסונים שמתחת לאלכסון הראשי. בהתבסס על עבודה קודמת שטיפלה בחיות d-ממדיות על סריג ריבועי רגיל, אנו מוכיחים נוסחה עבור שני האלכסונים הראשונים בטבלה של חיות d-ממדיות שהגרף המייצג אותן הוא עץ; בעזרת הנוסחאות לשני האלכסונים הללו אנו מסוגלים לתת הערכה טובה לגודלו של λ_d^T - קבוע הגידול המתאים לחיות אלו ב-d ממדים - כאשר d שואף לאינסוף.

שיטה שונה בה אנו נוקטים היא תיאור של polyominoes (דו ממדיים) באמצעות פרמוטציות. אנו מציגים העתקה חד-חד ערכית מ-polyomino בגודל n לפרמוטציות בגודל n, באמצעות מספור הריבועים המרכיבים את ה-polyomino בשתי דרכים שונות ובניית הפרמוטציה באמצעותן, (כך למשל, אם ריבוע מסויים סומן ב-2 בשיטה האחת וב-5 בשיטה השניה, אז הפרמוטציה שתתקבל ממנו תעביר את 2 ל-5.) בשיטה זו ניתן לאפיין מחלקות מסויימות של polyominoes באמצעות תבניות אסורות. שיטת אפיון זו לפרמוטציות נפוצה למדי בשנים האחרונות, ומבוצע מחקר רב אשר מטרתו למצוא אלגוריתמי ספירה ופונקציות יוצרות למחלקות של פרמוטציות המאופיינות על ידי תבניות אסורות. אנו מדגימים כיצד ניתן לתאר במדויק באמצעות תבניות אסורות שתי מחלקות של polyominoes - polyominoes קמורים, ו-polyominoes עצים קמורים.

שיטה נוספת בה אנו נוקטים היא חקירת מחלקות של polyominoes שניתן לספור באמצעות אלגוריתם Transfer-Matrix, ולכן, באופן שקול, לזהות באמצעות אוטומט סופי דטרמיניסטי. מחלקות שניתנות לאפיון בצורה זו הן פשוטות במיוחד לספירה, וניתן למצוא את הפונקציה היוצרת שלהן בצורה אלגוריתמית פשוטה. כמו כן, ניתן להוכיח כי קיימת עבורן נוסחת נסיגה. אנו מתמקדים ב-polyominoes המהווים "קירוב מלמטה" ל-polyominoes דו ממדיים - polyominoes על "גליל ספירלי" - מלבן בו כל משבצת בקצה הימני של שורה מחוברת למשבצת השמאלית ביותר בשורה שמעליה. על גליל מסוג זה אלגוריתם ה-Transfer-Matrix פשוט במיוחד למימוש, והסי-בוכיות שלו תלויה רק ברוחב הגליל. גלילים מרוחב גדול יותר מניבים חסמים תחתונים טובים יותר על λ עצמו.

תקציר

"חיות סריג" הן אובייקטים קומבינטוריים שניתנים להגדרה כתת-גרפים קשירים של הגרף הדואלי של הסריג הנתון. הדוגמה הנפוצה ביותר של חיית סריג היא polyomino, הניתן לתיאור כאוסף קשיר של ריבועי יחידה במישור, המחוברים לאורך צלעותיהם. הצורות המופיעות במשחק "טטר-יס" הן כולן polyominoes מגודל 4 (כלומר, הן מכילות 4 ריבועים).

חיות סריג צצות הן בהקשרים של שעשועים מתמטיים (למשל, משחקי ריצוף דוגמת טטריס) והן בתחומים מדעיים שונים דוגמת פיזיקה סטטיסטית (בה הם ממדלים תופעות מסויימות, למשל "קיפול" ו"פתיחה" של פולימרים) או גאומטריה קומבינטורית.

בעבודה זו אנו עוסקים בשאלת הספירה של חיות סריג - לחשב, או להעריך, כמה חיות סריג מגודל מסויים קיימות עבור סריג נתון. זוהי בעיה קומבינטורית קשה, ששורשיה נמצאים כבר בשנות ה-60 של המאה ה-20. אפילו עבור מקרה פשוט יחסית, הסריג הריבועי במישור, לא ידועה נוסחה סגורה למספר החיות מגודל נתון. עבור מספר זה, המסומן כ-$A_2(n)$, הוכיח Klarner בשנת 1968 כי מתקיים $\lim_{n\to\infty}\sqrt[n]{A_2(n)}=\lambda$, כך שמספר ה-polyominoes מגודל n בשני מימדים הוא בערך אקספוננציאלי ב-n עם בסיס λ. ערכו המדויק של הקבוע λ, המכונה "קבוע Klarner" אינו ידוע; כל שידועים כיום הם החסמים $3.9801\ldots \leqslant \lambda \leqslant 4.6496\ldots$. מציאת חסמים טובים יותר היא אחת מהבעיות הפתוחות המרכזיות של התחום.

