COUNTING CLASSES OF MATRICES AND MORE USING EXPERIMENTAL MATHEMATICS

By

GEORGE SPAHN

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Mathematics

Written under the direction of

Doron Zeilberger

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

March 4, 2024

ABSTRACT OF THE DISSERTATION

Counting Classes of Matrices and More Using Experimental Mathematics

By GEORGE SPAHN

Dissertation Director:

Doron Zeilberger

This thesis gives several examples of how finite state machines can be used to count rectangular objects. We associate a matrix with the object, fix the number of rows, and consider the sequence formed by increasing the number of columns. We then construct transfer matrices to count the number of accept strings for each finite state machine, and solve for a generating function. The process is completely automated using the maple computer algebra system. 4 cases are considered: logic puzzles, baxter matrices, voting districts, and seating arrangements that obey social distancing. At the end we provide code that is easily modifiable to handle any variation that the reader may be interested in.

Contents

Acknowledgements

I would first like to thank my advisor, Dr. Z, for introducing me to the field of experimental mathematics and giving me an endless supply of difficult but doable challenges. It has been a pleasure learning and collaborating with him, and I owe most of my findings to his ideas.

I would secondly like to thank the graduate secretary Katie Guarino for helping immensely with the administrative details of being a grad student.

I would like to thank Neil Sloane for introducing me to the Gerrymander Sequence and many other interesting integer sequences, as well as being on my dissertation committee.

I would like to thank the other members of my committee: Vladimir Retakh and Michael Saks.

Finally I would like to thank my fiance Xu Guo for support, encouragement, and bringing joy to my life.

Chapter 1

Introduction

How many ways can people sit in an auditorium so that no two are adjacent but no more people can be added? How many ways can a rectangular grid of towns be divided into two connected voting districts? How many solutions to a Ring-Ring puzzle are there on an empty grid? How many Baxter Matrices exist for specific dimensions? These questions may seem quite different at first, but a closer look shows they are all trying to count the number of ways to arrange objects in a two dimensional rectangular grid. In this thesis I will show that all of them can be solved using the same methodology.

The main idea of the approach is to fix the number of rows, and construct a finite state machine that uses an alphabet of the possible columns. By fixing the number of rows, r, we will get a sequence of answers as we increase the number of columns. We can then use a computer algebra system and experimental methods to solve for the generating functions of these sequences for small values of r.

Each chapter of the thesis will be dedicated to a separate problem, and each chapter is independently readable. At the end I will describe a generalized code base where users can implement their own variations on the problem with relative ease.

1.1 The role of Computers in Mathematics

Most of the work done in this thesis was enabled by a computer. We construct matrices with a million entries, and perform operations on them. With computer algebra we can let the entries be polynomials and symbolically compute inverses. The computations being done are enormous and a human alone would not be able to do them. The reader might expect that the results obtained from such methods would not be meaningful to humans. Yet experimental math is a branch of mathematics that has conclusively shown that idea to be false.

This paper will hopefully provide ample evidence of the power of computers. As we will see, sometimes the results of large computations are surprisingly simple. When this happens we often have a rigorous proof of a combinatorial fact. It then becomes a challenge to prove the combinatorial fact using combinatorial methods, which sometimes is doable. But without computers the fact may have never been discovered! Computers can provide not only proofs, but also the theorems and conjectures themselves.

Chapter 2

Puzzles Inspired by The New York Times

In this chapter we demonstrate a method for counting the number of solutions to various logic puzzles. Specifically, we remove all of the "clues" from the puzzle which help the solver to a unique solution, and instead start from an empty grid. We then count the number of ways to fill in this empty grid to a valid solution. We fix the number of rows k, vary the number of columns n, and then compute the sequence $A_k(n)$, which gives the number of solutions on an empty grid of size $k \times n$.

2.1 Ring-Ring

The New York Times has recently been publishing Ring-Ring puzzles. A solution consists of drawing rectangles so that no grid square remains empty. Rectangles are not allowed to share a side or a corner, however they may overlap. See Figure 2.1 for an example of a completed puzzle. A natural question to ask is: How many solutions are there on an empty grid (no black squares) of size $k \times n$?

Fix k, and define $A_k(n)$ to be the number of Ring-Ring solutions on an empty grid of size $k \times n$. This chapter gives a method for finding the generating function of the sequence $\{A_k(n) : n = 1, 2, \ldots\}$, and gives an explicit formula for the generating function for some small values of k.

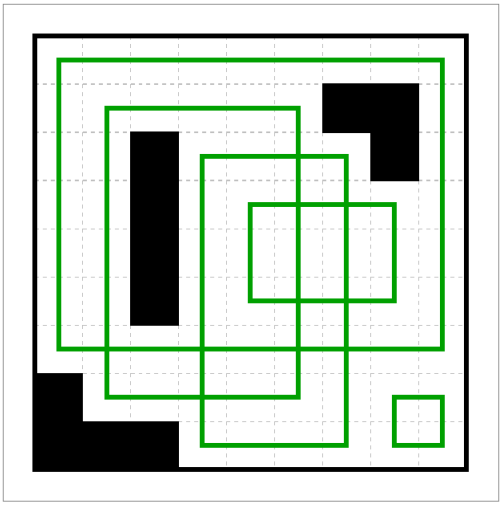


Figure 2.1: A solved puzzle with the solution in green

2.1.1 The 4 Row Case

First consider the case $k = 4$. Suppose you have a completed solution of an $n \times 4$ grid. Look at one particular column. It is not too difficult to show that it must look like one of the following 15 possibilities.

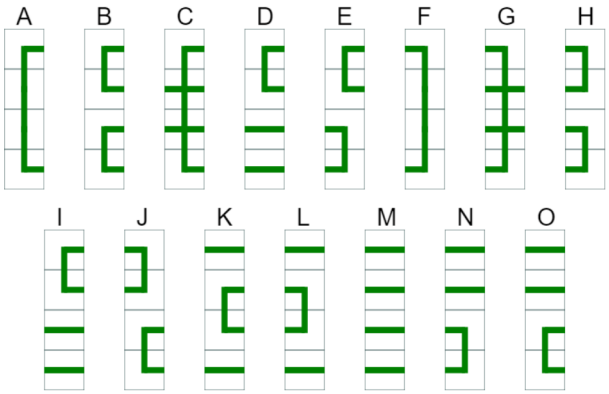


Figure 2.2: The 15 possible columns in the $k = 4$ case

If we call each of the above possibilities a symbol, we have that a solution to the grid must consist of some sequence of symbols. What determines whether such a sequence is legal? We need to ensure that if a symbol leaves some rectangles in progress, that those rectangles are continued in the next symbol. The information of what rectangles are currently in progress can be thought of as a state. When no rectangles are in progress, as is the case for the leftmost column of the grid, the only legal symbols are A and B. However if there is

currently a rectangle in progress that occupies rows 1 and 2 (the top two rows), then the legal symbols are J and O. A state can be described as a set of disjoint subsets of the rows, where each subset is of size 2. These subsets specify the rows in which a rectangle is currently in progress. Each rectangle occupies exactly two rows, one for the top edge, and one for the bottom edge.

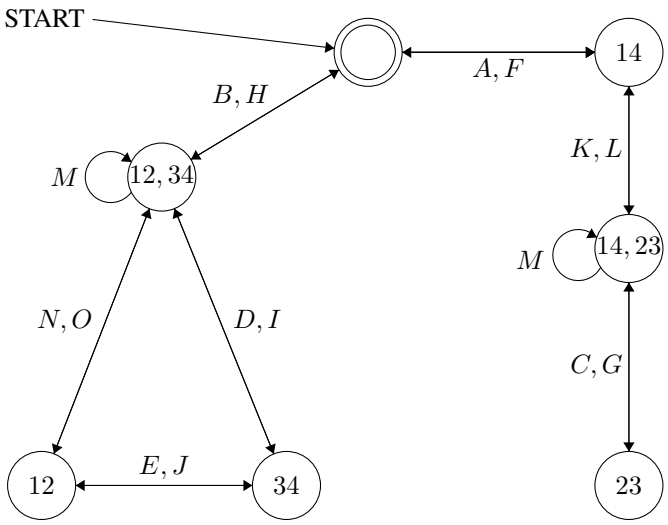


Figure 2.3: Finite state machine in the $k = 4$ case

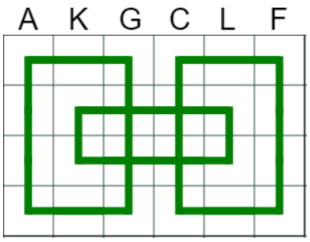


Figure 2.4: The solution corresponding to AKGCLF

We can model this situation nicely using a finite state machine. See [7] for an introduction on finite state machines. The machine in Figure 3 computes whether a string of symbols gives a legal solution for $k = 4$. Any legal solution must start in the state corresponding to the empty set (no rectangles in progress, labelled with START), and proceed to trace a path in the state machine, eventually returning to the empty state. For example, suppose we proceed from the empty state, to $\{1, 4\}$ and then to $\{1, 4\}, \{2, 3\}$ and then to $\{2, 3\}$ and then to $\{1, 4\}, \{2, 3\}$ and then to $\{1, 4\}$ and then to the empty state. This gives the sequence of symbols AKGCLF, which gives the 4×6 solution that is pictured in Figure 2.4.

To count the number of solutions on an $4 \times n$ grid, we need to compute the number of paths of length n that both start and end on the empty state. We can solve for the generating function of this sequence directly by solving a linear system of equations. Let F_i be the generating function such that the coefficient of x^n counts the number of paths of length n from the start state that end up at state i. Then

$$F_i = x \cdot \sum_j F_j$$

where the sum is taken over all states j that have a directed edge to i. In words, the number of paths to state i of length n is equal to the sum of the number of paths of length $n-1$ to states that have an edge to state i. We now must modify the equation slightly for the start state itself, to account for the fact that there is 1 way to get to the start state of length 0. So we have the above equations for $i \neq 0$, and that

$$F_0 = 1 + x \cdot \sum_j F_j$$

For $k = 4$, we can solve this system of equations to get

$$F_0 = \frac{(1+x)(1-2x)(1-2x-x^2)}{(1-3x-3x^2+10x^3+3x^4-5x^5-x^6)}$$

2.1.2 Summary of Results

The approach works for any k, and the author has Maple code that can output the corresponding generating function when given k as input. The states are automatically generated by constructing all possible partitions of $\{1, 2, \ldots, k\}$, into sets of size at most 2. The sets of size exactly 2 indicate the rows at which rectangles are currently in progress. The edges of the state machine are constructed by ensuring the rules of the puzzle are followed. Maple's built-in linear algebra package can solve the corresponding system of equations. Here is the output for some small values of k:

$k = 2$:

$$F = \frac{1-x}{1-x-x^2}$$

As described in [8], we can get a recurrence for the sequence by reading off the coefficients of the denominator of F.

$$A_2(n) - A_2(n-1) - A_2(n-2) = 0$$

This is the recurrence for the beloved Fibonacci sequence, and indeed if we start from $n = 1$ we have

$$A_2(n) = 0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

There is also a nice combinatorial explanation for why we get the Fibonacci recurrence. Consider the first two columns of a 2 by n solution. If it contains a 2 by 2 rectangle, than the number of ways to fill out the rest of the solution is equal to the number of 2 by $(n-2)$ solutions. Otherwise the first column must open a rectangle and the second column must continue it. We can now remove the second column to obtain a solution of length $(n-1)$.

$k = 3$:

$$F = \frac{1}{1 - x^2}$$

$$A_3(n) = 0, 1, 0, 1, 0, 1, 0, 1 \ldots$$

The only way to fill out a 3 by n solution is lining up 3 by 2 solutions next to each other. We also note that whenever k is odd, we will obtain an even function for $F(x)$. This is because there are no solutions if the total number of grid squares available is odd.

