Intensive use of computing resources for dominations in grids and other combinatorial problems

Alexandre Talon

Numéro National de Thèse : 2019LYSEN079

Thèse de Doctorat de l'Université de Lyon

opérée par

l'École Normale Supérieure de Lyon

École doctorale InfoMaths N° 512
École doctorale en Informatique et Mathématiques de Lyon

Spécialité de doctorat : Informatique

Soutenue publiquement le 29 novembre 2019 par

Alexandre Talon

Intensive use of computing resources for dominations in grids and other combinatorial problems

Utilisation intensive de l'ordinateur : dominations dans les grilles et autres problèmes combinatoires

---

Devant le jury composé de :

Tero Laihonen	Professeur, Université de Turku	Rapporteur
Mathieu Liedloff	Maître de conférences, Université d'Orléans	Rapporteur
Emmanuel Jeandel	Professeur des Universités, Université de Lorraine	Examinateur
Aline Parreau	Chargée de Recherche, ENS de Lyon	Examinatrice
Myriam Preissman	Directrice de Recherche, Université Grenoble Alpes	Examinatrice
Michaël Rao	Chargé de Recherche, ENS de Lyon	Directeur de thèse

Acknowledgements

Here is the place to thank all the people who, directly or not, helped me achieve this PhD. Such help fell into two main categories: science and friendship or social activities.

First, I want to thank Michaël for his guidance and support during all the duration of my PhD. This includes the knowledge he shared with me, as well as the numerous discussions we had about the problems we tackled. I also want to thank him, Guilhem, Nathalie, Silvère, Tero and Mathieu for reading part or the whole of this manuscript, and providing much helpful feedback. I also want to thank Silvère for his involvement in the work we did together, which gave me knowledge about subshifts which are very interesting objects. I thank Karl Dahlke for answering my questions with plenty of details about the work he did on polyominoes. I also want to congratulate him for his project Edbrowse, a text-based editor which was aimed to help blind people access the web. Many thanks to the MALIP team who helped me with administrative tasks, and Alice and Aurore for helping me for the final steps of my PhD.

I now want to thank many other people whom I have met, and who helped me evade from the world of the PhD. First, I want to thank Alice, Alma and Aurore for their very valuable friendship and support throughout these three years. The people I played badminton with, and most specifically Edmond, Juliette, Corentin, Leeroy, Hubert and Selva allowed me to avoid some stress through sport, and we spent very nice moments even outside badminton sessions. Michaël and Laetitia supported me regularly around some beers, helped me rethink part of the world, and we shared a lot of fun moments. Gabrielle also supported me around beers, being for some time in the same adventure as me. To continue in the register of beers, I am very glad I randomly met Alice, with whom I spent lots of great moments, with interesting discussions. I am also grateful to Flore for all the valuable discussions we had, and her integrity of reasoning over various matters, among which climate change. Janelle and Mathieu, with small attentions, helped me a lot at a period when my PhD moments were not so bright. All the messages I received from friends to cheer me up, at the end of my PhD, were very nice and helped me keep up, so thanks to all of you who reached me on this occasion. I also had nice discussions and jokes with the people of the MC2 team.

A lot of other people have been part of my life through several activities. I want to thank the members of ENvertS and the RF for all the projects and things we organised together. I also want to thank the Improfesseurs for all the training sessions and shows we did. Finally, the 'club jeux' (board games club) was a big part of my life during these three years; I am grateful to the support the people attending it provided me, particularly from Jean, Valentine and Henry. I thank my parents who always supported me, and even understood at some point that asking me what I had done during the week was not a good question to raise each week. Finally, I am very happy to have met so many people with whom I enjoyed very good moments. I am glad to have kept contact with older friends too. Even though I cannot put all your names here, I think to all of you while writing this and am happy to know you.

Résumé

Nous cherchons à prouver de nouveaux résultats en théorie des graphes et combinatoire grâce à la vitesse de calcul des ordinateurs, couplée à des algorithmes astucieux. Nous traitons quatre problèmes.

Le théorème des quatre couleurs affirme que toute carte d'un monde où les pays sont connexes peut être coloriée avec 4 couleurs sans que deux pays voisins aient la même couleur. Il a été le premier résultat prouvé en utilisant l'ordinateur, en 1989. Nous souhaitions automatiser encore plus cette preuve. Nous expliquons la preuve et fournissons un programme qui permet de la réétablir, ainsi que d'établir d'autres résultats avec la même méthode. Nous donnons des pistes potentielles pour automatiser la recherche de règles de déchargement.

Nous étudions également les problèmes de domination dans les grilles. Le plus simple est celui de la domination. Il s'agit de mettre des pierres sur certaines cases d'une grille pour que chaque case ait une pierre, ou ait une voisine qui contienne une pierre. Ce problème a été résolu en 2011 en utilisant l'ordinateur pour prouver une formule donnant le nombre minimum de pierres selon la taille de la grille. Nous adaptons avec succès cette méthode pour la première fois pour des variantes de la domination. Nous résolvons partiellement deux autres problèmes et fournissons des bornes inférieures pour ces problèmes pour les grilles de taille arbitraire.

Nous nous sommes aussi penchés sur le dénombrement d'ensembles dominants. Combien y a-t-il d'ensembles dominant une grille donnée ? Nous étudions ce problème de dénombrement pour la domination et trois variantes. Nous prouvons l'existence de taux de croissance asymptotiques pour chacun de ces problèmes. Pour chaque, nous donnons en plus un encadrement de son taux de croissance asymptotique.

Nous étudions enfin les polyominos, et leurs façons de paver des rectangles. Il s'agit d'objets généralisant les formes de Tetris : un ensemble de carrés connexe (« en un seul morceau »). Nous avons attaqué un problème posé en 1989 : existe-t-il un polyomino d'ordre impair ? Il s'agit de trouver un polyomino qui peut paver un rectangle avec un nombre impair de copies, mais ne peut paver de rectangle plus petit. Nous n'avons pas résolu ce problème, mais avons créé un programme pour énumérer les polyominos et essayer de trouver leur ordre, en éliminant ceux ne pouvant pas paver de rectangle. Nous établissons aussi une classification, selon leur ordre, des polyominos de taille au plus 18.

Abstract

Our goal is to prove new results in graph theory and combinatorics thanks to the speed of computers, used with smart algorithms. We tackle four problems.

The four-colour theorem states that any map of a world where all countries are made of one part can be coloured with 4 colours such that no two neighbouring countries have the same colour. It was the first result proved using computers, in 1989. We wished to automatise further this proof. We explain the proof and provide a program which proves it again. It also makes it possible to obtain other results with the same method. We give potential leads to automatise the search for discharging rules.

We also study the problems of domination in grids. The simplest one is the one of domination. It consists in putting a stone on some cells of a grid such that every cell has a stone, or has a neighbour which contains a stone. This problem was solved in 2011 using computers, to prove a formula giving the minimum number of stones needed depending on the dimensions of the grid. We successfully adapt this method for the first time for variants of the domination problem. We solve partially two other problems and give for them lower bounds for grids of arbitrary size.

We also tackled the counting problem for dominating sets. How many dominating sets are there for a given grid? We study this counting problem for the domination and three variants. We prove the existence of asymptotic growths rates for each of these problems. We also give bounds for each of these growth rates.

Finally, we study polyominoes, and the way they can tile rectangles. They are objects which generalise the shapes from Tetris: a connected (of only one part) set of squares. We tried to solve a problem which was set in 1989: is there a polyomino of odd order? It consists in finding a polyomino which can tile a rectangle with an odd number of copies, but cannot tile any smaller rectangle. We did not manage to solve this problem, but we made a program to enumerate polyominoes and try to find their orders, discarding those which cannot tile rectangles. We also give statistics on the orders of polyominoes of size up to 18.

Contents

Introduction

This PhD falls into what is called 'computer science', or 'informatique' in French. This phrase means the science, hence the study, of computers. One may find it strange that computers are not more part of it. In some countries, a large part of computer science is called 'discrete mathematics'. It is considered part of this topic since for instance combinatorics and graph theory consist in studying some discrete objects, which intuitively means objects we can count. Rational numbers are also discrete objects whereas real numbers belong to the field of continuous mathematics: there are much too many of them to be able to count them. However, a certain number of people use computers in their research works in computer science, but only a few proved big results thanks to computers.

In this PhD, we put the computers in the limelight and used them as tools to help us find new results. There are mainly two ways of using a computer to prove a result: using its huge computational power, and using its rigour to validate proofs and certify their correctness. In this thesis we focus on the first means: devising and implementing programs to be run by computers, the result of which are then used to prove a theorem. However, a computer still has finite resources[1] and many problems involve checking an infinite number of objects. This means that theory is needed to reduce this infinity of cases into a finite number of them, even if that number is huge (but not too huge). This part is fundamental, but may turn out not to be enough. A second step may be to think more to reduce again the number of objects the computer will have to examine, thanks to symmetry reasons for instance.

The first big result which was proved by harnessing computers power is the four-colour theorem. It can be formulated as follows: any map in which countries are connected[2] can be coloured using only four colours, and such that any two neighbouring countries have different colours. In computer science, it is stated as 'every planar graph is four colourable'. It was first proved by Appel and Haken in 1976 [1]. There were some doubts on this proof since it contained a flaw, which was quickly fixed by the authors. In 1996, Robertson and al. proved again this theorem in [45] and their proof was somewhat more concise. In fact, both proofs rely on two lists of graphs (forbidden graphs, and discharging rules) and the proof of Robertson and al. contained much fewer forbidden configurations (633 versus 1476) and rules (32 versus 487). The four-colour theorem was proved using clever ideas and the computational power of computers. Actually, the idea of using the discharging method was first suggested by Heesch during the 70s, and he had thought about it since the 60s. He might have, with Durre, proved it earlier than Appel and Haken, but they did not have enough resources to run their program. The introduction of computers and the increase of their speed was crucial for the proof of this theorem.

[1] As does the Earth, see Chapter 3.

[2] In particular, all islands must be a proper country of their own.

Other problems were solved or tackled using the power of computers, like the ones we talk about in this thesis. For instance, Rao recently closed a problem which had remained open for a century about tiling the plane with convex pentagons in [41]: he showed that no other families of convex pentagons than the ones already known could tile the plane. However, the use of computers is not restricted to computer science nor to graph theory. Some people in cryptography or number theory also resort to the raw power of computers: some try to factor RSA integers, and others try to find, for example, which natural integers can be written as $x^3 + y^3 + z^3$ where $x, y, z \in \mathbb{Z}$. We now know[3] the answer for all numbers lesser than or equal to 100 and there are only ten numbers lesser than 1000 for which we do not have the answer yet.

In a completely different domain, here geometry with applications in chemistry, a 400-year-old conjecture made by Kepler was proved in 1998 by Hales and Ferguson in a series of papers. The whole proof and some comments may be found in [26]. This conjecture states that any packing of spheres of equal radius has a density at most around 0.7405. It was proved by examining a finite list of configurations to check the conjecture, despite the infinite number of possible configurations. It was Fejes Tóth who showed that this problem could be reduced to checking only a finite number of configurations, and provided them in 1953. The proof by Hales and Ferguson was very talked about since it closed a 400-year-old problem which was in the list of most important problems Hilbert made. Some people were not convinced of it because it was obtained with the help of a computer, like the proofs of the four-colour theorem some had earlier doubted.

The other use for computers we mentioned above belongs more to the fields of compilation and logic (and even arithmetic). It uses the automation of computers to certify things like correctness of proofs, correctness of programs, or decimals in floating-point arithmetic. Indeed, apart from using the raw power of computers, one can use the automation and the ability of a computer to follow some rules strictly. Some logicians try to certify proofs, with software tools like Coq, a proof assistant/engine. This means that they input each step of a proof in a certain fashion to the proof assistant and the software validates these steps, which certifies that the proof contains no flaws. This was done with the four-colour theorem by Gonthier [22] in 2005. Concerning the Kepler conjecture, the first proof was in fact checked during four years by a team of twelve reviewers, which is quite exceptional. They said they estimated the probability of the proof to be true to be at least 99%, and the Kepler conjecture was widely accepted as a theorem. To remove any remaining doubts, in 2014 the Flyspeck project team, headed by Hales, showed in [27] that their proof of the conjecture was correct, combining two proof assistants to do so. People studying compiler design and/or logic also try to certify some aspects of programs: they prove, with the help of a computer, specific properties a program will have, for instance that never a division by zero would occur, or that the program will always terminate[4]. These so-called certified programs are used in critical real-time embedded systems, as in planes. We recall that this thesis is not about certification or proof assistants.

[3] The answer for the famous number 42 was found very recently.

[4] It may be possible to show that some programs do terminate, however, there is no way to decide if any program given as input always terminate.

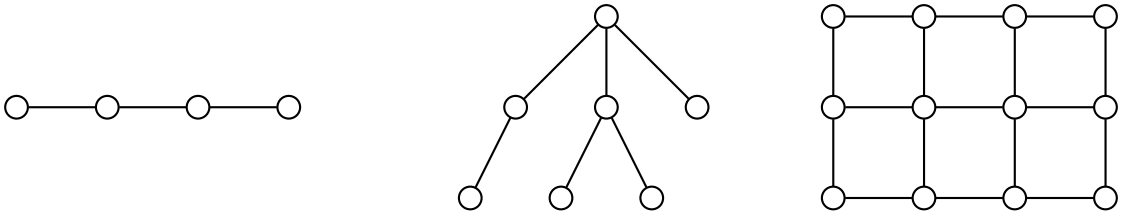


Figure 1: From left to right: a path, a tree and a grid. The vertices are the circles, and the edges are the line segments joining them.

In this thesis we study mostly problems on graphs, and some combinatorics. A graph can be seen as a list of relations between objects (see Figure 1). For instance, any social network would be seen as the set of friendships among its members. The members are called the *vertices* of the graph and the friendships are called the *edges*: they connect pairs of vertices. Graphs can be used to model a lot of problems occurring in real life: when asking for a route for public transport, the website or application of the transport company uses graphs to model its network. Each stop is a vertex, and two successive stops on a same line are connected by an edge, weighted by the average time between the two stops. In case a bus and the underground both have the two stops as successive stops, we may have multiple edges between them. To answer the customer of the public transport who would like to know the fastest route, we have to study the *shortest-path problem*: given two vertices, what is the path of minimal total weight between them? This problem has been much studied in graph theory and we know efficient algorithms to solve it.

However, not all problems on graphs are easy and a lot of people study graphs in more abstract ways, without direct applications. These questions include categorising graphs into classes the elements of which share a lot of properties. Often, the elements of a same class are constructed according to specific patterns and rules. Other questions include studying some problems on graphs, showing that some have a particular property, and designing fast algorithms to solve some problems. Part of the motivation to split graphs into classes is to make it easier to study them. For instance, a certain problem may be very hard on graphs in general, but it may turn out to be easier on some classes of graphs. Usually, paths (a line of vertices connected only to their left and right neighbours) and trees (like a family tree, see Figure 1) are the simplest families of graphs on which to study a problem. Other classes of graphs which have a bounded treewidth, that is 'look like trees', may also be easier to study than general graphs. Another class, which is harder to study because it does not have a bounded treewidth, or even a bounded cliquewidth, is the class of grids: each cell of the grid is a vertex, and it is connected to the four adjacent cells.

The main part of this thesis addresses some problems of *domination* in this class of grids. This problem consists in selecting a set of *dominating* vertices such that any vertex not in this set is connected to a dominating vertex. The goal is to select as few vertices as possible while dominating the graph. The domination problem can model real-life problems such as choosing where to build fire stations and hospitals such that every city is not too far away from one of each type. Multiple variants of this problem have been studied. They are obtained by modifying the condition for a vertex to be dominated. The Roman domination, for instance, looks like a game: we put zero, one or two troops

of soldiers in each vertex, and the set of troops is dominating if any vertex with zero troops has a neighbouring vertex with two troops. Intuitively, a military strategy could consider that a troop would defend the vertex it is placed on and two troops could split into two, one of the two leaving to defend a neighbouring vertex[5]. These domination problems are very hard on general graphs, but we managed to solve some and approximate others in the case of grids. The simplest domination problem was already solved in grids by Gonçalves et al. [21] in 2011. Among the multiple variants of the domination, we study here the domination, minimal-domination, 2-domination, Roman-domination, total-domination, minimal-total-domination and distance-two-domination problems. We introduce the meta-k-domination and the minimal meta-k-domination, which extend the domination and total-domination problems in a certain direction. We already mentioned that the goal is to find, given a graph, the minimum size of a dominating set. We study this optimisation problem for variants of the domination, in Chapter 2. Another problem which we study consists in trying to estimate how many dominating sets there are in a large grid, in Chapter 3. Whereas the minimum-domination problem in grids belongs to graph theory and combinatorics, this problem leads us to the side of combinatorics, a bit outside graph theory.

As we mentioned above, computers are able to operate in an automatic way and very fast. They may try a lot of possible solutions of a problem, provided we program them to do so by giving a set of instructions in the right syntax. A computer program would for instance solve a Sudoku in a matter of seconds, or even less than one if correctly optimised. The same goes for many puzzles which may take us some, or even a lot of time to solve. One other kind is the Rubik's Cube: people usually follow an algorithm to solve it, hence a computer can solve it way faster than any human would.

Some people exploited the fact that even a computer cannot solve a problem if the number of configurations to examine is too big, and if no ways to reduce this number are known. The Eternity game was a puzzle game consisting of 209 irregularly shaped small polygons, and a dodecadron board. The goal was to fit all the 209 pieces together to tile the board, and a prize of one million pounds was promised to whomever would solve it first within four years. A pair of mathematicians made it in 2000 and were paid. A sequel, called Eternity 2, was released in 2007. It was composed of 256 decorated squares: they should be placed on the board so that the decorations match on the edges of two adjacent squares. This problem is strongly related to objects called Wang tiles, which are equivalent to the concept of SFTs we discuss in this thesis. A prize of two million dollars was offered if someone could solve it within four years, but no one achieved it.

We tackle another 'geometric' problem, similar to the first Eternity game, which also belongs to the field of combinatorics. It involves polyominoes, which are generalisations of Tetris pieces: they may contain fewer or more than four squares. We are interested in a more complex puzzle: given a polyomino and as many copies of it as you want, try to tile a rectangle with these pieces. The rectangle is not given, it is up to you to find if there exists one which is tilable by the polyomino or if the polyomino cannot tile any rectangle. Also, if it can tile a rectangle, please find one which requires as few copies of the polyomino as possible as in Figure 2. It would be awesome if this minimum number

[5]provided the enemy does not attack too many vertices at once!

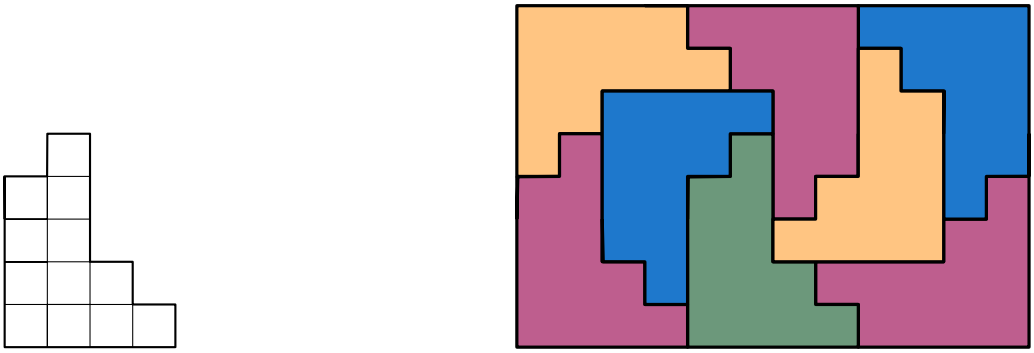


Figure 2: A polyomino of size 12 and order 8 and a tiling of the associated rectangle.

of copies could be an odd integer greater than one. Unfortunately, we still do not know if this is possible. Here, contrarily to the games we mentioned, the number of configurations is not finite, for there are infinitely many rectangles we could try. Even trying to tile a single rectangle with about a hundred copies of a polyomino may require a huge amount of configurations to try. Fortunately we may use clever arguments to reduce this number, which makes it possible to find the order of some polyominoes, but not for every one we tested.

To tackle any problem with the help of a computer, a certain methodology must be followed. The power of computers is huge, all the more so that we can use parallelism and we have increasingly powerful machines. However, there is still a finite and fixed number of operations doable in a certain amount of time by a given machine. This means that if the problem we want to tackle implies examining an infinite number of objects, the first step is to reduce it to a finite number, thanks to theoretical arguments. The second step is to consider the angle of attack: what type of algorithms should we use? Also, what are the optimal data structures we need? Once this is done, it may be useful to estimate the running time and memory consumption of the program we are devising, if this is possible. If it exceeds the available resources, more care must be given to reducing the number of objects it is to examine. One classical way to do so is to observe the symmetries of the objects we study and try to reduce their numbers according to this fact. Indeed, if the objects of a group are equivalent, we may as well examine thoroughly one of them and detect that the others do not need to be examined. Another way to use fewer resources can be to think more about the algorithm and data structures, and find better ones. In case the bottleneck is only time, we may do some precomputations and store intermediate results to be reused. This trades some memory for a smaller running time. If on the contrary the amount of memory used is problematic, we may store fewer results and compute them again several times, to benefit from the opposite trade-off.

Nowadays the frequencies of computers no longer increase, or by very little. The way to gather more computing resources is to resort to parallelism. This can usually be done by launching a lot of instances of the same program, at the same time, with different inputs. Another way is to use threads: a program splits itself into several children which run the same code. We split the amount of computations between the threads so that if we have k threads, each would do one k^{th} of the computations. To benefit fully from parallelising the code, we must think carefully of how to make parts of the computations

independent from one another. Indeed, the speed up will be much higher if the parallel computations do not write to the same locations in memory and do not need to wait for results of another computation to start. One problem is that when using threads, we may tolerate simultaneous accesses to a same location of the memory provided none is writing to it. If one of the simultaneous accesses is a write, the behaviour of the program is undefined: if some thread reads this location 'at the same time', should the value read be the new one or the old one? In the case of two writes, which value should be the one to be stored at the end?

Another hint to achieve better performances is to choose the right programming languages and libraries[6]. Interpreted languages such as Python may be very elegant and have a nice syntax, but they can be as much as 100 times as slow as C/C++. When the critical operations are classical ones, like matrices products, it may be better to use a library written for that purpose, which will be faster. Using libraries also make it less likely to find bugs in the final program.

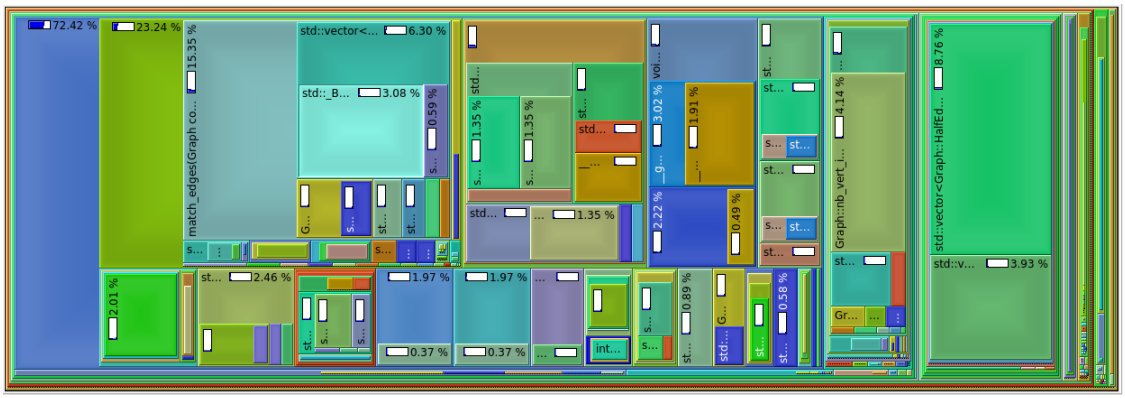


Figure 3: A screenshot of qcachegrind displaying visually some logs of the profiler callgrind. It shows each function as a rectangle whose size is proportional to the time spent in the function.

Finally, there are some good practices to adopt when programming something rather big. They may take a bit of time and some energy, but they can save a huge amount of time and frustration later on. The first thing is to always write some simple code first, before optimising it. It may be sufficient even if not optimal, and if we need to optimise it we will have a reference to test the new code against. The second thing is then to have tests, if possible automatic ones. For instance, running the first simple code on small inputs can generate test cases for the subsequent versions. Third thing, **do not overoptimise, or optimise too early**: there is no use optimising some parts (small rectangles in Figure 3) of the code which take only one or two percent of the running time. It is only making the code harder to read, maintain, and more prone to bugs. The objective is to find a good balance between a simple code and a fast one. To see which parts of the code are important to optimise we may use tools called *profilers*: they tell us how many times each function was called, and what portion of the running time these calls took. We used a lot the callgrind software in this PhD (see Figure 3), with the

[6]Sorts of programs made by other people, which contain routines to do specific tasks.

visualiser qcachegrind. When bugs occur or are suspected, it is better to use a *debugger*: it enables us to stop at specific points of the program (or at the point where it crashes) and have some information about the current values of the variables, the position in the code, which functions were currently being called, and so on. We used GDB for this purpose. For similar goals, we also used valgrind, a piece of software which performs further checks to detect some possible memory errors or misuse. All these low-level tools, though, are to be used in the last resort: they do not replace a careful design and implementation of the algorithms.

Organisation of the manuscript

This thesis deals with four topics: the proof of the four-colour theorem, several dominations numbers of grid graphs, counting some dominating sets in grids, and a tiling problem with polyominoes. The second and third topics are very related: they study similar problems. The last problem is to some extent related to the second topic, as we will see later. All these four topics share the approach to solve them, which resorts at some point to the use of computer programs. This thesis is divided into four chapters, one dealing with each of the problems we have just listed.

Chapter 1 is about the four-colour theorem and its proof in [45]. The theorem states that any finite planar graph can be coloured properly using only four colours. We first explain the proof from Robertson et al. It works by contradiction: it assumes the existence of a minimal counter-example to the theorem. The minimality forbids some graphs to appear in this counter-example: we generate a list of *forbidden (or reducible) configurations*. The reason the proof works is that no planar graphs can avoid all these reducible configurations. This is shown with the *discharging method*, which consists in assigning weights to the vertices and moving them according to a set of *rules*. This rules are used in an exhaustive search for the hypothetical minimal counter-example. They make it possible to cut all the branches of explorations. Finally, only a finite set of graphs are examined and none can lead to a minimal counter-example, hence the theorem is true. We provide an interface for a use with Python, so that the community may prove some other results with the discharging method, without having to implement everything from scratch. It may later be ported to Sage. We describe some aspects of the program in this thesis.

Chapter 2 tackles variants of the domination number, i.e. the minimum size of a dominating set, in grid graphs. The domination number problem in grids was solved in 2011 by Gonçalves et al. [21]. We reuse for the first time their method and apply it to the 2-domination, the Roman domination, the total domination and the distance-2-domination. We explain there the method, and give the results we obtained using a new program we wrote. We solve the 2-domination and the Roman domination, that is we provide closed formulas giving the 2-domination number and the Roman-domination number according to the height and the width of the grid. We give, for the distance-2-domination and the total domination, the numbers for up to 15 lines and an arbitrary number of columns. For the total loss, this confirms the results of Crevals and Ostergård [9], who went up to 28 lines. We also give, for the total domination, a lower bound provided by our program. We also try to explain why our method does not seem to be able to give a full result for

the total domination, which we relate to some covering problems. Most of the results of this chapter are published in [42].

The dominations problems can be viewed from another angle: trying to count the number of different dominating sets of a specific graph. In Chapter 3, we attack this problem in grid graphs for several domination variants: the domination, the total domination, and their minimal variants. Using the notion of *subshifts* and some known results about them, we show that the number of dominating sets, for each of these four problems, admits a growth rate which is furthermore *computable*. We show this property by analogy with the *entropy* of the domination subshifts, which we show are *block gluing*. The number of dominating sets in grids is showed to be $\nu^{nm+o(nm)}$ for some constant ν depending on the problem studied. For each of these constants, we give numerical bounds obtained by our program. We also introduce a new domination family, which generalises the domination and total domination problems: the *meta-k-domination* family of problems. We also study one particular property of the subshifts associated to this family. The work of this chapter is a joint work of Silvère Gangloff and myself [17].

In Chapter 4, we study a problem related to the covering of rectangles we mentioned for Chapter 2. We study polyominoes and the way they can or cannot tile some rectangle. We are interested in *rectifiable* polyominoes: when there exists a rectangle the polyomino can tile. The question we attacked, and has been open for 30 years, is the following: is there a polyomino of *odd order* greater than one? This means we look for a polyomino which can tile a certain rectangle with an odd number $k > 1$ of copies and which cannot tile any rectangle with fewer copies. We tried to find such a polyomino, unfortunately with no success. We describe several ways to find the order of a polyomino. Apart from these algorithms, we also describe some methods to show, again with the help of a computer program, that a polyomino is not rectifiable. Some ideas come from the work of Karl Dahlke [11], which we programmed anew, along with some of our own optimisations.

To access the source codes of the programs I developed and used during my PhD, see the version of this thesis stored on arxiv; it can be downloaded as a tar archive at: `https://arxiv.org/e-print/2002.11615`. In total, the three different programs account for around 16 000 lines of code.

Chapter 1

Discharging and the four colours theorem

One thing which strongly characterises human beings is their relation to waste. No other species produce that much waste and then do not care about it. This waste can take many shapes: from the plastic of our packagings, to some residues of the cleaning product we use (or other products used by the industry), buildings and machines left beside when no longer used... and obviously nuclear waste, which can last for 100 000 years. But more shockingly is how we process this waste: some of it is simply discharged at the middle of some nature, illegally[1]. Some nuclear waste was just dumped into water: between 1946 and 1993, before it was agreed to stop doing this[2] France alone has put around 15 000 tons of nuclear wast into the sea and the world.

As we said in the introduction, the four-colour theorem is very famous because it is the first big problem which was solved with a computer.

Theorem 1.1 ([1])**.** *Every planar graph is four colourable.*

We will explain each term in Section 1.1, so for the moment we consider an equivalent version.

Theorem 1.2. *Every planar map can be coloured with four colours.*

Le us assume that we draw a finite number of lines on a sheet of papers, the lines delimiting regions of the sheet. The theorem means that we can colour the regions using only four colours and such that any two adjacent regions (sharing an edge[3]) have different colours. For instance, it implies that it is not possible for five regions to be pairwise adjacent, for they could not be coloured with only four colours in that case. The theorem is however stronger than this fact.

The conjecture was apparently first proposed when, in 1852, Francis Guthrie was colouring the counties of England and noticed that four colours sufficed in this task to

[1]and the authorities do not seem to put the necessary means to stop it, even when these places are noticed

[2]Or maybe not, TEPCO company plans on putting 777 000 tons of contaminated water in the sea following the Fukushima accident.

[3]non reduced to a point

guarantee that two neighbouring counties are given different colours. A lot of people tackled this problem with little or no success until 1976. The first big attempt was made by Kempe [32] in 1879. It was welcomed by his peers until, eleven years later, Heawood [28] showed it was false. Similarly, a claim made by P. Guthrie Tait in 1880 was only disproved in 1891 by Petersen. However, Heawood reused one argument in Kempe's proof, the notions of *Kempe chain* and *Kempe interchange* to prove a weakened version of the four-colour theorem. These notions turned out to be important in later proofs.

Theorem 1.3 ([28])**.** *Every planar graph is five colourable.*

During the 20$^{\text{th}}$ century, much progress was made and the theorem was eventually proved. In 1913, Birkhoff [3] formalised the notion of *reducible configurations*, i.e. subgraphs which cannot appear in a minimal counter-example. He notably showed that the theorem holds for graphs with fewer than 26 vertices. Beginning in the 60's, Heesch worked on the problem and introduced a crucial step towards the resolution of the problem: the *discharging method*. Much excitement went about the problem, and it was thought solved by Shimamoto. He showed that the whole problem could be reduced to checking some property on a particular graph, the 'horseshoe'. Heesch announced that it was one of the graphs for which they had shown this property, called *D-recucibility*, but it turned out that it was an incorrect result due to a flaw in the program he had devised with Dürre. Finally, Appel and Haken [1] proved it in 1976, using the reducibility and discharging methods. A small flaw in the program they used cast some doubts on their proof. This proof was criticised by some sceptical people. The program was quickly fixed by their authors, but this did not end the doubts on their proof. This proof relied on some approximatively 2000 (later reduced to 1476) *reducible configurations* which had to be checked by hand by the authors and Haken's daughter, Dorothea Blostein. It also relied on 487 *discharging rules*. In 1996, Robserton et al. [45] gave another proof, based on the same principles, but needing much fewer configurations (633) which were checked by a computer program, and also much fewer discharging rules (32). The same authors announced in 1999 an alternate proof, by proving the 'snark theorem' which implied the four-colour theorem. The proof is still not completely published. All doubts on the four-colour theorem were disappeared in 2005 when Gonthier [22], using the Coq proof assistant, made a certified proof of the theorem.

We begin by giving general definitions about graphs, including planarity, and colourings, in Section 1.1. Section 1.2 is about the proof of the theorem by Robertson et al. We first give a one-page sketch of the main ideas of their proof. After this, we consider the false proof of Kempe, which introduced a central concept named the *Kempe chains*. The rest of this section is devoted to the main two ingredient of the proofs of the four-colour theorem: *reducible configurations* and *discharging rules*. The former consist in subgraphs which cannot appear in a minimal counter-example, and the latter help us realise that no planar graph can avoid all the reducible configurations. The last section is about the work we did in this topic. It contains some details of the program we developed to reproduce the proof of Robertson et al., and some ideas on how to further automate the proof.