אנו נוקטים במספר גישות לבעיית הספירה. ראשית, אנו מציגים ומממשים אלגוריתם המסוגל לספור באופן ישיר את חיות הסריג מגודל נתון עבור כל סריג אפשרי. אלגוריתם זה הוא הכללה של אלגוריתם Redelmeier לספירת polyominoes, וכוחו נעוץ בכך שהוא מייצר כל חיית סריג פעם אחת בדיוק, בעוד שאלגוריתמי הספירה הנאיביים יוצרים את אותה חיה מספר פעמים ובכך מבזבזים זמן. אלגוריתם זה אינו היעיל ביותר הידוע עבור polyominoes (האלגוריתם של Jensen הרבה יותר מהיר), אך עבור מחלקות רבות של חיות סריג הוא מניב את התוצאות הטובות ביותר הידועות כיום, ואשר נתגלו לראשונה באמצעות המימוש שלנו.

ניתן לחשוב על האלגוריתם כעל אלגוריתם לספירת תתי-גרפים קשירים של גרף קשיר לא מכוון, הנקבע על ידי הסריג (לעתים תחת מגבלות נוספות, גאומטריות באופיין, דוגמת זה שהחיה המתקבלת תהיה קמורה, או שחיה d-ממדית לא תהיה ניתנת לשיכון במרחב עם פחות מ-d ממדים, וכדומה). האלגוריתם בונה את החיות בהדרגה, על ידי הוספת צומת אחת בכל צעד וה-משך רקורסיבי של המניה; לב האלגוריתם טמון במניעת ספירות כפולות המושגת על ידי הוספה "חכמה" של צמתים לרשימת ה"מועמדים הפוטנציאליים" להוספה לחיה בהמשך האלגוריתם. בכל הוספה של צומת חדש לגרף, האלגוריתם מוסיף לרשימת המועמדים גם את שכניו של אותו צומת שעדיין אינם שייכים לחיה שנבנתה וגם אינם שכנים של צומת קיים בחיה. תנאי זה, לפיו כל השכנים שיתווספו לרשימת המועמדים הם חדשים, הוא שמונע ספירות כפולות.

המחקר נעשה בהנחיית פרופ' גיל ברקת בפקולטה למדעי המחשב.

ראשית כל, ברצוני להודות למנחה שלי, פרופ' גיל ברקת, שליווה אותי עוד מאמצע התואר הראשון ועד לסיום הדוקטורט. גיל תמיד הפתיע אותי ביכולת שלו לאתר סדר ותבניות במקומות בהם ראיתי רק כאוס, ובגישות היצירתיות שלו להתמודדות עם בעיות, שהיוו לי השראה לכל אורך הלימודים. הוא גם היה מדריך מעולה בנבכי העולם האקדמי הפתלתל.

ברצוני להודות למרצים נוספים רבים מספור אשר עוררו בי את ההתלהבות והאהבה למתמטיקה ומדעי המחשב. במיוחד ארצה לציין את פרופ' איל קושלביץ שלימד אותי מספר קורסים בתורת הסיבוכיות; את פרופ' רון אהרוני שלימד אותי לוגיקה מתמטית; ואת פרופ' אלי ביהם שלימד אותי קריפטוגרפיה. מרצים רבים נוספים לימדו אותי בצורה מעוררת השראה ומלאה התלהבות ודלתם הייתה תמיד פתוחה לשאלות שלי. לבסוף ברצוני להודות לפרופ' דוד הראל, אשר אף כי מעולם לא למדתי אצלו, ספרו "אלגוריתמיקה" הוא שחשף עבורי לראשונה את יופיים של מדעי המחשב ובמידה רבה בזכותו פניתי ללימודיהם.

אני רוצה להודות למשפחתי שתמכה בי תמיד, ובפרט בלימודים, ושעוררה אצלי כבר בגיל צעיר את אהבת הידע וחדוות הלמידה.

לבסוף, אני רוצה להודות לבת זוגי, עדי וולף, שליוותה אותי מתחילתו של התואר הראשון ובזכותה תקופה זו הייתה המהנה ביותר בחיי עד כה.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי. כמו כן ברצוני להודות לקרנות על שם ג'ייקובס ופיין על המלגות הנדיבות שהעניקו לי במהלך הלימודים.

ספירה של חיות סריג

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

גדי אלכסנדרוביץ׳

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
סיוון ה׳תשעא   חיפה   יוני 2011

ספירה של חיות סריג

גדי אלכסנדרוביץ׳