$k = 4$:

$$F(x) = \frac{(1+x)(1-2x)(1-2x-x^2)}{1 - 3x - 3x^2 + 10x^3 + 3x^4 - 5x^5 - x^6}$$

$$A(n) = 0, 2, 1, 8, 12, 45, 98, 292 \ldots$$

$k = 5$:

$$F(x) = \frac{-(2x^2-1)(x^4-3x^2+1)}{x^8 - 14x^6 + 19x^4 - 8x^2 + 1}$$

$$A(n) = 0, 3, 0, 12, 0, 51, 0, 221 \ldots$$

$k = 6$:

$$F(x) = \frac{-(64x^{23} + 518x^{22} - 660x^{21} - \ldots)}{(x+1)(68x^{24} + 496x^{23} - 1685x^{22} - \ldots)}$$

$$A(n) = 0, 5, 1, 45, 51, 573, 1365, 8995 \ldots$$

$k = 7$:

$$F(x) = \frac{-(672x^{90} - 177832x^{88} + \ldots)}{(144x^{92} - 55476x^{90} + \ldots)}$$

$$A(n) = 0, 8, 0, 98, 0, 1365, 0, 19982 \ldots$$

$k = 8$:

$$F(x) = \frac{-(41419800576x^{201} + \ldots)}{(507706343424x^{202} + \ldots)}$$

$$A(n) = 0, 13, 1, 292, 221, 8995, 19982, 346281 \ldots$$

Beyond $k = 8$, the Maple code runs into computational limits. In general, the degree of the denominator is bounded by the number of states, which is equal to the number of partitions of $\{1, 2, \ldots, k\}$, into sets of size at most 2. This is A000085 in the OEIS [6], which exhibits super-exponential growth.

Since the generating functions are rational polynomials, we can get formulae for the corresponding sequences by computing the roots of the denominators. In general, we have

$$A_k(n) = \sum_r \frac{p_r(n)}{r^n}$$

where the sum is taken over all the roots of the denominator of F, and the p_r are complex polynomials in n. See [8] for more details on this. Let R be the root of the denominator with the smallest absolute value, and let s be its multiplicity. Using big O notation, we have $A_k(n) = O(n^{s-1} \cdot |1/R|^n)$. If all the roots of the denominator are distinct, then $p_R(n)$ will be a constant, so we have $A_k(n) = O(|1/R|^n)$. In practice, this always was the case. Below is a table of $|1/R|$ from $k = 2$ to $k = 8$.

2	1.618
3	1
4	2.667
5	2.093
6	4.431
7	3.908
8	7.392

Figure 2.5: $1/R$ for various k

2.2 Circuit Board

Secondly, we consider the puzzle "Circuit Board." Here there exist dots at the center of each grid square. We are allowed to connect a dot to any subset of its four immediate neighbors (up, down, left, right). The goal is to connect all the dots so that each dot has either one edge or three edges emanating from it, without forming any closed loops. See Figure 7 for an example puzzle, where the squares in the bottom right are blocked out and will not be used. We assume that there are no blocked out squares and count solutions on a k by n grid.

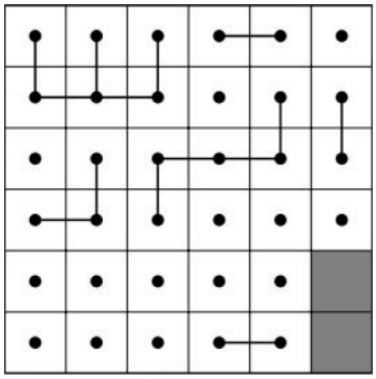


Figure 2.6: An unsolved circuit board puzzle.

Transitioning to the language of graph theory, we are looking for spanning trees of the grid graph where each vertex has degree 1 or 3. To check whether solutions with k rows are valid, we can easily ensure that each vertex has the appropriate degree as we read the columns. However we must also ensure that the graph is connected and that there are no cycles. To do this, our state also keeps track of which subsets of the current column are connected. Thus a state is essentially a set-partition of the rows, each partition indicating locations of the most recent column that are connected using the columns that have been read so far.

The final state consists of set partition with only one sub-partition since everything must be connected. In the

start state, the set partition is a group of singletons because nothing is connected yet. The number of states is a function of the number of set partitions on a set with k elements. This count grows super-exponentially, and is given by the Bell numbers.

The above has all been implemented in Maple, and we can solve for the generating functions using the same technique as described in Section 1. The Maple code was able to complete for $k \leq 6$, but for $k = 7$ ran into computational limits.

For $k = 4$, it turns out there are never any solutions, regardless of the number of columns. Intrigued by this, I found that the number of vertices must be congruent to 2 mod 4 for there to be solutions. Any solution can be constructed by starting with 2 vertices that are connected, and then repeatedly expanding degree 1 vertices to be degree 3. If we think of the vertices on a chess board, then some vertices lie on black squares while others lie on white squares. Expanding a vertex either adds 2 black squares or 2 white squares. Thus our final state must have an even number of vertices, and therefore has the same number of white and black squares covered (since it forms a rectangular grid). Because we started with 2 squares covered, the total amount of squares covered must be 2 mod 4.

For $k = 3$ the first 10 terms of the sequence (starting from $n = 1$) are given by: 0, 1, 0, 0, 0, 10, 0, 0, 0, 36

The corresponding generating function is

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

Using this, we can show that after the first 10 terms, our sequence satisfies the recurrence $A_3(n) = 4 \cdot A_3(n-4)$. Adding 4 more columns gives a factor of 4 more solutions! In this case, there are four distinct 3×4 puzzle pieces that can be sequentially laid down after the first two columns to produce solutions. Also note that the generating function has no constant term, reflecting the fact that the start state is not an accept state.

For $k = 5$ the generating function has approximate degree 50, and for $k = 6$ it has approximate degree 100. Download the code to try it yourself.

2.3 Code

My code for the above computations is all available *here*[4]. It is written in Maple, and makes use of Maple's built-in linear algebra package to solve very large systems of equations.

Chapter 3

Baxter Matrices

3.1 Introduction

Baxter permutations are a class of permutations that come up in numerous places. They were introduced in 1964 by Glen Baxter, who was studying fixed points of commuting continuous functions [2]. They have since been connected to Hopf Algebras [3], planar graphs [4], and tilings [5]. The number of Baxter permutations of length n is given by A001181 in the OEIS and forms a holonomic sequence of order 2 and degree 2 [6]. More recently, Donald Knuth suggested a generalization of Baxter permutations to matrices [1].

Donald Knuth defines a Baxter matrix as a matrix of 0s and 1s that satisfy 4 conditions. Before we state the conditions, we must define a pinwheel in a matrix.

Definition 1. Let M be an $r \times k$ matrix. For $1 \leq x \leq r-1$ and $1 \leq y \leq k-1$, the **clockwise pinwheel** of index (x, y), denoted $P_{x,y}$, is a specific subset of the entries of M. $P_{x,y}$ is divided into 4 **segments**.

- Segment $A_{x,y}$ contains the entries $M[i, y+1]$, for $1 \leq i \leq x$.
- Segment $B_{x,y}$ contains the entries $M[x, i]$, for $1 \leq i \leq y$.
- Segment $C_{x,y}$ contains the entries $M[x+1, i]$, for $y+1 \leq i \leq k$.
- Segment $D_{x,y}$ contains the entries $M[i, y]$, for $x+1 \leq i \leq r$.

Similarly, for $1 \leq x \leq r-1$ and $1 \leq y \leq k-1$, the **counterclockwise pinwheel** of index (x, y), denoted $P'_{x,y}$ also contains four segments.

- Segment $A'_{x,y}$ contains the entries $M[i, y]$, for $1 \leq i \leq x$.
- Segment $B'_{x,y}$ contains the entries $M[x+1, i]$, for $1 \leq i \leq y$.

- Segment $C'_{x,y}$ contains the entries $M[x, i]$, for $y + 1 \leq i \leq k$.
- Segment $D'_{x,y}$ contains the entries $M[i, y + 1]$, for $x + 1 \leq i \leq r$.

Definition 2. A **Baxter Matrix** is a matrix of 0s and 1s that satisfy 4 conditions.

1. Each row contains at least one 1.
2. Each column contains at least one 1.
3. For each clockwise pinwheel, at least one of the four segments has only 0s.
4. For each counterclockwise pinwheel, at least one of the four segments has only 0s.

A pinwheel is said to be satisfied if at least one of its four segments has only 0s. In an $r \times k$ matrix, there are $(r - 1)(k - 1)$ pinwheels of each direction that must be satisfied. As an example consider the two matrices in the figure. The left one is indeed a Baxter matrix, and the reader is encouraged to check that all the pinwheels are satisfied. The right one is almost a Baxter matrix; every pinwheel except $P'_{2,2}$ is satisfied. Note that in general a Baxter matrix could have multiple 1s in a row or column.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \left(\begin{array}{c|c|cc} 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \end{array}\right)$$

A Baxter matrix (left), and a non-Baxter matrix (middle), with its unsatisfied pinwheel (right).

It turns out that $r \times r$ Baxter matrices that only have a single 1 in each row and column are in bijection with Baxter permutations of length r. This can be seen by viewing the matrix as a permutation matrix. For each column i, the corresponding permutation π maps i to r minus the row in which the 1 appears in column i.

What is the maximum number of 1s that can appear in a Baxter matrix of size $r \times k$? Knuth conjectured that this maximum is equal to $r + k - 1$, and exhaustively checked it for all r and $k \leq 7$. In this paper we will prove that the conjecture is true, i.e. that for any $r, k \geq 1$, the number of 1s in an $r \times k$ Baxter matrix is less than $r + k$.

3.2 A Finite State Automaton for Baxter Matrices with r rows

In this section we will describe a finite state automaton for determining whether a matrix is Baxter. We will fix the number of rows, and have the automaton read the columns of the matrix as symbols. When it is

done reading the columns, it should accept or reject according to whether the matrix is Baxter. For ease of explanation, consider first the $r = 2$ case.

3.2.1 The 2 Row Case

Let's construct a finite state machine for determining whether a 0-1 matrix with 2 rows is a Baxter Matrix. The symbols that our machine should recognize as input should be the possible columns in the matrix. There are 4 possible columns in a 0-1 matrix with 2 rows, $[0,0]^T$, $[0,1]^T$, $[1,0]^T$, and $[1,1]^T$. We can ignore the column $[0,0]^T$ because any column in a baxter matrix must not be all zeros.

As we move through the columns of our input matrix, our machine will keep track of the following information for each row: whether the row is all 0s up to this point (so that it can be used to satisfy future pinwheels) and whether the row must be all 0s in the future (because a pinwheel from earlier is depending on it). Thus each row can be in one of 4 possible states; we will refer to them as the 4 rowstates:

1. This row has a 1 in the most recent column.
2. This row only contains 0s up to now.
3. This row must only contain 0s for the rest of the columns, including the most recent one.
4. This row had a 0 in the most recent column but does not fit 2. or 3.

Our machine will have 16 states, one for each ordered pair of rowstates. We next remove the states which have all rows in rowstates 2, 3, and 4. This is because if all rows had a 0 most recently, then the corresponding column contains only 0s, and the matrix cannot be Baxter. This leaves us with $16 - 9 = 7$ states. I claim that the information contained in such a state is enough to determine which columns can come next. Suppose we just read column m, and have a proposed column $m + 1$. For both pinwheels of index $(1, m)$ we can mostly check whether they are satisfied with the information stored. The information about whether the vertical strips are all 0s is known because we store the exact contents of the most recent column in the state. The horizontal strip going left is available if and only if the corresponding row is in rowstate 2. The only thing we don't know yet is whether a horizontal strip going off to the right is all 0s, but if a pinwheel requires it to be so, we can set the rowstate of the corresponding row to 3, and keep track of it for later. We note some other basic constraints:

- Once a row is in rowstate 3 it can never leave rowstate 3.
- Once a row leaves rowstate 2 it can never come back to rowstate 2.
- A row cannot transition from 2 to 4.

- A row cannot transition from 2 to 3 directly, or else it will be all zeros.

As the automaton proceeds reading columns, it checks whether each new pinwheel can be satisfied. When it encounters such a pinwheel that cannot be satisfied, it can immediately reject the sequence of columns. If a pinwheel cannot be satisfied given the first j columns, there is no Baxter matrix that begins with those first j columns. Similarly, if a pinwheel is satisfied given the first j columns, we do not need to keep track of that pinwheel any longer. It will still be satisfied in any Baxter matrix with those first j columns provided we keep track of which rows must be all 0s in the future.