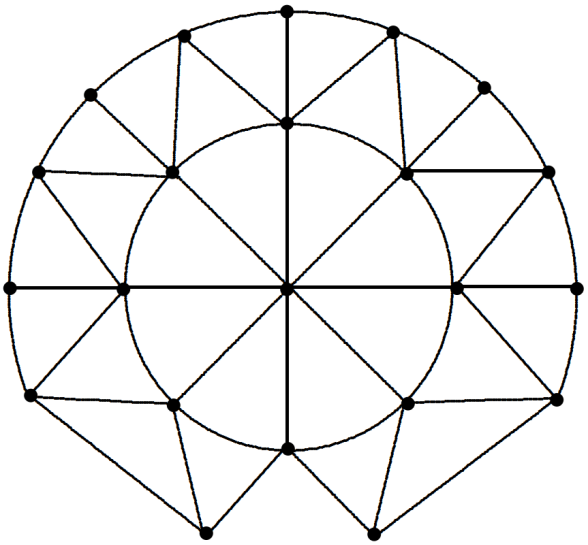


Figure 1.1: Shimamoto's 'horseshoe' graph.
The figure comes from [7].

1.1 Graph definitions

In this chapter, Chapter 2 and Chapter 3 we talk about graphs. We define here what they are, some vocabulary related to them, some of which is specific to this chapter.

Definition 1.1. A **graph** is an ordered pair $G = (V, E)$: $V \subset \mathbb{N}$ is the set of **vertices** and $E \subset \{\{u, v\} \,|\, u, v \in V,\ u \neq v\}$ is the set of **edges**.
If V is finite, then the graph is **finite**.

Remark 1.1. Our definition is the one of *simple* graphs: it forbids loops (an edge between a vertex and itself) and parallel edges (multiple edges having the same endpoints). In all this thesis we only deal with simple graphs.

Definition 1.2. If $\{u, v\} \in E$ we say that u and v are **neighbours**, or **connected**.
The number of neighbours of a vertex u is called the **degree** of u and denoted by $\mathrm{d}^\circ(u)$.

Definition 1.3. We define the **open neighbourhood**, or **neighbourhood** of a vertex u, and denoted by $N(u)$ as the set of neighbours of u. The **closed neighbourhood** of u is $N[u] = N(u) \cup \{u\}$.

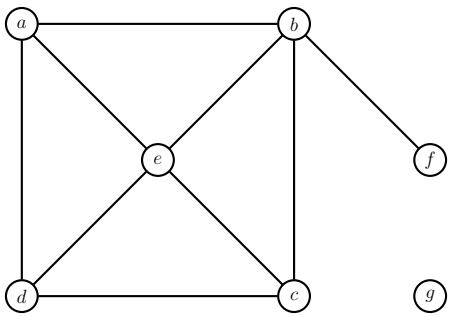


Figure 1.2: An example of a graph.

Figure 1.2 illustrates a graph with 7 vertices (a to g) and 9 edges. a and e are connected, but c and a are not. e has degree 4, f has degree 1 and g is not connected to any vertex: its degree is 0. The neighbourhood of b is $\{a, e, f\}$ and its closed neighbourhood is $\{a, b, e, f\}$.

Definition 1.4. An **embedding** of a graph G into the plane is a representation of G on $\mathbb{R}^2$: vertices are given coordinates and every edge $\{u, v\}$ is drawn as a connected arc whose endpoints are the points assigned to u and v.

Usually when we speak about graphs, in addition to giving V and E we also give embeddings of them, i.e. we draw a representation of them.

Definition 1.5. A graph is said to be **planar** when it can be embedded into the plane such that no two edges cross.

This means that there is a way to draw the graph on the plane such that no pairs of edges cross. It does not mean, of course, that any embedding of the graph would respect this property.

Definition 1.6 (Figure 1.3)**.** Any embedding of a planar graph the edges of which only intersect at the vertices is called a **plane graph**.

Remark 1.2. Any planar graph can be embedded on a sphere such that edges only intersect at the vertices.

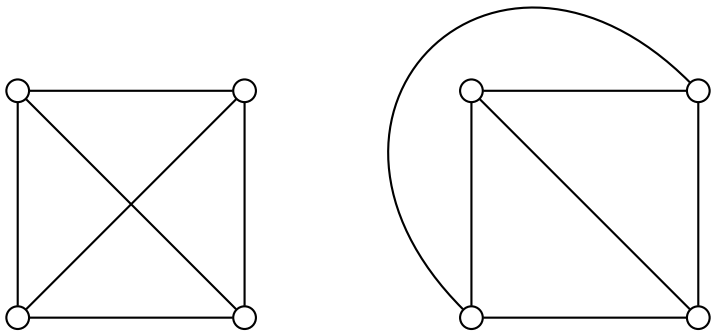


Figure 1.3: Two embeddings of the same planar graph (K_4). Only the one on the right is a plane embedding.

In this chapter we focus on planar graphs, and we reason in fact on plane embeddings of them.

Definition 1.7. A **face** of a plane graph is a region of the map delimited by vertices and edges, and which contains no vertices and no edges except the ones of its boundary.
A **triangle** is a face delimited by three vertices.

In Figure 1.3, the plane graph has 4 faces, all of them being triangles.

Definition 1.8. We say that a plane graph is **triangulated** when all its faces are triangles. It is **almost-triangulated** when all its faces except at most one are triangles. If there is one, the non-triangular face is then called the **outer face**.

Definition 1.9 (see Figure 1.4)**.** $c : V \mapsto [\![1, k]\!]$ is called a ***k*-colouring** of $G = (V, E)$.
A k-colouring c is **proper** when for any $\{u, v\} \in E$ $c(u) \neq c(v)$.
A graph is ***k* colourable** when it admits a proper k-colouring.

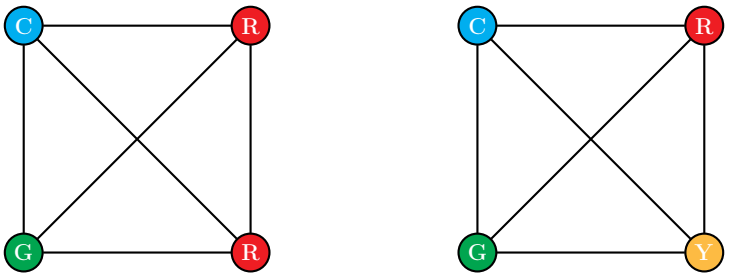


Figure 1.4: Two colourings of K_4. Only the one on the right is proper: the two vertices coloured in red on the left are connected.

A proper colouring means that the vertices of a graph are given colours among a set of k colours, such that any two connected vertices receive different colours. The idea of giving different colours to connected vertices arises from practical problems. In the case of colouring a map, it makes the map clearer by making the frontiers more visible: since adjacent countries have different colours we may not think that they are the same big country. We can also model a room assignment problem with graph colourings. Suppose we have n different lessons to be given and we fixed the schedule of the lessons but would like to know how many rooms we need. We can think of the lessons as vertices, and two vertices are connected if the lessons cannot be put in the same room because their schedules overlap. Each colour we assign to vertices will represent a room, different colours representing different rooms. Then any proper colouring of the graph with k colours gives an assignment of the lessons to k different rooms. The minimum number of colours needed for a proper colouring gives the minimum number of rooms needed.

1.2 The proof

1.2.1 Scheme of the proof

We give here a five-minute overview of the proof: we abstracted it very much skipping most details and above all technical ones. We boiled down the proof into nine notions, some very short, other more detailed.

The first idea is that we make a *proof by contradiction*: we assume that there is indeed a planar graph which is not 4 colourable. Let us call this guy G_0. Actually, we will reason on plane graphs (see Definition 1.6): a planar graph may admit several non-isomorphic plane embeddings. The second idea is to reduce the number of candidates for G_0: to show that it cannot exist, we must find contradictory properties it has. So we first assume that we look for a *critical* counter-example: G_0 must be 5 colourable. We also restrict ourselves to minimal counter-examples: G_0 is chosen such that no graphs smaller than G_0 can be counter-examples. Idea three is then to use the previous conditions to

show properties our graph has, to restrict the number of candidates for G_0. For instance, we can show that G_0 must be triangulated (see Lemma 1.4) and have no vertex of degree less than five. The fourth ingredient is the notion of *reducible configurations*: they are graphs which cannot appear in G_0. There are several reasons for a configuration to be reducible, and they can be tested by a computer. The goal here is to obtain a list of reducible configurations 'big enough'. Now, at step five, we want to show that no plane graphs can avoid all the reducible configurations we found. To do this, we use the discharging technique. First we decide on a way to assign weights to the vertices of G_0, such that the sum of the weights over any plane graph is 120. We then design *discharging rules*: each time some specific configuration[4] appear in a weighted graph G, we transfer some weight from some special vertex to another special one, depending on the rule. At the sixth step, we know that any application of a rule preserves the total weight of the graph. Hence after applying our set of rules to G_0, the sum of the weights is still 120. Therefore, some vertex v_0 has a positive weight after applying the rules. We want to show that some reducible configuration necessarily appears in the second neighbourhood of v_0. The seventh ingredient is to enumerate all possible neighbourhoods for v_0 with the help of a computer. We start with v_0 and use a branch-and-bound algorithm to generate these neighbourhoods. At each step of the algorithm we precise a little more our neighbourhood, or we extend it. Before going further, at each step we test our current neighbourhood. If it cannot be triangulated, or if we can see that any extension of this neighbourhood will necessarily contain a reducible configuration or forces v_0 to have a non-positive weight after applying our set of rules, we discard it and backtrack. The eighth thing to do is to find a set of discharging rules which is good enough. By this we mean a set which, together with the list of reducible configurations, leads to discarding all possible neighbourhoods and show that G_0 does not exist. Robertson et al. found, by trial and error, a sufficient set of discharging rules. Beginning with only a few rules, the exploration program did not seem to finish. By looking at some neighbourhoods which could not be discarded, they designed new discharging rules. A new run of the program led to other new rules, and so on... until at some point the program terminated, proving the theorem.

1.2.2 An interesting wrong proof

We give here a proof which turned out to be false and could not be fixed. However it introduces an important notion in the colouring of planar graphs: the one of *Kempe chains*. It was published in 1879 by Kempe [32]... and the flaw was discovered by Heawood [28] 11 years later! The proof uses some lemmas which are true and reused in the proof by Thomas et al., in particular the concept of *Kempe chains*.

We us assume we colour our vertices with four colours: α, β, γ and δ.

Lemma 1.4. *Any minimal counter-example to the four-colour theorem is triangulated.*

Proof. Let assume $G_0 = (V_0, E_0)$ is a minimal counter-example for the four-colour theorem and has a face with at least 4 vertices. Then let u and v be two non-adjacent vertices of this face. We create G'_0 to be the graph where u and v are identified: $G'_0 = (V'_0, E'_0)$ with $V'_0 = V \backslash \{u, v\} \cup \{uv\}$ and $E'_0 = E_0 \backslash \{\{a, b\} \,|\, a \in \{u, v\}\} \cup \{uv, b\} \,|\, \{u, v\} \in E_0 \text{ or } \{v, b\} \in E_0\}$.

[4] not the reducible ones, some other ones

Now any colouring c' of G'_0 extends itself to a colouring c of G_0 by giving to u and v the colour uv has in c'. Indeed, identifying u and v preserved the vertices connected to u and v: any of their neighbours is neighbour of uv. This works because u and v were not neighbours in G_0. This implies that the set of colourings of G'_0 is included in the one for G_0. Since G_0 admits no four-colourings, the same applies to G'_0. Therefore G'_0, which is still planar, is a smaller counter-example, which is a contradiction. □

Lemma 1.5. *Any triangulated planar graph has a vertex of degree less than 6.*

Proof. We use Euler's formula for planar graphs: $n - m + f = 2$, where n is the number of vertices, m the number of edges and f the number of faces. Besides, since all faces are triangles, the number of edges satisfies $f = 2m/3$: each edge belongs to two faces and the sum of edges over the faces, which are triangles, is $3f$. Euler's formula can then be rewritten to $3n - m = 6$ or $m = 3n - 6$. Summing the degrees of the vertices equals $2m$: each edge has two incident vertices. The average degree of the (finite) graph is $2m/n$ so here it is $2m/n = (6n - 12)/n < 6$. Since the average degree is lesser than 6, at least one vertex must have a degree at most equal to 5. □

Now we want to show that no counter-examples to the four-colour theorem exist. To show this, Kempe used strong induction: we look for a minimal counter-example G_0. By Lemma 1.4, we may know that G_0 has a maximum number of edges, namely that it is triangulated. We proceed by contradiction, assuming that such a counter-example G_0 exists. We proceed by disjunction of cases since G_0 has a vertex of degree $d < 6$:

First case: $d < 4$**.**
In this case let u be a vertex of degree $d < 4$. We remove u and its edges, so the remaining graph must be four colourable by the minimality hypothesis. Let us consider the colours its $d < 4$ neighbours receive. There is at least one colour which is not given to them (since we have four colours available). We can extend the four-colouring of $G_0 \setminus u$ to u by colouring it with the remaining colour, hence our graph G_0 is four colourable, which is a contradiction.

Note that the graph obtained by removing u might no longer be triangulated. However this is not a problem: $G_0 \setminus u$ is still smaller than G_0.

Second case: $d = 4$**.** (see Figure 1.5)
We need here to introduce the very useful concept Kempe introduced in his proof.

Definition 1.10. Let $G = (V, E)$ be a four colourable graph and $c : V \to \{1, 2, 3, 4\}$ a four colouring of it. Let $G_{\{\alpha,\beta\}}$, more concisely written $G_{\alpha,\beta}$, be the subgraph of G whose vertices are coloured with colour α or β. For any pair of $\alpha \neq \beta \in \{1, 2, 3, 4\}$, any maximal connected component of $G_{\alpha,\beta}$ is called a **Kempe chain**, or more precisely an ***αβ*-chain** of c.

Definition 1.11. Given G, c and a pair $\alpha \neq \beta$ a **Kempe interchange** with respect to the colour partition $\{\{\alpha, \beta\}, \{\delta, \gamma\}\}$ is the colouring c' obtained by switching colours α and β in one of the $\alpha\beta$-chains, or by switching the colours γ and δ in one of the $\gamma\delta$-chains.

One important and useful fact about Kempe chains is the following:

Fact 1.1. *Performing a Kempe interchange on a proper four-colouring c always yields a proper four-colouring c'.*

Indeed, by definition of a Kempe chain, we are switching two colours of a maximal component of vertices which had these colours. This means that if we exchanged colours α and β, any vertex which had its colour changed did not have any neighbour coloured α or β outside its $\alpha\beta$-chain.

Remark 1.3. Performing a Kempe interchange along an $\alpha\beta$-chain may create or remove $\alpha\gamma$-chains, $\alpha\delta$-chains, $\beta\gamma$-chains and $\beta\delta$-chains. However such a Kempe interchange does not modify the $\alpha\beta$-chains and $\gamma\delta$-chains.

Now let us consider a vertex u of degree 4. We remove it from the graph, and colour the resulting (smaller) graph with four colours. If two of u's neighbours share the same colour, then we use one spare colour for u. Otherwise, the neighbours have four different colours, let us say Orange, Cyan, Pink and Red. Let us call v_O, v_C, v_P and v_R the neighbours of u labelled with these colours. We assume that in clockwise direction, the neighbours of u are v_O, v_C, v_P and v_R. We take for instance v_O and the OP-chain G_{OP} which contains v_O. If v_P does not belong to this subgraph, then we perform a Kempe interchange along this chain. This frees colour O because v_O gets colour pink. Hence we can colour u with orange. On the contrary, if v_P belongs to the OP-chain containing v_O then we consider the RC-chain G_{RC} containing v_R. v_C cannot belong to this chain because the chain G_{OP} acts as a barrier: it contains only orange and pink vertices, hence no red or cyan vertex of G_{RC}. Since the graph is plane, no edges can cross G_{OP}. This means that we can perform a Kempe interchange along G_{RC}. v_R becomes cyan and we can colour u in red.

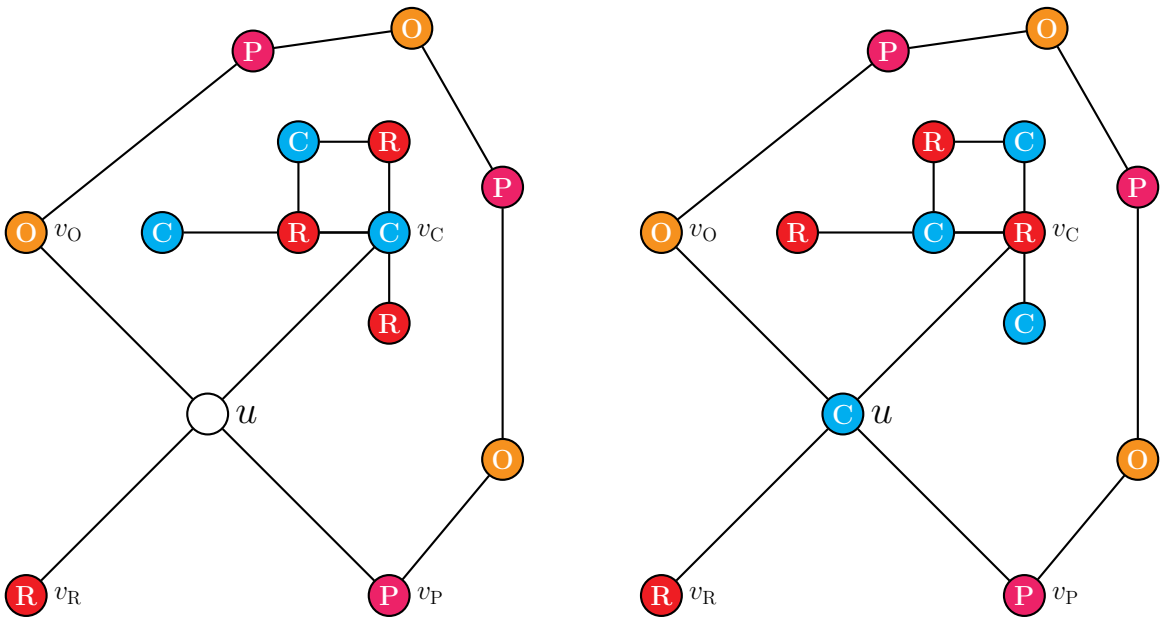


Figure 1.5: Illustration of the Kempe interchange for the case $d = 4$ in Kempe's proof. The PO-chain acts as a barrier: it guarantees that the CR-chain containing v_C cannot contain v_R. On the right we have performed a CR-interchange so that the colour cyan is free for u.

Third case: $d = 5$.

This last case is the most difficult and uses the ideas of the previous case. As usual, let u be a vertex of degree 5. We four colour the graph $G_0 \setminus \{u\}$. If the neighbours of u are coloured with fewer than four different colours, we use a spare one for u. If this is not the case, two vertices have the same colour, say pink, and each of the other three has a colour from the remaining three ones. There are two cases: either the two pink vertices are adjacent when we list the neighbours of u in clockwise order, or they are not. The first case cannot occur since G_0 is triangulated: two consecutive neighbours of u are connected themselves, hence they cannot receive the same colour in G nor in $G \setminus u$.

So we deal with the case when the two pink vertices are not adjacent. We may assume that the vertices, listed in clockwise directions are (their names reflect their colours): v_{P}, v_{O}, v'_{P}, v_{R}, and v_{C}. We consider the OC-chain G_{OC} which contains v_{O}. If it does not contain v_{C} then for the same reason as in the case $d = 4$ we are done. So we assume G_{OC} contains v_{C}. We then consider the OR-chain containing v_{O}. We again assume that we are in the worst case: $v_{\mathrm{R}} \in G_{\mathrm{OR}}$. We now consider two new Kempe chains: the PR-chain G_{PR} which contains v_{P} and the PC-chain G_{PC} which contains v'_{P}. Like before, G_{OR} and G_{OC} act as barriers. Therefore, using the same arguments we conclude that both $v_{\mathrm{C}} \notin G_{\mathrm{PC}}$ and $v_{\mathrm{R}} \notin G_{\mathrm{PR}}$. We can then perform a Kempe interchange on G_{PR} and one on G_{PC} such that v_{P} becomes red and v'_{P} becomes cyan. We finish by colouring u with pink.[5]

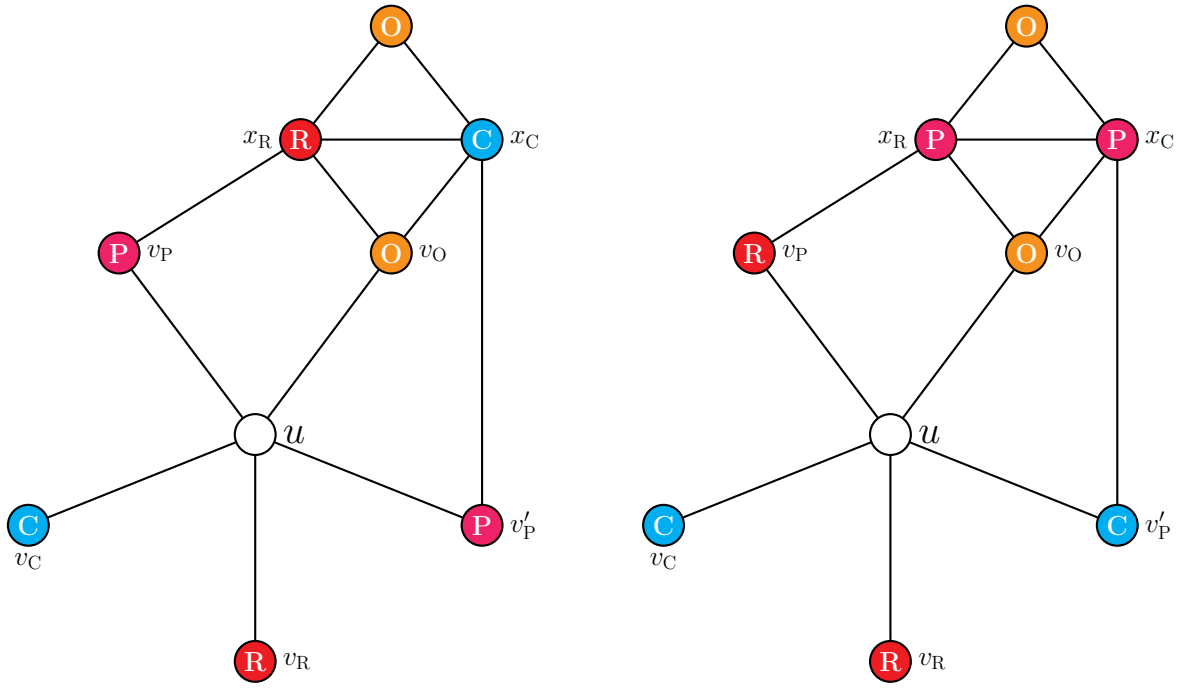


Figure 1.6: Illustration of the flaw in Kempe's proof for the case $d = 5$. When performing the two interchanged mentioned in the proof, x_{R} and x_{C} both receive the colour pink, hence the colouring is not proper.

A fourth case for the prosecution. (see Figure 1.6)

There is subtle flaw in this proof, which you may have overlooked. Indeed, v_{O} may have

[5] QED

one neighbour x_R coloured in red and x_C coloured in cyan. Let us assume that x_R and x_C are connected. It is moreover possible that x_C belongs to G_{PC} and that x_R belongs to G_{PR}. In this case, after performing the Kempe interchanges in G_{PC} and in G_{PR}, a problem arises: x_R and x_C are both pink, which makes the colouring invalid.

In his PhD manuscript, in 1921, Errera[6] found a graph with 17 vertices and 45 edges, known as the Errera Graph, on which Kempe's proof fails. Therefore, the proof cannot be fixed.

However, Kempe's proof can be used to show that any planar graph is five colourable as Heawood did in [28]. Indeed, the only non-trivial case in the proof of this result is when $d = 5$. We can use the arguments of the case $d = 4$ of Kempe's proof: taking two disjoint Kempe chains and using one as a barrier.

1.2.3 Forbidden subgraphs

To reduce the number of possible counter-examples to explore, one way is to find a list of subgraphs which cannot be contained in any triangulated minimal counter-example: the forbidden configurations. This way, if at some point our current partial graph contains any graph in the list of forbidden configurations, we may discard it because any final graph we would obtain from this one would contain the forbidden configuration, hence could not be a counter-example. These forbidden configurations can be obtained thanks to the properties we imposed on our counter-example G_0: triangulation and minimality. We begin by giving a property shared by minimal counterexamples. We then define what a configuration is, and afterwards explain how to find some of the forbidden configurations.

The idea behind it is, assuming the correctness of the theorem, that if we list enough configurations then no plane graphs forbidding them all can exist. Section 1.2.4 describes a tool to realise 'quickly' that indeed no plane graphs can exclude all the given forbidden configurations.

We already proved something with Kempe's false proof.

Lemma 1.6. *The minimum degree of a minimal counter-example is five.*

Proof. The proof is simple. Let G be a minimal counter-example. Let us assume that G has a vertex u of degree less than 5. By minimality of G, $G \setminus u$ is four colourable. We showed in the correct part of Kempe's false proof that there exist a 4-colouring of $G \setminus u$ which can be extended to a colouring of G. Therefore, G is not a counter-example, which concludes the proof. □

Definition 1.12. A **separating short circuit** C of a plane graph G is a cycle of size at most five such that: if C is of length 3 or 4 then each of the two open[7] regions bounded by C contains at least one vertex, and if C is of length 5 then both open regions contains at least 2 vertices.

Lemma 1.7 ([3])**.** *Any minimal counter-example contains no short cycles.*

We prove this lemma in the cases of cycles of lengths 3 and 4 to illustrate a bit the concept of reducibility. We leave aside the case with a short cycle of length 5, which is longer to prove and does not bring much more understanding.

[6] who bears the same first name as Kempe

[7] excluding C

Partial proof. Let us assume that G_0 is a minimal counter-example to the theorem and that C is a separating short circuit of G_0 of length 3 or 4. We define G_{in} and G_{out} be the two closed regions bounded by C. Note that they both contain C.

Let us assume that C is a triangle with vertices a, b and c. Since G_{in} and G_{out} are smaller than G_0, each of them admits a four-colouring, and the vertices of C must receive different colours since they are pairwise connected. Up to renaming the colours, we may assume that in both colourings a receives colour 1, b colour 2 and c colour 3. The two colourings agree on the vertices of C so that they can be combined to form a four colouring of G_0, which is a contradiction.

We now assume that C has four vertices: a, b, c and d. We define in the same way G_{in} and G_{out}. For the same reason as before, each of them is four colourable. Up to renaming the colours, each colouring must be $(1, 2, 3, 4)$, $(1, 2, 1, 3)$, $(1, 2, 1, 2)$ or $(1, 2, 3, 2)$: either all vertices receive different colours, only two opposite vertices receive the same colour, or each pair of opposite vertices receive the same colour. If both G_{in} and G_{out} admit a colouring of the shape $(1, 2, 3, 4)$ then we are done: each of this colouring can be extended to G_0 as for the previous case.

We then assume that G_{out} does not admit $(1, 2, 3, 4)$ as a proper colouring. We will show that it admits both $(1, 2, 1, 3)$ and $(1, 2, 3, 2)$ as proper colourings, and that G_{in} admits one of the two. First, we prove that G_{out} must admit colourings of the shape $(1, 2, 1, 3)$ and $(1, 2, 3, 2)$. Indeed, the graph G_{out} to which we add the edge $\{a, c\}$ is still smaller than G_0, hence it admits a proper 4-colouring giving a and c different colours. This colouring is a proper colouring for G_{out}. The same argument applies if we instead add the edge $\{b, d\}$. Now we know that G_{out} admits both $(1, 2, 1, 3)$ and $(1, 2, 3, 2)$ as proper colourings. If G_{in} admits one of them, we are done. If it is not the case then G_{in} only admits colourings of the shape $(1, 2, 3, 4)$. We can then look at the 13-chain containing c: if it does not contain a we may perform a Kempe interchange and obtain $(1, 2, 1, 4)$. Up to renaming the colours, we may assume it is the desired $(1, 2, 1, 3)$. Otherwise, as in the case $d = 4$ of the Kempe proof in Section 1.2.2 we know that the 24-chain containing d does not contain c, hence we can obtain $(1, 2, 3, 2)$, which concludes the proof for a short circuit of length 4. □

We now introduce the notion of configuration used by the correct proofs of the four-colour theorem. It enables us to define 'partial' graphs, which will be shown to be excluded from any minimal counter-example. We recall that an almost-triangulated plane graph is an embedding in which every face is a triangle, except for at most one.

Definition 1.13. A **configuration** is a couple $C = (H, \gamma)$ where $H = (V, E)$ is an almost-triangulated plane graph and $\gamma : V \to \mathbb{N}$. It verifies $\gamma(v) = \mathrm{d}^\circ(v)$ except for the vertices of at most one face, which has to be the non-triangular face if any. This face is called the outer face, and its vertices verify $\gamma(v) > \mathrm{d}^\circ(v)$.

We can see an example of configurations in Figure 1.7. Each configuration is drawn such that its contour is the outer face. We can see that, in the second line, two configurations have a vertex which does not belong to the outer face. In both cases it has a γ value of five, and its degree in the configuration is indeed five. The other vertices have a degree in the configurations at most equal to their γ value.

Definition 1.14. A configuration $C = (H, \gamma)$ **appears** or **is contained** in a triangulated graph G when:

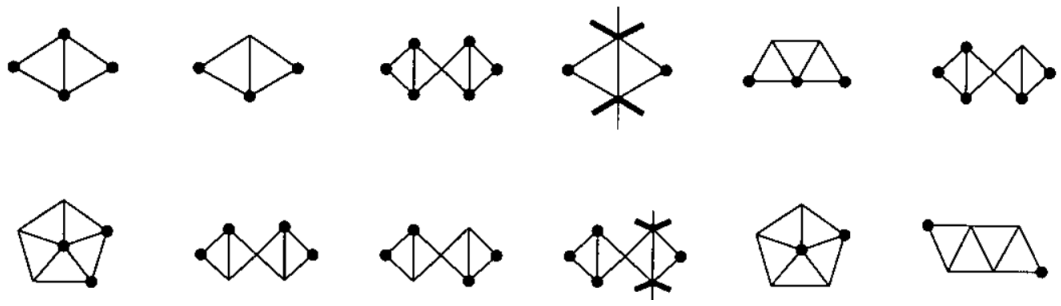


Figure 1.7: An example of reducible configurations found by Robertson et al. The vertices represented by a black circle have a γ-value of 5, and the others a γ-value of 6.
The figure comes from [45].

- H is an induced subgraph of G;
- except for the outer face, any face of H is mapped to a face in G;
- if $u_H \in H$ is mapped to $u_G \in G$ then $\mathrm{d}^\circ(u_G) = \gamma(u_H)$: γ represents the degrees of the vertices once mapped in a graph.

Definition 1.15. When a configuration C cannot appear in any minimal triangulated counter-example, we say that C is a **forbidden configuration**, or that C is **reducible**.

Figure 1.7 gives an example of twelve reducible configurations from [45]. In that paper, great care is brought to the list of reducible configurations. Their configurations have stronger structural properties than in Definition 1.13. For instance, if removing a vertex splits the graph into several components then there are at most two of them and this vertex has exactly two other neighbours than the one appearing in the configurations (its γ-value is 2 more than its degree in the configuration). If a vertex is not incident to the region which is not a triangle, then its gamma value is its degree in the configuration. Some other properties are true for their list of reducible configurations, one of which saying that the sum of the $\gamma(v) - \mathrm{d}^\circ(v) - 1$ over some specific sets of vertices is at least two.

One way to show that a configuration $G = (H, \gamma)$ is reducible is to show that if a graph G_0 contains G and is not four colourable, then there exists a graph G_0', still not colourable, but with fewer vertices than G_0. For instance, G_0' could be constructed from G_0 by removing H and replacing it with a smaller subgraph. We describe a bit further how to detect this case, as well as another way to find that a configuration is reducible.

We consider a configuration G, almost-triangulated, and denote by L the cycle defining its outer face. We call L the *crown* of the configuration. We generate the list X_{true} of all 4-colourings of L which induce a proper colouring on G, i.e. all colourings pf L which can be extended into a proper colouring of G. If G appears in some counter-example, then every proper 4-colouring of G is guaranteed to be non-extendible to the whole graph it would appear in, for otherwise the counter-example would have a 4-colouring. We study the restrictions of the colourings of G to its crown because if G appears in a counter-example, its crown is its 'interface' with the other vertices. We now try to find a smaller configuration G' which has the same crown bounding its outer face, with some property on its colourings. If, denoting the set of the restrictions of its proper 4-colourings to the

crown L by X'_{true}, we find that $X'_{\text{true}} \subseteq X_{\text{true}}$, we deduce that G is reducible. Indeed, let G_0 be a counter-example containing G. Let us replace G by G' in G_0 to obtain G'_0. G'_0 is smaller than G_0 and admits no 4-colourings. Indeed, any 4-colouring of G'_0 would induce a 4-colouring of G_0: this 4-colouring restricted to the crown L would also be proper for G, hence a proper colouring for G_0. We would have achieved our goal: any time G would appear in a counter-example, it could be replaced by G', hence this would show its reducibility.