We additionally add a start state that transitions to all states that have each row in either rowstate 1 or rowstate 2 (rowstates 3 and 4 cannot be reached using only a single column). We designate all states that have no rows in rowstate 2 as accept states (2 must be excluded so that no row of the final matrix is all 0s). Enforcing all of the rules we have described so far yields the automaton in Figure 3.1. The state label 12 indicates that the first row is in rowstate 1 and the second row is in rowstate 2. The transition label 10 indicates the column $[1,0]^T$.

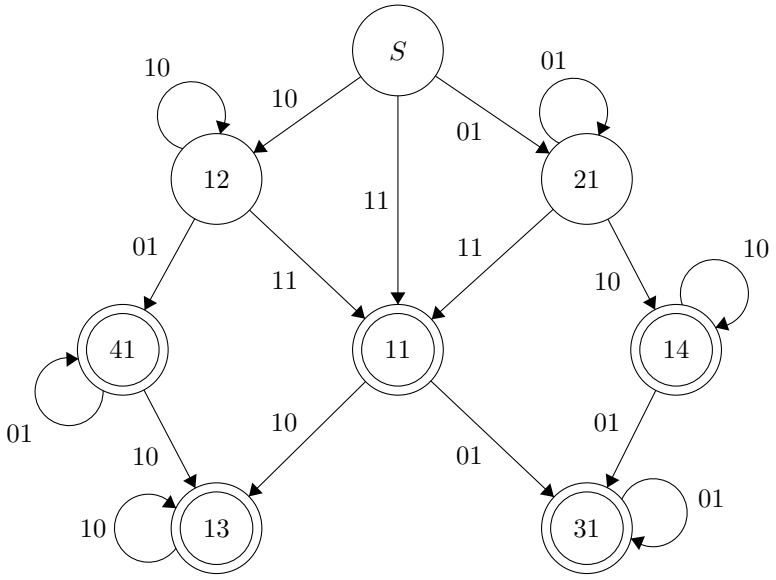


Figure 3.1: The automaton for the 2 row case.

We can do this same process for any fixed number of rows. There will be 2^r symbols, and $4^r - 3^r$ states. Let's call this automaton A_r. Figure 3.2 contains A_3:

3.2.2 Depth of States

I have drawn the automata like this to motivate the following definition.

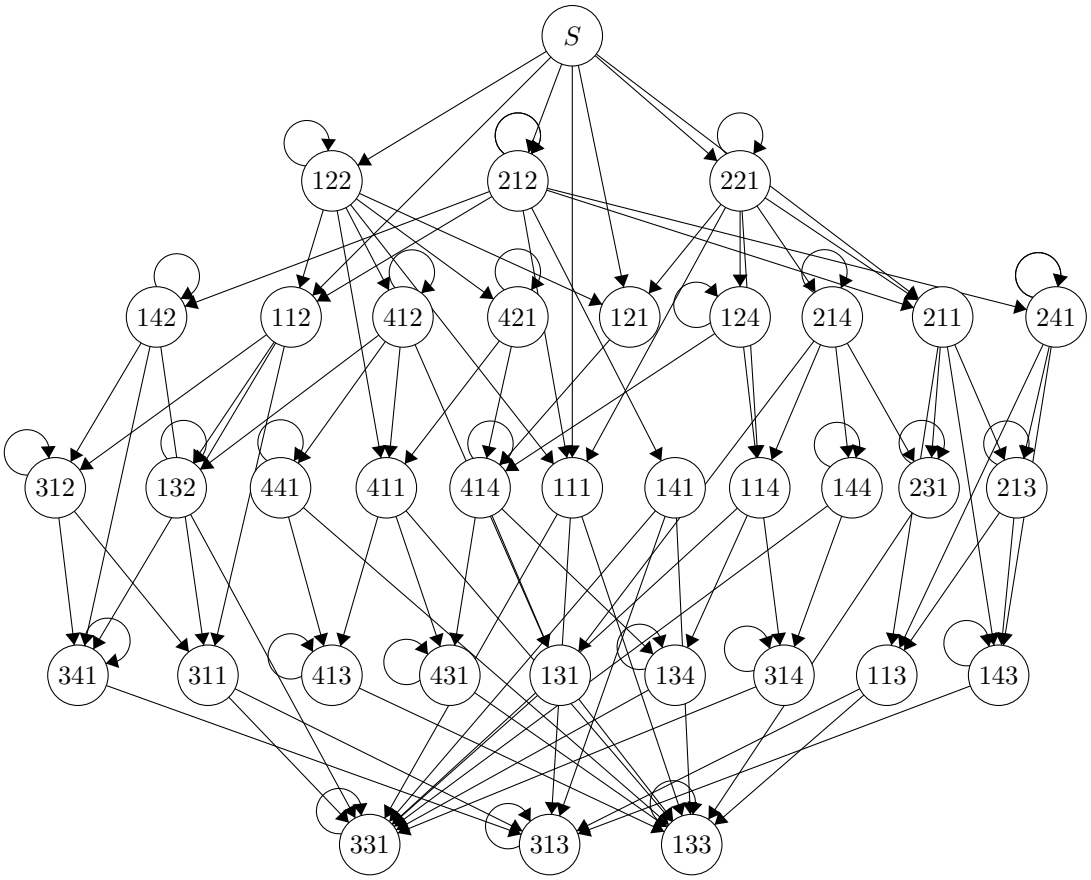


Figure 3.2: The automaton A_3.

Definition 3. The **depth** of a state, s, in A_r, is equal to $r + t_3 - t_2$, where t_3 is the number of 3s that can be found in the rowstates of s, and t_2 is the number of 2s that can be found in the rowstates of s. Let $d(s)$ denote the depth.

Lemma 1. *For a fixed number of rows, r, any transition in A_r must either be a self-loop or increase depth. Additionally, a self-loop emerges from a state if and only if the corresponding column has a single 1.*

Proof. Consider an arbitrary transition. The two states in the transition, let's call them s and s', fully specify the contents of two consecutive columns of the matrix, let's call them m and $m + 1$ respectively. If a row in s is in rowstate 3, it must be in rowstate 3 in s' as well. Similarly if a row in s' is in rowstate 2, it must be in rowstate 2 in s as well. Therefore the contribution to depth from a single row cannot decrease. The total depth is computed as a sum of contributions from each row, so it follows that the total depth cannot decrease. To show that if the depth remains the same, we must have a self transition, we first note that each individual row's contribution in a depth preserving transition must not increase depth. Otherwise, if such a row did increase its contribution, then we can apply the above reasoning to the other rows, and conclude that the total depth must increase. Therefore all 2's in s must remain 2's, and all 3's in s' must have come from 3's in s.

We know that both columns must contain a 1 somewhere. Say the 1 in column m is in row i. It cannot be the case that the only 1 in column $m+1$ is in row i, otherwise we would have a self-loop. Therefore there exists a row $j \neq i$ so that column $m+1$ contains a 1 in row j. Let's assume $j < i$, and note that the other case will be covered by symmetry. The clockwise pinwheel of index (j, m) indicates that row $j+1$ must end up in rowstate 3. Since the depth is preserved it must have started in rowstate 3 as well. Now consider the clockwise pinwheel of index $(j+1, m)$. Since we cannot have row $j+1$ starting in rowstate 2, we require row $j+2$ to end up in rowstate 3 as well. This pattern continues all the way down to row i where we start in rowstate 1 and end in rowstate 3, which necessarily increases the depth.

In fact we have shown something stronger: whenever there exists 1s in consecutive columns that are not in the same row, the corresponding transition must increase depth. Thus a self-loop can only emerge from states that contain a single 1. It is straightforward to check that if a state has a single 1, then the corresponding self-loop does not break any rules, and is present. □

3.2.3 Counting Baxter Matrices

Suppose we fix the number of rows, r, and want to count the number of Baxter matrices with k columns. They are in bijection with paths of length k on our state transition graph. If we fix r and ignore the self-loops, the lemma shows that there are only finitely many possible paths. Thus we can classify all Baxter matrices with r rows into finitely many classes according to the paths they take with all the self-loops removed.

How many k column matrices are there of a particular class? Let P denote a path of length l in the automaton that starts from S and avoids loops, but goes through q states that contain loops. Let $N(P, k)$ be the number of paths of length k arising from P. To compute $N(P, k)$, we just have to choose how many self-loops to put at each node where it is possible to put self-loops, so that the total path length is k. This is the number of ways to choose q natural numbers that sum to $k - l$. Using stars and bars we get that if $k \geq l$:

$$N(P, k) = \binom{k - l + q - 1}{q - 1}$$

which is a polynomial in k of degree $q - 1$. Note that the polynomial is also correct for $l + 1 - q \leq k < l$, since it correctly outputs 0 in these cases.

Notice that the minimum possible depth of a state is 0 at the start state, and the maximum possible depth is $2r - 1$. Thus $l \leq 2r - 1$, and therefore $q \leq 2r - 1$ as well. Thus for $k \geq 2r - 1$, $N(P, k)$ coincides with a polynomial in k of degree at most $2r - 2$.

We can also show that the bound is tight, and that there exists a path P^* with corresponding polynomial having degree exactly $2r - 2$. To do this it suffices to force P^* to go through exactly $2r - 1$ states with

self-loops, i.e. set $q = 2r - 1$. Let M^* be the $r \times (2r-1)$ Matrix that is all 0s except for the following locations: $M^*[i,i] = 1$ for $1 \leq i \leq r$, and $M^*[i, 2r-i] = 1$ for $1 \leq i \leq r$. Thus M^* has two diagonal stripes of 1s that intersect in the middle. M^* has no self-loops since it has no repeated columns, but since each column has a single 1, all of the states that is passes through have legal self-loops. It is straightforward to check that M^* is indeed a Baxter matrix, and then we can set P^* to be the path in A_r that corresponds to M^*.

Now for each possible path P without self-loops, $N(P,k)$ is eventually a polynomial in k of degree at most $2r-2$. The total number of matrices with r rows and k columns will be the sum over all these polynomials, which will be a polynomial of degree exactly $2r-2$. The only small caveat is that for small values of k we must take the maximum of each polynomial and 0, so some paths cannot contribute negatively. Summarizing:

Theorem 1. *Let $a_{r,l,q}$ be the number of paths of length l in A_r that start from S, do not use self-loop transitions, but pass by q states carrying a loop. Let $P_r(x)$ be the polynomial in x defined by*

$$P_r(x) = \sum_{l,q \geq 1} a_{r,l,q} \binom{x-l+q-1}{q-1}$$

Then $P_r(x)$ has degree $2r-2$.

Let $N_{r,k}$ be the number of Baxter matrices of size $r \times k$. Then, for $k \geq 2r-1$ we have $N_{r,k} = P_r(k)$.

This shows that for a fixed number of rows, r, the number of Baxter matrices with r rows and k columns eventually satisfies a polynomial in k of degree $2r-2$. Computing the polynomial for a fixed r is straightforward once the transition graph has been constructed. The author has Maple code that constructs the graph and computes the corresponding polynomial using the above method. The code completes instantly for $r \leq 5$ and within a couple minutes for $r = 6$. The polynomial for $r = 2$ also appears in Knuth's paper, who found it using a combinatorial approach.

rows	formula	works for
2	$k^2 + 3k - 4$	$k \geq 2$
3	$(1/3)k^4 + 3k^3 - (16/3)k^2 + 2k + 3$	$k \geq 3$
4	$(1/18)k^6 + (21/20)k^5 - (5/18)k^4 - (151/12)k^3$ $+(443/9)k^2 - (1012/15)k + 28$	$k \geq 4$
5	$(23/4032)k^8 + (937/5040)k^7 + (853/1440)k^6 - (2671/360)k^5$ $+(15697/576)k^4 - (341/720)k^3 - (1274363/5040)k^2$ $+(98659/140)k - 643$	$k \geq 5$
6	$(361/907200)k^{10} + (403/20160)k^9 + (5177/30240)k^8 + \ldots$	$k \geq 6$

The reader will notice that the polynomials are correct for $k \geq r$, which is a much better bound than $k \geq 2r - 1$. Does this pattern continue for larger r? The answer is yes, and we can prove it using Lemma 2 from the next section. Recall that for any path P in A_r without self-loops, the polynomial corresponding to P is correct for $k \geq l + 1 - q$, where l is the length of P, and q is the number of states with self-loops that P passes through. Thus if we can show that for any P, $l + 1 - q \leq r$, we can show that $N_{r,k} = P_r(k)$, for $k \geq r$. To compute the maximum of $l + 1 - q$, we note that $l - q$ is equal to the number of states without self-loops that P passes through. Using the terminology of the next section, $l - q$ is equal the number of states with extra 1s that P passes through. By Lemma 2, P cannot pass through two states with extra 1s on consecutive depth levels. We know $l \leq 2r - 1$, and since the first possible depth a state with extra 1s can occur on is depth 2, we get that $l - q \leq r - 1$. Thus $l + 1 - q \leq r$.

3.3 Resolving one of Knuth's conjectures

Donald Knuth conjectured that the number of 1s in any $r \times k$ Baxter matrix is fewer than $(r + k)$. We know from the definition of Baxter matrices that each column must contain at least one 1. Then we can refer to any 1 that is not the topmost in its column as an extra 1.

Theorem 2. *The number of extra 1s in a Baxter matrix with r rows is less than r.*

To prove this, we will use the following lemma.

Lemma 2. *The total number of extra 1s that appear in two consecutive columns is at most the change in depth of the corresponding state transition in A_r.*

For now let's assume the lemma is true. Suppose M is some Baxter matrix with r rows and k columns. Let p be its corresponding path in A_r, and let T be the set of transitions in p. Let t^* be the final state in p. If

we use the fact that the start state does not contain any extra 1s and assume that t^* does not contain any extra 1s, we get that

$$(\#\text{ of extra 1s in } M) = \frac{1}{2}\left(\sum_{\tau \in T}(\#\text{ of extra 1s in the columns associated with } \tau)\right) \tag{3.1}$$

Suppose t^* does contain extra 1s. Let t' be the state obtained by replacing all the extra 1s in t^* with rowstate 3. We now modify p to contain an extra transition from t^* to t', regardless of whether this transition exists in A_r. Note that t' does not have any extra 1s by construction. This extra transition satisfies Lemma 2, because for each extra 1 in t^*, the depth has increased by 1 in that row. Now we can apply Lemma 2 to the (potentially modified) p.

$$\begin{aligned}
(\#\text{ of extra 1s in } M) &= \frac{1}{2}\left(\sum_{\tau \in T}(\#\text{ of extra 1s in the columns associated with } \tau)\right) && (3.2)\\
&\leq \frac{1}{2}\left(\sum_{\tau \in T}(\text{depth increase of } \tau)\right) && (3.3)\\
&\leq \frac{1}{2}(2r-1) && (3.4)\\
&< r && (3.5)
\end{aligned}$$

Now for a proof of Lemma 2:

Proof. Consider 2 consecutive columns, column m and column $m+1$. We will denote them by c_0 and c_1 for short. Let i be the minimum row which contains a 1 in c_0, j be the minimum row that contains a 1 in c_1, k be the maximum row which contains a 1 in c_0, and l be the maximum row that contains a 1 in c_1. We can assume $i \leq j$, for if not the proof is symmetric. We will consider 3 cases.

Case 1: $i \leq k \leq j \leq l$. We will show that each extra 1 in c_0 forces the depth to increase by at least one. If $i = k$ then there are no extra 1s so there is nothing to show. So we have $i < k \leq j$. Let's look at the counterclockwise pinwheel of index $(k-1, m)$. The only way to satisfy it is if row $k-1$ ends up in rowstate 3. We can then look at the counterclockwise pinwheel of index $(k-2, m)$, and conclude that row $k-2$ must end up in rowstate 3 as well (the left segment cannot be used otherwise that row would be all 0s). We get that each row from $k-1$ up to i ends up in rowstate 3, so we have an increase in depth for each 1 in c_0 from rows i to $k-1$, which is an increase in depth by 1 for each extra 1 in c_0. Now look at the counterclockwise pinwheel of index (j, m), and recall that $j \geq k$. It forces row $j+1$ to start in rowstate 2. Applying the same process, all rows from row $j+1$ to l must start in rowstate 2. This gives an increase in depth for each 1 in c_1

between row $j+1$ and l.

If we are not in case 1, then it must be the case that $k > j$.

Case 2: $i \leq j = l < k$. We will show that for each 1 that is not on row j, there must be an increase in depth. If there is a 1 above row j, then the counterclockwise pinwheel of index $(j-1, m)$ can only be satisfied if row $j-1$ ends up in rowstate 3. Continuing to look at the counterclockwise pinwheels upward, all rows must end up in rowstate 3 from row $j-1$ up to row i. Thus all 1s in those rows cause there to be transitions from rowstate 1 into rowstate 3, causing an increase in depth. Similarly the clockwise pinwheel of index (j, m) requires row $j+1$ to end up in rowstate 3, and all rows below it down to row k as well. Thus each 1 below row j also causes an increase in depth.

Case 3: $i \leq j < l$ and $k > j$. It is not hard to show that in this case $i < j$ and there are no 1s in the first column on rows j through l. Let i' be the maximum row that contains a 1 that is less than j, and k' be the minimum row that contains a 1 that is greater than l. We will show that each 1 except for the 1s in rows i' and k' cause an increase in depth. By the same reasoning that we applied in case 1, there must be an increase in depth for each 1 above row i' and for each 1 below row k'. If there are at least two 1s in the c_1, then each of those 1s must have transitioned from rowstate 2 in c_0, causing an increase in depth. If there is a single 1, then we can look at the clockwise pinwheel below it to conclude it either transitioned from rowstate 2 or the next row ends up in rowstate 3. In the first case we are done and in the second case we can progress the rowstate 3's down until we get that row k' must end up in rowstate 3. □

Note that the number of extra 1s in a Baxter matrix is completely determined by its path in A_r. Also Lemma 2 shows that a self-loop never can produce extra 1s. Therefore the self-loops in a path have no effect on the number of extra 1s. If we want to compute the number of $r \times k$ Baxter matrices with t 1s for some $k \leq t < k + r$, we can use the same method that we used to count them before, except only add the polynomials for paths without self-loops that add the appropriate amount of extra 1s. Here are some results:

$r = 3$, correct for $k \geq 3$

extra 1s	total weight	formula
0	k	$(1/3)k^4 - k^3 + (2/3)k^2$
1	$k+1$	$4k^3 - 12k^2 + 15k - 8$
2	$k+2$	$6k^2 - 13k + 11$

$r = 4$, correct for $k \geq 4$

1s	weight	formula
0	k	$(1/18)k^6 - (3/10)k^5 + (2/9)k^4 + (3/2)k^3 - (77/18)k^2 + (24/5)k - 2$
1	$k+1$	$(27/20)k^5 - (47/6)k^4 + (235/12)k^3 - (157/6)k^2 + (226/15)k$
2	$k+2$	$(22/3)k^4 - (121/3)k^3 + (335/3)k^2 - (500/3)k + 106$
3	$k+3$	$(20/3)k^3 - 32k^2 + (238/3)k - 76$

There is a github repository for the maple code, and the code is open source under the MIT license. https://github.com/DarthCalculus/BaxterMatrices

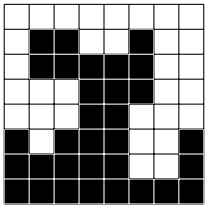
Chapter 4

The Gerrymander Sequence

This chapter is joint work with Manuel Kauers and Christoph Koutschan.

4.1 Introduction

In a guest lecture on April 28, 2022, in Zeilberger's famous course on experimental mathematics at Rutgers [17], Sloane talked about some of his favorite entries of the OEIS. One of the entries he highlighted in his lecture was A348456. The nth term of this sequence is defined as the number of ways to dissect a $2n \times 2n$ chessboard into two polyominoes, each of area $2n^2$. Here is one of the solutions for $n = 4$:



Finding such dissections can be viewed as a combinatorial version of gerrymandering, and it has thus been suggested to call the sequence the gerrymander sequence. When Sloane gave his lecture, only the first three terms of the gerrymander sequence were known (they are 2, 70, 80518). He declared, perhaps exaggerating a bit, that he considers the next term of this sequence as the "most wanted number" in the whole OEIS. This statement motivated Zeilberger to offer a donation of $100 to the OEIS in honor of the person who first manages to compute this most wanted number. In this paper, we explain how we computed not only the next

term of A348456, but in fact the next four terms. They are

$$7157114189,$$

$$49852157614583644,$$

$$2828935859304341472594435 3, \text{ and}$$

$$133505657942308037118645688854373216 2,$$

respectively. In addition, we confirm the correctness of the previously known three terms. Our Mathematica source code is available at http://www.koutschan.de/data/gerry/.

We employ the transfer-matrix method, a classical technique in enumerative combinatorics whose general idea is nicely explained in Sect. 4.7 of Stanley's textbook [18]. We tweak this method by introducing catalytic variables, in order to apply it to the problem at hand; this is explained in Sect. 4.2. Besides the computation of specific terms, the transfer-matrix method also allows us to derive structural information about the generating function for boards of rectangular shapes $m \times n$ when m is fixed and n varies. The case $m = 3$ was proposed by Knuth as a Monthly problem a few years ago [16]. It turns out that if a_n is the number of ways to break a $3 \times 2n$ board into two connected components of the same size (A167242), then

$$\sum_{n=0}^{\infty} a_n x^n = \frac{1+\sqrt{1-4x}}{(\sqrt{1-4x}+x)^2} \frac{1}{\sqrt{1-4x}} - \frac{1-x^2+2x^3}{(1-x)^3}.$$

It is not a coincidence but a consequence of a theorem of Furstenberg [10] that the generating function is algebraic. In principle, it can be computed by combining the transfer-matrix method with the method of creative telescoping [19, 9, 13]. However, this quickly becomes expensive when m increases. In Sect. 4.4, we report on some computations we did in this direction.

4.2 The transfer matrix

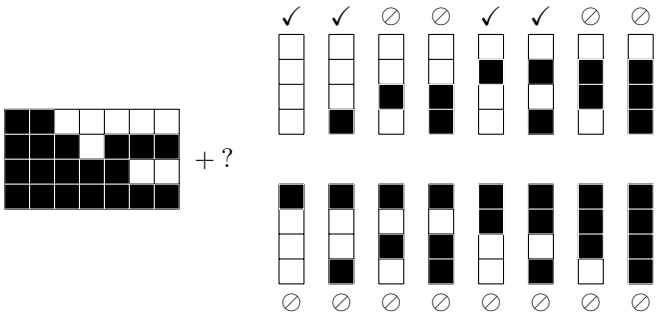
The transfer-matrix method was invented in the context of statistical mechanics [14, 15], in order to express the partition function of a statistical model in a simpler and more succinct form than its plain definition as a multi-dimensional sum. The method is applicable whenever the mechanical system can be decomposed into a sequence of N subsystems, each of them interacting only with the previous and the next one. Let ℓ denote the number of states that each of these subsystems can have, and $m_{i,j}(k)$ a "statistical weight" that is associated with state i of subsystem $k-1$ being next to state j of subsystem k. The relation between these two adjacent subsystems is then described by the *transfer matrix* $M(k) = \big(m_{i,j}(k)\big)_{1 \le i,j \le \ell}$, and the partition function of

the whole system can be written in the form

$$v_{\text{init}}^{\top} \cdot M(1) \cdot M(2) \cdots M(N) \cdot v_{\text{final}},$$

where v_{init} and v_{final} are vectors of dimension ℓ.

Recall that we are interested in counting the number of ways that an $m \times n$ grid can be divided into two (or, more generally, q) connected regions, each of which is represented by a different color. We can apply the transfer-matrix method to this gerrymandering problem by decomposing the grid into a sequence of columns that are added one after the other. In each step, not all of the 2^m (resp., q^m) potential columns can be added, because some would violate the rules, e.g., by creating two disconnected regions of the same color:



Clearly, the information how the squares in the right-most column are colored is not sufficient to decide which columns can come next. For example, we need to know that the two black squares in the last column belong to the same connected region, otherwise we could not add a column with only white squares.