However, having to enumerate all smaller configurations with the same crown (and its 4-colourings inducing a proper 4-colouring of G!) is very costly in time. Also, this technique was not efficient enough for the proof of Robertson et al.: it misses a lot of reducible configurations. Indeed, when examining a configuration, it leaves aside the completions of the configuration into a plane graph. It may be possible to show that some configuration forces any graph extending it to be four colourable. There are indeed others techniques to show that a configuration is reducible: in fact the researchers who worked on the four-colour conjecture categorised the reducibility property into several classes. One of them, used in the proof of Robertson et al. is the *D-reducibility*. It once again uses an argument based on Kempe chains. Let again G be a configurations which we want to show is reducible. Let us call X_{false} the list of the 4-colourings of the crown L of G which do not induce a proper 4-colouring of a configuration. Let us assume that G appears in G_0. We then know that $G' = (G_0 \setminus G) \cup L$ is four colourable, because it is smaller than G_0. Let us call X'_{true} the list of the 4-colourings of L which induce a proper 4-colouring of G'. G appears in G_0 and G_0 is not four colourable, therefore like previously $X'_{\text{true}} \subseteq X_{\text{false}}$. Our goal is to show that in fact $X'_{\text{true}} = \varnothing$, which is a contradiction: G', being smaller than G_0, is four colourable. To do so, we try to find a maximal *consistent* set X'_{C} of allowed 4-colourings for G'. We begin by setting $X'_{\text{C}} = X_{\text{false}}$. Then, for each colouring in X'_{C}, we first look for other colourings of the crown which should be allowed for G' based on Kempe interchange arguments. If, for some colouring $c \in X'_{\text{C}}$, one such colouring c_1 is not allowed, i.e. not in X'_{C} then this implies that c is in fact not allowed either. This is what is called the consistency of a set: if a colouring is allowed, but not some colourings obtained by Kempe interchanges, then the set of allowed colourings is inconsistent. Removing c may in turn lead to some other removals of possibly valid colourings. We iterate this process until the set X'_{C} stabilises. If at the end $X'_{\text{C}} = \varnothing$, this means that the configuration is D-reducible. We do not dive into the details of how to deduce which colourings should also be valid, given that some c belongs to the current X'_{C}. The algorithm to compute these is quite complex. In our code, the search for a maximal consistent set of colourings included in some X_{false} is done by the function `get_maximal_consistent_colouring_subset`.

We may notice that we can combine this method with the previous one, when we enumerated smaller subgraphs. Indeed, we looked for some smaller configuration G' such that $X'_{\text{true}} \subseteq X_{\text{true}}$, or, equivalently, such that $X_{\text{false}} \subseteq X'_{\text{false}}$. This condition is more frequently met if we reduce the set X_{false} to a maximal consistent subset. Robertson et al. used this D-reducibility notion, but they also used another type of reducibility (the C-reducibility) which we do not define here.

1.2.4 The discharging method

We detail here more the idea of discharging, mentioned in Section 1.2.1[8]. It helps us realise that the class of minimal counter-examples is empty. Discharging requires us to work on *weighted* graphs.

Definition 1.16. A **weighted** graph is a couple (G, w) where $G = (V, E)$ is a graph and $w : V \mapsto \mathbb{Z}$ is the weight function.

The weight function assigns a weight to each vertex of the graph. The discharging method will consist in locally moving parts of the weights between certain vertices and some of their neighbours whenever certain conditions are met. Note that here we only put weights on vertices, but it is possible to also put weights on edges and faces.

Definition 1.17. A discharging rule is a quadruplet (F, u, v, q): F is a configuration, u (the **source**) and v (the **sink**) are vertices of F, and q is the weight of the rule.

Definition 1.18. Applying rule (F, u, v, q) to a weighted graph (G, w) results in the weighted graph (G, w') where for all $u \in V$, $w'(u) = w(u) + (a - b)q$ if, in G, F appears in a different times with u as a sink and b different times with u as a source.

Applying a rule consists in, each time F appears in G such that $u \in F$ is matched with $u' \in G$ and $v \in F$ is matched with $v' \in G$, transferring a weight of q from u' to v'.

Remark 1.4. Note that, like for weighting a graph, we put weights on the vertices, but other uses of the discharging method can also weight edges and faces. In this case, the discharging rules would also transfer weights between vertices, edges and faces.

Fact 1.2. *Applying any discharging rule to a graph does not change its total weight.*

Fact 1.3. *If a graph has a positive total weight then it has a vertex of positive weight.*

These two facts are trivial. Now, let us assume that the class of minimal counter-examples to the four-colour theorem is not empty. **Let G_0 be a minimal counter-example.** We assign the weights in G_0 in the following way:

$$w(v) = 10(6 - \mathrm{d}^\circ(v)).$$

Claim 1.1. *If G is triangulated, then the sum of the weights of (G, w) is 120.*

Proof. To show this claim, we use again Euler's formula for planar graphs: $n - m + f = 2$ where n, m and f are respectively the number of vertices, edges, and faces of our graph.

Since our graph is triangulated, we have $3f = 2m$: each face has three edges, but each edge is shared by two faces. Now:

$$\begin{aligned}
\sum_{v \in V} w(v) = \sum_{v \in V} 10(6 - \mathrm{d}^\circ(v)) = 10(\sum_{v \in V} 6 - \sum_{v \in V} \mathrm{d}^\circ(v)) &= 10(6n - 2m) && (1.1)\\
&= 10(6m - 6f + 12 - 2m) && (1.2)\\
&= 10(12 + 4m - 6f) && (1.3)\\
&= 120. && (1.4)
\end{aligned}$$

(1.2) comes from Euler's formula and (1.4) from the relations between f and m we mentioned above. □

[8] We hope we were not too efficient so that there are still things to learn or understand here.

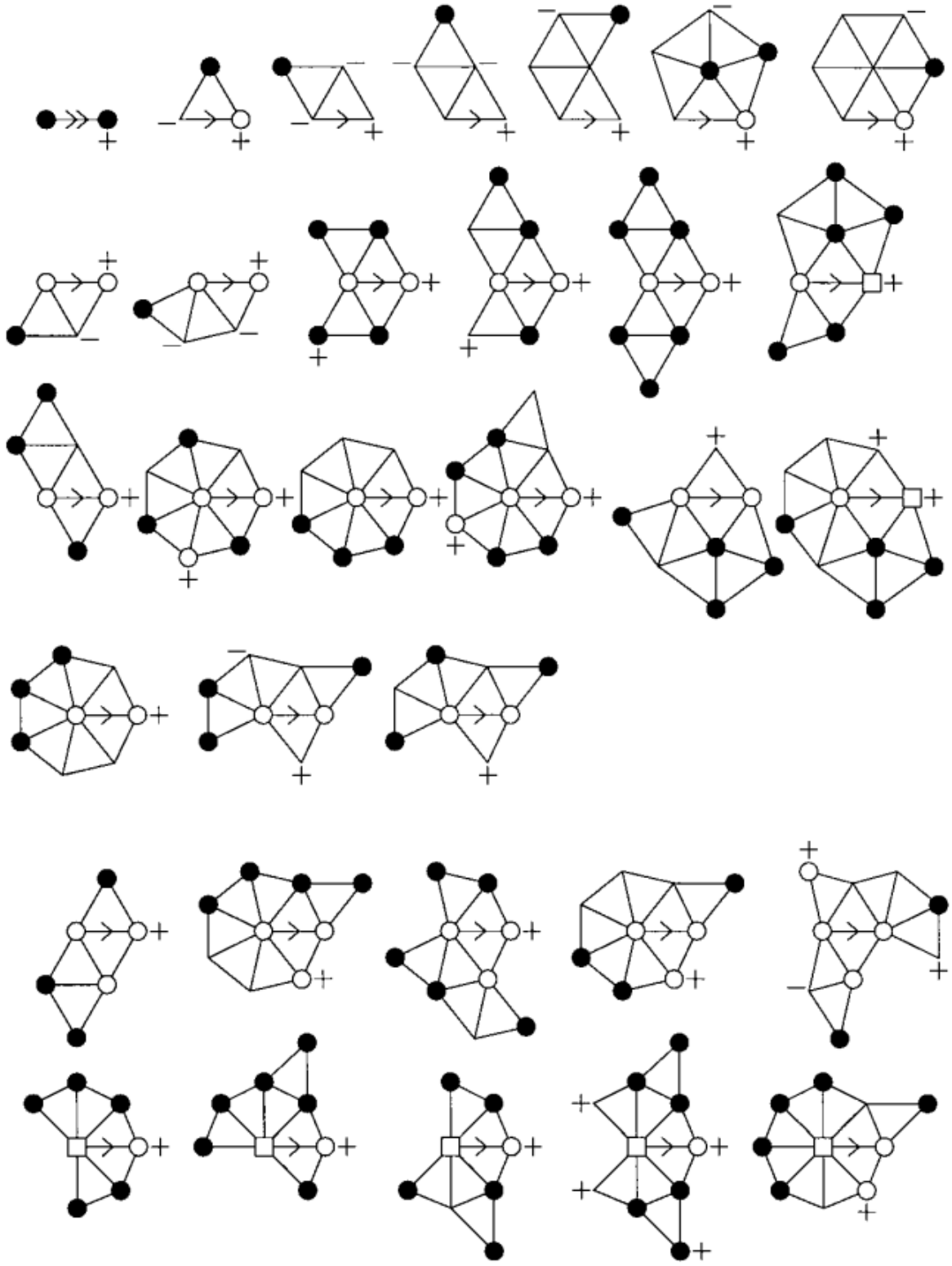


Figure 1.8: The 32 rules used in the proof by Robertson et al. The degrees are given by symbols: black circles, points, white circles, white squares and white triangles respectively represent integers from 5 to 9. A '-' (resp. '+') sign as exponent means 'degree at most' (resp. 'degree at least'). All rules have weight 1 except for the first one. The arc in the configuration originates at the source of the rule and ends at its sink.
The figure comes from [45].

Since G_0 has a total weight of 120, and from Facts 1.2 and 1.3, **there is a vertex which has a positive weight after applying all the discharging rules once**. **We name it v_0**. Our goal will be to show that it cannot exist: any neighbourhood for v_0 will necessarily contain reducible configurations. We will enumerate the possible neighbourhood with the help of a computer. This enumeration is detailed in Section 1.3.1. We show in Figure 1.8 the 32 discharging rules of the proof of Robertson et al.

1.3 Contribution

We began by proving again the four-colour theorem, using the data (the lists of forbidden subgraphs and discharging rules) from the proof of Robertson et al. [45]. We explain here the different parts of the program and some of the algorithms we implemented. A modification of this program is released with a Python interface. It can be included with the Sage software, a free mathematical assistant. It enables anyone to try to prove a result using the discharging method: the user has to provide the forbidden subgraphs and discharging rules for their problem, as well as a weight function. Then our program tries to prove that no plane graphs avoiding all the forbidden subgraphs exist. We provide some flexibility to the user since they can provide a python function which, given a partial current graph, decide whether or not it should be discarded: some properties may be better coded than expressed in terms of forbidden subgraphs.

1.3.1 The program

We describe here several parts of the program, like the scheme of the branch-and-bound enumeration of the possible neighbourhoods for v_0. We recall that G_0 is a minimal counter-example we assume the existence, and v_0 is one of its vertex which has positive weight after applying all the rules once.

The enumeration.

We start the search for G_0 with v_0: it is the first vertex we build. We then explore the possible neighbourhoods for v_0. In the exploration, we can do several things: add vertices and edges, or choosing the final degree of a vertex (at first, they have some degree interval). By doing so, we will indeed explore every possible neighbourhood for v_0, until building G_0 or showing that G_0 does not exist[9].

Let us describe the algorithm with more details. We first start with the vertex v_0, or in fact a triangle containing v_0 since G_0 is triangulated. The degree of each vertex of the triangle has degree between 5 and 12, except for v_0 which has degree between 6 and 11. These restrictions were shown by Robertson et al. (we already showed each degree must be at least 5).

Let us assume that we are at some step with a partial neighbourhood of v_0 we call G. We have two options. We may choose a vertex u whose current degree lies between k_1 and k_2 with $k_1 < k_2$. We subdivide its degree interval into two smaller ones: G' derived from G by fixing the degree of u to be k_1 and G'' in which the degree of u is between $k_1 + 1$ and k_2. We will continue the exploration first with G', then with G''. The second

[9]The course of History has taught us that it is this option which happens.

option is to create a new edge from a vertex u whose degree interval is not $[\![0;0]\!]$. This edge may create a new vertex, or its other endpoint may be an existing vertex. In both cases, we create every free triangle we can: if a vertex has degree one, we know it will belong to a triangle we can describe.

Sometimes creating free triangles may fail: a triangle cannot be constructed because a vertex cannot accept new neighbours for instance. When this occurs, we may discard our current graph and backtrack. Before calling recursively our exploration function on a graph G', we do some checks. If G' contains a reducible configuration, we may also discard it. Finally, we also apply all the discharging rules which involve v_0 and obtain an interval for the final weight of v_0. For instance, for the upper bound we apply the rules we are sure apply, and apply also the rules which may apply and contribute to increasing the weight of v_0. If the upper bound is non-positive, this is a contradiction[10] and we may also discard G' and backtrack. Besides, we may use the information about some configurations which almost appeared, or some rule which almost applied to choose how to expand our current graph or which degree to refine. Good heuristics for this choice lead to reducing the number of graphs we explore, hence reducing the running time.

Storing plane graphs.

We manipulate partial plane graphs all the time in the program so they must be stored efficiently for our uses. First, before storing them we had to read the reducible configurations and rules from Robertson et al. The way the reducible configurations are encoded is available at `http://people.math.gatech.edu/~thomas/FC/ftpinfo.html`. To parse them requires a good comprehension of their proof. Each one is given as the coordinates of an embedding of the configurations. The format of the rules is described in [46] and, is also not that handy. Our program uses a different format which we find easier and more convenient. Since a plane graph is completely determined by the **directed** list of the edges of each vertex, we decided to store a graph as the directed adjacency lists of each of its vertices. We chose the counter-clockwise direction in our program. As we mentioned, each half-edge contains the information about the minimum and maximum number of future edges between itself and the next known half-edge of the vertex.

Concerning the storing of the graphs properly speaking, it essentially boils down to a doubly-linked list of 'half-edges' for each vertex. We call them half-edges because an edge $\{a, b\}$ is both stored for a and for b. Also, any edge defines an angle: the interval giving the number of other edges between this edge and the next known edge of the vertex. This means that the half-edge joining a to b and the one joining b to a are not the same: they define different angles. For easy triangle making and detection we also store, for each half-edge, a link to its other half.

Detecting if a configuration appears.

To detect when reducible configurations appear and which rules apply, we need some subgraph detection routine. This is called the induced subgraph isomorphism problem, and is known to be very hard for general graphs. However, here it is on plane graphs that we want to detect some type of isomorphism, which modifies the rules of the problem,

[10] We chose v_0 such that its weight is positive after applying the rules.

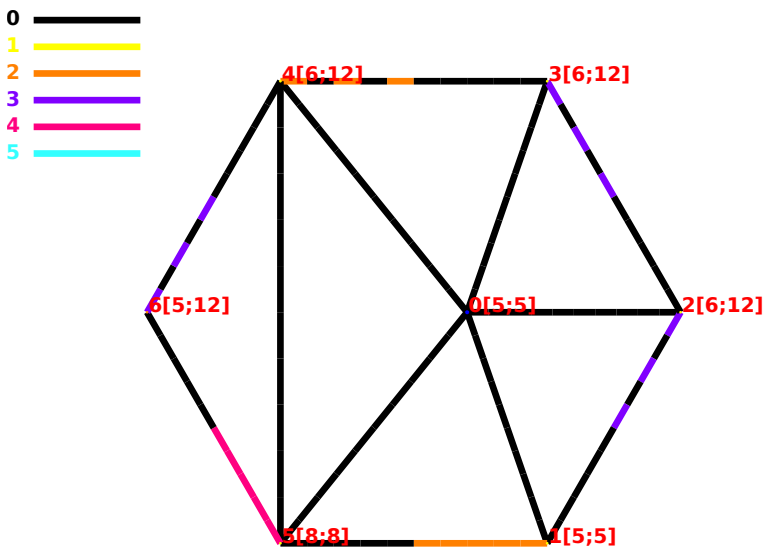


Figure 1.9: A screenshot of our program showing a partial graph during the exploration. The colours are indicated on the top left corner. The label of each vertex is of the form '$a[b;c]$': vertex a has degree between b and c. Each edge is split in two in the middle: the two different half-edges. The colour of the half-edges indicated the degree interval of the angle until the next edge (in counter-clockwise direction). If a half-edge has two colours, the smallest one is the lower bound on the degree of the angle, the other one is the upper bound.

and make it polynomial.

Indeed, let us assume that G and C are plane graphs and not configurations: each vertex has a degree instead of a degree interval, and all vertices and edges are known. We want to know if, or even (for the discharging rules) where C appears in G.We may choose an edge e_C and try to match it to every edge e_G. Now, to verify if one matching of this edge extends to an isomorphism between C and G, we may do a graph traversal by walking from faces to faces. We are forced to map the faces which contain e_C to the faces which contain e_G. There are two faces, and as soon as one is chosen, then the rest of the isomorphism is forced: there is always one choice (or 0 if G does not contain C) for matching every other vertex and edge.

Yet, here we do not test for exact isomorphism because in the rules and reducible configurations, vertices of the crowns do not have every neighbour instantiated. This happens a bit in C, and more in G. We precise here how a configuration appears in such 'partial' graphs G we manipulate here: a configuration appears if whatever the completion of G into any G_{final}, C appears in G_{final}. Due to this problem we sometimes cannot conclude: C might for instance be a subgraph but we cannot decide yet if it will be an induced one.

In order to optimise the induced subgraph detection we may first look at the maximum

and minimum degree of C: each of these two vertices must be assigned a vertex in G with the same degree. After this, we may look at the list of degrees in C and in G. By choosing to match first an edge belonging to a vertex of G whose degree appears the least often in the list of degrees of G, we may reduce the running time of the check. Indeed, ideally the vertex would only be matched to a single vertex of G, if it is alone to verify the degree condition.

A Python/Sage library to use the discharging method.

We decided, since we had programmed it, to let the community use our engine for discharging. We created a Python interface to our program. The user has to provide a file containing the reducible configurations, and a file containing the discharging rules[11]. They must also provide a function to compute the weights of the vertices. They can provide a python function which has access to a current neighbourhood in the exploration, and decides whether it should be discarded based on some other properties. The program then enumerate the possible neighbourhoods of a vertex which should have a positive weight after applying the rules. It can print the neighbourhoods which could not be discarded (see Figure 1.9).

Making this program available to Sage meant porting it to Python. Instead of writing the code from scratch in Python, which would in addition make it slower, we decided to interface it with Python. We used the Boost.Python library, which allows us to expose some C++ functions to a python program. It creates a Python module including the specified functions. To do so, we needed to add a bit of code: tell Boost.Python how to translate the functions, i.e. what types they should have. Some problems occurred during this translation, in particular with C++ functions using some C++ features. We sometimes had to define a new C++ function to wrap a problematic one into something with a simpler header (the types of the arguments of the function and its return type). For instance, no functions exposed to Python could contain pointers.

We also said that the user could provide a function to check for properties easier to check with some code than by forbidding graphs. They also need to provide a function to compute the weight of a vertex. This means the C++ code[12] should be able to execute a Python function[13]. This was harder than running C++ code from Python. We managed to do it with the same Boost.Python library, which provided some 'python object' type, which could be anything passed by Python to the C++ program. It also provides functions to specify this object, that is for instance specify its type, and if it is a function. In the latter case, it is possible to run the Python function and retrieve its output. After some time testing how all this was working, we achieved our goal. A good thing is that the modifications involved adding a bit more code but are rather transparent: the exploration function calls the function `compute_weights` which can either be a C++ function (as we did ourselves for the four-colour theorem) or a Python function. Our exploration function contains a unique call to the `compute_weights` function, which means that the syntax of the calls was not modified and that a C++ function and a

[11] for the moment, only with weights on vertices

[12] called from a python code

[13] If you followed carefully, it means that a python code would execute (some C++ code which executes) some Python code!

Python function can be unified in a single C++ Boost.Python type.

1.3.2 A possible more automated approach

The original goal which motivated us to study and redo the proof of the four-colour theorem was to automatise the proof a bit further. The set of reducible configurations was generated by a computer, so this part is fine. The set of discharging rules, however, was found by hand, by error and trial. This part is somewhat tedious because, as we will see, it is repetitive. It is the part we wanted to improve by making it more automatic, so this section gives ideas on how achieve this.

As we have just written, looking for discharging rules is repetitive. We start with a limited set of discharging rules, and run the program. If it seems not to terminate, it is possible to look at some of the graphs it did not manage to discard, notably (for the four-colour theorem) the ones going further than a neighbourhood of distance two from the first vertex, which was supposed to be non-necessary. Then it is possible to devise some rules, and rerun the program again. Once more, if the program does not terminate in a reasonable amount of time, we can have a look at the graphs which were not discarded. This makes it possible to craft new rules, taking care of not going backwards on the ones which were introduced earlier: a graph which was discarded before should still be discarded after adding a new rule.

The objective was to find a way to generate automatically new discharging rules: the program would be the one to analyse the non-discarded graphs, and find some 'optimal' rules. A rule would for instance be optimal if it makes it possible to discard 'as many graphs as possible'. One obvious way to generate rules would be to enumerate a certain number of 'small' configurations and generate rules by defining sources and sinks. Then we could replay the beginning of the run with each rule tested as the new one to be adopted. The one which would lead to discarding the most graphs would be kept.

Another idea could be to try to optimise an existing set of rules: we assume we already have a few rules. We may discard more graphs without adding a new rule by modifying the weight of each rule. A linear program may model our needs: given the list of graphs we have explored so far (the ones we already discarded and some others we would like to discard), we may associate to each one an equation. Let us consider a graph G and $x_{i,G}$ be the net effect of Rule i on our vertex v_0 in G. If Rule i leads to transferring 4 towards v_0 and 7 from v_0, then its net effect would be $4 - 7 = -3$. Now we obtain the condition, for each G:

$$w(v_0) + \sum_i x_{i,G} \cdot w_i \leq 0.$$

The w_i's are proper to the rules, independent from the explored graphs. If solving the linear program shows that some solution exists, it means that with the same set of rules, only by tweaking the coefficients, we may discard more graphs.

We did not investigate much the possible methods to automate the search for good discharging rules. However, finding some automatic ways to obtain them would be a big step forward in the field of automatic proofs for planar graphs. It would make it a lot faster and easier to try the discharging method to attack some other problems on planar graphs.

Chapter 2

The domination numbers in grid graphs

Societies have always (or for long, at least) seen a group of dominant people emerge and set the rules. Nowadays, it is both publicly said by some people and conveyed by some media that some groups dominate the majority of people. This is refuted or ignored by many in the dominating groups, and little is done to correct the state of things. Such groups may include (very) rich people and countries, men, cisgender[1] people, valid[2] people, white people and so on. Let us state a trivial thing: not all of these people, who enjoy some privileges due to the colour of their skin, their gender or other factors, consciously oppress women or some minorities. Not every person in these groups takes part in the wrongdoings, but many do or take part in the system, which is oppressing. Thus the overall behaviour of these groups is bad. Well, yes, a lot of women contribute to sexism, because society framed them into doing so. But guess what? Women suffer from this problem which comes from, and is mostly maintained by men: the ones who refuse to see the problem, and those who do nothing to try to fix this state of things. The dominating groups come from all sort of problems such as patriarchy and sexism, queerphobia[3], disabilism, and a lot of other systemic discriminations. For instance, society leads to the invisibility of disabled people in society: how many times did you watch a sport played by disabled people, either on TV or in real life, or read about it somewhere? Do you realise that in some cities, some disabled people cannot take the underground or the bus, or go to some buildings, because they have mandatory stairs? In Paris for instance, only one line and a dozen stations (out of 303) of the underground are accessible to people in a wheelchair. Patriarchy and sexism also induce a lot of problems: in France women who have a job are four times as likely as men to have a part-time job. They earn overall 18.5% less than men, 16.3% less than men when restricting to full-time employees, and they are still paid 12.8% less than men when considering equivalent positions. They also compose only 23% of the people in the French parliament. In addition to this, there were 149 people killed in domestic violence in France in 2018: 128 of them were women. 25% of women between 20 and 65 years declared having suffered some violence in a public place during the last year, and this rate jumps to more than 60% if we consider women

[1] a person whose (social) gender is the same as the biological sex which was assigned to them at their birth

[2] not disabled

[3] including, but not limited to transphobia and homophobia: (when) did you learn about asexuality and/or stopped thinking this was a disease or a problem?

between 20 and 24 years. Among these violences, 1 million women declare having, over the last year, suffered harassment or sexual harassment. During the year 2018 in France, 1905 acts of LGBTQ-phobia[4] have been recorded, among which 231 involving physical violence. These numbers keep increasing and a poll suggested that only 27% of the victims of physical violence report it to the police. Two thirds of LGBT people have at some point avoided holding the hands or kissing their companion and 12% have considered moving to another city to avoid being harassed or assaulted. This means that a majority of LGBT people are denied the right of walking freely in the street without fearing for themselves.

However, we study here some forms of domination which makes no one suffer... except maybe the people studying these problems. Indeed, the domination number problem, that is the problem of finding the minimum size of a dominating set[5] is a NP-complete problem for general graphs. This informally means that if we ask, 'Does G admit a dominating set of size less than or equal to k?', then it is easy to check that one solution we are given is indeed of size at most k and dominating (the problem is in NP); however we do not know any systematic and 'fast' algorithmic way to prove the 'no' answer (the problem is NP-hard: it is as hard as other NP-complete problems).

The basic domination problem consists in selecting a set of vertices in a graph such that any other vertex has a neighbour in that set. As we will see, many variants of this problem exist. These problems can be used to model optimising problems arising in real life, as the Roman-domination problem illustrates: it is said to have been used as a model by the Romans to defend their territory. The domination problem can also be used in other contexts, such as some public services: where to put hospitals, fire stations, and other critical places such that every person in a country can benefit from it.

The domination number problem is one of many problems which are hard for general graphs, but are easy to solve for graphs of bounded treewidth. Indeed, Courcelle [8] showed in 1990 that a particular class of properties, the one being expressible in monadic second-order logic, are decidable in linear time on the class of bounded-treewidth graphs. The treewidth can be understood, intuitively, as a measure of how much a graph 'looks like' a tree. The grids are among the simplest graphs which neither have a bounded treewidth nor a bounded cliquewidth (another graph parameter), and for which these kinds of problems are usually difficult to tackle.

The first values (for a number of lines $n \in \{2, 3, 4\}$) of the domination number in grids were discovered by Jacobson and Kinch in 1983. Then, ten years later Chang and Clark found the ones for 5 and 6 lines. All these results were found without using any computer. Also in 1993, Chang [6] conjectured that the domination number for a grid graph of arbitrary size, that is the minimum size of a dominating set, was $\gamma(G_{n,m}) = \left\lceil \frac{(n+2)(m+2)}{5} \right\rceil - 4$. He also showed that this was actually an upper bound. Fisher [14], using computer resources, found the values for $n \leq 19$ and showed that these values were conform to the conjecture. He also found a method to detect and prove the periodicity of the domination number, so as to establish formulas for a fixed number of lines and arbitary number of

[4]LGBT stands for Lesbian, Bisexual, Transgender, Queer. The acronym designates the union of people in these non-exclusive sets.

[5]or to be more formal, the decision problem associated to this optimisation problem

columns. After a few papers by a few other people, Gonçalves et al. [21] finally proved Chang's conjecture in 2011 by showing that his formula was also a lower bound, improving a bound by Guichard [24]. After 2011, several papers gave formulas for small number of lines for various domination problems. Several generalisations of the domination problem have also been studied in the literature (see for example [4]).

The goal of this chapter is to solve the 2-domination and Roman-domination problems. We achieve this by giving closed formulas computing the minimum cost of the respective dominating sets for any size of grids as in [42]. The formulas we give are simple: they involve multiplications, additions, division and rounding to lower or upper integer. This shows that, like the domination number, the 2-domination and the Roman domination numbers problems are solvable in constant time in grids.

This chapter is organised as follows. After giving some definitions in Section 2.1, we will explain in Section 2.2 how to solve the 2-domination problem on grid graphs: first on fixed-height grids and then on arbitrary grids, relying on the notion of *loss*. We also use the *Rauzy graphs* to give some complexity information. In Section 2.3, we will study the distance-two-domination problem and the total-domination problem, but are only able to give formulas for grids of small number of lines. We also give a lower bound for the total loss when both the number of lines and the one of columns are arbitrary. We continue by giving in Section 2.4 some insight on why the method for arbitrary-size grids what we believe the method works in some cases and seems not to in others. We define some properties like the *fixed-height-border-fixing* one which we conjecture explains when the method for arbitrary grids works. In Section 2.5, we finally explain some of the optimisations we made and give some implementation details and statistics.

2.1 Basic definitions and notations

We define here formally the different types of problems we will study in this chapter. We list them by increasing complexity.

Since we work with two dimensions in this chapter and the following, we will try to be coherent all along. There may be some differences on the order of the indices between these chapters and the associated papers. In what follows, n will always be the number of lines and m the number of columns. When speaking of a grid $G_{n,m}$ or the domination number $\gamma(n, m)$ we first put the number of lines. When using coordinates, we will use the standard order (x, y). The indices will be j when referring to the columns numbers, and i when we refer to the lines indices.

In this thesis, the x-values are increasing from left to right, and the y-values are when going from top to bottom. In this chapter, the indices and coordinates will always begin at 0 (and not 1), and $(0, 0)$ are the coordinates of the top-left cell of a graph or rectangle.

We denote by $G_{n,m}$ the grid graph with n lines and m columns. In the illustrations to come, the vertices of a grid will be its cells (and not the intersections of the lines).

Definition 2.1. A set S of vertices of G is **dominating** when any vertex not in S has at least one neighbour in S.

(a) (b)

Figure 2.1: Illustration of a dominating set on $G_{5,7}$:
(a) the cells pointed by arrows are not dominated;
(b) the set of black cells is dominating.

Definition 2.2. A set S of vertices of G is **2-dominating** when any vertex not in S has at least two neighbours in S.

(a) (b)

Figure 2.2: Illustration of a 2-dominating set on $G_{5,7}$:
(a) the cells pointed by arrows are not 2-dominated: they have 0 or 1 black neighbour;
(b) the set of black cells is 2-dominating.

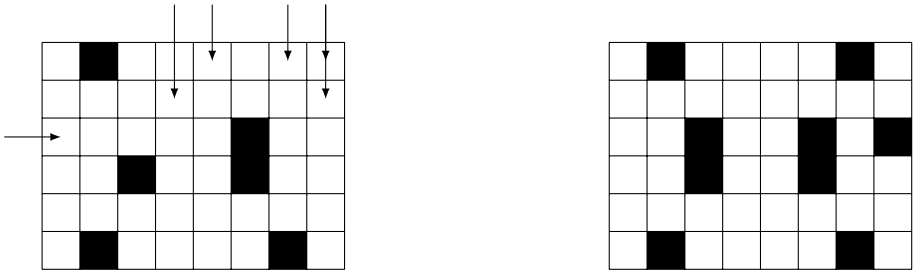
Definition 2.3. A set S of vertices of G is **distance-two-dominating**[6] when any vertex v not in S has a neighbour at distance at most two in S: either a neighbour of v or a neighbour of one of v's neighbours.

It is easy to note that for instance any 2-dominating set is dominating. Similarly, any dominating or 2-dominating set is distance-two-dominating.

Definition 2.4. A set S of vertices of a graph G is **total dominating** when any vertex $v \in G$ has at least one neighbour in S.

Notice that, by contrast with the domination, even dominant vertices must have a neighbour which dominates them. In the case of grids, since there are no loops, a vertex is never its own neighbour, therefore each $v \in S$ must be connected to some vertex in $S \setminus \{v\}$.

[6]As for the other types of domination, a point of English grammar arises. We write: 'This child is six years old.' but 'This is a six-year-old child.' From that we infer that while there is no need to hyphenate the group when following 'to be', we have to when the whole group is an adjective which precedes a noun. We also hyphenate when necessary to avoid confusions.



(a) (b)

Figure 2.3: Illustration of a distance-2-dominating set on $G_{6,8}$:
(a) the cells pointed by arrows are not distance-two-dominated: the closest black cell is at distance at least three;
(b) the set of black cells is distance-two-dominating.

(b) (b)

Figure 2.4: Illustration of a total-dominating set on $G_{5,7}$:
(a) the set is not total dominated: the two cells pointed by arrows are not dominated;
(b) the set of black cells is total dominating.

Definition 2.5. A **Roman-dominating 'set'** is a pair (S_1, S_2) such that every vertex $v \notin S_1 \cup S_2$ has at least one neighbour in S_2.

Informally, a Roman-dominating set consists in placing troops of soldiers on the vertices. We can either put no troops, one troop or two troops. A single troop can defend the vertex it is placed on while two troops placed on a vertex defend both it and its neighbours. The cost is the total number of troops.

Definition 2.6. The cost of a dominating, 2-dominating or distance-2-dominating set S is its size: $|S|$. For a Roman-dominating set, the cost is $|S_1| + 2|S_2|$. The **2-domination number** of a graph G, denoted by $\boldsymbol{\gamma_2(G)}$, is the minimum cost of a 2-dominating set of G.
We define similarly the total-domination number $\boldsymbol{\gamma_T}$, the distance-2-domination number $\boldsymbol{\gamma_{d2}}$ and the Roman-domination number by $\boldsymbol{\gamma_R}$.

Note that in the previous figures illustrating the various domination problems, the correct dominating sets are not necessarily of minimum size. For some, we may even trivially remove some vertices from the dominating set without breaking the domination property. For instance in Figure 2.5 one grey cell has a black cell neighbour. The grey

(a) (b)

Figure 2.5: Illustration of a Roman-dominating set on $G_{5,7}$ (ells with two troops are black and cells with one troop are grey):
(a) the cells pointed by arrows are not dominated.
(b) the set of black and grey cells is Roman dominating

cell can be removed because it is still dominated by its black neighbour and it does not dominate any cell.

Notation 2.7. For concision we will denote $\gamma_2(G_{n,m})$ by the shorter notation $\boldsymbol{\gamma_2(n,m)}$. The same applies to the other domination problems.

2.2 Method for finding the 2-domination number for grids of arbitrary size

Here, we set up a framework for finding the minimum size of dominating sets in a grid. We first explain how to proceed when the number of lines is fixed, then we introduce the notion of *loss* to extend it, when possible, to grids of arbitrary height. The first method is only usable when the number of lines is small, because the number of objects we examine with our computer program becomes too big at some point. The second method begins to work when the number of lines is big enough, but does not always work, depending on some characteristics of the problem. Also, even if it should work on a problem, it may fail by lack of computational power. In fact, the number of lines for which the method works needs to be small enough to make it possible to run the method on a computer. We first explain one after the other the two methods, on the 2-domination problem (see [21] for an alternate explanation of the loss method, applied to the domination problem). Both methods rely on the notions of *states* and *compatibility relations*, which are translated into transfer matrix products in the $(\min, +)$-algebra (it can also be viewed as a dynamic algorithm). The **$(\min, +)$-algebra** consists in substituting, in the computations, the operator $+$ by the operator min as well as replacing the multiplication by the addition. It is a standard method. The second method uses the more recent and more complex method of *loss* introduced by Gonçalves et al. [21]. We are the first ones to use this method to find other results. The optimisations we made and some details on the manner they were implemented will also be discussed.