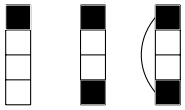
Thus, in order to decide which columns can be added and to tell whether the completed grid has the desired number of connected regions, we introduce *states* for remembering connectivity information that is implied by all previous columns. More precisely, a state is described by a pair (c, P), where $c \in \{0,1\}^{2n}$ encodes the content of the last column (the colors white and black are now represented by the numbers 0 and 1, respectively), and where $P = \{P_1, \ldots, P_k\}$ is a partition of $\{1, \ldots, 2n\}$ that indicates which squares in that column belong to the same connected region. For example, in the figure above, the previous columns imply that the two black squares are connected, while the two white squares are not. The fact that they could (and should!) be connected by adding further columns is not relevant at this moment. Hence, in the partially completed grid we have k different regions, and P_j gives the positions of squares belonging to the j-th region. In particular, all these squares must have the same color, that is $|\{c_i \mid i \in P_j\}| = 1$ for all $1 \le j \le k$.

For example, $\big((0,0,1,0), \{\{1,2\},\{3,4\}\}\big)$ or $\big((1,1,0,0), \{\{1,2\},\{3\},\{4\}\}\big)$ are not valid state descriptions because the first violates the same-color condition, while the latter claims that the squares at positions 3

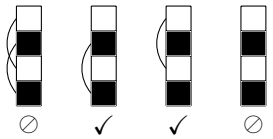
and 4 belong to different regions although they are obviously connected. In contrast,

$$\big((1,0,0,0),\{\{1\},\{2,3,4\}\}\big),$$
$$\big((1,0,0,1),\{\{1\},\{2,3\},\{4\}\}\big),$$
$$\text{or } \big((1,0,0,1),\{\{1,4\},\{2,3\}\}\big)$$

are valid state descriptions, which we depict graphically as follows (connections that happen in previous columns are symbolized by an arc):

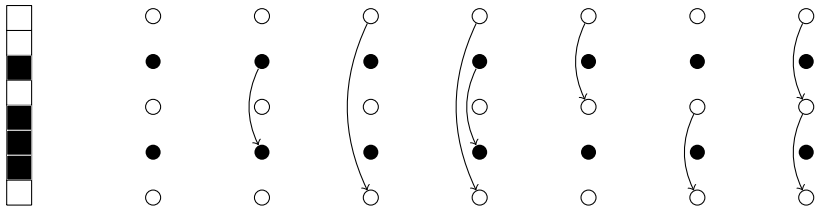


But even if we stick to the above rules, there are still many pairs (c, P) that describe *impossible* states, for example, $\big((0,1,0,1),\{\{1,3\},\{2,4\}\}\big)$. Connecting 1 with 3 and 2 with 4 produces arcs that cross each other, meaning that this connectivity cannot be achieved by extending the column to the left. Similarly, by considering how a column could be extended to the right, we can discard *uninteresting* states, i.e., states that could possibly be reached, but which represent "hopeless" situations that will never allow us to complete the grid in a satisfactory manner. For example, the state $\big((0,1,0,1),\{\{1\},\{2\},\{3\},\{4\}\}\big)$ is uninteresting, because all four squares are declared to belong to different regions, and it is impossible to connect 1 with 3 and 2 with 4 by adding more columns to the right. Hence, for the column $c = (0,1,0,1)$ we consider only two out of four possible states:



Algorithmically, we construct the set of states as follows: in step (1) we enumerate the set of all states that are not impossible, and in step (2) we discard those states which are uninteresting. In both steps it is clear from the construction that no necessary state is discarded, ensuring the correctness of our method.

1. For each tuple $c \in \{0,1\}^{2n}$, all non-crossing arc configurations are generated, where the vertices for these configurations are maximal chunks of squares of the same color. All arcs point in the same direction and each arc connects two vertices of the same color. Each vertex has at most one outgoing and at most one incoming arc. For example, the following column of size 8 admits seven such arc configurations:



2. A state (c, P) is uninteresting if there are two connected components $A_1, A_2 \in P$ of the same color and $B_1, B_2 \in P$ of the opposite color such that the following condition holds:

$$\max A_1 + 1 = \min B_1 \wedge \max B_1 = \min A_2 - 1$$
$$\wedge (\max A_2 < \max B_2 \vee \min B_2 < \min A_1).$$

It corresponds to situations where we cannot draw another non-crossing arc configuration on the right side such that all vertices of the same color get connected:

uninteresting: 

interesting state: 

There is one subtle issue one still has to take care of: if the current column equals $(0, \ldots, 0)$ or $(1, \ldots, 1)$, then we need to store the information whether the other color has not yet appeared (in which case this column can be followed by any other column), or whether the other color has appeared previously (in which case this column can only be followed by more copies of the same column). In our graphical notation, we decorate the latter of these two states by a prime.

Let us denote by L the set of states that is constructed according to the above rules. The number of states $\ell = |L|$ grows (at least) exponentially with n, since each of the 2^{2n} possible columns appears in at least one state. For example, for $n = 2$ we have 16 different columns but 26 states in total; they are explicitly enumerated in Figure 4.1. The number of states for $1 \leq n \leq 7$ is given in Table 4.1.

To construct the transfer matrix M, for each state we need to determine which other states it can move into by adding an appropriate next column. Note that for this purpose it does not matter in which particular column of the grid we are: in each step we can use the same matrix M, which is in contrast to the general situation sketched at the beginning of this section. The rows and columns of the transfer matrix are indexed by the states; hence we obtain an $\ell \times \ell$ matrix. Let $s = (c, P)$ be any of the ℓ states and let $c' \in \{0, 1\}^{2n}$ be

$2*n$	$2*\ \ \ell$	v_{init}		v_{final}		M	
		$\#_{\neq 0}$	sparsity	$\#_{\neq 0}$	sparsity	$\#_{\neq 0}$	sparsity
1	6	4	33.33 %	6	0.00 %	16	55.56 %
2	26	14	46.15 %	16	38.46 %	178	73.67 %
3	154	32	79.22 %	34	77.92 %	2546	89.26 %
4	1026	58	94.35 %	60	94.15 %	44008	95.82 %
5	7222	92	98.73 %	94	98.70 %	832454	98.40 %
6	52650	134	99.75 %	136	99.74 %	16505486	99.40 %
7	393878	184	99.95 %	186	99.95 %	337332580	99.78 %

Table 4.1: Number ℓ of states and number $\#_{\neq 0}$ of non-zero entries in v_{init}, v_{final}, and in the transfer matrix M.

an arbitrary column. If attaching c' to the state s would violate the connectivity requirements, then the matrix entries at positions (s, s') are set to 0, where s' is any state containing c'.

But what should be put as the matrix entry when a transition is actually possible? Here another condition has to be considered that so far has not been taken care of: the two regions must have the same area. Since this requirement can only be checked at the very end when the whole grid is filled, we need to propagate information about the number of squares of either color through the whole computation. For this purpose, we introduce a "catalytic" variable x that counts the number of white squares that have been used so far. In each transition this counter has to be increased accordingly. Assume that state $s = (c, P)$ admits attaching column c' to it, which basically means that no existing region gets disconnected (this happens when c and c' have opposite colors at all positions given by some $P_j \in P$), except when all squares of c' have the same color and only a single $P_j \in P$ is related to the other color, in which case we move to one of the two primed states. In any case, the new connectivity information P' is uniquely determined from c, c', P, and therefore yields a new state $s' = (c', P')$. It may well happen that $s' \notin L$ is an uninteresting state, in which case no matrix entry is generated. Otherwise the matrix entry at position (s, s') is set to $x^{\#_0(c')}$, where $\#_0(c')$ denotes the number of 0's in c'.

Now that we have constructed the transfer matrix M, it remains to consider the start and the end of this process. We start the grid with a single column. There cannot be any additional connectivity information other than what can be seen in this column. We define a start vector v_{init}, which is indexed by the states and hence is ℓ-dimensional. Its entry at position $s = (c, P)$ equals $x^{\#_0(c)}$ if the parts of P correspond exactly to the consecutive runs of entries of the same color in c (in other words, if the graphical representation of s has no arcs (and no prime!)), and 0 otherwise.

When we have filled the grid up to the last column, we have to decide which states are "accept" states. Clearly, this is the case for states $s = (c, P)$ such that $|P| \leq 2$. Since we just want to add up the results of all acceptable states, we define a vector v_{final} that is 1 at accept states and 0 otherwise (see Table 4.1).

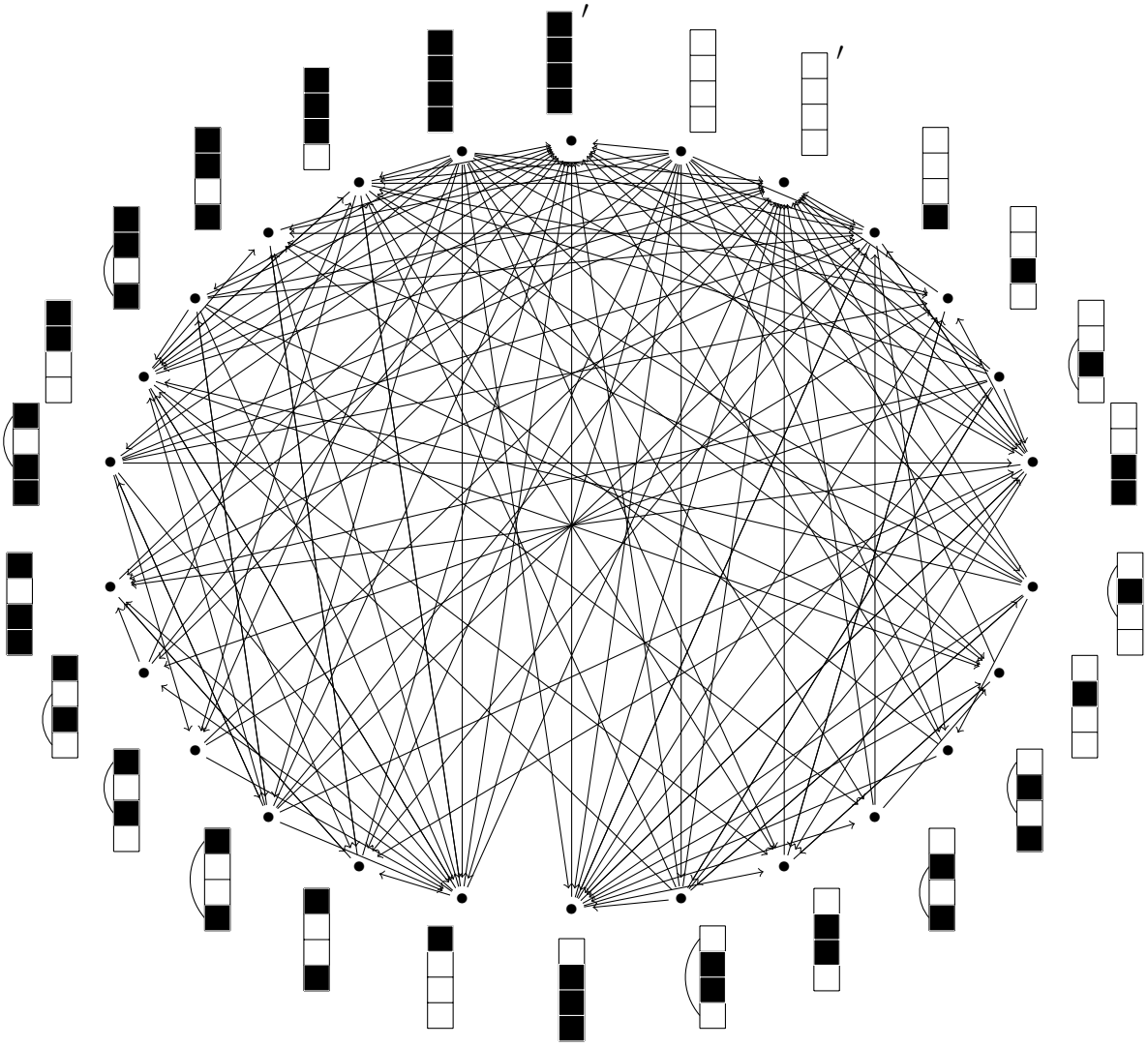


Figure 4.1: All 26 states for a grid with 4 rows and their possible transitions; note that each state can also be followed by itself — these loops are not depicted.

Having all this at hand—the transfer matrix M, the start vector v_{init}, and the end vector v_{final}—one can compute

$$p(x) = v_{\text{init}}^{\top} \cdot M^{2n-1} \cdot v_{\text{final}}, \tag{4.1}$$

which is a polynomial in x. The coefficient of x^k in p gives the number of ways that the $2n \times 2n$ grid can be divided into a white polyomino of k squares and a black polyomino consisting of $4n^2 - k$ squares. This means that for sequence A348456 we need to extract the coefficient of x^{2n^2} and divide the result by 2 in order to eliminate the distinction between black and white.

4.3 Optimizations

In this section, we discuss some optimizations that one can employ when implementing the method described in Section 4.2. For our purposes, we have used the computer algebra systems Maple and Mathematica.

First, it is clear that in (4.1) one should not compute M^{2n-1} explicitly, using expensive matrix multiplications. Instead, it is more efficient to exploit the associativity of matrix multiplication and perform only (cheap) matrix-vector multiplications:

$$p(x) = (\cdots((v_{\text{init}}^{\top} \cdot M) \cdot M) \cdots) \cdot v_{\text{final}}.$$

Then we address the problem that the matrices get large: for example, for $n = 7$ the transfer matrix M has 393878^2, i.e., about 115 billion entries, which poses memory challenges when one does not have a supercomputer at hand. However, we realize that the matrix is very sparse. Clearly, in each row we can have at most 2^{2n} nonzero entries (because this is the number of different grid columns we can add), but actually there are much fewer, since many of these columns are not admissible (e.g., because they disconnect existing regions). Table 4.1 shows the sparsity of the transfer matrix M and of the vectors v_{init} and v_{final}. In Mathematica, one can use the command `SparseArray` to store such matrices in a memory-efficient way, which has the additional advantage that it also speeds up the matrix-vector multiplications. Note that the vector will quickly become dense as we multiply the matrix to it, reflecting the fact that we have no unreachable states in L.

The next observation concerns the structure of the transfer matrix. Since each specific column of the matrix is responsible for producing the counting polynomial for a specific state (c, P), by matrix multiplication, all entries of this column of M must either be 0 or $x^{\#_0(c)}$. No other powers of x can occur in the same column. It is more efficient to work with the $\{0, 1\}$-matrix $M' = M\big|_{x \to 1}$ and store the x-powers in a

transfer matrix		mult. in $\mathbf{Z}[x]$	in $\mathbf{Z}[x]/(x^{2n^2+1})$	eval-int. + CRT
dense	M	23.1 s	22.3 s	69.6 s
dense	$M'X$	9.1 s	8.3 s	89.7 s
sparse	M	16.3 s	15.3 s	1.8 s
sparse	$M'X$	1.9 s	2.0 s	0.6 s

Table 4.2: Effect of different strategies on the runtime, for $n = 4$.

2*n	2*create states	2*build M	total time for matrix-vector multiplications		
			mult. in $\mathbf{Z}[x]$	in $\mathbf{Z}[x]/(x^{2n^2+1})$	eval-int. + CRT
3	0.01 s	0.39 s	0.10 s	0.12 s	0.06 s
4	0.06 s	10 s	1.9 s	2.0 s	0.6 s
5	0.87 s	5 min	56 s	54 s	21 s
6	13.6 s	5 h	49 min	44 min	10 min
7	4 min	5 d	29 h	25 h	5 h

Table 4.3: Runtime observed for various strategies and various problem sizes, using the decomposition $M'X$ and the sparse matrix representation.

separate diagonal matrix

$$X = \operatorname{diag}\big(\big(x^{\#_0(c)}\big)_{(c,P)\in L}\big)$$

such that $M = M' \cdot X$. Computing $v_{\text{init}}^\top \cdot M$ has the disadvantage that large intermediate expressions are produced that can only be combined after expansion, since they are sums of products of monomials times polynomials. This is avoided by computing $\big(v_{\text{init}}^\top \cdot M'\big) \cdot X$.

We know that the result is a polynomial $p(x)$ of degree $4n^2$, whose coefficient of x^{2n^2} we wish to extract. Hence, the whole computation can be done modulo x^{2n^2+1}, which does not change the coefficient of x^{2n^2}, but which reduces the size of intermediate expressions. Alternatively, one can apply the evaluation-interpolation technique combined with the Chinese remainder theorem (CRT). Not only the degree of $p(x)$ is known, we also know that it is palindromic, i.e., $p(x) = x^{4n^2} p(1/x)$. Therefore $p(x)$ can be interpolated by using only $2n^2 + 1$ evaluation points. In addition, we can determine an a-priori bound on the height of $p(x)$: let k be the maximal number of nonzero entries in any row (or column) of M, then the entries of M^{2n-1} are polynomials with coefficients at most k^{2n-2} and the height of $p(x)$ is thus bounded by $\ell^2 k^{2n-2}$.

Table 4.2 illustrates the effect of the different strategies on the runtime of the computation. Table 4.3 shows computation times for $3 \le n \le 7$. Note that we used parallelization for some of the tasks, but the timings are given in CPU time. They were measured on Intel Xeon E5-2630v3 processors at 2.4 GHz. As a curiosity, we realized that Mathematica takes about twice as long for computing $v_{\text{init}}^\top \cdot M$ (row vector times matrix) compared to computing the equivalent product $M^\top \cdot v_{\text{init}}$ (transposed matrix times column vector). The timings in the last column of Table 4.3 refer to the minimal number of primes that are needed to

reconstruct the correct result (which we know already from the two previous computations). In order to obtain a provably correct result with the CRT approach, one would have to use enough primes to exceed the bound on the height of $p(x)$. For example, the result for $n = 7$ is approx. $1.335 \cdot 10^{36}$, therefore requiring 4 primes of size 2^{31}, while our bound $\ell^2 k^{2n-2}$ with $\ell = 393878$ and $k = 16384$ yields $5.804 \cdot 10^{61}$, corresponding to 7 primes of the same size.

4.4 Further results

While the gerrymandering problem for a rectangular board of size $m \times n$ is symmetric in m and n, the cost of the transfer-matrix method is highly asymmetric. As explained above, the cost depends exponentially on the side length that determines the transfer matrix but only polynomially on the side length that appears in the exponent. Because of this discrepancy, slim rectangular boards are somewhat easier to handle than boards that are quadratic or close to quadratic. For small values of m, it is not too hard to let n grow into the hundreds or even thousands.

For fixed m and varying n, we are interested in the number of solutions to the gerrymandering problem for a board of size $m \times n$. Obviously there is no solution when both m and n are odd. Therefore, for fixed and even m, we define a_n as the number of solutions for a board of size $m \times n$, and for fixed odd m, we define a_n as the number of solutions for a board of size $m \times 2n$. For certain vectors $v_{\text{init}}, v_{\text{final}}$ and a certain matrix M whose entries are polynomials in x, we then have $a_n = \frac{1}{2}[x^{nm/2}](v_{\text{init}}^\top M^{n-1} v_{\text{final}})$ if m is even and $a_n = \frac{1}{2}[x^{nm}](v_{\text{init}}^\top M^{2n-1} v_{\text{final}})$ if m is odd. We know from linear algebra that the entries of a matrix power M^n are C-finite sequences with respect to n, i.e., they satisfy linear recurrences with constant coefficients, or in other words, their generating functions are rational. An explicit formula is given in Thm. 4.7.2 of Stanley's book [18]: for a fixed $\ell \times \ell$-matrix M and any $i, j \in \{1, \ldots, \ell\}$, the generating function of the sequence appearing in the (i, j)th entry of M^n is

$$(-1)^{i+j} \frac{\det(I_\ell - t\, M)^{[j,i]}}{\det(I_\ell - t\, M)},$$

where the exponent $[j, i]$ indicates the removal of the jth row and the ith column of the matrix $I_\ell - tM$. The generating function for a sequence defined as $v_{\text{init}}^\top M^{n-1} v_{\text{final}}$ is just a certain linear combination of such rational functions.

In particular, the rational generating function for the sequence $(v_{\text{init}}^\top M^{n-1} v_{\text{final}})_{n=0}^\infty$ (or $(v_{\text{init}}^\top M^{2n-1} v_{\text{final}})_{n=0}^\infty$, if m is odd) can be explicitly computed from the vectors $v_{\text{init}}, v_{\text{final}}$, and the matrix M. At least in principle. In practice, for large matrices M involving a symbolic parameter x, computing the determinant of $I_\ell - t\, M$,

m	1	2	3	4	5	6	7
$\deg_t$ of numerator	2	4	7	17	36	75	203
$\deg_x$ of numerator	2	4	23	34	190	236	1425
monomials in numerator	3	9	76	194	1955	4312	55218
$\deg_t$ of denominator	2	4	7	17	36	75	203
$\deg_x$ of denominator	2	4	22	34	188	235	1422
monomials in denominator	4	9	76	194	1935	4310	55188

Table 4.4: Sizes of rational generating functions for various values of m.

which involves an additional symbolic parameter t, can be a hassle. We have managed to compute the rational expressions using evaluation/interpolation techniques for $m = 3, \ldots, 7$. The computation is non-rigorous in so far as the number of evaluation points was only determined experimentally. In Table 4.4, we summarize their sizes. The cases $m = 1$ and $m = 2$ are quite simple, for example, for $m = 2$ the generating function is

$$\begin{aligned}&\frac{-t^4x^4 - t^3x^4 - 2t^3x^3 - t^3x^2 + 4t^2x^2 - tx^2 + 2tx - t + 1}{(t-1)^2(tx^2-1)^2}\\ &= 1 + (x^2 + 2x + 1)t + (x^4 + 4x^3 + 4x^2 + 4x + 1)t^2\\ &\quad + (x^6 + 6x^5 + 6x^4 + 6x^3 + 6x^2 + 6x + 1)t^3 + \cdots\end{aligned}$$

For $m = 3$, the expression is already too big to fit in a line, and as can be seen in the table, the sizes increase significantly with respect to m.

The number of colors affects the growth of the expressions even more significantly. We have considered a variant of the problem where besides black cells and white cells we also have gray cells. The question is then how many ways there are to dissect the $m \times n$ grid into three connected regions, with prescribed areas for each color. The corresponding rational generating function contains three variables: one marking the length n of the board, one marking the area of black cells, and one marking the area of white cells. (There is no need for a variable marking the area of the gray cells because we know that the areas of the three colors must sum to mn.) We were only able to construct the rational generating function for the case $m = 3$. Its numerator has degree 29 in t and degree 32 in x_1 and x_2, it altogether consists of 7939 monomials. The denominator has degree 28 in t and degree 30 in x_1 and x_2, and it consists of 7412 monomials.

Returning to the case of two colors, it remains to discuss the coefficient extraction operator. If $a(x, t)$ is a rational generating function in x and t, viewed as power series in t whose coefficients are polynomials in x, extracting the coefficient of $x^{\alpha n}$ (with $\alpha = m/2$ or $\alpha = m$ depending on whether m is even or odd) from the nth term of the series is the same as extracting the coefficient of x^{-1} from $x^{-1}a(x, t/x^{\alpha})$. This can be done by creative telescoping [19, 9, 13], as follows: using computer algebra, we can compute polynomials

$p_0(t), \ldots, p_r(t)$ which only depend on t as well as a rational function $b(x,t)$ such that

$$p_0(t)\frac{a(x,t/x^\alpha)}{x} + \cdots + p_r(t)\frac{d^r}{dt^r}\frac{a(x,t/x^\alpha)}{x} = \frac{d}{dx}b(x,t).$$

Applying $[x^{-1}]$ on both sides, we get zero on the right, because the derivative of a rational function cannot have a residue. On the left, observe that taking the residue with respect to x commutes with the polynomials $p_i(t)$ and the derivations in t. Therefore the series $a(t) := \frac{1}{2}[x^{-1}]\big(x^{-1}a(x,t/x^\alpha)\big)$ satisfies the differential equation

$$p_0(t)a(t) + \cdots + p_r(t)\frac{d^r}{dt^r}a(t) = 0.$$

In practice, using the second-named author's implementation [12], this approach works nicely for $m = 3$ (A167242), where we obtain a computer proof of Knuth's result mentioned in the introduction, and for $m = 4$ (A167247), where we find a differential equation of order 5 and polynomial coefficients of degree 208. Using guessing [11], we can also find a differential equation of order 2 with polynomial coefficients of degree 96 as well as a polynomial equation of degree 2 with polynomial coefficients of degree 51. The correctness of these guessed equations can be proved by showing that the corresponding differential operators are right factors of the differential operators obtained via creative telescoping, and checking an appropriate number of initial values. The equations, as well as the rational generating functions, are available at `http://www.koutschan.de/data/gerry/`.