Throughout the explanations of this section, we will prove the following theorem, which confirms the results found by [34, 47] for $n \leq 4$ and slightly corrects the result by [47] for $n = 5$:

Theorem 2.1 ([42]). *For all $1 \leq n \leq m$, the 2-domination number equals:*

$$\gamma_2(n,m) = \begin{cases} \left\lceil \frac{m+1}{2} \right\rceil & \text{if } n = 1 \\ m & \text{if } n = 2 \\ m + \left\lceil \frac{m}{3} \right\rceil & \text{if } n = 3 \\ 2m - \left\lfloor \frac{m}{4} \right\rfloor & \text{if } n = 4 \text{ and } m \mod 4 = 3 \\ 2m - \left\lfloor \frac{m}{4} \right\rfloor + 1 & \text{if } n = 4 \text{ and } m \mod 4 \neq 3 \\ 2m + \left\lceil \frac{m}{7} \right\rceil + 1 & \text{if } n = 5 \text{ and } m \mod 7 \in \{0,6\} \\ 2m + \left\lceil \frac{m}{7} \right\rceil & \text{if } n = 5 \text{ and } m \mod 7 \notin \{0,6\} \\ 2m + \left\lfloor \frac{6m}{11} \right\rfloor + 1 & \text{if } n = 6 \text{ and } m \mod 11 \in \{0,2,6\} \\ 2m + \left\lfloor \frac{6m}{11} \right\rfloor + 2 & \text{if } n = 6 \text{ and } m \mod 11 \notin \{0,2,6\} \\ 3m - \left\lfloor \frac{m}{18} \right\rfloor + 1 & \text{if } n = 7,\ m > 9,\ m \mod 18 \leq 9 \text{ and } m \mod 18 \neq 7 \\ 3m - \left\lfloor \frac{m}{18} \right\rfloor & \text{if } n = 7 \text{ and } (m \leq 9 \text{ or } m \mod 18 > 9 \text{ or } m \mod 18 = 7) \\ 3m + \left\lfloor \frac{m}{3} \right\rfloor & \text{if } n = 8 \text{ and } m \mod 3 = 1 \\ 3m + \left\lfloor \frac{m}{3} \right\rfloor + 1 & \text{if } n = 8 \text{ and } m \mod 3 \neq 1 \\ \left\lfloor \frac{(n+2)(m+2)}{3} \right\rfloor - 6 & \text{if } n \geq 9. \end{cases}$$

2.2.1 Fixed (small) height and width

We present here the technique of transfer matrices, a well-known method to solve many problems of this kind. We adapt it here to establish the 2-domination values for grids of fixed height and width. The technique follows a dynamic programming approach. One parameter of this approach (the number of states) is exponential in the number of lines, which is why the latter needs to be small enough: we want the computations we run on a computer to finish within a reasonable[7] time.

In this paragraph, we introduce the notion of column *states*, which enables us to 'enumerate' the partial (i.e. the first columns of) 2-dominating sets by remembering only their 'fingerprint' on their last column. This way, when we consider adding a new column, we do not need to remember the whole partial dominating set but only some information stored in the current column. This is possible because the domination problems we study have a **local characterisation**[8]: while the property is global (defined on a whole object, which can be arbitrarily large), it is possible to make local queries (querying neighbours at most at a fixed distance from the vertex we consider) in such a way that we can check if the global property is verified by querying local information at each vertex. This is most helpful for us because we can enumerate a dominating set column by column and, when looking at column j, we can forget columns with index less than $j-2$: designing our algorithm smartly, we do not recall the full columns with indices $j-1$ and $j-2$ but only the necessary information.

[7] i.e. less than the time until the end of the PhD... or more truthfully a few hours up to a couple of days on a machine with a hundred of cores

[8] We speak a bit about this notion in Chapter 3

Before defining the notions, we indicate that the number of lines n is supposed fixed. We show in Proposition 2.2 that all the states and compatibility relations we will define implicitly appear when the number of columns m is at least some number. We assume here that m is big enough in this respect, that is 8 for the 2-domination. We define $\mathcal{S} = \{$ STONE, NEED_ONE, OK $\}$ to be the set of **cell values**. STONE means that the cell belongs to the 2-dominating set D: in the rest, we refer to the cells of D as **'containing a stone'**. OK means that the previous and current columns suffice to 2-dominate the cell, before adding the next column. Finally, NEED_ONE means that the cell had so far one neighbour in D, so it needs a new one in the next column. If D is a 2-dominating set of cells of the grid $G_{n,m}$ let $f(D) \in (\mathcal{S}^n)^m$ be such that[9] $f(D)[i][j]$ is STONE if[10] $(j,i) \in D$, OK if at least two among $(j-1,i)$, $(j,i-1)$ and $(j,i+1)$ are in D, or NEED_ONE otherwise. Note that, since D is 2-dominating, a cell is NEED_ONE if exactly one among $(j-1,i)$, $(j,i-1)$ and $(j,i+1)$ is in D. Note that the value of a cell does not depend on the values of the cells of the next column(s). We also define, for $0 \leq j < n$, $f_j(D)$ to be the vector containing the values of column j of $f(D)$: for all $i \in [\![0, n-1]\!]$, $f_j(D)[i] = f(D)[i][j]$.

We now define the set of (column) **states** $\mathcal{V} = \cup_{0 \leq j < m} \{f_j(D) : D \text{ is 2-dominating}\}$. $\mathcal{V}$ is the set of states which appear in some 2-dominating set. Among these states, we define the set of **first states** $\mathcal{F} = \{f_0(D) : D \text{ is a 2-dominating set}\}$. Finally, we define the set of **end** (or **dominated**) **states** $\mathcal{E} = \{f_{m-1}(D) : D \text{ is a 2-dominating set}\}$. $\mathcal{F}$ is the set of states which can be the first column of a 2-dominating set, that is whose entries only depend on themselves and not on a presupposed previous column. $\mathcal{E}$ is the set of states which can be the last column of a dominating set because they do not need a next column to be 2-dominated: they are dominated by themselves and their previous column.

We now define the **relation of compatibility** R: we say that a state $S' \in \mathcal{V}$ is **compatible** with $S \in \mathcal{V}$, and write $S\mathrm{R}S'$ if there exist a 2-dominating set D and some $j \in [\![0, m-2]\!]$ such that $\mathrm{f}_j(D) = S$ and $\mathrm{f}_{j+1}(D) = S'$. Defining these states enables us to use the principles of dynamic programming: instead of enumerating all possible 2-dominating sets, we realise that the information conveyed in $f(D)$ is enough, and that we only need the information at a column j to continue to column $j+1$. In particular, we do not need to know what happened in previous columns with indices less than j. This corresponds to the notion of *nearest-neighbour* recoding in the world of SFTs which we will define in Chapter 3: a translation *subshift* into an equivalent one on a different alphabet such that the forbidden patterns are only of size two. In our case, it means that we only forbid couples of columns states S, S' when S' cannot be put just after S.

To illustrate these concepts, we give the rules defining the sets $\mathcal{V}$, $\mathcal{F}$, $\mathcal{E}$ and the relation R. These explicit definitions correspond exactly to the implicit way we defined these concepts in the previous paragraph. It is easy to convince ourselves of this fact as they are translations of the implicit definitions of $\mathcal{V}$, $\mathcal{F}$, $\mathcal{E}$ and R considered along with the definition of the cell values STONE, NEED_ONE and OK. The concept of states and compatibility relation are shown on Figure 2.6.

[9] $f(D)$ is considered as a matrix so i is the index for the lines and appears first.

[10] Here (j,i) are coordinates, hence their inverted order compared to indexing $f(D)$.

$S \in \mathcal{V}$ if and only if for all $i \in [\![0, n-1]\!]$:
- if $S[i] =$ NEED_ONE then at most one among $S[i-1]$, $S[i+1]$ is STONE;
- if $S[i] =$ OK then at least one among $S[i-1]$ and $S[i+1]$ is STONE.

$S \in \mathcal{F}$ if and only if for all $i \in [\![0, n-1]\!]$:
- if $S[i] =$ NEED_ONE then exactly one among $S[i-1]$, $S[i+1]$ is STONE;
- if $S[i] =$ OK then both $S[i-1]$ and $S[i+1]$ are STONE (so $1 \leq i < m-1$).

A state S belongs to $\mathcal{E}$ if and only if $S \in \mathcal{V}$ and none of its entries is NEED_ONE.

Finally, $S\mathrm{R}S'$ if and only if for all $i \in [\![0, n-1]\!]$:

- if $S[i] =$ NEED_ONE then $S'[i] =$ STONE;
- if $S'[i] =$ NEED_ONE then exactly one among $S'[i-1]$, $S'[i+1]$ and $S[i]$ is STONE;
- if $S'[i] =$ OK then at least two among $S'[i-1]$, $S'[i+1]$ and $S[i]$ are STONE.

. . .	OK	N	*	N	*	N	*	*	. . .
	*	*	*	OK	*	OK	N	*	
	OK	*	OK	*	OK	*	N	*	
	*	OK	*	OK	N	*	OK	*	
	N	*	OK	*	*	OK	*	OK	

Figure 2.6: Illustration of the states and compatibility relations for the 2-domination problem. 'N' corresponds to the state value NEED_ONE and '*' corresponds to STONE. The data on one column only depends on this column and the columns at its left.

We said above that the number of states is exponential in the number of lines. We can in fact compute the growth rate of the number of states thanks to a concept named the *Rauzy graph* of a language. We will explain in Section 2.2.3. It allows us to find the number of states in our technique.

Fact 2.1. *There are $\Theta(x_0^n) \approx \Theta(2.485584^n)$ states in $\mathcal{V}$ for n lines, where x_0 is the real root of the polynomial $x^3 - 2x^2 - 3$.*

Notation 2.8. If V is a vector, then we denote by $|V|_{foo}$ the number of its entries equal to *foo*.

Claim 2.1. *Let F be the vector of size $|\mathcal{V}|$ such that $F[S] = |S|_{\text{STONE}}$ if $S \in \mathcal{F}$ or $+\infty$ otherwise.*
Let E be the vector of size $|\mathcal{V}|$ such that $E[S] = 0$ if $S \in \mathcal{E}$ or $+\infty$ otherwise.
Let T be the square matrix with $|\mathcal{V}|$ lines such that $T[S][S'] = |S'|_{\text{STONE}}$ if $S\mathrm{R}S'$ or $+\infty$ otherwise.
Then for any $m > 0$, $\gamma_2(n, m) = F^T T^{m-1} E$.
(We recall that the products of matrices are done in the $(\min, +)$-algebra.)

Proof. Let $j \geq 1$. $V_j = F^T T^{j-1}$ is a vector such that if $S \in \mathcal{V}$ then $V_j[S]$ is the minimum size of a set X which 2-dominates the subgrid with n lines and $j-1$ columns, and such that the last column of X is in state S. However, we are interested in a 2-dominating set, therefore the last column (assumed to be in state S) should be 2-dominated as well. Thus $V_m E = \min_{S \in \mathcal{E}} V_m[S]$ gives us the minimum size of any 2-dominating set. □

This claim leads to a simple algorithm to compute $\gamma_2(n, m)$: generate the different sets and the compatibility relation, and then compute the exponentiation of matrices, and two matrix-vector products. The matrix T is a transfer matrix, whose exponentiation propagates the fact of being 2-dominated one column further. This is enough to compute the 2-domination numbers for fixed n and m, but our goal here is to find all the numbers for fixed n and arbitrary m. The next section fills this hole.

2.2.2 Fixed number of lines but arbitrary number of columns

The method we have just presented works for a fixed number of lines. Since it is linear in the number of columns, this number must be finite. We use here a less known method to obtain values for arbitrary width of the grid, establishing recurrence relations whose existence are guaranteed by some properties of the transfer matrices. The computations lead to closed formulas for a fixed number of lines, hence we obtain a constant-time algorithm at the end, exploiting this formula. We also, on a theoretical point, make the connection between the effectiveness of the method and the *primitivity* of the transfer matrix we use.

By observing that the transfer matrix we manipulate is *primitive*, we show that this periodicity was expected. That is, even without finding the recurrence relation, it is possible to prove that the formula follows some recurrence relation without running any program. Furthermore, this observation extends to any problem like this one. We can, for instance, guarantee that the distance-three-domination (any vertex not in S must have a neighbour at distance at most three in S) number satisfies a recurrence relation. The same applies to all the dominating problems we study in this chapter.

Definition 2.9. A matrix M is **primitive** (in the $(\min, +)$-algebra)[11] when there exists an integer $k > 0$ such that $\max_{i,j}(\{M^k[i][j]\}) < +\infty$.

Proposition 2.2. *T is primitive.*

This proposition means that given two states S_1 and S_2, it is always possible to find some 2-dominating set D and some i_1 and $i_2 \leq i_1 + k$ such that $f_{i_1}(D) = S_1$ and $f_{i_2}(D) = S_2$. This can be related to the notion of *irreducibility* of Markov chains. As we mentioned above, it certifies that any possible state appears in some dominating set of a grid if there are k columns separating it from the special first states and the special end states. So any state appears if there are at least $2k+1$ columns. Now, to be sure to see every compatibility relations, we must add another column: each possible state appears at the $k+1^{\text{th}}$ column, and any next state must be separated from the end states by k columns. Therefore, $m \geq 2k+2$ is enough, which means 8 for the 2-domination (see the proof just below).

[11] (almost) last reminder that we do not work in the standard algebra

Proof. Let $S_0, S_2 \in \mathcal{V}$. Let S_* be the state whose entries all are STONE. There exists some $S_1 \in \mathcal{V}$ such that $S_0 \mathrm{R} S_*$, $S_* \mathrm{R} S_1$ and $S_1 \mathrm{R} S_2$. We leave the construction of S_1 to the reader: put the necessary stones and fill the rest accordingly. We conclude that $T^3 < +\infty$. □

We now prove an interesting property that primitive matrices have, in the (min, +)-algebra: the series of their powers satisfy some recurrence relation. This is true for any primitive matrix in this algebra.

Theorem 2.3. *Let M be a primitive matrix with coefficients in $\mathbb{N} \cup \{+\infty\}$. Let k be such that $\max(\{M^k[i][j]\}) < +\infty$. Then there exist some l_0, p and r such that for all $l \geq l_0$, $M^{l+r} = M^r + p$, where $M^r + p$ means[12] that we add p to every element of M^r.*

Proof. Notice that, since $\max(\{M^k[i][j]\}) < +\infty$, each line of M has at least one element different from $+\infty$. The same goes for the columns. We denote by α the maximum value of M^k. Let $l > k$. For every i and j,

$$M^l[i][j] = \min_u(M^{l-k}[i][u] + M^k[u][j]) \leq \min_u(M^{l-k}[i][u] + \alpha) \leq \min_u(M^{l-k}[i][u]) + \alpha. \quad (2.1)$$

This quantity is finite since M^{l-k}, like M, has in each row and each column one element different from $+\infty$.

Now let us bound the value of $M^l[i][j]$ from below:

$$M^l[i][j] = \min_u(M^{l-k}[i][u] + M^k[u][j]) \geq \min_u(M^{l-k}[i][u]) \quad (2.2)$$

By subtracting Equation (2.2) from Equation (2.1), we obtain

$$0 \leq \max_{i,j}(M^l[i][j]) - \min_{i,j}(M^l[i][j]) \leq \alpha.$$

This allows us, for any $l > k$, to define $M'_l \in [\![0; \alpha]\!]^{nm}$ by the following decomposition:

$$M^l = \min(M^l) + M'_l.$$

This means that each M^l is decomposed into a matrix which has all its coefficients equal to $\min(M^l)$ plus a matrix whose coefficients are bounded by a value independent from l. Since there are finitely many matrices in $[\![0; \alpha]\!]^{nm}$, we conclude that there are two equal matrices $M_{l_0} = M_{l_1}$ for $l_1 > l_0 > k$. By letting $p = \min(M^{l_1}) - \min(M^{l_0})$ we obtain

$$M^{l_1} = M^{l_0} + p. \quad (2.3)$$

Now we choose $r = l_1 - l_0$, and for $l \geq l_0$:

$$M^{l+r} = M^{l-l_0} M^{l_0+r} = M^{l-l_0}(M^{l_0} + p) = M^{l-l_0} M^{l_0} + p = M^l + p.$$

Note that in our tropical algebra, if A, B and C are matrices, and '+' denotes, as before, the standard plus operation, then $A(B + C) = AB + C$. □

A direct implication of Theorem 2.3 is that the transfer matrix T verifies some recurrence relation. Thanks to Claim 2.1, the relations we obtain for the transfer matrix T directly apply to the 2-domination number. Here are the relations we obtain for $n \leq 12$:

[12] We recall that the replacement of $(+, \times)$ by $(\min, +)$ only applies in the internal operations of products between matrices and matrices or vectors.

- $\forall m \geq 3, \gamma_2(1,m) = \gamma_2(1,m-2) + 1$;
- $\forall m \geq 3, \gamma_2(2,m) = \gamma_2(2,m-1) + 1$;
- $\forall m \geq 5, \gamma_2(3,m) = \gamma_2(3,m-3) + 4$;
- $\forall m \geq 8, \gamma_2(4,m) = \gamma_2(4,m-4) + 7$;
- $\forall m \geq 14, \gamma_2(5,m) = \gamma_2(5,m-7)+15$;
- $\forall m \geq 20, \gamma_2(6,m) = \gamma_2(6,m-11)+28$;
- $\forall m \geq 31, \gamma_2(7,m) = \gamma_2(7,m-18)+53$;
- $\forall m \geq 16, \gamma_2(8,m) = \gamma_2(8,m-3)+10$;
- $\forall m \geq 17, \gamma_2(9,m) = \gamma_2(9,m-3)+11$;
- $\forall m \geq 14, \gamma_2(10,m) = \gamma_2(10,m-1)+4$;
- $\forall m \geq 16, \gamma_2(11,m) = \gamma_2(11,m-3)+13$;
- $\forall m \geq 17, \gamma_2(12,m) = \gamma_2(12,m-3)+14$.

These relations were obtained by running our program on a computer. For instance, finding the relation for $n = 12$ takes around 17.5 seconds on a personal laptop using only one CPU, and it uses 56Mib of RAM. There are around 26.5k states and 10M compatible pairs between states.

Thanks to these relations, and to the first values we obtain for each n, we deduce the formulas for $\gamma_2(n,m)$, for $1 \leq n \leq 12$. For instance, for $n = 5$ we only need to know the recurrence relation, plus the first twelve[13] values. We stopped here at $n = 12$ here because the method for arbitrarily large n works from $n \geq 13$.

We may observe that these figures are the ones reflecting the periodicity of the transfer matrix: for $n = 12$, this begins when $m \geq 17$. However, once we have this relation we can 'go back in time' and check from what point the relations begins to be valid for the γ_2 numbers we computed. By doing so, we find for instance that the recurrence relation for $\gamma_2(12,m)$ is actually valid from $m = 13$ instead of 17.

Now thanks to Theorem 2.3 we can prove a meta-theorem for a class of problems.

Theorem 2.4. *Any minimisation problem on grids which admits a local characterisation and a primitive transfer matrix has, when the number of lines is fixed, a closed formula for answer.*

Proof. It suffices to apply the same method as here. The recurrence relation the transfer matrix satisfies guarantees a recurrence relation on the parameter studied, hence the answer is a closed-form expression. This expression involves basic arithmetic operations (+,-,*,/) and the remainder of a Euclidean division. □

Remark 2.1. We can notice that any problem to which we can associate an SFT with the *block-gluing* property (see Chapter 3) will have a primitive transfer matrix and hence falls into the hypothesis of Theorem 2.4.

This method can work for other classes of graphs, as long as we find an 'acyclic' way of enumerating parts of the dominating sets, like for the fasciagraphs defined by Bouznif et al. [5]. Our method, and the theory behind stated by Theorem 2.4 can for instance be used to prove parts of what they investigate in [5]. We used it to prove that some local problems with a primitivity condition admits a closed formula as answer, hence can be answered in constant time. In fact, the same applies to rotagraphs: instead of computing $F^T T^{m-1} E$ (for a fasciagraph, with a first and last 'column'), we compute

[13] $20 = 14 + 6$

$\min_{S \in \mathcal{V}}(T^m[S][S]$ (for $m > 1$): the first and last state must be the same. In their paper, they prove similar results, using a different technique and a result similar to the one we prove for the recurrence of powers of primitive matrices.

The method we described can work also in these structures for other types of problems. We show in Chapter 3 that we can also count some types of dominating sets, using different arguments (and working in the standard algebra). Basically, this approach will work when the problem has some sort of local characterisation or local property.

2.2.3 The number of states

We present here a technique to find the growth rate of the number of words of a *factorial* language: a set of words which is defined by forbidding specific patterns to be *factors*. This means for instance that we choose to allow only words which do not contain 'an' or 'on': 'tomato' and 'cherry' would be authorised whereas 'banana' and 'cinnamon' would not. The method consists in constructing the Rauzy graph of the language, and take its largest eigenvalue, which turns out to be the desired growth rate.

Let $\mathcal{L}$ be a language of words forbidding patterns of sizes up to k. Without loss of generality, we may assume that all the forbidden factors are of length equal to k. The (directed) Rauzy graph $R_i(\mathcal{L}) = (V_i, E_i)$ of $\mathcal{L}$ of order i is defined by: V_i is the set of factors of size i appearing in words of $\mathcal{L}$, and $(u, v) \in E_i$ (note that the relation is not symmetric) when there exist u' and two letters a and b such that $u = au'$ and $v = u'b$. We will refer to this relation as R_i in the rest of this subsection. The way this graph works is similar to the one in Section 3.5.4, in which we use transfer matrices in the standard algebra to count dominating sets. The number of arcs going to some vertex u of a Rauzy graph is the number of ways to obtain this factor by adding a letter to a smaller authorised factor of $\mathcal{L}$. This way, $R_i^n[u]$ counts the number of words of size n the suffix of which is u, forbidding all forbidden factors of $\mathcal{L}$ of size up to $i + 1$. In the case when the matrix is primitive (in the standard algebra), the Perron-Frobenius theorem states that it admits a largest eigenvalue which is positive and simple. Let us call it λ_i for the graph R_i. This λ_i is a lower bound on the growth rate. However, for $i \geq k - 1$, this λ_i becomes the growth rate of the number of words of the language: all forbidden factors of size up to $k - 1 + 1 = k$ are guaranteed not to appear.

Unfortunately here our languages of the valid states are not factorial: for the first and last lines of a state, we forbid specific factors which are not forbidden in the 'middle' of the word. Still, if we remove this limitation, the factorial language which we would obtain would have the same growth rate as our language of real valid states. As we showed in Section 2.2.2, the matrices of compatibility between the states are primitive. Therefore we could compute the matrices of the associated factorial languages (without the limitations at the extremities of the states). However, we proceeded here by smartly reusing the existing code: we take, as vertices of the Rauzy graph R_i, the factors of size i of our languages which are far enough away from the beginning and from the end of the states. Their behaviour is similar to the behaviour of the middle parts of the states, if the distance to $i = 0$ and $i = n - 1$ is big enough. With the help of the Sage software, we could compute an approximate form of the λ_i's by providing the adjacency matrix of the Rauzy graph, computed by our program. We also obtain a polynomial there are root of (the minimal polynomial of the matrix). We give each time the minimal polynomial of the

Rauzy graph, or more precisely this polynomial divided by its lowest degree monomial.

2.2.4 Arbitrary height

We adapt here the method Gonçalves et al. introduced in [21]. This method is much less known than the previous ones, and to our knowledge it is the first time it is adapted. Quite some people provided formulas for some domination problems when the number of lines is small and fixed, but no one provided a formula for the real 2D case.

The idea is to assume that for a sufficiently large grid, there are always dominating sets of minimum size in which the positions of the stones would be a projection of an optimal dominating set (i.e. one with minimum ratio of stones, one fifth here) of $\mathbb{Z}^2$, except on a fixed-height border of the grid. The border are special because not every cell has 4 neighbours at the frontier of a finite grid. This assumption turns out to also be true for the 2-domination problem. The height of the border which needs some rearrangement is a constant: it does not depend on the size of the grid we consider (provided n and m are large enough).

We need to count how many stones we need to make up for the problematic fact that the cells of the border have fewer than four neighbours. We use the dual concept of *loss* to integrate this number to the number of stones at the centre of the grid. The loss denotes how much 'influence' produced by the stones of a 2-dominating set is wasted. For instance, two neighbouring stones would cause a loss of 2: each stone cell dominates its stone neighbour which did not need to be dominated by another cell since it has a stone. Instead of computing the minimum number of stones needed, we will compute a lower bound on the minimum loss possible on the border. It happens that this lower bound gives us a direct lower bound on the 2-domination number, and that these bounds are sharp.

More formally, given a 2-dominating set D of the $n \times m$ grid, we define the **loss** to be

$$\ell(D, n, m) = 4|D| - 2(nm - |D|).$$

The idea behind this formula is simple: each stone contributes to the domination of its four neighbours, and each cell not in D should be dominated twice. The difference between these two quantities is the influence of stones that was 'lost', or 'wasted', i.e. not necessary. The loss function should have several characteristics: while it should be 'easy' to compute, it should also be reversible so that given the loss, we can deduce the 2-domination number.

The loss for the 2-domination problem can be computed in the following way. Let us consider a 2-dominating set S and a cell which belongs to S. If it is a corner of the grid, it induces a loss of 2 because of this; else, if it is on the border, it induces a loss of 1. If a stone cell has p neighbours also in S, then its loss is increased by p. Now if a cell does not belong to S and has p neighbours in this set, then it has a loss of $p-2$. We illustrate these computations in Figure 2.7. For instance, the top-left cell has a loss of two because it is a corner: its top and left neighbours do not exist. Two cells at the right, we have a loss of two: this cell in on the border and has a neighbour in S. The cell with a three on the one before last column is in S and has $p = 3$ neighbours in S. Last example: the cell on the second column and one before last line has a loss of two: it does not belong to S

but has $p = 4$ neighbours in S: two of them are enough to 2-dominate it, hence it has a loss of two.

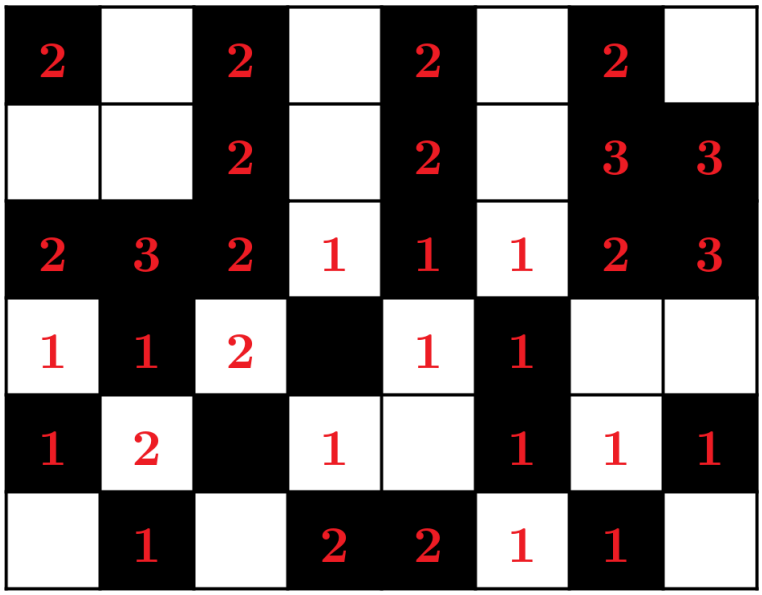


Figure 2.7: Illustration of the local computation of the loss for the 2-domination problem. The red numbers indicate the loss induced by the cell they are on. The cells with no number have a loss of 0.

Notation 2.10. We denote by $\ell(n, m)$ the **minimum possible loss** over every 2-dominating set of $G_{n,m}$.

Here, we can indeed reverse the formula as we wanted: $|D| = (2nm + \ell(D, n, m))/6$. We then obtain

$$\gamma_2(n, m) = \frac{2nm + \ell(n, m)}{6}. \tag{2.4}$$

The method solves the problem, that is the lower bound we obtain by the loss technique matches the 2-domination numbers: we prove later that it is also an upper bound. Computing the minimum loss over a big grid seems very hard, but we managed to obtain the right values by computing a lower bound for $\ell(n, m)$ which happens to be equal to it: we achieve this by computing the loss only on the border of size h (where $2h < \min(n, m)$, see Figure 2.8).

Definition 2.11 (see Figure 2.8)**.** The **border** of height h of an $n \times m$ grid is the set of cells (j, i) such that either $\min(i, n - 1 - i) < h$ or $\min(j, m - 1 - j) < h$. We define the **corners** as the four connected parts of the grid composed of cells (j, i) such that both $\min(i, n - 1 - i) < h$ and $\min(j, m - 1 - j) < h$. The remaining four connected parts of the border are called the **bands**.

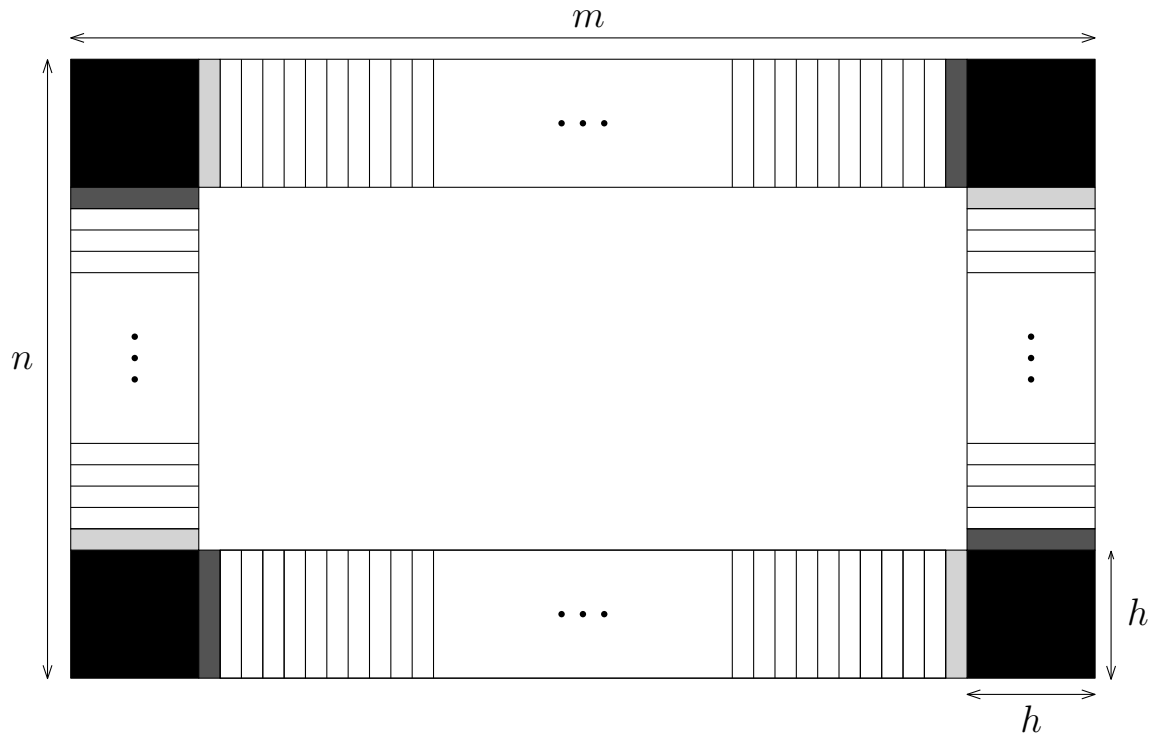


Figure 2.8: The borders of size h of the grid. The four parts coloured in black are the *corners*, and the white parts are the *bands*. The grey cells belong to both the bands and the corners: the ones filled with light grey are the output of a band and input of the corner next to it; the ones in dark grey are the output of a corner and input of the following band.

Once again, we use an algorithm which is faster than an exhaustive search over the dominating sets by working with the notion of states. However, we need to adapt the sets $\mathcal{V}, \mathcal{F}, \mathcal{E}$ and the relation R we worked with.

Let us begin with computing the loss over the bottom band. We will, using transfer matrices as in the beginning of the chapter, compute the loss column by column. We remember, for each state placed on each column, what the minimum possible loss is so that the column is in this state. Thus we need to adapt the sets $\mathcal{V}, \mathcal{E}$ and the relation R. In the following we assume, as mentioned before, that $2h < \min(n, m)$.

Notation 2.12. We define the function $\hat{f}_j$ such that, if D is a 2-dominating set, $\hat{f}_j(D) = f_j(D)[0], \cdots f_j(D)[h-1]$. $\hat{f}_j$ consists of the bottom h lines of f_j. As previously, $\hat{f}_j(D)$ denotes the column j of $\hat{f}(D)$.

We continue by defining the set of **almost valid states**

$$\mathcal{V}_a = \bigcup_{0 \leq j < m} \{\hat{f}_j(D) : D \text{ is a 2-dominating set}\}.$$

This set contains the states we will enumerate to compute the loss. It is in fact almost the same set as $\mathcal{V}$: $S \in \mathcal{V}_a$ if and only if, for $i \in [\![0, h-1]\!]$:

- if $S[i] =$ NEED_ONE then at most one among $S[i-1]$, $S[i+1]$ is STONE;
- if $S[i] =$ OK then $i = 0$ or at least one among $S[i-1]$ and $S[i+1]$ is STONE.