For $m \geq 5$, we have not been able to find equations either by creative telescoping or by guessing, although Furstenberg's theorem guarantees their existence. Following Zeilberger's example, the first-named author (M.K.) will therefore offer a donation of €100 to the OEIS in honor of the person who first manages to find a differential equation for some $m \geq 5$, either experimentally or rigorously.

4.5 Acknowledgment

We thank Doron Zeilberger for encouraging us to work on this problem and to write this paper. We are also indebted to the anonymous referee for checking our manuscript very carefully, by even producing an own implementation of our method. This process helped to clarify some inaccuracies and thus greatly improved the readability of the paper. M.K. was supported by the Austrian FWF grant P31571-N32.

Chapter 5

Social Distancing

This chapter is joint work with Doron Zeilberger.

5.1 Introduction

It all started when we came across the delightful article [21] whose motivation had nothing to do with the now fashionable *social distancing*. Their starting point was an $r \times s$ housing development with rs building lots, each with room for a house. They wanted to enumerate the number of ways (out of the total of 2^{rs} possibilities, including building nothing, and building on all the lots) for which

- No house is 'blocked from the sun' (see below).
- You can't build any house on a currently empty lot without violating this condition.

They were not only interested in the number of such configurations, let's call it $T(r,s)$, but among those, the statistical distribution of the number of houses, or equivalently, the density (the number of houses divided by rs), in other words, the *generating function*, or *weight-enumerator*, of such maximal configurations according to the weight $z^{NnmberOfHouses}$.

It turns out that from an abstract point of view, this is nothing but enumerating **maximal** (w.r.t. the number of ones) $r \times s$ $0-1$ matrices that avoid the pattern

1	1	1
	1	

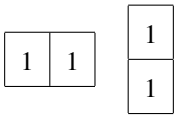
In other words, $0-1$ matrices (a_{ij}) $(1 \leq i \leq r,\, 1 \leq j \leq s)$ where it is forbidden to have a location (i_0, j_0), where

$$a_{i_0,j_0-1} = 1 \quad , \quad a_{i_0,j_0} = 1 \quad , \quad a_{i_0,j_0+1} = 1 \quad , \quad a_{i_0+1,j_0} = 1 \quad ,$$

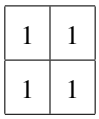
and changing any of the entries that are currently 0 to 1, would create this undesirable pattern.

But why just this particular pattern? The question makes sense for enumerating such $0-1$ matrices avoiding *any* pattern, and in fact, any *set of patterns*.

In particular this question has immediate relevance to *social distancing*. We have a rectangular classroom with r rows, each with s seats, and you can't have two students sitting next to each other in the same row, or having anyone immediately in front of you, or behind you (i.e. you can't have any two students sitting next to each other in the same column). In addition none of the currently empty seats can accommodate a newcomer without breaking this restriction. This is equivalent to the problem of weight-enumerating $r \times s$ **maximal** (with respect to the number of ones) $0-1$ matrices avoiding the patterns

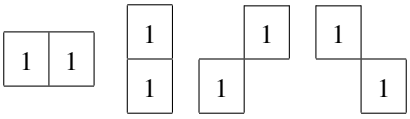


The problem is already interesting for one row (equivalently one column), and one can impose lots of possible restrictions. Going back to two dimensions, one can think of prohibiting the pattern



and the possibilities are endless.

The famous problem of *maximal non-attacking kings* can be formulated as counting (and weight-counting according to the number of 1s) maximal $r \times s$ $0-1$ matrices avoiding the patterns



We will probably never know the exact number of doing it for a 100×100 board, and in general, for any fixed pattern, or set of patterns, the problem of enumerating such maximal $0-1$ **square** matrices, seems intractable. But thanks to the **transfer matrix** method one can efficiently find explicit bi-variate generating functions for such enumeration problems with a **fixed** number of columns, s (not too big, of course), but arbitrary number of rows r (or vice versa). So the problem of enumerating (and weight-enumerating) maximal configurations of non-attacking kings, say on a 1000×5 board can be found exactly.

To see their total number, and the number of these maximal placements with 668 kings (the smallest possible number), see the following output file:

https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDk5Story.txt .

We will now describe how to do it.

5.2 A Finite State Machine for Maximal Seating

Suppose that we have r rows of seats. We wish to compute the sequence giving the number of maximal seating assignments as a function of the number of columns. As input we receive r, the number of rows, and P, the set of patterns describing the shapes to be avoided.

There are 2 ways a seating arrangement can fail.

1. It may fail to meet the social distancing requirement. That is there may be some people sitting in a place prohibited by one of the patterns. Call this a failure of type 1.

2. It may fail to be maximal. That is there may be an empty seat which could be filled without violating any of the patterns. Call this a failure of type 2.

We note that both of these problems are *local*. We don't need to know the entire seating arrangement to determine with certainty that a possible assignment is not valid. This allows us to create a finite state machine with relative ease. We treat a seating arrangement as a sequence of columns. The possible columns will serve as inputs to our state machine. Here, we represent a column as a binary string of 0s and 1s of length r. The 1s correspond to occupied seats, and the 0s correspond to empty seats. There are thus 2^r possible symbols that our state machine must be able to process.

As we read in columns, we must determine whether the constraints have been satisfied *so far*. In order to detect whether the proposed next column causes a violation, we need to store some data about the preceding columns. In this case, we just need to store the contents of some number of previous columns.

Definition 4. For a given pattern, p, define it's width, $w(p)$, to be the largest x-value of any of its lattice coordinates minus the smallest x value. Define the width of our state machine, $W(r, P)$ or just W, to be the greatest width of all the pattern in P.

To check whether the proposed next column causes a violation of type 1, we need to have stored the previous W columns. If we have that information stored, we can just loop through all the patterns and check that each new 1 does not add a violation of that pattern shape.

To check whether the proposed next column causes a violation of type 2, we must be a bit more clever. For each 0 that is added, there must be a group of nearby 1's that ensures it cannot be changed to a 1 without causing a violation. A 0 that has an appropriately shaped group of nearby 1s is called a satisfied 0. Since we

do not know the contents of columns we might read in the future, we are unable to determine whether the 0s in the current column are satisfied. Sure, they may not be satisfied yet, but we should not consider them to be violations yet. If the current column is column j, then we check that the 0s in column $j - W$ are satisfied.

In total, we store the contents of the previous $2W$ columns in the state machine. Thus for column $j - W$ we know exactly the contents of the previous W columns and the next W columns so we can determine with certainty whether the 0s are satisfied.

Since we require a state for each of the possible $2W$ previous columns, this gives a total of a 2^{2Wr} possible states.

Now that we have constructed our set of states, it is simple to determine the transitions. For each state, we have a transition for each possible next column. If the transition causes no violations, it is valid, otherwise the state machine should immediately REJECT. Once a violation has been detected we do not need to look at any more columns.

When we are done reading our input, we must do a little more work to determine whether to ACCEPT or REJECT. Specifically, the 0s in the most recent $W - 1$ columns have not been checked yet, so we must check that each is satisfied before accepting.

We also add some special initialization states for when the previous $2W$ columns do not have values yet. The details of this are implementation specific and omitted.

5.3 Constructing the Transfer Matrix

Once the finite state machine has been constructed, we seek to count the number of input strings of length n that cause the machine to ACCEPT. To do this we use the transfer matrix method. Each state corresponds to a row and column of the matrix, M. Then the i, j entry $M[i][j]$ is 1 if there is a valid transition from state i to state j, and 0 otherwise.

We can count the number of paths from state i to state j of length s by computing $M^s[i][j]$. Thus the number of maximal seating arrangements with s columns is given by an entry of the matrix M^s. We just set i to be the initial state and j to be the final state of our state machine.

5.3.1 Density of Seating arrangements

We can modify the transition matrix by replacing each 1 with a power of z. For a transition corresponding to adding a column with t 1s, the corresponding matrix entry will be z^t. Then $M^s[i][j]$ is a polynomial in z. The coefficient of z^k is the number of seating arrangements with a total of k seats occupied.

5.3.2 Getting the bi-variate generating function

The previous section explained how to compute the transfer matrix, let's call it $M_{\mathcal{P},r}(z)$, for *any* set of patterns $\mathcal{P}$, and any **fixed** number of rows r. Altogether it had, say, d, states, including the two special states, the initial, and the final, that our program labeled 1 and d, respectively.

We are interested, for an *arbitrary* number of columns, s, in the weight-enumerator, let's call it $W_{\mathcal{P},r,s}(z)$, according to the weight $z^{NumberOfOnes}$ of all maximal $r \times s$ $0-1$ matrices avoiding $\mathcal{P}$. Define the bi-variate generating function

$$f_{\mathcal{P},r}(z,x) := \sum_{s=0}^{\infty} W_{\mathcal{P},r,s}(z) x^s \quad .$$

Then we have

$$f_{\mathcal{P},r}(z,x) = (I \, - \, x \, M_{\mathcal{P},r}(z))^{-1}[1,d] \quad .$$

What's nice about computer algebra is that it can all be streamlined.

Using this generating function, we can automatically compute the *limiting average density* (among maximal seatings) for a fixed r and arbitrary $\mathcal{P}$, as s goes to infinity, namely

$$\frac{1}{r} \lim_{s \to \infty} \frac{W'_{\mathcal{P},r,s}(1)}{s \, W_{\mathcal{P},r,s}(1)} \quad .$$

We omit the details, since this is standard residue calculus, and the interested reader can look at the Maple source code of procedure `AsyAv(f,x,z)` in the Maple package `SD.txt`, accompanying this paper.

5.4 Random Sequential Adsorption

The polynomial $W_{\mathcal{P},r,s}(z)$ is the weight enumerator of maximal $0-1$ $r \times s$ matrices that avoid the patterns in $\mathcal{P}$. Hence the coefficient of z^m in that polynomial is the exact number of such matrices with exactly m ones (or equivalently the number of maximal seatings in an r by s classroom obeying the restrictions in $\mathcal{P}$ and having exactly m occupied chairs). It follows that the *probability generating function* (for the uniform distribution), let's call it $V_{\mathcal{P},r,s}(z)$, is:

$$V_{\mathcal{P},r,s}(z) = \frac{W_{\mathcal{P},r,s}(z)}{W_{\mathcal{P},r,s}(1)} \quad ,$$

and the expected number of occupied seats is $\frac{d}{dz} V_{\mathcal{P},r,s}(z)|_{z=1} = V'_{\mathcal{P},r,s}(1)$.

There is a quick way to generate random maximal seatings, described in the special case of T-avoiding seatings in [21], but that makes sense in general, called *Random Sequential Adsorption*, that they abbreviated

to RSA, but since for us RSA means 'Rivest-Shamir-Adleman', we will refrain from using this abbreviation.

In that random-generation process, one picks, *uniformly at random*, a permutation of the rs initial empty seats, representing rs people each having their favorite seat, all distinct from each other. When someone enters the classroom they attempt to occupy their favorite seat, and do so if they do not cause a violation. But if sitting there does create a violation, they are not allowed to sit elsewhere and must leave the room (and cut the class).

This process is very easy to simulate, and it generates a random maximal seating. Alas, it is not the uniform one. Thanks to our closed-form expressions for the probability distribution for $r \times s$ maximal seatings with a fixed (not too big) r but arbitrarily large s, we were able to compare notes, and estimate (via simulations) how close is random sequential adsorption to the uniform distribution.

The Maple package SD.txt

This article is accompanied by a Maple package `SD.txt`, available from

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/SD.txt` .

The front of this article

`https://sites.math.rutgers.edu/~zeilberg/mamarim/mamarimhtml/social.html`

,

contains numerous sample output files, a selection of which we will describe next.

5.5 Avoiding Dimers in $3 \times s$, $4 \times s$, and $5 \times s$, $0-1$ matrices

To see explicit expressions, limiting average densities, and comparison with Random Sequential Adsorption simulation, for the $3 \times s$ case, see the output file

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDd3.txt` .

In particular, the bi-variate generating function for $3 \times s$ maximal dimer-avoiding $0-1$ matrices is:

$$-\frac{\left(2x^5z^4+2x^3z^3+2x^2z^3-6x^2z^2-x\,z^2-xz-z-1\right)xz}{x^5z^4+2x^4z^4-x^4z^3+x^3z^4-4x^3z^3-x^2z^3-xz+1} \quad .$$

For similar information for the $4 \times s$ case, see the output file

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDd4.txt` ,

in particular, the bi-variate generating function for $4 \times s$ maximal dimer-avoiding $0-1$ matrices: is,

$$\frac{\left(x^6z^6-x^5z^6+x^5z^5-2x^5z^4-3x^4z^4+2x^3z^4-7x^3z^3+2x^3z^2-4x^2z^3+7x^2z^2-x\,z^2+4xz+3\right)x\,z^2}{x^6z^6+x^5z^6+x^5z^5+2x^4z^5-x^4z^4+2x^3z^5-4x^3z^4-x^3z^3-2x^2z^3-x^2z^2-x\,z^2+1}$$

For similar information for the $5 \times s$ case, see the output file:

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDd5.txt` .

5.6 Maximal Non-Attacking Kings

The generating function for maximal configurations of non-attacking kings on a $3 \times s$ chessboard is

$$-\frac{\left(x^5z^3+x^5z^2-x^3z^3+x^3z+2x^2z^2+x^2z+x\,z^2-x^2-3xz-2x-z-1\right)xz}{x^6z^4+x^6z^3-x^5z^4-x^5z^3+x^4z^3+x^4z^2+x^3z^3-x^3z^2-x^3z-x^2z^2-x^2z-xz+1} \quad .$$

For more details see: `https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDk3.txt` .

The generating function for maximal configurations of non-attacking kings on a $4 \times s$ chessboard is

$$-\frac{\left(6x^6z^3+9x^5z^2-6x^4z^3+3x^4z^2-3x^3z^2+3x^3z+3x^2z^2+2x^2z-3x^2+3xz-12x-3\right)x\,z^2}{6x^7z^5-6x^6z^5+9x^6z^4-6x^5z^4+3x^4z^4+x^4z^3+3x^3z^3-6x^3z^2-4x^2z^2-xz+1} \quad .$$

For more details see: `https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDk4.txt` .

For the bi-variate generating function of maximal configurations of non-attacking kings on a $5 \times s$ chessboard and more details, and results of simulations, see:

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSDk5.txt` .

See also the output files

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSD22C3.txt` ,

`https://sites.math.rutgers.edu/~zeilberg/tokhniot/oSD22C4.txt` ,

for avoiding a 2×2 block of *ones*.

Readers are welcome to generate more data, and experiment with other patterns (and set of patterns) to their heart's content, using the Maple package `SD.txt`. Enjoy!

5.7 The Uniform vs. the Random Sequential Adsorption Distributions

It seems much harder to get the *exact* probability generating functions for the density (equivalently, the number of 1s) by using Random Sequential Adsorption in general, and we have to resort to (very reliable!) simulations.

But for the very special case of seatings on a line with n seats, and avoiding the single pattern 11, i.e. non adjacent seatings, we can get exact, closed-form expression for **both** the uniform and the random sequential adsorption process.

Let $g_n(z)$ be the enumeration generating function, according to 'number of occupied seats' under the uniform distribution, then as we saw above

$$\sum_{n=0}^{\infty} g_n(z)\, x^n = -\frac{x^2 z + xz + 1}{x^3 z + x^2 z - 1} \quad .$$

Equivalently the sequence $\{g_n(z)\}$ satisfies the **linear** recurrence

$$g_n(z) = z(g_{n-2}(z) + g_{n-3}(z)) \quad ,$$

subject to the initial conditions

$$g_0(z) = 1 \quad , \quad g_1(z) = z \quad , \quad g_2(z) = 2z \quad .$$

To turn it into a *probability generating function*, we have to divide by $g_n(1)$ and define

$$G_n(z) := \frac{g_n(z)}{g_n(1)} \quad .$$

Recall that in random sequential adsorption the input is a permutation π. If $\pi(1) = i$ then the persons that like locations $i-1$ and $i+1$ would not be able to sit, since i is already occupied. There are $(n-1)(n-2)$ ways where they can be placed. The remaining students independently seat in locations $1 \le x \le i-2$ and $i+2 \le x \le n$ and it follows that the probability generating function, let's call it $F_n(z)$ (after dividing by $n!$), satisfies the **non-linear** recurrence

$$F_n(z) = \frac{z}{n}\left(2\, F_{n-2}(z) + \sum_{i=2}^{n-1} F_{i-2}(z) F_{n-i-1}(z)\right) \quad , \quad n \ge 2 \quad ; \quad F_0(z) = 1 \quad , \quad F_1(z) = z \quad .$$

To get the expected number of occupied chairs under the uniform and the sequential adsorption distributions, we compute $G_n'(1)$ and $F_n'(1)$ respectively, or more informatively, the *densities* $G_n'(1)/n$ and $F_n'(1)/n$. They both converge to limits. The former can be found exactly. We have (as already stated above, with less precision, the number is a certain algebraic number)

$$\lim_{n\to\infty} \frac{G_n'(1)}{n} = 0.411495588662645763381900381335531940800608649354765817635803356939840\ldots$$

The other limit we estimated numerically. we have (by taking large n)

$$\lim_{n\to\infty} \frac{F_n'(1)}{n} \approx 0.4323\ldots \quad ,$$

agreeing with the exact value $(1-e^{-2})/2$ stated in Steven Finch's wonderful book [20] (section 5.3.1) and due to several people (see [20] for references).

Hence the average occupation densities, for large n, of the random sequential adsorption process is approximately 1.052 times those of the uniform distribution. Of course, the maximal density, obtained by the regular spacing $101010\ldots$ is 0.5.

Chapter 6

Do It Yourself Guide

All of the projects seen so far are essentially carrying out the same process with slightly different specifics. We now provide some maple code which will serve as a toolkit for anyone else who would like to consider their own variations of this setup. The maple code is HERE. We next describe an outline of the work needed to implement an arbitrary setup.

1. Decide the set of symbols which will be allowed in your rectangular arrangements. For the social distancing project, it was just the set $\{0, 1\}$. Put this inside the gen_ symbols function.

2. Decide the list of states. This should depend on the number of rows, r. Fill out the function gen_ states(r,symbols), returning a list of states, with the initial state as the first entry.

3. Decide which states are accept states. Fill out the function is_ final(s) which takes a state as input and returns true/false.

4. Given two states, decide whether there is a valid transition between them. This is the function valid_ trans(s1,s2,r,symbols). It should return true if there exists a column, c, such that if the state machine is in state s1 and reads c it will transition to s2.

Once that work is done, the function comp_ seq(r,n) will compute the first n terms of the sequence, and the function gen_ fun(r) will compute the corresponding generating function.

6.1 An Example

Suppose we want to count the number of matrices with entries $\in \{0, 1, 2\}$ such any two horizontally or vertically adjacent entries are not equal. The stencil code currently shows this example. The set of symbols

is $\{0, 1, 2\}$ so for gen_ symbols we can return this set.

In this case, we can use the contents of the most recent column as the state, which makes thing very easy. The set of states is thus the set of possible columns. We can use the function gen_ columns(r,symbols) to construct this set, which is included in the stencil code. It includes a column of all -1 for the initial state.

In this case, we will do all the checking in the transition function, and we can let all states be accept states. This is because all errors can be detected immediately on reading the column that contains the error. Put another way, whenever there is a valid arrangement, all partial arrangements obtained by stopping early are also valid arrangements. Thus for the function is_ final(s), we can return true for all s.

It only remains to implement valid_ trans(s1,s2). s1 contains the previous column and s2 contains the proposed next column. Thus we can loop over all entries in the proposed next column and check that they do not introduce any errors. This is shown in the stencil code, and the reader can play with changing it however they like.

If we run the code we can easily compute the sequences for any number of rows. For 3 rows, comp_ seq(3,10) gives the sequence

$$12, 54, 246, 1122, 5118, 23346, 106494, 485778, 2215902, 10107954$$

gen_ fun(3) gives the rational function:

$$\frac{-4x^2 + 7x + 1}{2x^2 - 5x + 1}$$

Bibliography

[1] D. E. Knuth, Baxter matrices, https://cs.stanford.edu/~knuth/papers/baxter-matrices.pdf

[2] Baxter, Glen (1964), On fixed points of the composite of commuting functions, Proceedings of the American Mathematical Society, 15: 851–855, https://doi.org/10.2307/2034894

[3] Giraudo, Samuele (2011), Algebraic and combinatorial structures on Baxter permutations, 23rd International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2011), Discrete Math. Theor. Comput. Sci. Proc., vol. AO, Assoc. Discrete Math. Theor. Comput. Sci., Nancy, pp. 387–398, 1011.4288

[4] Bonichon, Nicolas; Bousquet-Mélou, Mireille; Fusy, Éric (October 2009), Baxter permutations and plane bipolar orientations, Séminaire Lotharingien de Combinatoire, 61A, Art. B61Ah, 29pp, 0805.4180

[5] Korn, M. (2004), Geometric and algebraic properties of polyomino tilings. https://dspace.mit.edu/bitstream/handle/1721.1/16628/56018440-MIT.pdf?sequence=2&isAllowed=y

[6] The On-Line Encyclopedia of Integer Sequences, founded in 1964 by N. J. A. Sloane. http://oeis.org/

[7] Sipser, M. (2006). Introduction to the theory of computation. Boston: Thomson Course Technology.

[8] Kauers, M., Paule, P.: The Concrete Tetrahedron. Symbolic Sums, Recurrence Equations, Generation Functions, Asymptotic Estimates. Springer, Wien/New York (2011)

[9] Frédéric Chyzak. The ABC of creative telescoping. Habilitation à diriger des recherches (HDR), 2014. http://specfun.inria.fr/chyzak/Publications/Chyzak-2014-ACT.pdf.

[10] Harry Furstenberg. Algebraic functions over finite fields. *J. Algebra* **7** (1967), 271–277.

[11] Manuel Kauers. Guessing handbook. Technical Report 09-07, RISC Report Series, Johannes Kepler University, Linz, Austria, 2009. http://www.risc.jku.at/research/combinat/software/Guess/.

[12] Christoph Koutschan. HolonomicFunctions (user's guide). Technical Report 10-01, RISC Report Series, Johannes Kepler University, Linz, Austria, 2010. https://risc.jku.at/sw/holonomicfunctions/.

[13] Christoph Koutschan. Creative telescoping for holonomic functions, in Carsten Schneider and Johannes Blümlein, eds., *Computer Algebra in Quantum Field Theory: Integration, Summation and Special Functions*, Springer, 2013, pp. 171–194.

[14] Hendrik A. Kramers and Gregory H. Wannier. Statistics of the two-dimensional ferromagnet. Part I. *Phys. Rev.* **60** (1941), 252–262.

[15] Hendrik A. Kramers and Gregory H. Wannier. Statistics of the two-dimensional ferromagnet. Part II. *Phys. Rev.* **60** (1941), 263–276.

[16] Donald E. Knuth (proposer) and Editors (solver). Balanced tilings of a rectangle with three rows. Problem 11929. *Amer. Math. Monthly* **125** (2018), 566–568.

[17] Neil J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. An illustrated guide with many unsolved problems. Math 640 Guest Lecture, Rutgers University, April 28, 2022. https://vimeo.com/704569041/4ffa06b95e [talk]; https://sites.math.rutgers.edu/~zeilberg/EM22/C27.pdf [slides].

[18] Richard P. Stanley. *Enumerative Combinatorics, Volume 2*. Cambridge University Press, 1999.

[19] Doron Zeilberger. The method of creative telescoping. *J. of Symbolic Comput.* **11** (1991), 195–204.

[20] Steven R. Finch, *Mathematical Constants* (Encyclopedia of mathematics and its applications, v. 94) , Cambridge University Press, 2003.

[21] Mate Puljiz, Stjepan Sebek, and Josip Zubrinic, *Packing density of combinatorial settlement planning models*, Amer. Math. Monthly **130**(10) (Dec. 2023).

[22] Vince Vatter, *Social Distancing, Primes, and Perrin Numbers*, , Math Horizons, **29**(1) (2022). https://sites.math.rutgers.edu/~zeilberg/akherim/vatter23.pdf .