We need neither first states nor dominated states to compute the loss, so we will neither define a set $\mathcal{F}_a$ nor $\mathcal{E}_a$. We define the relation of **almost-compatibility**, which differs a bit from R. The main difference with the relation of compatibility we used earlier concerns the top line (of the bottom border) $i = 0$: the first cell will have a neighbour above it, in the centre of the grid, so we need to consider the case when this neighbour has a stone. More explicitly, if $S, S' \in \mathcal{V}_a$, $S\mathrm{R}_a S'$ if and only if for $i \in [\![0, n-1]\!]$:

- if $S[i] =$ NEED_ONE $S'[i] =$ STONE;
- if $S'[i] =$ NEED_ONE and $i \neq 0$ then exactly one among $S'[i-1]$, $S'[i+1]$ and $S[i]$ is STONE;
- if $S'[0] =$ NEED_ONE then at most one among $S'[1]$ and $S[0]$ is STONE;
- if $S'[i] =$ OK and $i \neq 0$ then at least two among $S'[i-1]$, $S'[i+1]$ and $S[i]$ are STONE;
- if $S'[0] =$ OK then at least one among $S'[1]$ and $S[0]$ is STONE.

As said just before, we allow a state whose first cell has value NEED_ONE not to be dominated by the next column: in this case, we consider that its neighbour above dominates it. This is a conservative assumption: we may underestimate the loss this way, but this is fine since we compute a lower bound on the loss.

We use again the exponentiation of a transfer matrix to compute the minimum loss over some border of a grid. We define the **band matrix T_a** such that $T_\mathrm{a}[S][S']$ contains the loss induced by putting state S' after state S. By exponentiating the matrix T_a we can compute the minimum loss over a border, excluding the loss induced by the first state alone on itself.

The next step is to compute the loss for the corners. A corner is composed of an h by h square, plus an input column and an output column. Let us consider the bottom right corner of Figure 2.8. The last column of the bottom band is coloured in light grey: it is the input column of the square (and the output column of the band). At the other side of the square, the horizontal 'column' filled with dark grey is the output column of the square (and the input column of the next border). Suppose that the input column of the square is in state A and its output column is in state B. The **loss over the corner** is the sum of:

- the loss on the corner by A, B and the corner itself;
- the loss on B by the corner and B itself;
- the loss on A by the corner.

The explanation is simple: the input state was fixed by the loss computation on the band (so its loss so far was already counted) and the corner provides the first state for the next band (so we have to compute its loss so far). Similarly to T_a, we define the **corner matrix C_a** for a corner: $C_\mathrm{a}[S][S']$ contains the minimum loss over a corner whose input state is S and output state is S' as defined just before (we do not count the loss induced by S alone on itself).

Lemma 2.5. *The minimum loss over the border is*

$$\min_{S \in \mathcal{V}_a} ((T_\mathrm{a}^{m-2h-1} C_\mathrm{a} T_\mathrm{a}^{n-2h-1} C_\mathrm{a})^2[S][S]).$$

Before diving into the proof, we mention that, since we are in the (min,+)-algebra, the formula in the above lemma can be rewritten as $\mathrm{Tr}((T_{\mathrm{a}}^{m-2h-1}C_{\mathrm{a}}T_{\mathrm{a}}^{n-2h-1}C_{\mathrm{a}})^2)$.

Proof. T_{a}^{m-2h-1} is the minimum loss over a band starting on the output column of the bottom-left corner and ending on the input column of the bottom-right corner. Hence $T_{\mathrm{a}}^{m-2h-1}C_{\mathrm{a}}$ means computing the minimum loss on the bottom band we have just described, and extending it to the output state of the bottom right corner. As mentioned above, in the corner loss we take the input state of the corner as it is (which is exactly what T^{m-2h-1} provides: the loss on the last state by itself and its preceding column was already computed). Since C_{a} includes the loss induced by the corner onto the output state of the corner, $T_{\mathrm{a}}^{m-2h-1}C_{\mathrm{a}}T_{\mathrm{a}}^{n-2h-1}$ extends the loss to the right band. Now, $T_{\mathrm{a}}^{m-2h-1}C_{\mathrm{a}}T_{\mathrm{a}}^{n-2h-1}C_{\mathrm{a}}$ corresponds to the loss from the output of the bottom-left corner to the output of the top-right corner, that is the loss of half the border. By squaring this matrix, we obtain the minimum losses over the whole border of the grid: $(T_{\mathrm{a}}^{m-2h-1}C_{\mathrm{a}}T_{\mathrm{a}}^{n-2h-1})^2[S][S]$ means that we compute the minimum loss by starting from the leftmost column of the bottom band, excluding the bottom-left corner, which we suppose is in state S, and leaving it in state S after the bottom-left corner computation, after going through all the band in counter-clockwise direction. □

Lemma 2.6. *Let $\ell_h(n,m)$ be the minimum loss over the border of height h on a $n\times m$-grid for the 2-domination. Then $\lceil \frac{2nm+\ell_h(n,m)}{6} \rceil$ is a lower bound on $\gamma_2(n,m)$.*

Proof. When computing the loss over some part of the grid, we obtain a lower bound on the loss of the whole grid. The same applies when we compute the minimum loss. We then replace $\ell(n,m)$ by $\ell_h(n,m)$ in Equation (2.4). Since $\gamma_2(n,m)$ is an integer, we can take the upper bound. □

We now try to find some h for which the minimum loss over the border of height h matches the minimum loss over the grid. In the rest of this section, **we consider that $\mathbf{h=6}$**, unless explicitly specified otherwise. This value is sufficient to obtain the correct bounds with our program. Here again, we have the problem of computing the minimum loss over borders of arbitrary widths. However, we may notice that, if we let $H(n,m) = (T_{\mathrm{a}}^{m-13}C_{\mathrm{a}}T_{\mathrm{a}}^{n-13})^2$, then there exist some j_0, k and p such that $\forall\, r \geq r_0, T_{\mathrm{a}}^{r+k} = T_{\mathrm{a}}^{r}+p$, so that $H(n+i,m+j) = H(n,m)+2(i+j)p$ for all $n,m \geq 13+r_0$. Indeed, the matrix T_{a} is primitive for the same reasons as for the transfer matrix of Theorem 2.3. The factor 2 before $(i+j)p$ comes from the fact that the matrix T_{a} appears twice for the horizontal bands and twice for the vertical onew.

With the program considering a band of height 6, we find that

$$\text{for all } r \geq 20,\ T_{\mathrm{a}}^{r+3} = T_{\mathrm{a}}^{r} + 6. \tag{2.5}$$

Note that if we choose a border of height 7, the same recurrence relation on T_{a} is true from $r \geq 17$. From Equation (2.5) and because $\ell(n,m)$ is symmetric, we deduce:

Claim 2.2. *$\ell_6(n+3,m) = \ell_6(n,m+3) = \ell_6(n,m) + 12$ for every $n,m \geq 33$.*

The recurrence begins at 33 because in Lemma 2.5, T_{a} is put to the power $n-2h-1$ for instance, so that to have $n-2h-1 \geq 20$ we need to have $n \geq 20+2h+1 = 33$ for $h=6$. Despite Equation (2.5), it is 12 that we add and not 6, because in the formula

in Lemma 2.5: as we wrote just above, there are two vertical bands and two horizontal ones. The formula can be rewritten as $\ell_6(n,m) = \ell_6(n,m-3)+12$ for every $n \geq 36$. So, to complete the proof, we must check that the formula holds for $n, m \leq 36$ to initialise the recurrence. We check with the method for fixed height the values for $9 \leq n \leq 12$ because the loss method can only by used when $2h < \min(n,m)$. This means that, when the height is 6, we must check the γ_2 values for $n < 13$ by another method. Then, we can compute our lower bound for $\gamma_2(n,m)$ values for $13 \leq n, m \leq 36$ using the loss method with height 6. These values match the upper bound we show below, so we obtain the values of $\gamma_2(n,m)$.

Now the recurrence relation on T_{a}^r from Equation (2.5) and exploited in Claim 2.2 completes the proof of the theorem. Indeed, if $13 \leq n, m \leq 33$ then for all $k \in \mathbb{N}$:

$$\begin{aligned}
\gamma_2(n, m+3k) \geq \frac{2n(m+3k)+\ell(n,m+3k)}{6} &\geq \frac{2n(m+3k)+\ell_6(n,m+3k)}{6} \\
&\geq \frac{2nm+\ell_6(n,m)}{6} + nk + 2k \\
&\geq \left\lfloor \frac{(n+2)(m+2)}{3} - 6 \right\rfloor + nk + 2k \\
&\geq \left\lfloor \frac{(n+2)(m+3k+2)}{3} - 6 \right\rfloor.
\end{aligned}$$

The first inequality comes from Lemma 2.6. The transition from the first line to the second one comes from Claim 2.2. The transition from the second to the third line comes from the fact that we checked values of γ_2 for n and m between 13 and 33 and they match the equality we use to substitute $(2nm + \ell_6(n,m))/6$. This proves the lower bound for every $13 \leq n \leq 36$ and $m \in \mathbb{N}$. To prove it for $n > 33$, it suffices to do the exact same computation, namely computing $\gamma_2(n+3k, m)$ for any $n \geq 36$ and any $m \geq 2 \cdot 6 + 1 = 13$.

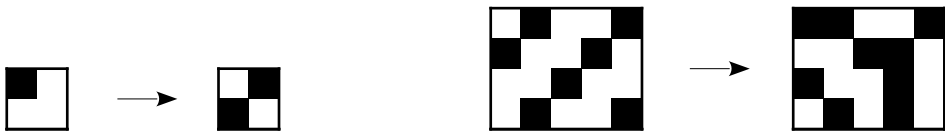


Figure 2.10: The two rules used to convert a restriction of an optimal 2-dominating set of $\mathbb{Z}^2$ into optimal 2-dominating set of a rectangle. For each corner, if one of the two patterns before the arrows appear, we replace them by the version on the right of the arrow.

To show that this bound is sharp, we show that our lower bound is also an upper bound, by giving general 2-dominating sets of the right sizes. To construct these 2-dominating sets we consider, for the infinite grid $\mathbb{Z}^2$, the 2-dominating set $D = \{(j,i) : i+j \mod 3 = 0\}$ and its rotations. We then take all the different restrictions of these 2-dominating sets for $\mathbb{Z}^2$ into a finite $n \times m$ grid. For each restriction, we modify each corner of size 6 according to two rules which depend on the pattern of that corner. The two rules are shown in Figure 2.10. A rule corresponds to removing some cells and adding some other cells to D in that corner. For instance, Rule 1 could be stated as follows: if the cell at the angle of the grid is in the dominated set, we remove it from the set and

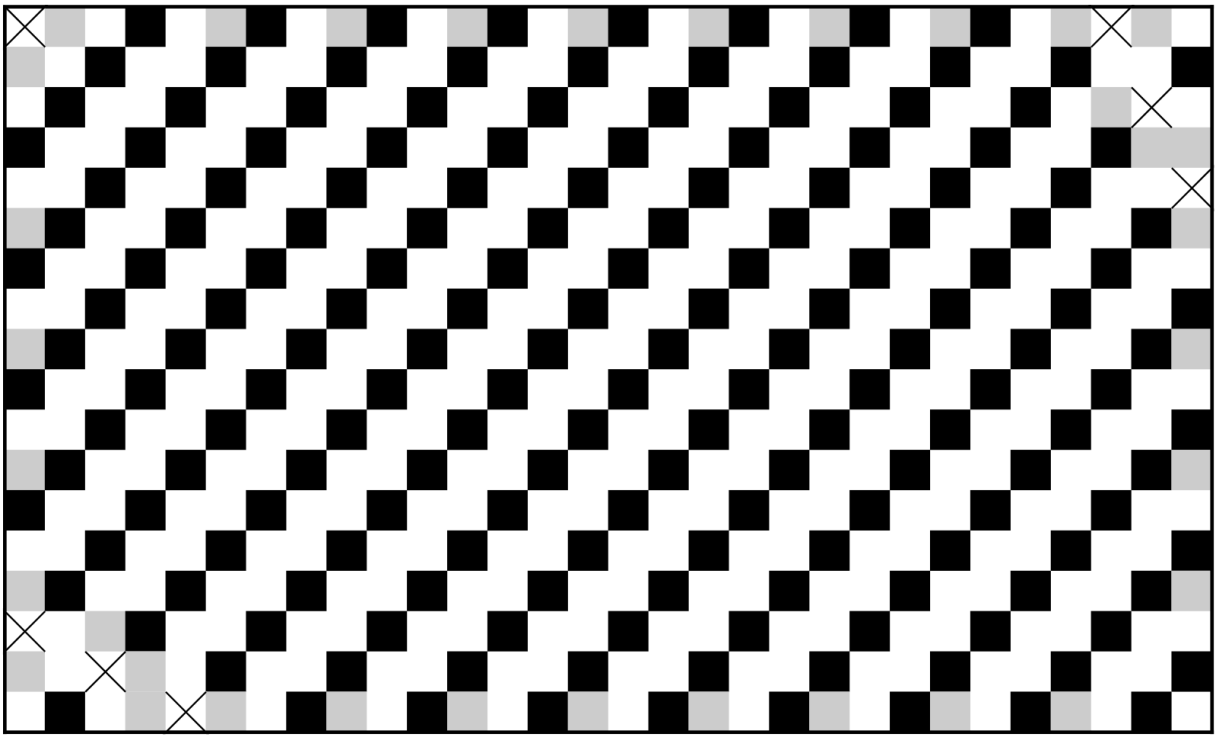


Figure 2.9: Example of an optimal 2-dominating set D on a 18×30 grid. D is the set of cells which are filled with grey or black. The black cells and the cells with a cross constitute the projection of a minimal 2-dominating set on the grid $\mathbb{Z}^2$.

add instead its two neighbours. Finally, we put a stone on the cells of the first and last rows and columns which are not 2-dominated. One can show that for $13 < n \leq m$ one of the resulting 2-dominating $D_{n,m}$ set has the right size. We can see an example of such a $D_{n,m}$ for a 18×30 grid in Figure 2.9. The first rule is used in the top-left corner and the second rule is used in the top right and bottom left corner. No modifications need to be done in the bottom right corner. By counting the number of stones in the regular pattern (black and crossed cells in Figure 2.9), removing the number of crossed cells, and adding the number of grey cells, we get $D = \frac{nm+2n+2m}{3} - 5$, which is equal to the number in Theorem 2.1 when n and m are multiple of 3.

The grid we show is of size 18×30, but it extends immediately to any $n \times m$ grid when n and m are both greater than 14 and multiple of 3. Applying the same method for $14 \leq n, m$ when the two numbers have other congruences modulo 3 leads to 2-dominating sets having the right size.

2.3 Adaptation to other problems and results

In this section we explain the adaptations needed to make the method presented in the previous section work for one other type of domination, namely the *Roman domination*. We will give the corresponding theorem obtained by applying this method. We will also give partial results on some dominations problems, namely the *distance-two-domination*

and the *total domination*, for which we could not get the method to fully work. We will conjecture the possible reasons explaining the partial failure of the method to obtain a closed formula for arbitrary heights and widths of the grid. We also give a lower bound for the total domination.

2.3.1 The Roman domination

We explain here how the code was adapted to the Roman domination to obtain the following result.

Theorem 2.7 ([42]). *The Roman-domination number is such that, for all* $1 \leq n \leq m$,

$$\gamma_{\mathrm{R}}(n,m) = \begin{cases} \left\lceil \frac{2m}{3} \right\rceil & \text{if } n = 1 \\ m+1 & \text{if } n = 2 \\ \left\lceil \frac{3m}{2} \right\rceil & \text{if } n = 3 \text{ and } m \mod 4 = 1 \\ \left\lceil \frac{3m}{2} \right\rceil + 1 & \text{if } n = 3 \text{ and } m \mod 4 \neq 1 \\ 2m+1 & \text{if } n = 4 \text{ and } m = 5 \\ 2m & \text{if } n = 4 \text{ and } m > 5 \\ \left\lfloor \frac{12m}{5} \right\rfloor + 2 & \text{if } n = 5 \\ \left\lfloor \frac{14m}{5} \right\rfloor + 2 & \text{if } n = 6 \text{ and } m \mod 5 \in \{0,3,4\} \\ \left\lfloor \frac{14m}{5} \right\rfloor + 3 & \text{if } n = 6 \text{ and } m \mod 5 \notin \{0,3,4\} \\ \left\lfloor \frac{16m}{5} \right\rfloor + 2 & \text{if } n = 7 \text{ and } m = 7 \text{ or } m \mod 5 = 0 \\ \left\lfloor \frac{16m}{5} \right\rfloor + 3 & \text{if } n = 7 \text{ and } (m > 7 \text{ and } m \mod 5 \neq 0) \\ \left\lfloor \frac{18m}{5} \right\rfloor + 4 & \text{if } n = 8 \text{ and } m \mod 5 = 3 \\ \left\lfloor \frac{18m}{5} \right\rfloor + 3 & \text{if } n = 8 \text{ and } m \mod 5 \neq 3 \\ \left\lfloor \frac{2(n+1)(m+1)-2}{5} \right\rfloor - 1 & \text{if } n \geq 9 \text{ and } n \mod 5 = 4 \text{ and } m \mod 5 = 4 \\ \left\lfloor \frac{2(n+1)(m+1)-2}{5} \right\rfloor & \text{if } n \geq 9 \text{ and } n \mod 5 \neq 4 \text{ or } m \mod 5 \neq 4 \end{cases}$$

First, for this problem the set of values for a cell is $\mathcal{S} = \{$ TWO_STONES, STONE, OK, NEED_ONE $\}$. STONE means that we put a troop (here we will talk of stones instead) on the cell, so it does not need to be dominated by another cell. TWO_STONES means that we put two troops on the cell, so it dominates its neighbours. Now a state S is in $\mathcal{V}$ if and only if for each $i \in [\![0, n-1]\!]$:

- if $S[i] =$ NEED_ONE then neither $S[i-1]$ nor $S[i+1]$ is TWO_STONES;
- if $S[i] =$ STONE then neither $S[i-1]$ nor $S[i+1]$ is TWO_STONES or STONE;

Note that the second rule is not required for the consistency of the state, but it is an optimisation which allows us to reduce a lot the number of states, as we will see in Section 2.2.3. It is justified by the fact that in a minimum Roman-dominating set, we

can always remove any stone neighbouring a cell with two stones, and if there are two neighbouring cells with a stone each we still have a dominating set of same value by removing one of the stones and putting a second stone on the other cell. This implies that there exist minimum Roman-dominating sets matching the extra rules we enforce.

Fact 2.2 (see Section 2.2.3)**.** *There are* $\Theta(x_0^n) \approx \Theta(2.956295^n)$ *states in* $\mathcal{V}$ *for* n *lines, where* x_0 *is the largest real root of the polynomial* $x^4 - 3x^3 - x^2 + 3x + 1$.

A state $S \in \mathcal{V}$ is in $\mathcal{F}$ if and only if for every $i \in [\![0, \dots, n-1]\!]$, if $S[i] = \text{OK}$ then at least one among $S[i-1]$ and $S[i+1]$ is TWO_STONES.

(S, S') is a compatible pair if and only if for $i \in [\![0, \dots, n-1]\!]$:

- if $S[i] = \text{NEED\_ONE}$ then $S'[i] = \text{TWO\_STONES}$;
- if $S'[i] = \text{NEED\_ONE}$ then $S[i] \neq \text{TWO\_STONES}$;
- if $S'[i] = \text{OK}$ then at least one among $S[i]$, $S'[i-1]$ and $S'[i+1]$ is TWO_STONES;
- if $S[i] \in \{\text{TWO\_STONES}, \text{STONE}\}$ then $S'[i] \neq \text{STONE}$;
- if $S[i] = \text{STONE}$ then $S'[i] \neq \text{TWO\_STONES}$.

Finally, a state $S \in \mathcal{V}$ is in $\mathcal{E}$ if and only if none of its entry is NEED_ONE. We may notice that we do not enumerate every $f_j(D)$: we forbid for instance two neighbouring cells when each contains one or two stones. This is possible because there exist Roman-dominating sets of minimum size which do not contain this pattern. We will discuss this optimisation in Section 2.5. Here we can always avoid this pattern by removing the single stone of one cell. If the other one also had a single stone, we add one stone to it..

We now need to adapt the loss. We define, for the Roman domination,

$$\ell(n, m) = 5|S_2| + \frac{5}{2}|S_1| - nm = \frac{5}{2}(2|S_2| + |S_1|) - nm.$$

Indeed, each cell with two stones dominates five cells, and each cell in S_1 dominates only itself, but we add to it an additional loss of $3/2$ to penalize its bad ratio of number of dominated cells compared to number of stones used. This allows us to obtain

$$\gamma_{\text{R}}(n, m) \geq \frac{2}{5}(\ell(n, m) + nm).$$

Note that in the program, what we compute is actually $2\ell(n, m)$ to avoid to manipulate fractions or floating numbers. Let us define the almost-valid states which, for this problem, coincide with the valid states: $\mathcal{V}_a = \mathcal{V}$. Now if $S, S' \in \mathcal{V}_a$, $S\text{R}_a S'$ if and only if for $i \in [\![0, n-1]\!]$:

- if $S[i] = \text{NEED\_ONE}$ then $S'[i] = \text{TWO\_STONES}$;
- if $S'[i] = \text{NEED\_ONE}$ then $S[i] \neq \text{TWO\_STONES}$;
- if $S'[i] = \text{OK}$ and $i \neq 0$ then at least one among $S[i]$, $S'[i-1]$ and $S'[i+1]$ is TWO_STONES;
- if $S[i] \in \{\text{TWO\_STONES}, \text{STONE}\}$ then $S'[i] \neq \text{STONE}$;
- if $S[i] = \text{STONE}$ then $S'[i] \neq \text{TWO\_STONES}$.

Here we do not give complete details on how we compute the loss. Each cell with two stones having $k < 4$ neighbours with one or two stones contributes for k, and each cell dominated by $k > 1$ cells (with two stones) also contributes for $k - 1$. Finally, each stone

with one cell contributes for 3/2. All these contributions sum up to make the loss. We recall that in the program we compute twice these values.

As in the previous section, we get exact values for 'small' values of n, and a lower bound for bigger values of n. This time, we obtain the following recurrence relation:

$$\text{for all } r \geq 12, T_{\mathrm{a}}^{r+5} = T_{\mathrm{a}}^{r} + 5. \tag{2.6}$$

We prove the formula for an arbitrary number of columns and lines the same way as for the 2-domination problem, using Equation (2.6) and the definition of the loss function. We conclude that our lower bound is the exact value of γ_{R} thanks to the thesis of Currò (see [10, Chapter 4, Theorem 10]). Indeed, he showed some upper bound for the Roman-domination number. The lower bound we find is the same as his upper bound, hence both are sharp and are the Roman-domination number.

2.3.2 The total domination

We present here the details for the total domination. Unfortunately, for reasons we will discuss in Section 2.4, we were not able to find the values for grids of arbitrary size, so we give only partial results. The total domination in grids was studied by Gravier [23]. He gives the values for up to 4 lines, and provides some lower and upper bound. We improve his lower bound. Crevals and Ostergård [9], on their side, gave values up to 28 lines, which is more than we do (up to 15 lines for us).

First, we can see that the domination, 2-domination and total domination are in fact part of a more general class of problems:

Definition 2.13. A set $S \subset V$ is **(a, b)-dominating** a graph $G = (V, E)$ when any vertex v in S has at least a neighbours in S and every vertex outside S has at least b neighbours in S.

It is clear that the domination is the (0,1)-domination while the total domination is the $(1, 1)$-domination. Given this fact, storing information about whether a cell has a stone, or whether one or zero of its 'current' neighbours have one is no longer enough. Fortunately, we can encode in a cell value the number of neighbouring stones, or alternately the number of stones it lacks to be (a, b)-dominated, plus the knowledge of whether or not it contains a stone. Depending on the problem we consider, we may need more or less information: for the total domination for instance, we also need to store for the cells with a stone whether or not they are dominated by another cell.

One way to encode the necessary information for an (a, b)-domination problem is to have the following set for the cells values: STONE_PREV, STONE, NONE_PREV, NONE. The '_PREV' suffix means that the neighbouring cell from the previous column also contains a stone. 'NONE' means that the cell does not contain a stone itself. We then compute which states are valid and the compatibility relations just from that. Indeed, we can recover how many times a cell is dominated from this piece of information.

However, as we will see in details in Section 2.2.3, we can make the computations a lot faster by choosing carefully how to encode the necessary information. So the

set of cell values is $\mathcal{S} = \{$STONE_OK, STONE_NEED_ONE, STONE_NEED_TWO, OK, NEED_ONE, NEED_TWO$\}$. Some of these values are not necessary: for the computations of the exact values, when the number of lines is fixed, any cell would have at most one new neighbour (in the next column). This makes the values ending by '_TWO' useless: they are used only for the loss computation. We do not give explicitly the set of valid states, first states, ending states and compatibility relation: the logic behind them is very similar to the one for the domination, and they are present in the code (see respectively functions `is_state_valid`, `can_cell_neighbourhood_be_first`, `is_state_dominated` and `are_state_compatible` in the source code).

Fact 2.3 (see Section 2.2.3)**.** *There are* $\Theta(x_0^n) \approx \Theta(2.618034^n)$ *states in* $\mathcal{V}$ *for* n *lines, where* x_0 *is the real root of the polynomial* $x^4 - 3x^3 + 3x - 1$.

As we warned above, we did not manage to get a closed formula which would work for every value of n and m. Hence we give here values for small number of lines (up to 15), and some bounds on the quantity γ_{T} when the number of lines is arbitrary. The lower bound is obtained by the same loss method, and again the transfer matrix we use for the bands verifies some recurrence property, hence we can extend our lower bound for an arbitrary number of lines and columns. However, this lower bound does not seem to match the actual value: it seems to increase little by little, and we exhaust the computing resources while it still wants to grow[14].

Theorem 2.8. *For* $1 \leq n \leq 15$ *and any* $m \geq n$*, the following equalities about the total domination number hold:*

$$\gamma_{\mathrm{T}}(1,m) = \begin{cases} \left\lfloor \frac{m}{2} \right\rfloor & \text{if } m \mod 4 = 0 \\ \left\lfloor \frac{m}{2} \right\rfloor + 1 & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(2,m) = \begin{cases} \left\lfloor \frac{2m+2}{3} \right\rfloor + 1 & \text{if } m \mod 3 = 1 \\ \left\lfloor \frac{2m+2}{3} \right\rfloor & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(3,m) = n$$

$$\gamma_{\mathrm{T}}(4,m) = \begin{cases} \left\lfloor \frac{6m+3}{5} \right\rfloor + 2 & \text{if } m \mod 5 \in \{0,3\} \\ \left\lfloor \frac{6m+3}{5} \right\rfloor + 1 & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(5,m) = \begin{cases} \left\lfloor \frac{6m+3}{4} \right\rfloor + 2 & \text{if } m \mod 4 = 0 \\ \left\lfloor \frac{6m+3}{4} \right\rfloor + 1 & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(6,m) = \begin{cases} \left\lfloor \frac{12m}{7} \right\rfloor + 4 & \text{if } m \mod 7 = 5 \\ \left\lfloor \frac{12m}{7} \right\rfloor + 3 & \text{if } m \mod 7 \in \{1,2,3\} \\ \left\lfloor \frac{12m}{7} \right\rfloor + 2 & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(7,m) = \begin{cases} 2m+2 & \text{if } n \mod 2 = 0 \text{ or } m \in \{9,11,15,21\} \\ 2m+1 & \text{otherwise} \end{cases}$$

[14] like a poor vegetable running out of water

$$\gamma_{\mathrm{T}}(8,m) = \begin{cases} \lfloor \frac{20m+6}{9} \rfloor + 4 & \textit{if } m \mod 9 \in \{0,7\} \textit{ and } n \notin \{9,16\} \\ \lfloor \frac{20m+6}{9} \rfloor + 3 & \textit{if } m \mod 9 \in \{2,3,4,5\} \\ \lfloor \frac{20m+6}{9} \rfloor + 2 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(9,m) = \begin{cases} \lfloor \frac{10m+3}{4} \rfloor + 3 & \textit{if } m \mod 4 = 2 \\ \lfloor \frac{10m+3}{3} \rfloor + 2 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(10,m) = \begin{cases} \lfloor \frac{30m+1}{11} \rfloor + 6 & \textit{if } m \mod 11 = 9 \textit{ and } m \neq 20 \\ \lfloor \frac{30m+1}{11} \rfloor + 5 & \textit{if } m \mod 11 \in \{2,5,7\} \textit{ and } m \notin \{13,18\} \\ \lfloor \frac{30m+1}{11} \rfloor + 4 & \textit{if } m \mod 11 \in \{0,1,3,6\} \textit{ or } m = 20 \\ \lfloor \frac{30m+1}{11} \rfloor + 3 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(11,m) = \begin{cases} 3m+4 & \textit{if } m \in \{12,22\} \\ 3m+3 & \textit{if } m \in \{13,15,17,19,23,27,29,33,37,43,47,57\} \\ 3m+2 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(12,m) = \begin{cases} \lfloor \frac{42m+9}{13} \rfloor + 6 & \textit{if } m \mod 13 \in \{0,11\} \textit{ and } m \notin \{13,24,26,37\} \\ \lfloor \frac{42m+9}{13} \rfloor + 5 & \textit{if } m \mod 13 \in \{2,4,7,9\} \textit{ and } m \notin \{15,17,20\} \\ \lfloor \frac{42m+9}{13} \rfloor + 4 & \textit{if } m \mod 13 \in \{3,5,6,8\} \textit{ or } m \in \{13,24,26,37\} \\ \lfloor \frac{42m+9}{13} \rfloor + 3 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(13,m) = \begin{cases} \lfloor \frac{14m+3}{4} \rfloor + 5 & \textit{if } m \in \{14,26\} \\ \lfloor \frac{14m+3}{4} \rfloor + 4 & \textit{if } m \mod 4 = 0 \textit{ or } m = 19 \\ \lfloor \frac{14m+3}{4} \rfloor + 3 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(14,m) = \begin{cases} \lfloor \frac{56m+2}{15} \rfloor + 8 & \textit{if } m \mod 15 = 13 \textit{ and } m \notin \{28,43\} \\ \lfloor \frac{56m+2}{15} \rfloor + 7 & \textit{if } m \mod 15 \in \{2,9,11\} \textit{ and } m \notin \{17,24,26,32,41\} \\ \lfloor \frac{56m+2}{15} \rfloor + 6 & \textit{if } (m \mod 15 \in \{0,5,6,7\} \textit{ and } m \notin \{15,21,22,30\}) \\ & \quad \textit{or } m \in \{28,43\} \\ \lfloor \frac{56m+2}{15} \rfloor + 5 & \textit{if } m \mod 15 \in \{1,3,4,10\} \textit{ or } m \in \{17,24,26,32,41\} \\ \lfloor \frac{56m+2}{15} \rfloor + 4 & \textit{otherwise} \end{cases}$$

$$\gamma_{\mathrm{T}}(15,m) = \begin{cases} 4m+6 & \textit{if } m \in \{16,30\} \\ 4m+5 & \textit{if } m \in \{21,23\} \\ 4m+4 & \textit{if } (m \mod 2 = 0 \textit{ and } m \notin \{16,30\}) \\ & \quad \textit{or } m \in \{17,19,25,27,31,35,37,39,41,45,49,53,55,59,63, \\ & \qquad 67,73,77,81,91,95,109\} \\ 4m+3 & \textit{otherwise} \end{cases}$$

Theorem 2.8 confirms the results from Crevals and Ostergård [9] for values up to 15. In their paper they managed to go up to $n = 28$, using another approach. They do not enumerate full dominating sets as we do, but have like us some notion of states. However, instead of enumerating the states of the columns, they only enumerate the states of partial dominating sets: they store only the elements of the minimal-dominating set. Using clever arguments they also manage to cut down on the number of states they enumerate, and it turns out this number grows less fast than in our method.

The loss can be adapted to the generic (a, b)-domination problem: each stone contributes to dominating its 4 neighbouring cells. The dominating $|D|$ cells need to be dominated a times and the cells $nm - |D|$ cells not in D need to be dominated b times. When reversing the formula we obtain:

$$\gamma_{a,b}(n, m) = \frac{b \cdot nm + \ell(n, m)}{4 - a + b}. \tag{2.7}$$

With our program to compute the loss, we find that, for a band of height 10:

$$\text{for all } r \geq 31,\ T_{\mathrm{a}}^{r+22} = T_{\mathrm{a}}^{r} + 10. \tag{2.8}$$

From this, as for the other problems we studied above, we deduce:

Claim 2.3. $\ell_{10}(n + 22, m) = \ell_{10}(n, m + 22) = \ell_{10}(n, m) + 10$ *for every* $n, m \geq 52$.

First, we may notice that our lower bounds obtained thanks to the loss are not too far from the actual values Crevals and Ostergård found. For 28 lines, they find 416 for $m = 56$ and 427 for $m = 58$ when we find that it must respectively be at least 411 and 426.

Also, the bounds we obtain agree to some extent to the conjecture of [9]. Indeed, their formulas imply that, when $m \mod 4 \in \{1, 3\}$ then $\gamma_{\mathrm{T}}(n, m) = \Theta(\frac{nm+n+m}{4})$ and our values imply

$$\gamma_{\mathrm{T}}(n, m) \geq \frac{nm + 10/11(n + m)}{4} + O(1). \tag{2.9}$$

We can even go a bit further on the constant after this equivalent. By using Claim 2.3 we deduce that $\ell_{10}(n, m) \geq 2\frac{10}{22}(n + m) + c = \frac{10}{11}(n + m) + c(n \mod 22, m \mod 22)$, where the $c(i, j)$ for $0 \leq i, j \leq 22$ are some constants depending on the actual values of $\ell_{10}(n, m)$. The factor 2 comes from the fact that a rectangle has two vertical bands and two horizontal bands, as we mentioned for the 2-domination. We determine, thanks to the values of the loss of height 10 for $52 \leq n, m \leq 74$, a lower bound on the $c(i, j)$ constants. From what just precedes and by Equation (2.7), we obtain:

Proposition 2.9.

$$\gamma_{\mathrm{T}}(n, m) \geq \frac{nm + \frac{10}{11}(n + m)}{4} - 1.$$

We may even notice that the fraction which is multiplied by $(n+m)$ in Equation (2.9) increases when the height increases and may converge towards 1, which is the values from their conjecture. Indeed, the fractions we obtain equal 6/7 for 6 and 7 lines, 8/9 for 8 and 9 lines, and 10/11 for 10 lines. It may even be possible that it takes all the values of the shape $2l/(2l + 1)$ when the height becomes arbitrarily big.

Conjecture 1.

$$\gamma_{\mathrm{T}}(n,m) = \frac{nm+n+m}{4} + O(1).$$

2.3.3 The distance-two-domination

As was written in the introduction, a grid is distance-two-dominated by S when any vertex not in S is at distance at most two of an element of S. The more general distance-k-domination problem was studied by Farina and Grez [13] who proved some upper bound on the associated domination number.

Here again we did not manage to get a closed formula for arbitrary numbers of lines and columns. The problem is not the same as for the total domination: we more likely just lacked of a bit a computing resources instead of the problem being much more difficult to tackle. Indeed, since now a vertex may be dominated by a vertex at distance two, we need to store more information on previous columns: not just some information about the previous columns, but also some about the one even before.

The set of cell values is $\mathcal{S} = \{$ STONE_PREV, STONE, OK_PREV, OK, NEED_DIST_TWO, NEED_DIST_ONE $\}$. The 'prev' suffix, here again, means that the cell of the previous column has a stone. OK means that the cell is dominated, whereas NEED_DIST_TWO means that the cell is not dominated so that it requires a cell at distance at most two with a stone. NEED_DIST_ONE is similar but it means here that the cell in the previous column was not dominated, hence the next cell needs to have a stone to dominate this ante-predecessor cell. As for the total domination we do not detail the other special sets and the compatibility relation, which are only a matter of logic and optimisations, and can be found in the source code (see the functions `is_state_valid`, `can_cell_neighbourhood_be_first`, `is_state_dominated` and `are_state_compatible`).

Fact 2.4 (see Section 2.2.3)**.** *There are* $\Theta(x_0^n) \approx \Theta(2.958770^n)$ *states in* $\mathcal{V}$ *for* n *lines, where* x_0 *is the largest real root of the polynomial* $x^{24}-5x^{23}+6x^{22}+2x^{21}-7x^{20}+6x^{19}-8x^{18}+8x^{17}+4x^{16}-13x^{15}+5x^{14}+6x^{13}+8x^{12}-14x^{11}-17x^{10}+14x^9-8x^8-10x^7-8x^6+5x^5+9x^4-x^3-5x^2+4x-1$.

As for the total domination, we give the formulas for small values of n.

Theorem 2.10. *For* $1 \leq n \leq 14$ *and any* $m \geq n$*, the following equalities about the distance-two-domination number hold:*

$$\gamma_{\mathrm{d2}}(1,m) = \left\lceil \frac{m}{5} \right\rceil$$

$$\gamma_{\mathrm{d2}}(2,m) = \left\lfloor \frac{m}{4} + 1 \right\rfloor$$

$$\gamma_{\mathrm{d2}}(3,m) = \left\lceil \frac{m}{3} \right\rceil$$

$$\gamma_{\mathrm{d2}}(4,m) = \begin{cases} \left\lfloor \frac{3m}{7} \right\rfloor + 1 & \text{if } m \mod 7 \in \{0,1,3,5\} \\ \left\lfloor \frac{3m}{7} \right\rfloor + 2 & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{d2}}(5,m) = \begin{cases} \left\lfloor \frac{m+1}{2} \right\rfloor & \text{if } m \mod 6 = 1 \\ \left\lfloor \frac{m+1}{2} \right\rfloor + 1 & \text{otherwise} \end{cases}$$

$$\gamma_{\mathrm{d}2}(6,m)=\begin{cases}\lfloor\frac{3m}{5}\rfloor+1 & \textit{if } m \mod 5\neq 3\\ \lfloor\frac{3m}{5}\rfloor+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(7,m)=\begin{cases}7 & \textit{if } m=9\\ \lfloor\frac{2m}{3}\rfloor+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(8,m)=\begin{cases}12 & \textit{if } m=13\\ \lfloor\frac{3m}{4}\rfloor+1 & \textit{if } m \mod 8\in\{4,7\}\\ \lfloor\frac{3m}{4}\rfloor+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(9,m)=\begin{cases}\lfloor\frac{5m}{6}\rfloor+1 & \textit{if } m\in\{11,18\}\\ \lfloor\frac{5m}{6}\rfloor+3 & \textit{if } m \mod 6\in\{2,3,9\} \textit{ and } m\notin\{4,15,20,21,27,32,39\}\\ \lfloor\frac{5m}{6}\rfloor+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(10,m)=\begin{cases}\lfloor\frac{10m}{11}\rfloor+3 & \textit{if } m \mod 11\in\{2,3,5,8\}\\ \lfloor\frac{10m}{11}\rfloor+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(11,m)=\begin{cases}m+1 & \textit{if } m \mod 30\in\{1,4,6,7,9,11,14,16,17,19,21,24,26,27,29,30\}\\ m+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(12,m)=\begin{cases}\lfloor\frac{15m}{14}\rfloor+4 & \textit{if } m \mod 14=11 \textit{ and } m\neq 25\\ \lfloor\frac{15m}{14}\rfloor+2 & \textit{if } m \mod 14\in\{1,4,7\} \textit{ or } m\in\{14,17,19,28\}\\ \lfloor\frac{15m}{14}\rfloor+3 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(13,m)=\begin{cases}\lfloor\frac{15m}{13}\rfloor+4 & \textit{if } m \mod 13=5 \textit{ and } n\neq 31\\ \lfloor\frac{15m}{13}\rfloor+3 & \textit{if } (m\neq 13 \textit{ and } m \mod 13\in\{0,2,3,6,8,10,11,12\}) \textit{ or } n=31\\ \lfloor\frac{15m}{13}\rfloor+2 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(14,m)=\begin{cases}\lfloor\frac{21m}{17}\rfloor+2 & \textit{if } m \mod 17=1 \textit{ or } n\in\{23,30,47\}\\ \lfloor\frac{21m}{17}\rfloor+4 & \textit{if } m=36 \textit{ or } (m>46 \textit{ and } n \mod 17\in\{2,3,8,11,14,16\}\\ & \qquad\qquad \textit{and } n\notin\{54,59,71\})\\ \lfloor\frac{21m}{17}\rfloor+3 & \textit{otherwise}\end{cases}$$

$$\gamma_{\mathrm{d}2}(15,m)=\begin{cases}\lfloor\frac{21m}{16}\rfloor+2 & \textit{if } m \mod 16\in\{1,4,7\}\\ \lfloor\frac{21m}{16}\rfloor+4 & \textit{if } m \mod 16\in\{2,3,5,8\} \textit{ and } n\notin\{19,21\}\\ \lfloor\frac{21m}{16}\rfloor+3 & \textit{otherwise}\end{cases}$$

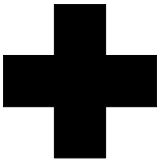


Figure 2.11: The shape corresponding to the domination tiling problem.

We adapt again the loss function: now a cell contributes to the domination of 12 cells, and each cell not in the dominating set needs to be dominated once. We then obtain:

$$\gamma_{2\mathrm{d}}(n,m) = \frac{nm + \ell(n,m)}{13}.$$

However, we were not able to find good bounds with the method we used for the other problems.

2.4 Conjectures about why the method works

As we said earlier, the method for a fixed number of lines should work for any problem the properties of which can be checked locally, that is by the means of a finite list of forbidden patterns. Some authors investigated the problem of domination in Cartesian products of cycles (see for instance [25, 38]). The first part of the technique (when n is fixed and small) may be adapted, as stated by Theorem 2.4 but the second part (for arbitrary number of lines) does not apply directly since a crucial property is that the loss can be concentrated inside the *borders* of the grids.

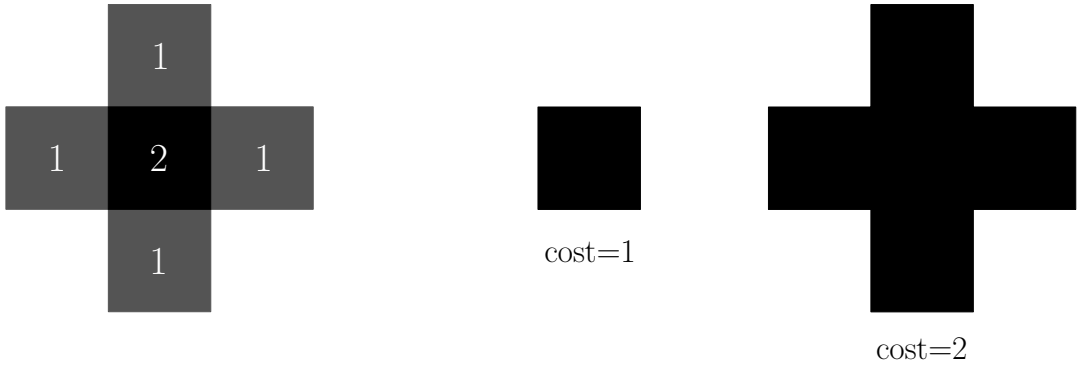
Some necessary and sufficient conditions for the loss method (the one described in Section 2.2.4, for arbitrary large number of lines) to work are yet to be discovered. We try here to infer what these conditions might be by giving some properties we believe to be related to the effectiveness of the method.

We believe that the reasons why the method gives sharp bounds can be expressed as some tiling properties. Indeed, the domination problems are related to covering problems. For instance, Figure 2.11 shows the shape associated to the domination problem. A smallest dominating set in a grid is equivalent to a smallest covering set of the rectangle with this shape. The method of Gonçalves et al. works thanks to the fact that the shape has the following two properties. First, it can tile (that is, cover without overlaps) the infinite plane. Second, we can find optimal solutions which consist of projecting a tiling of the plane, cropping it and modifying only tiles at bounded distance from the border.

In the case of the 2-domination and the Roman domination, it is not properly speaking a covering problem, but a generalised covering problem with some weights (see Figure 2.12). The properties we write below are rather focused on standard tilings than on generalised tiling.

One crucial point is the following property.

Property 2.1 (Fixed-height border-fixing)**.** Let X be a shape. X has the fixed-height border-fixing property if there exist k, n_0, m_0 such that, for any $n \geq n_0$ and $m \geq m_0$, there exists an optimal covering of the $n \times m$ rectangle whose cells at distance greater than k of the border are included in a tiling (so with no overlaps) of the plane.



(a) The 2-domination shape. (b) The Roman-domination shapes.

Figure 2.12: The shapes for the 2-domination and the Roman domination. For the 2-domination, we look for a covering such that the sum of the weights (in white) on a cell is at least 2. For the Roman domination, we cover with two tiles, but they have different costs. We are interested in a covering of minimum weight.

For instance, the 2-domination shape has this property for $k = 3$: any optimal solution to the 2-domination problem can be obtained from an infinite optimal 2-domination set of which we modify only cells at distance at most 3 from the border. Note that, due to the automation feature of the algorithm, this is indeed $k = 3$ here even if the program needs to explore borders of size 6 to find the correct bounds.

The fixed-height border-fixing property implies that the bounds given by the method are sharp for some constant height band, independent of the size of the rectangle. This seems to be related to the following property.

Property 2.2 (Crystallisation)**.** Let X be a shape. We say that X has the crystallisation property if there exists $k \in \mathbb{N}$ such that for every partial tiling of size k with the shape X, either this tiling cannot be extended to tile the plane, or there is a unique way to do so up to rotation/symmetry.

For instance, the domination shape has this property for $k = 2$: any two cross shapes put on a grid either cannot be extended into a tiling of $\mathbb{Z}^2$ or can be completed into only one such tiling. On the contrary, the total domination does not have this property. The total-domination problem has been studied a lot in other graphs (see [30] for example), but remains open for grids. It is related to the shapes in Figure 2.13. The small one corresponds to the influence of one 'stone': note that the centre cell does not dominate itself. The big ones are the unions of two copies of the small one. One can see that tiling the plane with the small shape is equivalent to tiling the plane with the set of the two big shapes: in the small shape, the middle cell must be dominated. As shown, the big shape can be vertical or horizontal. The problem with our technique is that a tiling of the plane can, with a certain degree of freedom, mix the vertical and the horizontal big shapes. This probably leads to some non-zero loss in the centre of a big grid to be necessary for a covering to be of minimum size. In this case, the assumption of the loss on the centre of the grid being zero would be false, making our technique unusable.

Conjecture 2. *If a shape X tiles the plane and has the crystallisation property then it also has the fixed-height border-fixing property.*

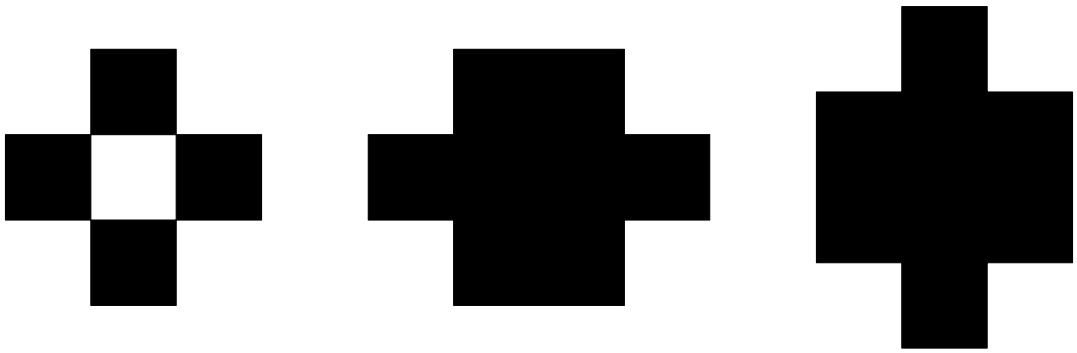


Figure 2.13: The shapes associated to the total domination. The big ones are the two different unions of two copies of the small one. Tiling the plane with the small one boils down to tiling the planes with the two big ones.

These properties could also be used on tiling problems with other shapes, even if they have no relation with any domination problem on grids.

2.5 Experimental details: implementation and optimisations

All of the results we obtained required a lot of computing resources. Some of them were obtained after quite a lot of optimisations, and some partial results could have been improved if the memory consumption and the running time had been smaller. We explain here some optimisations we applied, some of them being more theoretical while others are technical.

Problem-dependent optimisations.

As we said before, we made optimisations to reduce the number of states in some problems, using their properties. For instance for the Roman domination, there exist minimum dominating sets which never have one stone next to a cell with either one (case a) or two stones (case b). We can simply remove the stone in case a, or make it a two stones cell and remove its neighbours with one stone in case b. If we remove the optimisations for the states of the Roman domination, the growth rate of the number of states becomes 3.561553 instead of 2.956295. We achieved a very good improvement. For instance, for 10 lines the optimised version generates 32.5k states and 6.2M elements in the transfer matrix, while the unoptimised version generates 182k states and 2500M elements in the transfer matrix, which is very significant.

Another part where the improvements were big was about the (a, b)-domination. The first version we programmed encoded in the states whether or not each cell had a stone, and whether its predecessor had a stone. The set of states of this version is: $\mathcal{S} = \{$ STONE_PREV, STONE, NONE_PREV, NONE $\}$. The other version we programmed after, which we will call the fast version, instead encoded for each cell the fact of whether it has a stone and how many stones it needs (that is 0 or 1 for computing the domination numbers, and 0, 1 or 2 for the computation of the loss). The states of this

version for a fixed number of lines, is included[15] in { STONE_OK, STONE_NEED_ONE, STONE_NEED_TWO, OK, NEED_ONE, NEED_TWO }. This version is faster because if for instance a cell is dominated by its state, we do not need to convey the information of whether or not its predecessor cell had a stone. We will compare the two versions for the total domination, i.e. the (1,1)-domination. In the slow version, the growth rate of the number of states is 4, whereas in the fast version, it is 2.618034, which is a huge gain. Using 95 cores, computing the transfer matrix for 10 lines uses 150s for the slow version compared to 1.3s for the fast one. In the slow version there are 525k states and 252M compatible pairs in the transfer matrix; in the fast version there are 10.4k states and 2.2M compatible pairs, once again a very important gain.

Pruning symmetries in the states.

In order to reduce the number of states, we can do a simple observation: take a dominating set for any domination problem, and apply a horizontal symmetry to it. It remains dominating. This means that we do not need to store all the states: when they are not symmetrical we may keep only one representative out of the two since they both have the same number of stones. Also, let us assume that S and S' are compatible, and let us denote the symmetric of S by $\mathrm{rev}(S)$. Then let us assume that S and S' are compatible, but we only stored $\mathrm{rev}(S)$ and not S'. This is not a problem since S and S' being compatible implies that $\mathrm{rev}(S)$ and $\mathrm{rev}(S')$ are compatible. When computing the transfer matrix, it suffices to check for compatibility with the reversed versions of one of the two states. If a state can be put first, then its symmetrical can also. The same applies for the end states. This implies that, by using only one representative for each pair of symmetrical states, we compute the same domination numbers as when we store all states. This optimisation saves us around half of the states, half of the compatible pairs. This optimisation does not, however, change the growth rate of the number of states of a problem: it would otherwise make it decrease exponentially when the number of lines increase instead of roughly dividing it by two.

For the 2-domination with 12 lines, without pruning symmetrical states we have around 142k states and 10M compatible pairs, and the computation of the transfer matrix with 95 cores takes around 59s. When we prune symmetrical states it takes 43.5s: we have 71k states and 4.6M compatible pairs. This a slight improvement, however the optimised version has a memory peak at 236MB whereas the other uses 493MB at peak. This may make a difference, for some problems, when computing the transfer matrices: it may make it possible to go one line further. Unfortunately, this optimisation does not apply to the computation of the loss matrices. Indeed, the top cells have a neighbour above which can contain a stone

Checking validity while generating states.

To generate the valid states, we may recursively enumerate all the states in $\mathcal{S}^n$ and each time we complete a state (by choosing its bottom value) we check whether or not the state is valid. However, in the program the generation of the valid states was not parallelised and so it took a non-negligible part of the total running time when using a machine with a lot of cores. This justified trying to optimise this part of the program.

[15] If a or b equals zero, some states are no longer useful.

The optimisation we programmed was to partially check the validity of a state while generating the state: each time we fix the value of the next cell we check whether or not this beginning is valid instead of waiting until the state is complete.

This means that in many cases we can cut some branches of exploration of the states values before reaching the end. In the minimal-domination problem we investigate in Chapter 3, this helped reduce the time a lot: it takes 5.1s instead of 9.7s for a height of 7 and 46.4s instead of 121.4s for a height of 8. We did not implement this feature for the problems in this chapter since the generation of states did not take that long compared to the other parts of the computation. Computing the loss matrix is, as we will see, cubic in the number of states.

Pre-computing a subset of the potentially compatible states.

The previous paragraph was about precomputing the validity of possible states while we are generating the states. The idea here is similar: when considering a state and checking which other states might be compatible with it, we waste some time trying a lot of states which have no chance of being compatible. For instance, S may be incompatible with S' because of the values of the first three cells... in that case, any state S'' with the same first three cells will not be compatible with S'. For some problems, we may precompute a list of *candidates* to the compatibility relation with a specific state S': any state outside this list will not be compatible, but not every state in the list will be compatible. For other problems, like the simple domination or the 2-domination, precomputing the list of candidates to the compability slows down the program: the ratio of incompatible 'pairs' is less than the one of the minimal domination, for instance. This method will be used in Chapter 3 for the minimal versions of the domination problems.

In some problems, mainly the ones storing a lot of things in the past: the distance-two-domination, and especially the minimal and minimal total domination that we study in Chapter 3 are the best examples to illustrate this fact. Since the latter stores whether or not the current cell has a stone as well as the information about the most recent three columns, compatibility rules follow some structure. Indeed, if S' can be put after S then the information about columns '-1', '-2', and '-3' must respectively be the same as the information in S about its relative current column and columns '-1' and '-2'. So we can discard any state where the information differs. Since each value is stored in four bits (one for each column we store information about), the list of possible next states S' is obtained by shifting each value of S by one bit. This means that we test only one eights of the number of states as possible successors for a state S, as opposed to the whole set of states we would test otherwise. To simplify, for instance, 1110 cannot be compatible with 1111 because its one-before-last bit should match the last bit of 1110 (i.e. a zero). 1110 might, on the contrary, be compatible with 1100 or 1101: the zero has shifted towards the left.

For the minimal domination of height 7 (around 6.2 millions states), using a machine with 20 cores, the computation of the transfer matrix takes 8 seconds with the optimisation and more than 22 minutes without (we did not wait for it to end).

Optimising matrices products.

The matrices we handle are big, or even huge: as many lines and columns as there are states, which can be around one million (or a bit more) for some problems in this

chapter. So the products between matrices take a lot of time and this is a concern of primary importance. Yet, our matrices are very sparse, the ratio of the number of states squared divided by the number of actual (different from $+\infty$) values are the following:

- for the 2-domination, it is 32 for 10 lines and 153 for 14 lines;
- for the Roman domination, it is 314 for 10 lines and 2899 for 14 lines;
- for the distance-2-domination, it is 226 for 10 lines and 1884 for 14 lines.

This is very fortunate because the algorithm we use have a running time of roughly $\mathcal{O}(nbState^2)$ when the number of lines is fixed and $\mathcal{O}(nbState^3)$ for the loss method. Since the matrices are sparse, the running time is in fact proportional to the number of compatible pairs, that is the number of values of the matrix. Still, the matrices products take a very long time. One first observation is that since we know that our matrices are primitive (i.e. there exists k such that M^k has its full $nbState^2$ coefficents) products between matrices rapidly take a very very long time. This can be avoided, for the first method (fixed number of lines) by doing only matrix-vector products when computing the domination numbers. This way, we do a number of operations which is around the number of values in the matrix which are different from $+\infty$.

Ordering the corner computations.

The final optimisation we talk about here is done at a higher level: it is more a design detail of the algorithm. To compute the loss, we need to compute the corner transfer matrix (see Section 2.2.4) C_{a}. $C_{\mathrm{a}}[a][b]$ contains the minimum loss inside the corner if a and b are the 'internal' edges of the corner. So the standard algorithm is basically the following: for any two states a and b, compute the minimum loss over the corner (i.e. compute the loss over h columns, where h is the height of the border we chose). This leads, letting $nbState = |\mathcal{V}_a|$, to $nbState^2$ possibilities for the choices of a then b, multiplied by a factor of $h \cdot |\mathrm{R}_a|$ where $|\mathrm{R}_a|$ is the number of almost compatible pairs. Roughly, this would be some $\mathcal{O}(nbState^4)$. However, things can be more finely tuned: let us assume we compute the loss over the bottom-right corner, let us fix b as the output state of the corner (the 'upper horizontal' state). We then compute the minimum loss inside the corner, and finally obtain, for every possible input state of the corner ('left vertical' state) a the value of $C_{\mathrm{a}}[a][b]$ by considering all the possible states a as a $(h+1)^{\text{th}}$ compatibility computation. This leads to a roughly $\mathcal{O}(nbState^3)$ algorithm, which is much better: we handle matrices with $nbState = 20000$, so we save a factor around 20000 in this case.

Using threads.

When you lack time, you may want to save up time eating, so a brunch is a good compromise. Or rather, 'brunch' was the name of the machine with the most (efficient) cores in our laboratory: it has 96 cores. So this was a great incentive to parallelise the code. As we mentioned in the introduction, the key point when parallelising a code is to identify how to make the parallel computations independent. Here we were lucky: computing the transfer matrix (i.e. the compatibility and almost-compatibility relations) can be done fairly easily in parallel: we divide the states into p share of equal size and each thread computes, for each state S in its share, the set of states which are compatible with S. The good thing is that all threads write the list of compatible states at different points

of memory, so there is no risk of writing at a location of the memory at the same time as another write or a read. They may read the same variable at the same time but this is not an issue. Due to the fact that not all portions are easily parallelisable and to a few technical issues the speedup is not 1: if we use p cores, the running time is not divided by p. Some of the problems involve context switching (to a minor impact): sometimes a thread is executed in one core and then switched to another so that all the values in the registers need to be copied, and all the thread do not have exactly the same amount of work to do: when multiplying two matrices and giving a portion of the multiplication to each core, some may be 'lucky' and finish well before the others. For instance, computing the transfer matrix of height 13 for the 2-domination problem takes 167.6 seconds using one thread. On a machine with 96 cores, using 20 threads, it takes 15.2s and using the 100 threads, it takes 4.8s.

It is beautiful to create a lot of threads and then see the usage of CPU decrease little by little, as though the finishing times of the threads followed a normal law.

Chapter 3

Asymptotic growth of the number of dominating sets

We live in a world whose main concept is the one of growth. All countries try to achieve the maximum possible economic growth so that their people may enjoy a better life because they have more money and comfort. However, alongside with the growing number of people on the Earth, such an economical and technological growth leads to huge problems, some of which are climate change and the loss of biodiversity. For instance, we have lost 52% of the existing animal species between 1970 and 2012, according to the WWF. Some animals are small yet very useful to humans, like bees. They also have suffered a big loss over the past decades, and still decrease now. Concerning the resources and energy that we **consume**, we can cite the IEA[1] which says that between 1971 and 2017 the total of produced energy **increased by more than 2.5 times**. Our consumption of resources increased a lot as well, which is not sustainable. Indeed, in 2019 the **Earth Overshoot Day** was July 29th[2]: between January 1st and this date, the world consumed the amount of resources that the Earth can renew in one year. This means that we consume at least 1,42 times what the earth renews each year... asymptotically, this leads to the total depletion of the available resources.

However, in this chapter the only thing which grows is the number of inoffensive[3] dominating sets. In the previous chapter, we studied these sets from an optimisation point of view: trying to minimise the size of a dominating set. It is also of interest to study the related counting problem: how many different dominating sets are there? We will tackle this question on various domination problems: the domination, total domination, minimal domination and minimal total domination. This means that, for each of these problems, we will show the existence of some constant ν, depending on the problem, such that the number of appropriate dominating sets is $\nu^{nm+o(nm)}$ when both dimensions n and m of the grid tend to infinity. We will also give bounds and estimates on the growth rate ν. We will do the same for the other domination problems listed just above. In this journey, we will see that counting dominating sets is related to some problems in dynamical systems, and particularly to the notion of entropy in subshifts of finite type.

[1] International Energy Agency

[2] It happened to be the precise day I wrote this paragraph.

[3] yes, remember Chapter 2

This notion of subshift, which we define later, encompasses the problems which can be defined as colouring $\mathbb{Z}^d$ forbidden a specific set of patterns to appear. In dimension 2, the subshifts can be seen as factor-free languages of bidimensional words.

We begin in Section 3.1 by giving the missing definitions of some dominating sets. We then introduce in Section 3.2 the simple concept of *local characterisation* and the notion of *subshifts (of finite type)*, and explain their similarities with the domination problems. These concepts are tools we will use to obtain our results: our domination problems can be viewed as some subshifts of finite type (SFT). In Section 3.3, we explain the link between these tools and our objectives: counting the patterns of a certain size which can appear in one of our SFTs helps us count the number of dominating sets associated to this SFT. We then introduce in Section 3.4 the notion of *entropy* of a subshift, which, in our cases, turns out to be equal to the growth rates we are looking for. In one subsection we show that some of the subshifts we have defined are *block gluing*. This fact implies that their entropies are computable numbers, hence the same is true for their asymptotic growth rates. In Section 3.5 we finally show how to obtain bounds on the growth rates. We give the numerical approximations and bounds we obtained for each problem, thanks to computer resources, using a program similar to the one of Chapter 2. Finally, we introduce a family of subshifts with particular properties in Section 3.6. This family generalises the domination and total domination, and we provide a second family of problems to do the same for their minimal counterparts. We study the block-gluing property for this second family and show that each of them are block gluing, but the block-gluing constant is a function of the parameter of the family.

The work of this chapter was done with Silvère Gangloff. Most of the results can be found in [17]. Beware that some notations, like $D_{n,m}$, differ a bit from the article in order to keep a coherency with Chapter 2.

3.1 Basic definitions and notation

In this chapter, G will be an undirected graph of which we examine the diverse dominating sets, and we will focus on grid graphs only. Contrarily to the previous chapter, the graphs here may be infinite when we specify it. We recall here Definition 2.4: a set S of vertices of a graph G is **total dominating** when any vertex $v \in V$ has at least one neighbour in S

Definition 3.1. A **minimal dominating** set S is a dominating set which is inclusion-wise minimal: any $S' \subsetneq S$ is not dominating. Or equivalently: for each $v \in S$, the set $S \setminus \{v\}$ is not dominating.
Likewise, a **minimal total-dominating** set is a total-dominating set which is inclusion-wise minimal.

Notation 3.2. In the following, for all integers n and m, we denote by $\boldsymbol{D_{n,m}}$, $\boldsymbol{M_{n,m}}$, $\boldsymbol{T_{n,m}}$ and $\boldsymbol{MT_{n,m}}$ respectively the number of dominating sets of the grid $G_{n,m}$, the number of its minimal dominating sets, the number of its total-dominating sets and the number of its minimal total-dominating sets.

To familiarise the reader with the notions of domination we study here, some of them are illustrated in Figure 3.1.

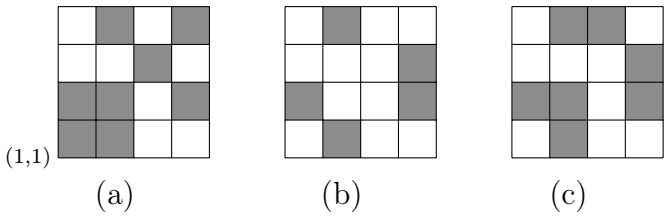


Figure 3.1: Illustration on $G_{4,4}$:
(a) a dominating set which is neither minimal dominating nor total dominating;
(b) a minimal dominating set which is not total dominating (the bottom-left dominant vertices are not dominated);
(c) a minimal total dominating set.

3.2 Local characterisations and relation with SFTs

In this section we recall, and for completeness prove, the local characterisations of some notions of dominating sets. This means that one can check if a set S is dominating (or minimal dominating, and so on) by examining, for each vertex, whether or not this vertex and its (possibly extended) neighbourhood are in S. This was what enabled Chapter 2 to work: checking properties locally makes it possible to encode only some information about the (few) previous column(s). This allows us to enumerate the dominating sets without keeping (and hence enumerating) them fully in memory. We then introduce and define the *subshifts of finite type* (SFT) which are dynamical systems objects strongly linked to the local characterisation property. Each domination problem will be associated to its SFT counterpart, with which it shares some properties like the growth rate (called *entropy* in the world of SFTs). We will later use the framework around the SFTs and prove some properties they have to deduce the counterpart in the domination growth rates.

3.2.1 Local characterisations

Definition 3.3. When a set S of a graph G is fixed, a vertex is called a **dominant** element of G when it is in S, and a **dominated** element when it has a neighbour in S. w is said to be a **private neighbour** of a dominant element v when v is the only neighbour of w in the set S.

Fact 3.1. *Let S be a set of vertices of a graph G. Then for all vertices v and w such that w is not a neighbour of v, w is dominated by S if and only if it is dominated by $S \setminus \{v\}$.*

Definition 3.4. Let S be a set of vertices of a graph G. We say that a dominant element is **isolated** in S when it has no neighbours in S.

In the previous three definitions, S is a dominating set for any of the variant of domination we study.

Proposition 3.1. *Let S be a dominating set of a graph G. S is minimal dominating if and only if every element of S is isolated in S or has a private neighbour not in S.*

Proof.

- ($\Rightarrow$): Let us assume that S is minimal dominating. Every vertex not in S has a neighbour in S because S is a dominating set. Now let us fix $v \in S$. From Fact 3.1 and by definition of a minimal dominating set, any w which is not in the neighbourhood of v is dominated by $S \setminus \{v\}$. Since $S \setminus \{v\}$ is not dominating, it means that:

  1. v is not dominated by $S \setminus \{v\}$, which means that v is isolated in S;
  2. or there exists some $u \notin S$ connected to v which is not dominated by $S \setminus \{v\}$, hence u is a private neighbour of v which is not in S.

- ($\Leftarrow$): Conversely, let us fix some dominating set S such that every $v \in S$ has a private neighbour not in S, or is isolated. Fix some $v \in S$. If it has a private neighbour u, then u is not dominated by $S \setminus \{v\}$, and thus $S \setminus \{v\}$ is not dominating. If it has no private neighbours, then it is isolated. This means that v is not dominated by $S \setminus \{v\}$, therefore the set is not dominating. In both cases, we conclude that S is minimal dominating.

□

With a similar proof, we obtain the following:

Proposition 3.2. *A total-dominating set S of a graph G is minimal total dominating if and only if any $v \in S$ has a private neighbour.*

3.2.2 Subshifts of Finite Type (SFTs)

We now introduce the notion of SFTs: they intuitively correspond to sets of possible colourings of the infinite grid $\mathbb{Z}^d$ which avoid some fixed finite set of forbidden patterns. We give the definitions in the general context of arbitrary dimension d, but in this chapter we will mostly work in dimension two, and a bit in dimension one in Section 3.5.1.

Definition 3.5. Let $\mathcal{A}$ be a finite set, and $d \geq 1$ an integer. A **pattern** p on alphabet $\mathcal{A}$ is an element of $\mathcal{A}^{\mathbb{U}}$ for some finite $\mathbb{U} \subset \mathbb{Z}^d$. The set $\mathbb{U}$ is called the **support** of p, and is denoted $\mathrm{supp}(p)$.

Definition 3.6. Given an alphabet $\mathcal{A}$, any colouring of $\mathbb{Z}^d$ with values in $\mathcal{A}$, that is any element of $\mathcal{A}^{\mathbb{Z}^d}$, is called a **configuration**.

Definition 3.7. Let C_1 and C_2 be two configurations. Let S_n be the square of size $2n+1$ centred in $(0,0)$. Let n be the maximum integer such that the pattern on support S_n of C_1 and the one on same support of C_2 coincide, or $+\infty$ if $C_1 = C_2$.

We define the distance on the set of configurations to be $d(C_1, C_2) = 2^{-n}$ if $C_1 \neq C_2$ or 0 otherwise.

Notation 3.8. For a configuration $x = (x_{\mathbf{u}})_{\mathbf{u} \in \mathbb{Z}^d}$ of $\mathcal{A}^{\mathbb{Z}^d}$ (resp. for a pattern $p \in \mathcal{A}^{\mathbb{U}}$ for some $\mathbb{U} \subset \mathbb{Z}^d$), we denote by $x_{|\mathbb{V}}$ the restriction of x to some subset $\mathbb{V} \subset \mathbb{Z}^d$ (resp. the restriction of p to $\mathbb{V} \subset \mathbb{U}$).

Definition 3.9. Let $\mathcal{A}$ be a finite set, and $d \geq 1$ integer. A **d-dimensional subshift** on alphabet $\mathcal{A}$ is a subset of $\mathcal{A}^{\mathbb{Z}^d}$ defined by a set of forbidden patterns. Formally, a subset X of $\mathcal{A}^{\mathbb{Z}^d}$ is a subshift when there exist some finite sets $\mathbb{U} \subset \mathbb{Z}^d$ and $\mathcal{F} \subset \mathcal{A}^{\mathbb{U}}$ such that:

$$X = \left\{ x \in \mathcal{A}^{\mathbb{Z}^d} : \forall\, \mathbf{u} \in \mathbb{Z}^d, x_{|\mathbf{u}+\mathbb{U}} \notin \mathcal{F} \right\}.$$

The elements of $\mathcal{F}$ are called the **forbidden patterns**.

Definition 3.10. Let X be a subshift defined by a set of forbidden patterns $\mathcal{F}$. When $\mathcal{F}$ is finite, then X is called a **subshift of finite type** or SFT.

Definition 3.11. For a subshift X, a **globally-admissible** pattern of size $[\![1, n]\!]^d$ is some pattern $p \in \mathcal{A}^{[\![1,n]\!]^d}$ which appears in a configuration of X, that is when $x_{|[\![1,n]\!]^d} = p$. When $d = 2$, we extend the definition to patterns $p \in \mathcal{A}^{[\![1,n]\!]\times[\![1,m]\!]}$ when there exists a configuration x of X such that $x_{|[\![1,n]\!]\times[\![1,m]\!]} = p$.

Although here we limit ourselves to SFTs, the world of subshifts contains many subshifts which are not of finite type. We will give a few examples in dimension one, on alphabet $\{a, b\}$. The set $X_{\leq 1}$ of words with *at most* one 'a' is not of finite type: an infinite word may contain two 'a's at arbitrarily large distance from each other, which we cannot forbid with finite forbidden patterns. However, there exists a SFT on an alphabet with three symbols which, after remaping one to a and the other two to b, gives the same language as $X_{\leq 1}$: we say that $X_{\leq 1}$ is *sofic*. Now, by modifying slightly the condition, we define $X_{=1}$ to be the set of words with *exactly* one 'a'. This set is not a subshift. It is due to the fact that the desired 'a' may be arbitrarily 'far away'. No sets of forbidden patterns can enforce that an 'a' is present. We can also consider the equivalent definition of subshifts to see that $X_{=1}$ is not one:

Definition 3.12 (Alternate definition of subshift)**.** A set X of configurations of $\mathbb{Z}^d$ with values in $\mathcal{A}$ is a subshift when it is closed and stable by translation.

We can see that the word containing only 'b's is in the closure of $X_{=1}$ but does not belong to $X_{=1}$, which shows that $X_{=1}$ is not closed, hence neither is it a subshift.

3.2.3 The domination subshifts

For our domination subshifts, the alphabet is $\mathcal{A}_0 = \{\square, \blacksquare\}$, and $d = 2$. We work in dimension two since we study grids.

Definition 3.13. The **domination** (resp. **minimal-domination**, **total-domination** and **minimal-total-domination**) subshift denoted by X^{D} (resp. X^{M}, X^{T} and X^{MT}), is the set of elements $x \in \mathcal{A}_0^{\mathbb{Z}^2}$ such that $\{\mathbf{u} \in \mathbb{Z}^2 : x_{\mathbf{u}} = \blacksquare\}$ is a dominating (resp. minimal dominating, total-dominating and minimal total-dominating) set of the infinite square grid $\mathbb{Z}^2$. In all these cases, a configuration x of the subshift is called a **dominated configuration**. We also say that $\mathbf{u}$ is a **dominant position** of the configuration x when $x_{\mathbf{u}}$ is grey. Likewise, a **private neighbour** is still a position which is dominated by exactly one dominant position.

The local characterisations for each type of dominating sets we gave earlier can straightforwardly be translated into finite sets of forbidden patterns. We then obtain the following result:

Proposition 3.3. *The sets X^{D}, X^{M}, X^{T} and X^{MT} are subshifts of finite type.*

As we just mentioned, the key point is that these domination problems have a local characterisation, that is a set of forbidden patterns of bounded sizes. All the problems which enjoy this property can also be associated to SFTs. For instance, the stable set problem for graphs consists in selecting vertices such that no two vertices are connected. In the world of SFTs, it is called the *hard-square* problem, and is associated to the Fibonacci subshift of dimension two. This SFT has been studied a lot, as well as its growth rate. We have good approximations of the growth rate (see for instance [39]), which corresponds to the *entropy* of the subshift. However we do not know its exact value, nor do we have closed formulas accounting for the number of stable sets of the finite grid $G_{n,m}$. These problems of counting turn out to be very hard to solve exactly, even if we only want to find the exact entropy. This chapter is a first step in studying the number of dominating sets for a few variants of domination, by proving that they have some interesting properties and approximating their entropies.

3.3 Comparing the growth of SFTs with the growth of dominating sets

Notation 3.14. For a subshift of finite type X, we denote by $N_n(X)$ the number of globally-admissible patterns of size $[\![1,n]\!]^d$. When $d = 2$, we extend the notation and denote by resp. $N_{n,m}(X)$ the number of globally-admissible patterns of size $[\![1,n]\!]\times[\![1,m]\!]$.

Definition 3.15. The **topological entropy** of a subshift of finite type is the number

$$h(X) = \inf_{n\to+\infty} \frac{\log_2(N_n(x))}{n^d}.$$

We will simply refer to this notion as the entropy in the rest of the manuscript. The following two lemmas are well known (see for instance [33]).

Lemma 3.4. *The infimum in the definition of $h(X)$ is in fact a limit:*

$$h(X) = \lim_{n} \frac{\log_2(N_n(x))}{n^d}.$$

Notation 3.16. Let us denote by σ the $\mathbb{Z}^d$-**shift** action on $\mathcal{A}^{\mathbb{Z}^d}$ defined such that for all $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^d$,

$$(\sigma^{\mathbf{u}}x)_{\mathbf{v}} = x_{\mathbf{v}+\mathbf{u}}\,.$$

Informally, σ acts on a configuration by translating it by the vector $\mathbf{u}$.

Definition 3.17. A **conjugation** between two d-dimensional subshifts of finite type X and Z is an invertible map $\varphi : X \to Z$ such that, for all $\mathbf{u} \in \mathbb{Z}^d$ and $x \in X$, $\varphi(\sigma^{\mathbf{u}}.x) = \sigma^{\mathbf{u}}.\varphi(x)$. In this case, X and Z are said to be **conjugated**.

Lemma 3.5. *If two subshifts of finite type X and Z are conjugated, then $h(X) = h(Z)$.*

The entropy is one parameter which is preserved by conjugation. It may be a way to show that two subshifts are not conjugated, by showing that their entropies are different. We will use this lemma and the concept of conjugation only in Section 3.5.1.

Lemma 3.6. *Let X be a bidimensional subshift of finite type. Then:*

$$h(X) = \lim_{n,m} \frac{\log_2(N_{n,m}(X))}{nm}.$$

This limit is to be understood as letting both n and m tend towards infinity at the same time (but possibly at different speeds). We can see that this limit exists and corresponds to the entropy by decomposing a square into rectangles and doing the other way around. This allows us to bound by below and by above this limit by numbers which tend towards the entropy $h(X)$.

We now need to show, for each variant of the domination, that the SFT and its associated dominating set problem grow at the same speed, that is the number of $n \times m$ patterns appearing in the SFT grows at the same pace as the number of dominating sets of the $n \times m$ grid. This is not trivial, since the dominating sets of a finite grid $G_{n,m}$ do not correspond exactly to the globally-admissible patterns on the same grid of the corresponding SFT type presented in Section 3.2.3. Indeed, in such a pattern, the positions of the border may for instance be dominated by a position outside the pattern in a configuration in which the pattern appears. Nonetheless, we will see that we can compare the number of globally-admissible patterns of size $n \times m$ for X^{D} (resp. X^{M}, X^{T} and X^{MT}) to the number of dominating sets (resp. minimal dominating sets, total-dominating set and minimal total-dominating sets) of $G_{n,m}$. We will later use this to prove the existence of an asymptotic growth rate for the grid, turning out to be equal to the entropy of the corresponding SFT.

For concision, we will assimilate the set of vertices of $G_{n,m}$ to any translate of $[\![1, n]\!] \times [\![1, m]\!]$. We will assimilate any dominating (for any domination problem) set S of vertices of a finite grid $G_{n,m}$ with the pattern p of $\mathcal{A}^{\mathbb{Z}^2}$ on $[\![1, n]\!] \times [\![1, m]\!]$ defined by $p_{\mathbf{u}}$ being grey if and only if $\mathbf{u} \in S$.

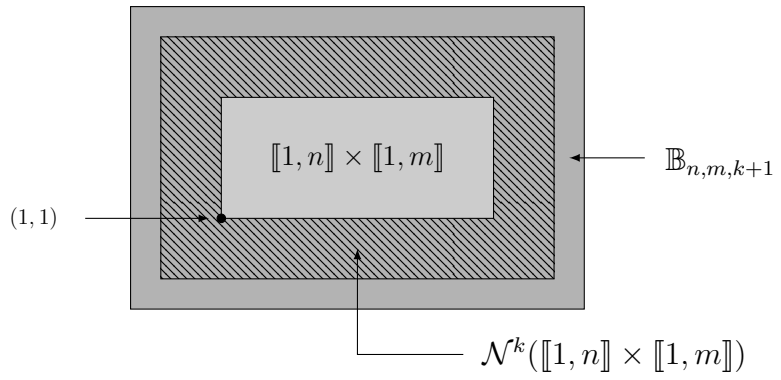
Definition 3.18. If $\mathbb{U}$ is a subset of $\mathbb{Z}^2$, we define the (extended) **neighbourhood** of $\mathbb{U}$ as

$$\mathcal{N}(\mathbb{U}) = \bigcup_{\mathbf{u} \in \mathbb{U}} \left(\mathbf{u} + [\![-1, 1]\!]^2\right).$$

Using iterates of the function $\mathcal{N}$ we also define, for all $n, m \geq 1$ and $k \geq 1$, the **border**

$$\mathbb{B}_{n,m,k} = \mathcal{N}^k([\![1, n]\!] \times [\![1, m]\!]) \setminus \mathcal{N}^{k-1}([\![1, n]\!] \times [\![1, m]\!]).$$

For convenience, we extend the notation to $\mathbb{B}_{n,m,0} = [\![1, n]\!] \times [\![1, m]\!] \setminus [\![2, n-1]\!] \times [\![2, m-1]\!]$.



Lemma 3.7. *For all $n, m \geq 3$, the following inequalities hold:*

$$N_{n-2,m-2}(X^{\mathrm{D}}) \leq D_{n,m} \leq N_{n,m}(X^{\mathrm{D}}).$$

Proof.

1. For all $n, m \geq 1$, any dominating set of $G_{n,m}$ can be extended into a configuration of X^{D} by defining the symbol of any position outside $[\![1,n]\!] \times [\![1,m]\!]$ to be grey. As a consequence, any dominating set of $G_{n,m}$ is globally admissible in X^{D} and thus $D_{n,m} \leq N_{n,m}(X^{\mathrm{D}})$.

2. Any pattern of X^{D} on $[\![1,n]\!] \times [\![1,m]\!]$ can be turned into a dominating set of $[\![0,n+1]\!] \times [\![0,m+1]\!]$ by extending it with grey symbols. Hence we obtain the inequality $N_{n,m}(X^{\mathrm{D}}) \leq D_{n+2,m+2}$ for all $n, m \geq 1$.

□

Using very similar arguments, we obtain the same inequality for the total domination.

Lemma 3.8. *For all $n, m \geq 2$, the following inequalities hold:*

$$N_{n-2,m-2}(X^{\mathrm{T}}) \leq T_{n,m} \leq N_{n,m}(X^{\mathrm{T}}).$$

We then address the minimal and minimal total domination. As we will see, the proofs of the following inequalities are more complex.

Lemma 3.9. *For all $n, m \geq 1$, the following inequalities hold:*

$$\frac{1}{2^{6(n+m)}} N_{n,m}(X^{\mathrm{M}}) \leq M_{n,m} \leq N_{n,m}(X^{\mathrm{M}}).$$

Proof.

1. **Second inequality.**

   (a) **A completion algorithm of a minimal dominating set into a configuration of X^{M}.**

   Let S be a minimal dominating set of $[\![1,n]\!] \times [\![1,m]\!]$. Let us extend it into a configuration x of X^{M} using the following algorithm: successively for every $k \geq 0$, we extend the current pattern into a pattern on $\mathcal{N}^{k+1}([\![1,n]\!] \times [\![1,m]\!])$ using the following operations, for all $\mathbf{u} \in \mathbb{B}_{n,m,k+1}$:

   i. if $\mathbf{u}$ is a corner then $x_{\mathbf{u}}$ is white;
   ii. if $\mathbf{u}$ is a neighbour of a corner in one of the vertical sides of $\mathbb{B}_{n,m,k+1}$ then $x_{\mathbf{u}}$ is white;
   iii. for every other $\mathbf{u}$, $x_{\mathbf{u}}$ is grey if and only if its neighbour in $\mathcal{N}^k([\![1,n]\!] \times [\![1,m]\!])$ is neither dominated by an element in this set, nor a dominant element.

   This algorithm is illustrated in Figure 3.2.

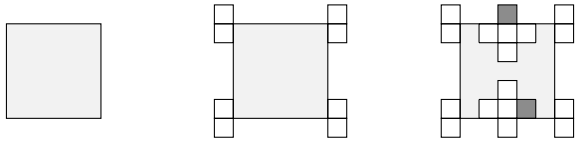


Figure 3.2: Illustration of the completion algorithm in X^{M}: steps of the algorithm are applied successively from left to right.

   (b) **The output obtained by repeating the algorithm is a configuration of X^{M}.**

   - **Every position is dominated.**
     This is verified for the positions in $[\![2,n-1]\!] \times [\![2,m-1]\!]$ because we start from a minimal dominating set. Outside this set, if a position in some $\mathcal{N}^k([\![1,n]\!] \times [\![1,m]\!])$ (for $k \geq 0$) is not dominated before extending the configuration on $\mathcal{N}^{k+1}([\![1,n]\!] \times [\![1,m]\!])$, then it gets dominated at this step by Rule iii and stays that way afterwards.
   - **Every dominant position is isolated or has a private neighbour.**
     Let us consider a dominant position $\mathbf{u}$ which is not isolated. If it lies in $[\![2,n-1]\!] \times [\![2,m-1]\!]$, then it has a private neighbour since the pattern on $[\![1,n]\!] \times [\![1,m]\!]$ is a minimal dominating set of $G_{n,m}$. Otherwise, it lies in some $\mathbb{B}_{n,m,k}$ for some $k \geq 0$ and there are two cases:
     – **$\mathbf{u}$ is not a corner.**
       Its neighbour $\mathbf{v} \in \mathbb{B}_{n,m,k+1}$ is white by the application of the algorithm. Also, since its neighbours in $\mathbb{B}_{n,m,k}$ are thus dominant or dominated, their neighbours in $\mathbb{B}_{n,m,k+1}$ are white. In addition, the neighbour of $\mathbf{v}$ in $\mathbb{B}_{n,m,k+2}$ is thus white. This is illustrated in Figure 3.3. As a consequence, $\mathbf{v}$ is a private neighbour for $\mathbf{u}$.

– **u is a corner.**
This case can only happen for the corners of $[\![1, n]\!] \times [\![1, m]\!]$: the other corners are white by Rule i. We apply a similar reasoning: u has two neighbours, so we may use the previous proof by considering for instance the one at its left or at its right, which is left white by Rule ii.

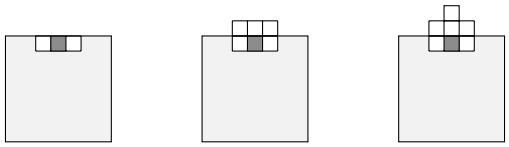


Figure 3.3: Illustration of the proof of a private neighbour for a non-isolated position. Steps of the completion algorithm for X^{M} applied from left to right.

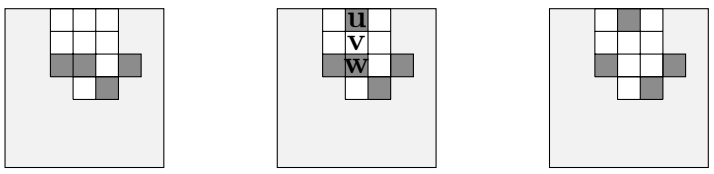
2. **First inequality.**

(a) **Transforming patterns of X^{M} into minimal dominating sets.**
Let us define an application $\phi_{n,m}$ which, to each pattern of X^{M} on $[\![1, n]\!] \times [\![1, m]\!]$, associates a minimal dominating set of $G_{n,m}$ defined by:

i. suppressing any dominant position in $\mathbb{B}_{n,m,0}$ (see Definition 3.18 for the definition of the borders) which has no private neighbours in $G_{n,m}$ and which is dominated by an element of $G_{n,m}$;

ii. changing successively any non-dominant position of $\mathbb{B}_{n,m,0}$ which is still not dominated into a dominant one;

iii. successively, for every dominant position $\mathbf{u} \in \mathbb{B}_{n,m,0}$: if one of $\mathbf{u}$'s neighbours $\mathbf{v}$ is the only private neighbour of a position $\mathbf{w}$ which is not isolated in $G_{n,m}$ then change $\mathbf{w}$ into a non-dominant position.
This step is illustrated in Figure 3.4.



After Step i. After Step ii. After Step iii.

Figure 3.4: Illustration of the second and then third steps of the algorithm defining $\phi_{n,m}$ for X^{M}, from left to right. $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$ are instances of the positions described in Rule iii.

(b) **Verifying that images of $\phi_{n,m}$ are minimal dominating sets.**
Let us consider a globally-admissible pattern p of X^{M} on $[\![1, n]\!] \times [\![1, m]\!]$. Let us show that the set $\phi_{n,m}(p)$ is a minimal dominating set of $G_{n,m}$:

- **Any vertex of $G_{n,m}$ is dominated or dominant in $\phi_{n,m}(p)$.**
  Before Step ii, if a position is not dominant and not dominated, it becomes dominant during this step. Every position in $[\![2, n-1]\!] \times [\![2, m-1]\!]$ was already dominated since the pattern was in X^{M}. Moreover, during Step iii, any position which is modified to no longer be dominant is necessarily a dominated position ('**w**' is chosen among non-isolated positions).
- **Any non-isolated dominant position has a private neighbour.**
  Before applying $\phi_{n,m}$, only the positions on the border $\mathbb{B}_{n,m,0}$ might not have any private neighbour. After Step i, every dominant position on $\mathbb{B}_{n,m,0}$ is isolated, or has a private neighbour. After Step ii, some positions may be dominant, non-isolated, and have no private neighbours. Such positions obtain a private neighbour in Step iii.

(c) For all n, m, the number of preimages of $\phi_{n,m}$ for any minimal dominating set of $G_{n,m}$ is bounded by $2^{3(2n+2m)}$: any symbol modified by the application is at distance at most two from $\mathbb{B}_{n,m,0}$, and there are $3(2n+2m)$ such symbols. Therefore $N_{n,m}(X^{\mathrm{M}}) \leq 2^{6(n+m)} M_{n,m}$.

□

Lemma 3.10. *For all n, the following bounds hold:*

$$\frac{1}{2^{8(m+n)}} N_{n,m}(X^{\mathrm{MT}}) \leq MT_{n,m} \leq N_{n,m}(X^{\mathrm{MT}}).$$

For readability, we reproduce the structure of the proof of Lemma 3.9, but simplify the arguments and refer to this proof.

Proof.

1. **Second inequality.**

   (a) **A completion algorithm of a minimal total-dominating set into a configuration of X^{MT}.**

   Let us consider a minimal total-dominating set of $G_{n,m}$. Any element in $[\![2, n-1]\!] \times [\![2, m-1]\!]$ is dominated by an element of $[\![1, n]\!] \times [\![1, m]\!]$, and any dominant element in $[\![2, n-1]\!] \times [\![2, m-1]\!]$ is not isolated and has a private neighbour in $[\![1, n]\!] \times [\![1, m]\!]$ (which may or may not be a dominant position). Let us extend this set into a configuration x of X^{MT} using an algorithm very similar to the one in the corresponding point in the proof of Lemma 3.9. The condition in the third point is different:

   i. if **u** is a corner then $x_{\mathbf{u}}$ is white;
   ii. if **u** is a neighbour of a corner in one of the vertical sides of $\mathbb{B}_{n,m,k+1}$ then $x_{\mathbf{u}}$ is white;
   iii. for every other **u**, $x_{\mathbf{u}}$ is grey if and only if its neighbour in $\mathcal{N}^k([\![1, n]\!] \times [\![1, m]\!])$ is not dominated by an element in this set.

(b) **The result of the algorithm is a configuration of X^{MT}.**

- **Every position is dominated.**
  Similar to the corresponding point in the proof of Lemma 3.9. This implies that no dominant positions are isolated.
- **Every dominant position has a private neighbour.**
  Let us consider a dominant position **u**. If it is in $[\![3, n-2]\!] \times [\![3, m-2]\!]$, since the pattern on $[\![1, n]\!] \times [\![1, m]\!]$ is a minimal total-dominating set of $G_{n,m}$, we know that it has a private neighbour. Otherwise, it lies in some $\mathbb{B}_{n,m,k}$ for $k \geq 0$, or in $[\![2, n-1]\!] \times [\![2, m-1]\!]$. Then there are two cases:
  - **u is not a corner.** If it has no dominant neighbours in $\mathcal{N}^k([\![1, n]\!] \times [\![1, m]\!])$, let us call **v** its neighbour in $\mathbb{B}_{n,m,k+1}$. Note that, depending on whether or not **u** is dominated inside $\mathcal{N}^k(\mathbb{B}_{n,m,0})$, **v** may be white or grey. Since the neighbours of **u** in $\mathbb{B}_{n,m,k}$ are dominated, **v**'s neighbours in $\mathbb{B}_{n,m,k+1}$ are white. Finally, since **v** is dominated by **u**, its neighbour in $\mathbb{B}_{n,m,k+2}$ is white, hence **v** is a private neighbour for **u**.
  - **u is a corner.** We apply a similar reasoning.

2. **First inequality:**

(a) **A transformation of patterns of X^{MT} into minimal total-dominating sets.**

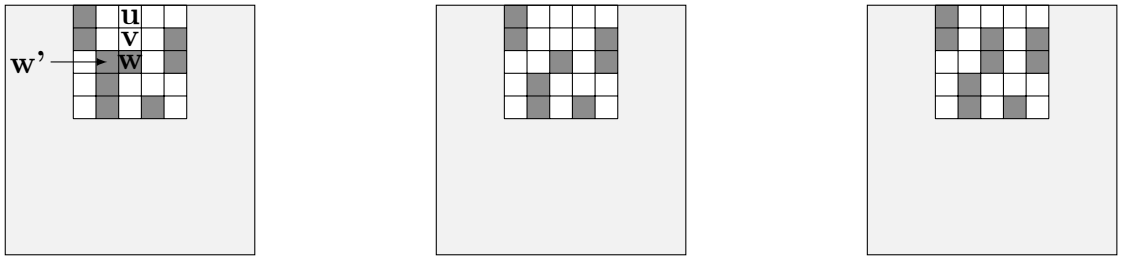
Let us define once again an application $\phi_{n,m}$ which, to each pattern of X^{MT} on $[\![1, n]\!] \times [\![1, m]\!]$, associates a minimal total-dominating set of $G_{n,m}$. It is defined in a similar way as in the corresponding point in the proof of Lemma 3.9, but the proof is more complex.

i. Suppress any dominant position on the border $\mathbb{B}_{n,m,0}$ which has no private neighbours in $G_{n,m}$.

ii. Successively, for every non-corner undominated position **u** on the border $\mathbb{B}_{n,m,0}$, do the following:
   - Consider the position **v**, neighbour of **u** in $[\![2, n-1]\!] \times [\![2, m-1]\!]$. For each dominant position **w** in the neighbourhood of **v**, and for each dominant position **w'** in the neighbourhood of **w**, if **w** is the only private neighbour of **w'**, then change **w'** into a non-dominant position.
   - Change **v** into a dominant position.

   Then do the same operations for the corners of $\mathbb{B}_{n,m,0}$, except that **v** is replaced by any neighbour of the corner.

This Step is illustrated on Figure 3.5.



After Step i. After Step ii, first point. After the second point.

Figure 3.5: Illustration of the second and then third steps of the algorithm defining $\phi_{n,m}$ for X^{MT}, from left to right. $\mathbf{u}, \mathbf{v}, \mathbf{w}$ and $\mathbf{w'}$ are instances of the positions described in Rule ii.

(b) **Verification that images of $\phi_{n,m}$ are minimal total-dominating sets.** Consider a pattern p of X^{MT} on $[\![1, n]\!] \times [\![1, m]\!]$. The set $\phi_{n,m}(p)$ is a minimal total-dominating set of $G_{n,m}$:

- **Any vertex of $G_{n,m}$ is dominated in $\phi_{n,m}(p)$.**
  Any (dominant or not) position which was dominated before applying Rule i is still dominated afterwards: if some position $\mathbf{u}$ lies in the neighbourhood of a dominant position $\mathbf{v}$ suppressed by Rule i, then since $\mathbf{v}$ had no private neighbours in $G_{n,m}$, $\mathbf{u}$ is dominated by another position. For similar reasons, no positions become undominated after the application of Rule ii: only the neighbours of some $\mathbf{w'}$ could be affected and if $\mathbf{w'}$ becomes non-dominant it means that they were dominated by other positions, so that they stay dominated. Since all the positions inside $[\![2, n-1]\!] \times [\![2, m-1]\!]$ were dominated before applying the rules, it only remains to show that the positions inside $\mathbb{B}_{n,m,0}$ are dominated after applying Rule ii. This is true thanks to this rule: any undominated position $\mathbf{u}$ inside the border sees its neighbour $\mathbf{v}$ inside $[\![2, n-1]\!] \times [\![2, m-1]\!]$ become dominant. The same applies to the corners, except that the neighbour comes from the border.
- **Any dominant position has a private neighbour.**
  At the end of Step i, any dominant position has a private neighbour. Only the creation of a dominant position $\mathbf{v}$ during the execution of Rule ii on position $\mathbf{u}$ could affect this property, by disabling the private neighbour of a position $\mathbf{w}$ in its neighbourhood, or by not having any private neighbour itself. The first case cannot happen since any dominant position $\mathbf{w'}$ having $\mathbf{w}$ as its unique private neighbour is suppressed. The second one also never happens since the position $\mathbf{u}$ is a private neighbour for $\mathbf{v}$.

(c) For all n and m, the number of preimages of $\phi_{n,m}$ for any minimal dominating set of $G_{n,m}$ is bounded by $2^{4(2m+2n)}$, since any symbol modified by the application is at distance at most 4 of the border of $[\![1, n]\!] \times [\![1, m]\!]$. As a consequence, $N_{n,m}(X^{\mathrm{MT}}) \leq 2^{8(m+n)} MT_{n,m}$.

□

Theorem 3.11 (Asymptotic behaviour)**.** *There exists some $\nu_{\mathrm{D}} \geq 0$ (resp. ν_{M}, ν_{T} and ν_{MT}) such that*

$$D_{n,m} = \nu_{\mathrm{D}}^{nm+o(nm)}$$

(resp. $M_{n,m} = \nu_{\mathrm{M}}^{nm+o(nm)}$, $T_{n,m} = \nu_{\mathrm{T}}^{nm+o(nm)}$ and $MT_{n,m} = \nu_{\mathrm{MT}}^{nm+o(nm)}$).

Proof. Let us prove this for the sequence $(M_{n,m})$ (the proof is similar for the other sequences).

As a consequence of Lemma 3.9, for all n, m:

$$-\frac{6(m+n)}{nm} + \frac{\log_2(N_{n,m}(X^{\mathrm{M}}))}{nm} \leq \frac{\log_2(M_{n,m})}{nm} \leq \frac{\log_2(N_{n,m}(X^{\mathrm{M}}))}{nm}.$$

As a consequence,

$$\frac{\log_2(M_{n,m})}{nm} \to h(X^{\mathrm{M}}).$$

This means that $M_{n,m} = 2^{h(X^{\mathrm{M}})\cdot nm+o(nm)} = \nu_{\mathrm{M}}^{nm+o(nm)}$, where $\nu_{\mathrm{M}} = 2^{h(X^{\mathrm{M}})}$. □

3.4 Computability of the entropy: the block-gluing property

In this section, we prove that the growth rates ν_{D} (resp. ν_{M}, ν_{T} and ν_{MT}) are computable numbers, meaning that there exists an algorithm which computes approximations of these numbers with arbitrary given precision. For this purpose, we rely on the *block-gluing* property. If a subshift of finite type has this property then it allows us to compute it with the known Algorithm 1 on page 73. We will finally prove that X^{D} (resp. X^{M}, X^{T} and X^{MT}) are block gluing.

3.4.1 Definition and properties

For two finite subsets $\mathbb{U}, \mathbb{V}$ of $\mathbb{Z}^2$, we write

$$\delta(\mathbb{U}, \mathbb{V}) = \min_{\mathbf{u}\in\mathbb{U}}(\min_{\mathbf{v}\in\mathbb{V}} ||\mathbf{v}-\mathbf{u}||_\infty).$$

This corresponds to the shortest distance between a point in $\mathbb{U}$ and one in $\mathbb{V}$. Note that a square of length one contains four points in $\mathbb{Z}^2$ if placed on integer coordinates. This means that if there is one column between $\mathbb{U}$ and $\mathbb{V}$ then the distance between them is one (and not zero). The usual definition of the block-gluing property is the following one.

Definition 3.19. For a fixed integer $c \geq 0$, we say that a bidimensional subshift of finite type X on alphabet $\mathcal{A}$ is c-block-gluing when, for every $n \geq 0$ and any two globally-admissible patterns p and q of X on support $[\![1, n]\!]^2$, for all $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^2$ such that $\delta(\mathbf{u} + [\![1, n]\!]^2, \mathbf{v} + [\![1, n]\!]^2) \geq c$, there exists a configuration $x \in X$ such that $x_{|\mathbf{u}+[\![1,n]\!]^2} = p$ and $x_{|\mathbf{v}+[\![1,n]\!]^2} = q$.
We say that X is block gluing if it is c-block-gluing for some integer c.

Informally, this means that any pair of rectangular patterns placed at whatever positions can be completed into a configuration of X provided that there are at least c lines or columns separating the two patterns.

Notation 3.20. For any subshift of finite type X, we denote by $c(X)$ the smallest c such that X is c-block-gluing. If X is not block gluing for any integer c, we write $c(X) = +\infty$.

In the following, we will use the notations $\mathbb{Z}_- =]\!]-\infty, 0]\!]$ and $\mathbb{Z}_+ =]\!]0, +\infty[\!]$. We give in Proposition 3.12 an equivalent characterisation of the block-gluing property:

Remark 3.1. In Proposition 3.12 we extend the notion of *patterns* to **infinite supports**. This will be the case whenever their supports are infinite.

Proposition 3.12. *Let $c \geq 0$ be an integer. A bidimensional subshift X is c-block-gluing if and only if for all $k \geq c$ and p and q globally-admissible patterns on supports $\mathbb{Z}_- \times \mathbb{Z}$ (resp. $\mathbb{Z} \times \mathbb{Z}_-$) and $\mathbb{Z}_+ \times \mathbb{Z}$ (resp. $\mathbb{Z} \times \mathbb{Z}_+$), there exists a configuration $x \in X$ such that $x_{|\mathbb{Z}_- \times \mathbb{Z}} = p$ and $x_{|(k,0)+\mathbb{Z}_+ \times \mathbb{Z}} = q$ (resp. $x_{|\mathbb{Z} \times \mathbb{Z}_-} = p$ and $x_{|(0,k)+\mathbb{Z} \times \mathbb{Z}_+} = q$).*

Informally, this means that in order to check the block-gluing property, it is sufficient to prove that any two patterns on half-planes glued at arbitrary distance, provided it is greater than c, can be completed into a configuration of X.

Proof.

- ($\Leftarrow$): Let us assume that X satisfies the second hypothesis. Let us consider some integer n, and two globally-admissible patterns $\overline{p}, \overline{q}$ of X on support $[\![1, n]\!]^2$. We prove the result in the case of columns separating the two patterns: the proof when the patterns are separated by lines is completely symmetrical. We place $\overline{p}$ and $\overline{q}$ such that $k \geq c$ columns separate them. Since $\overline{p}$ and $\overline{q}$ are globally admissible, there exist p and q globally-admissible infinite patterns of X on respective supports $\mathbb{Z}_- \times \mathbb{Z}$ and $(k, 0) + \mathbb{Z}_+ \times \mathbb{Z}$ which extend respectively $\overline{p}$ and $\overline{q}$. By hypothesis, there exists some configuration $x \in X$ which extends p and q separated by k columns, hence X is c-block-gluing.

- ($\Rightarrow$): Let us assume that the first hypothesis on X is true, and let p and q be two patterns on supports $\mathbb{Z}_- \times \mathbb{Z}$ and $(k, 0) + \mathbb{Z}_+ \times \mathbb{Z}$ for some $k \geq c$ (the case with $\mathbb{Z} \times \mathbb{Z}_-$ and $\mathbb{Z} \times \mathbb{Z}_+$ is proved in the same way). From the block-gluing property, for all $n > 0$ one can extend the restriction of p on $[\![1, n]\!]^2 - (n, 0)$ and the restriction of q on $[\![1, n]\!]^2 - (n, 0) + (k + 1, 0)$ into a configuration $x_n \in X$. By compactness of the subshift X for the product of the discrete topology, this sequence admits a subsequence which converges to some $x \in X$. This x satisfies the equalities $x_{|\mathbb{Z}_- \times \mathbb{Z}} = p$ and $x_{|(k,0)+\mathbb{Z}_+ \times \mathbb{Z}} = q$.

□

3.4.2 Algorithmic computability of the entropy

Definition 3.21. A function $f : \mathbb{N} \to \mathbb{N}$ is **computable** when there exists a Turing machine which, on any integer n written in binary[4] on its input tape, terminates with $f(n)$, also written in binary, on its output tape.

Definition 3.22. Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. A real number x is said to be **computable** with rate f when there exists an algorithm which, given an integer n as input, outputs in at most $f(n)$ steps a rational number r_n such that $|x - r_n| \leq \frac{1}{n}$.

This definition corresponds to Definition 1.3 in [40]. The following theorem is Theorem 1.4 in the same reference. Its proof provides an algorithm to compute $h(X)$.

Theorem 3.13 ([40])**.** *Let X be a block-gluing bidimensional subshift of finite type. Then $h(X)$ is computable with rate $n \mapsto 2^{O(n^2)}$.*

Remark 3.2. Let us note that in general the entropy of a bidimensional subshift of finite type is not computable at all (see Theorem 1.1 in [31] and the existence of non-computable right-recursively-enumerable numbers). This motivates our detour by the block-gluing property, which is needed for Algorithm 1.

Definition 3.23. Given an SFT, every pattern which does not contain any forbidden pattern is called **locally admissible**.

Any globally-admissible pattern is locally admissible: it can be extended to a configuration of the subshift in which no forbidden patterns appear. However the contrary is not true: if for instance we define a one-dimensional SFT on alphabet $\{a, b\}$ by forbidding patterns aa, bb and aba then a is locally admissible but not globally admissible since any word of $\mathbb{Z}^{\{a,b\}}$ extending it will necessarily contain a forbidden pattern.

The following lemma gives us an effective way to compute the entropy of a SFT. It relates the entropy to the locally-admissible patterns instead of using globally-admissible patterns the way it is defined (Definition 3.15). Indeed, it is possible to check that a pattern is locally admissible by verifying that no forbidden patterns appear. On the contrary, there is no automatic way to check that a given pattern is globally admissible: one way would be to find an extension of it to $\mathbb{Z}^2$, but this might not end in finite time.

Lemma 3.14 ([40])**.** *Let X be a c-block-gluing bidimensional subshift of finite type on alphabet $\mathcal{A}$. For all $k \geq 1$, the number $N_k(X)$ is equal to the number of $k \times k$ patterns which appear in a $(|\mathcal{A}|^{2c+1} \cdot (c + k) + 1) \times (2c+k+2)$ locally-admissible rectangular pattern whose restrictions on the two extremal vertical (resp. horizontal) edges are equal.*

The algorithm is as follows:

[4]Note that all binary digits are either 0 or 1. Sex, however, is not binary: there are intersex people who, on **biological sex** criterion, are neither male nor female. Gender also is not binary: some people categorise themselves as men, some as women, some as agender, gender-fluid, transgender, and a lot of other genders.

Algorithm 1: Computing the entropy of a c-block-gluing bidimensional SFT.

Input: An integer n, an alphabet $\mathcal{A}$ and a set of patterns $\mathcal{F}$ of $\mathcal{A}^{\mathbb{U}}$ for some finite $\mathbb{U} \subset \mathbb{Z}^2$

Output: A rational approximation of $h(X)$ up to $1/n$, where X is the SFT on alphabet $\mathcal{A}$ defined by the set of forbidden patterns $\mathcal{F}$

$k \leftarrow 0$
$r \leftarrow +\infty$
while $r \geq 1/2n$ **do**
  $k \leftarrow k + 1$
  $m \leftarrow N_k(X)$ (this is a sub-procedure using Lemma 3.14).
  $r \leftarrow$ some rational approximation up to $1/2k$ of $\frac{\log_2(N_k(X))}{k^2} - \frac{\log_2(N_k(X))}{(k+c)^2}$
end
Return a rational approximation up to $1/2n$ of $\log_2(N_k(X))/k^2$

3.4.3 Some dominating subshifts are block-gluing

It is straightforward to check that the domination subshift X^{D} and the total-domination subshift X^{T} satisfy the block-gluing property, with $c(X^{\mathrm{D}}) = 1$ (just fill every cell with grey). In this section, we prove that X^{M} and X^{MT} also satisfy this property. However, to show that this problem is not trivial, we first show that not all domination problems have the block-gluing property.

Definition 3.24. Let σ and ρ be two sets of integers (subsets of $[\![0, 4]\!]$ in the case of grids). S is said to be **(σ, ρ)-dominating** when

1. the number of neighbours in S of each vertex in S belongs to σ;
2. the number of neighbours in S of each vertex outside S belongs to ρ.

For instance, the domination is the $(\{0, 1, 2, 3, 4\}, \{1, 2, 3, 4\})$-domination and the total domination is the $(\{1, 2, 3, 4\}, \{1, 2, 3, 4\})$-domination.

Proposition 3.15. *The $(\{3\}, \{1\})$-domination subshift is not block-gluing.*

Notation 3.25. In the following, for all $j \in \mathbb{Z}$, we denote by C_j the column $\{j\} \times \mathbb{Z}$ of $\mathbb{Z}^2$.

Proof. We prove this result by providing a half-plane pattern which can be glued with itself only if the number of columns between them is of the form $4k + 2$.

We use Proposition 3.12 so that instead of finite pattern we may use a half-plane pattern. We take the half-plane pattern p whose rightmost two columns (i.e. C_0 and C_{-1}) are filled with grey (i.e. dominant elements), the two at their left are filled with white, the two at their left with grey, and so on: C_j for $j \leq 0$ is grey if $-j \mod 4 \in \{0, 1\}$ and is white otherwise (see Figure 3.6). We take q as the vertical symmetric of p, as in Figure 3.6. Since in the column called C_0 in the figure, each vertex has exactly 3 neighbours in the dominating set, its neighbour in C_1 must be white. Every element in C_1 having a neighbour which dominates them, each neighbours of an element of C_1 must be white, hence C_2 is also white. Now the elements of C_2 are not dominated, so C_3 must

be grey. Every element of C_2 now has only 2 grey neighbours, so that C_4 must be grey, and so on. q forces every column C_{4k-1} and C_{4k} to be grey, and every C_{4k+1} and C_{4k+2} to be white. However, q forces the exact opposite, which means that for any $k \geq 0$, p and q cannot be glued with a gap of size $4k$ for instance, hence the subshift is not block gluing.

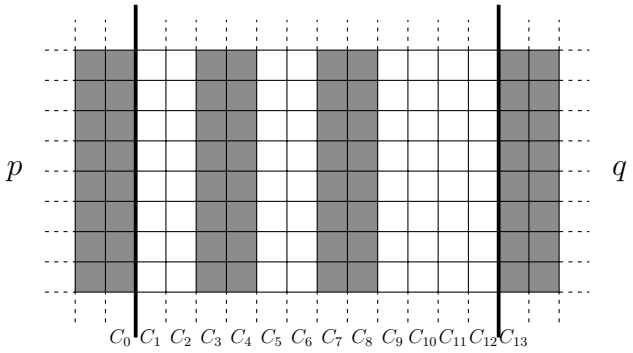


Figure 3.6: The $\{\{3\},\{1\}\}$ domination is not block gluing. The half-plane pattern p forces every C_{4k-1} and C_{4k} to be grey and the rest to be white, whereas q forces the opposite. Hence they cannot be glued whenever $4k$ columns separate them.

□

Remark 3.3. Proposition 3.12 is very useful to avoid some technicalities, which we briefly mention here for the sake of insight. If we only considered finite patterns p and q as they are in Figure 3.6, the proof would have been longer. Let us consider in this remark that p and q are patterns with 8 lines an 2 columns as in the figure. Then the top visible cell of C_2 could have been dominated by either its upper neighbour or the one in C_3. So we would have had to argue that this is not possible, or that it is not a problem: the sagging of the grey cells is linear in the size of the gap, so taking arbitrarily large (but finite) patterns will show that any gap of constant size would not work.

Theorem 3.16. *The minimal-domination subshift is block gluing and* $c(X^{\mathrm{M}}) = 5$.

Idea of the proof: *In order to simplify the proof of the block-gluing property, we rely again on Proposition 3.12. The proof of the block-gluing property for two half-plane patterns consists in determining successively how to fill the k intermediate columns from the patterns towards the 'centre' (chosen, for concision, to be column* C_{k-2}*). The completion follows an algorithm which enforces that, when the number of intermediate columns is large enough, any added dominant element is isolated or has a private neighbour in an already-constructed column. This ensures that the rules of the subshift are not broken. We now give the full proof.*

Proof.

- **Filling the intermediate columns between two half-plane patterns.**

  Let p and q be two patterns respectively on $\mathbb{Z}_- \times \mathbb{Z}$ and $\mathbb{Z}_+ \times \mathbb{Z}$ (the proof for the vertical case is similar). Let us determine a configuration of $\mathcal{A}_0^{\mathbb{Z}^2}$ such that $x_{|\mathbb{Z}_- \times \mathbb{Z}} = p$ and $x_{|(k,0)+\mathbb{Z}_+ \times \mathbb{Z}} = q$. The intermediate columns $C_1, ..., C_k$ are determined by the following algorithm. It intuitively puts grey cells only when it is really necessary to dominate the neighbour in the previous column:

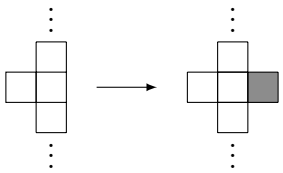


Figure 3.7: Illustration of the rule for filling the non-central intermediate columns for X^{M}. It is also applied symmetrically if the neighbour in C_{k-1} is not dominated.

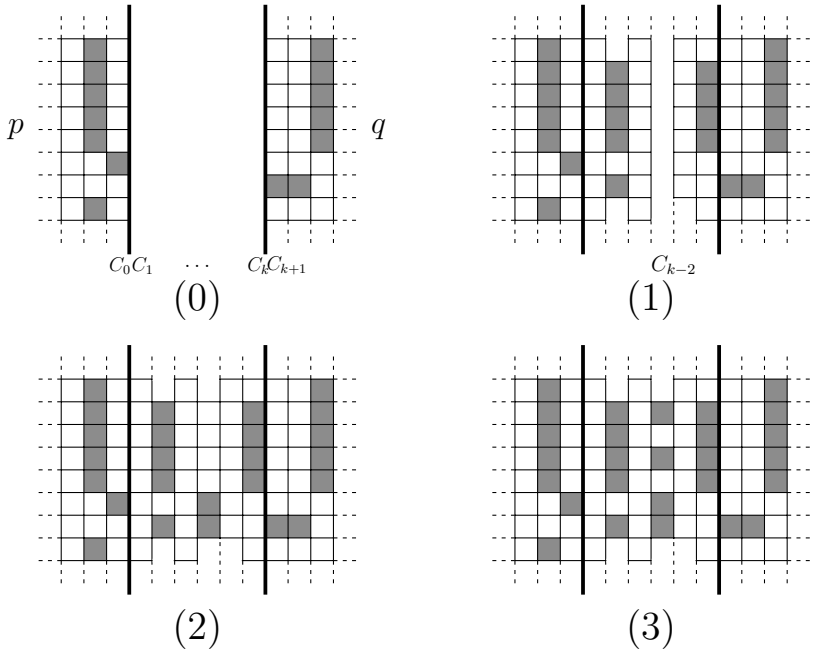


Figure 3.8: Illustration of the algorithm filling the intermediate columns between two half-plane patterns p and q for the minimal domination.
(0) Initial setting of the two patterns.
(i) After Step i of the algorithm
Since we show only finite sub-parts of p and q, the values of some cells in between remain unknown. We left them non filled and remove one of their boundary. We chose $k = 6$ for the illustration, still the proof works from $k = 5$ upwards.

1. **Filling the intermediate columns, from C_1 to C_{k-3}, then C_k, C_{k-1}.**
   Successively, for all j from 1 to $k-3$, we determine the column C_j according to the following rule: for all $\mathbf{u} \in C_j$, $x_{\mathbf{u}}$ is ■ when $x_{\mathbf{u}-(1,0)}$, $x_{\mathbf{u}-(1,1)}$, $x_{\mathbf{u}-(1,-1)}$ and $x_{\mathbf{u}-(2,0)}$ are □ (see Figure 3.7). Else, $x_{\mathbf{u}}$ is set to □. Similarly, for $j = k$ and then $j = k-1$, we determine x on any position $x_{\mathbf{u}}$ for $\mathbf{u} \in C_j$ by applying a symmetrical rule: $x_{\mathbf{u}}$ is ■ when $x_{\mathbf{u}+(1,0)}$, $x_{\mathbf{u}+(1,1)}$, $x_{\mathbf{u}+(1,-1)}$ and $x_{\mathbf{u}+(2,0)}$ are □. Else, $x_{\mathbf{u}}$ is set to □.
2. **The central column C_{k-2}.**
   We now determine x on the central column C_{k-2}. For all $\mathbf{u} \in C_{k-2}$, $x_{\mathbf{u}}$ is ■ when $x_{\mathbf{u}+(1,0)}$, $x_{\mathbf{u}+(1,1)}$, $x_{\mathbf{u}+(1,-1)}$ and $x_{\mathbf{u}+(2,0)}$ are equal to □, or when $x_{\mathbf{u}-(1,0)}$, $x_{\mathbf{u}-(1,1)}$, $x_{\mathbf{u}-(1,-1)}$ and $x_{\mathbf{u}-(2,0)}$ are equal to □. Else, it is □.

3. **Eliminating non-domination errors in the central column.** Choose any position $\boldsymbol{u_0} \in C_{k-2}$ and check if this position has a symbol ■ in its neighbourhood. If not, then set the symbol ■ on this position. Repeat this from $\boldsymbol{u_0} + (0,1)$ upwards, and in parallel from $\boldsymbol{u_0} - (0,1)$ downwards.

See an illustration of this algorithm on Figure 3.8.

- **The obtained configuration is in X^{M}.**

We have to check that the configuration x we constructed satisfies the local rules of the minimal-domination subshift. We divide this part of the proof according to whether or not the cells belong to p or q, or lie outside. For concision, this division is approximate: columns C_{-1} and C_{k+2} are checked in the second and third point instead of the first one.

1. **The local rules are satisfied in the 'interior' of the half-planes.**
   By hypothesis, the patterns p and q are globally admissible in X^{M}. As a consequence, for all $\mathbf{u}$ in $]\!]-\infty,-2]\!] \times \mathbb{Z}$ or $[\![k+3,+\infty[\![\times\mathbb{Z}$, the pattern $x_{|\mathbf{u}+[\![-2,2]\!]^2}$ is not forbidden. It remains to check that no forbidden patterns are created through the execution of the algorithm described in the first point of the proof, for columns C_{-1} to C_{k+2}.
2. **Every position in $C_{-1}, \cdots, C_{k+2}$ and not in S is dominated.**
   In the columns C_{-1} and C_{k+2}, this comes from the fact that the patterns p and q are globally admissible. For j between 0 and $k-3$, and $\mathbf{u} \in C_j$, if $\mathbf{u}$ is not dominated by a position in C_j or C_{j-1} the position $\mathbf{u}+(1,0)$ contains the symbol ■ (by the first and second steps of the algorithm), and thus $\mathbf{u}$ is dominated. A symmetrical reasoning works for the positions in the columns C_{k+1}, C_k, C_{k-1}. For a position in the central column C_{k-2}, this is guaranteed by Step 3.
3. **Every dominant position in $C_{-1}, \cdots, C_{k+2}$ is isolated or has a private neighbour not in S.**
   Let us consider a non-isolated dominant position $\mathbf{u}$.
   (a) If it lies in C_{-1} (resp. C_{k+2}), $\mathbf{u}$ has a private neighbour in a configuration $x \in X$ that extends p (resp. q). If this private neighbour is in fact in p (resp. q), in column C_{-2} or C_{-1} (resp. C_{k+2} or C_{k+3}), then it stays a private neighbour of $\mathbf{u}$ in x since the intermediate columns cannot make it not private. If it is $\mathbf{u}+(1,0)$ (resp. $\mathbf{u}-(1,0)$), then it stays a private neighbour in x: since this position is dominated by $\mathbf{u}$ according to the first step (resp. second step) of the algorithm, it is not dominated in x by a position in C_0 (resp. C_{k+1}). The same reasoning is applied to positions in columns C_0 and C_{k+1}.
   (b) In the other columns C_j for $j < k-2$, the first step guarantees, for any position $\mathbf{u}$ in C_j that is dominant, that the position $\mathbf{u}-(1,0)$ is a private neighbour. A symmetric reasoning applies to column C_k and C_{k-1} : $\mathbf{u}+(1,0)$ is a private neighbour in that case.

(c) If $\mathbf{u}$ is in the column C_{k-2}, it means that it was introduced in either the second or the third step, meaning that it has a private neighbour in column C_{k-3} or C_{k-1} (if introduced in the second step), or in C_{k-2} (if introduced in the third step).

- **The subshift X^{M} is not 4-block-gluing.**

  We consider the two half-plane patterns p and q on respective supports $\mathbb{Z}_- \times \mathbb{Z}$ and $\mathbb{Z}_+ \times \mathbb{Z}$ such that for all $j \leq 0$, if $-j \equiv 0, 1 \mod 4$, then for all $\mathbf{u} \in C_j$, $p_{\mathbf{u}}$ is ■, else for all $\mathbf{u} \in C_j$, it is □, and q is obtained from p by vertical symmetry. It is easy to see that these patterns are globally admissible. We leave 4 columns between p and q (see Figure 3.9). To ensure that the dominant positions in columns 0 and 5, which are not isolated, have private neighbours, every cell of the middle four columns needs to be □, as in Figure 3.9. This filling implies that the cells in column 2 and 3, which are not dominant, are also not dominated. This shows that the subshift is not 4-block-gluing.

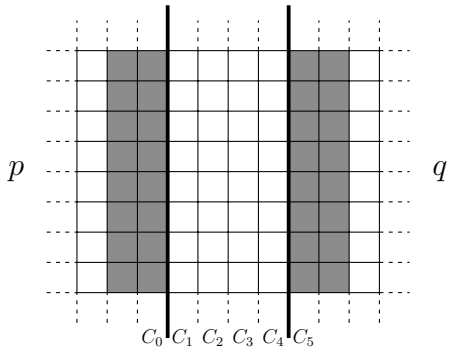


Figure 3.9: Illustration of the fact that X^{M} is not 4-block-gluing: when attempting to glue p and q, ensuring the existence of private neighbours according to the rules of X^{M} forces the presence of undominated positions in columns C_2 and C_3. It is also a counter-example for X^{MT} being 4-block-gluing.

As a consequence, $c(X^{\mathrm{M}}) > 4$. Since it is 5-block-gluing, $c(X^{\mathrm{M}}) = 5$.

□

Theorem 3.17. *The minimal total-domination subshift is block gluing and $c(X^{\mathrm{MT}}) = 5$.*

Idea of the proof: *We follow the same scheme as in the proof of Theorem 3.16, except that we have to take into account the variations in the definition of the subshift X^{MT}. For the sake of readability, we reproduce the structure of the proof. We now give the proof.*

Proof.

- **Filling the intermediate columns between two half-plane patterns.**

  We provide here an algorithm to fill these columns between two patterns p and q respectively on $\mathbb{Z}_- \times \mathbb{Z}$ and $\mathbb{Z}_+ \times \mathbb{Z}$ into a configuration $x \in X^{\mathrm{MT}}$. As in the proof of Theorem 3.16, intuitively, it puts grey cells only when it is really necessary to dominate the neighbour in the previous column:

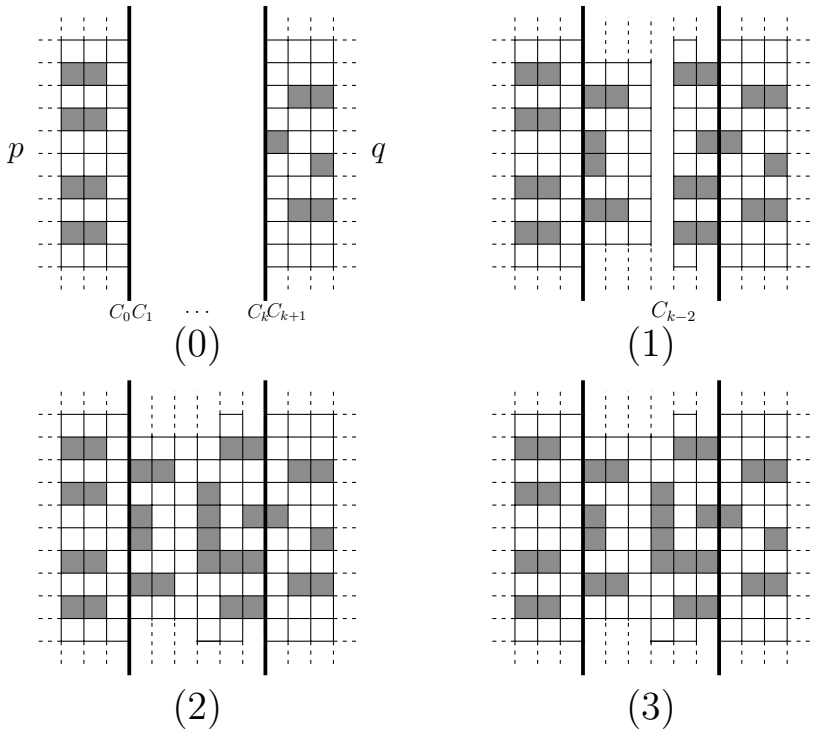


Figure 3.10: Illustration of the algorithm filling the intermediate columns between two half-plane patterns p and q for the minimal-total-domination subshift. In the last step, the position $\boldsymbol{u_0}$ is the bottommost represented position of the central column, and the central column is coloured with a possible colouring. We chose $k = 6$ for the illustration, still the proof works from $k = 5$ upwards.

1. **Filling the intermediate columns, from C_1 to C_{k-3}, then C_k, C_{k-1}.** Successively, for all j from 1 to $k-3$, we determine the column C_j according to the following rule: for all $\mathbf{u} \in C_j$, $x_{\mathbf{u}}$ is ■ when $x_{\mathbf{u}-(1,1)}$, $x_{\mathbf{u}-(1,-1)}$ and $x_{\mathbf{u}-(2,0)}$ are □ (the difference with the proof of Theorem 3.16 is that the symbol $x_{\mathbf{u}-(1,0)}$ is not imposed). Else, $x_{\mathbf{u}}$ is set to □. This rule is illustrated in Figure 3.11.

Figure 3.11: Illustration of the local rules for the completion algorithm for the intermediate columns.

   For $j = k$ and then $j = k-1$, we determine x on any position $x_{\mathbf{u}}$ for $\mathbf{u} \in C_j$ by applying a symmetrical rule: $x_{\mathbf{u}}$ is ■ when $x_{\mathbf{u}+(1,1)}$, $x_{\mathbf{u}+(1,-1)}$ and $x_{\mathbf{u}+(2,0)}$ are □. Else it is □.

2. **The central column ($j = k-2$).**
   We then determine x on the central column C_{k-2}. For all $\mathbf{u} \in C_{k-2}$, $x_{\mathbf{u}}$ is ■ when $x_{\mathbf{u}+(1,1)}$, $x_{\mathbf{u}+(1,-1)}$ and $x_{\mathbf{u}+(2,0)}$ are equal to □, or when $x_{\mathbf{u}-(1,1)}$, $x_{\mathbf{u}-(1,-1)}$ and $x_{\mathbf{u}-(2,0)}$ are equal to □. Else, it is □.
3. **Eliminating total-domination errors in the central column.**
   Choose any position $\boldsymbol{u_0} \in C_{k-2}$. From this position upwards, check for every position $\mathbf{u}$ if it is dominated. If this is not the case, then change the symbol on $\mathbf{u}+(0,1)$ into ■. As soon as $\boldsymbol{u_0}$ has been processed, do the same symmetrically (change the symbol in $\mathbf{u}-(0,1)$ when $\mathbf{u}$ is not dominated) in parallel downwards, beginning from $\boldsymbol{u_0}-(0,1)$.

See an illustration of this algorithm on Figure 3.10.

- **The obtained configuration is in X^{MT}.**

  We have to check that the local rules of the minimal-total-domination subshift are satisfied over the whole constructed configuration x.

  1. **The local rules are satisfied inside the 'interior' of the half-planes.**
     Same as the corresponding point in the proof of Theorem 3.16.
  2. **Every position in $C_{-1}, \cdots, C_{k+2}$ is dominated.**
     Same as the corresponding point in the proof of Theorem 3.16 for the positions outside the central column C_{k-2}. In this column, let us assume that a position $\mathbf{u}$ above $\boldsymbol{u_0}$ (without loss of generality) is not dominated. Then the last step of the algorithm, when examining this position, would have changed the symbol on position $\mathbf{u}+(0,1)$, which is a contradiction. In particular, no dominant positions are isolated.
  3. **Every dominant position in $C_{-1}, \cdots, C_{k+2}$ has a private neighbour.**

  (a+b) Outside the central column, the proof is similar to the corresponding points in the proof of Theorem 3.16.

  (c) In the column C_{k-2}, the dominant positions added in Step 2 necessarily have a private neighbour in column C_{k-3} or C_{k-1}. Let us take a dominant position $\mathbf{u}$, assumed without loss of generality to be above $\boldsymbol{u_0}$, which was added in the last step. This implies that $\mathbf{u}-(0,1)$ was not dominated when the algorithm checked this position. As a consequence, it is a private neighbour for $\mathbf{u}$.

- **The subshift X^{MT} is not 4-block-gluing.**

  Let us consider the patterns p and q defined in the corresponding point in the proof of Theorem 3.16. It is easy to see that these two patterns are also globally admissible in the subshift X^{MT}. Using the proof for X^{M}, we only have to check that in the constructed configuration, no dominant positions are isolated, which is straightforward.

  Using the same arguments as the ones for the minimal domination case, it is easy to see that any configuration in $\mathcal{A}^{\mathbb{Z}^2}$ where p and q are glued at distance 4 contains some forbidden patterns.

As a consequence $c(X^{\mathrm{MT}}) > 4$. Since it is 5-block-gluing, $c(X^{\mathrm{MT}}) = 5$.

□

As a direct consequence of Theorem 3.13:

Theorem 3.18. *The numbers ν_{D}, ν_{T}, ν_{M} and ν_{MT} are computable with rate $n \mapsto 2^{O(n^2)}$.*

3.5 Bounding the growth rates with computer resources

Although the algorithm presented in Section 3.4.2 provides a theoretical way to compute the growth rates of various dominating sets of the finite square grids, it is not efficient enough for practical use on a computer. In this section, we use other tools which make it possible to obtain bounds for the growth rates. These bounds are obtained using computer resources, by running a C++ program made for the occasion. The program is a modification of the program used in Chapter 2. Its modularity enabled us to only (re)write partly the local characterisation rules of domination problems and use the same main code. The main difference is that we work in the $(+, \times)$- algebra instead of the previously used $(\min, +)$-algebra.

We explain here the method differently, relating it to unidimensional SFTs. We do not dive much into the details of the specific problems, but give the abstract framework. As we have just mentioned, the technique relies on, for a fixed m, assimilating the dominating sets of $G_{n,m}$ to patterns of a unidimensional subshift of finite type, whose entropy is known to be computable through linear algebra computing. Note that, in its use in Chapter 2 there are no such things as lower or upper bounds we investigate now: we only enumerated sets which are precisely 2-dominating or Roman dominating. Since we were interested in finding the minimum size of such a set, we had the right to apply some optimisations to avoid having to enumerate all dominating sets, in particular the ones we knew were not of minimum size. Since we now want to count *all* the different dominating sets, these optimisations do not apply here: we must enumerate all possible states.

3.5.1 Nearest-neighbour unidimensional subshifts of finite type

In this section $\mathcal{A} = (a_1, ..., a_k)$ is a finite set, and X is a unidimensional subshift of finite type on alphabet $\mathcal{A}$. The elements of $\mathcal{A}$ are the cell values we defined in Chapter 2. Let us denote by $(e_1, ..., e_k)$ the canonical basis of $\mathbb{R}^k$.

Definition 3.26. The subshift X is said to have the **nearest-neighbour** property when it is defined by forbidding a set of patterns on support $\{0, 1\}$.

Definition 3.27. The **adjacency matrix** of a nearest-neighbour SFT X is the matrix $M \in \mathcal{M}_k(\mathbb{R})$ such that $M[e_i][e_j] = 1$ if the pattern $a_i a_j$ is not forbidden, or 0 otherwise.

The following is well known (see [33]):

Proposition 3.19. *Let $||.||$ be any matricial norm. The entropy of X is equal to the spectral radius of M:*

$$h(X) = \log_2 \lim_n ||M^n||^{1/n}.$$

Our matrix has non-negative coeffecients and is primitive for the same reason as in Proposition 2.2. Thanks to these properties, the Perron-Frobenius theorem states that the matrix has a largest eigenvalue in $\mathbb{R}^+$.

3.5.2 Unidimensional versions of the domination subshifts

We define here the unidimensional versions of the domination subshifts introduced in Section 3.2.3. We use them to describe and prove the method we use to obtain the bounds on the growth rates. We recall that n refers to the number of lines and m to the number of columns. The first sequence of SFTs $(X^{D,n})$ is used to obtain the lower bound, whereas we use the second one $(X_*^{\mathrm{D},n})$ to obtain the upper bound.

Notation 3.28. Let us fix some integer $n \geq 1$. We denote by $X^{\mathrm{D},n}$ the undimensional subshift on alphabet $\mathcal{A}_0^n$ such that a configuration x is in $X^{\mathrm{D},n}$ if and only if the set of positions $(j,i) \in \mathbb{Z} \times [\![1,n]\!]$ such that the symbol $x_{(j,i)}$ is grey forms a dominating set of the grid $\mathbb{Z} \times [\![1,n]\!]$.

With arguments similar to the ones of the proofs of Lemma 3.9 and Lemma 3.10, we obtain the following asymptotic formula when n is fixed and m grows to infinity. We recall Notation 3.2: $D_{n,m}$ denotes the number of dominating sets of $G_{n,m}$.

$$D_{n,m} = 2^{h(X^{\mathrm{D},n})\cdot m + o(m)}.$$

Notation 3.29. For all $n \geq 3$, we also denote by $X_*^{\mathrm{D},n}$ the undimensional subshift on alphabet $\mathcal{A}_0^n$ such that a configuration x is in $X_*^{\mathrm{D},n}$ if and only if the set of positions $(j,i) \in \mathbb{Z} \times [\![2,n-1]\!]$ such that the symbol $x_{(j,i)}$ is grey forms a dominating set of the grid $\mathbb{Z} \times [\![2,n-1]\!]$.

3.5.3 Recoding into nearest-neighbour subshifts

Let us set $\mathcal{A}_1 = \{\square, \blacksquare, \blacksquare\}$, and let us consider the map $\varphi : (\mathcal{A}_0^n)^{\mathbb{Z}} \to (\mathcal{A}_1^n)^{\mathbb{Z}}$ which acts on configurations of $(\mathcal{A}_0^n)^{\mathbb{Z}}$ by changing the i^{th} symbol of any position $j \in \mathbb{Z}$ into $\blacksquare$ whenever it is not dominant and dominated by an element of $C_{j-1} \bigcap (\mathbb{Z} \times [\![1,n]\!])$ or $C_j \bigcap (\mathbb{Z} \times [\![1,n]\!])$. Informally, from lightest to darkest, the symbols stand for an undominated cell (which is not dominant), a dominated cell which is not dominant and a dominant cell. This is illustrated in Figure 3.12. The nearest-neighbour property makes it possible to count the dominating sets without enumerating them fully: it is enough to store the information about a small number of the latest columns, proceeding from left to right in the grid. We encode this information from the possibly several columns we need to recall into a single column, for practical reasons. This is why it coincides with the definition of the nearest-neighbour property.

φ is a conjugation and the entropy is stable by conjugation by Lemma 3.5, therefore:

$$h(X^{\mathrm{D},n}) = h(\varphi(X^{\mathrm{D},n})).$$

Moreover, the subshift $X^{\mathrm{D},n}$ has the nearest-neighbour property.

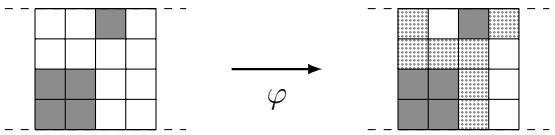


Figure 3.12: Illustration of the map recoding $X^{\mathrm{D},n}$ into a nearest-neighbour SFT.

3.5.4 Numerical approximations

We give here the numerical bounds we can compute, and prove them. The proof is only given for the simple domination, the other ones use the same method, hence are very similar.

Theorem 3.20 (Domination)**.** *The following inequalities hold:*

$$1.950022198 \leq \nu_{\mathrm{D}} \leq 1.959201684.$$

Proof.

- **Lower bound:**

  1. For all integers n_1, n_2 and m, $D_{n_1+n_2,m} \geq D_{n_1,m} \cdot D_{n_2,m}$.
     Indeed, let us consider two sets dominating respectively $G_{n_1,m}$ and $G_{n_2,m}$. By gluing the first one on the top of the second one, we obtain a dominating set of $G_{n_1+n_2,m}$. This is true because any position in this grid is either in the copy of the grid $G_{n_1,m}$ and thus dominated by an element in this grid, or in the copy of $G_{n_2,m}$. Since this construction is invertible our remaping is indeed a conjugation. We obtain the announced inequality.
  2. As a consequence, for all $k \geq 0$,

     $$D_{18k,m} \geq D_{18,m}^k = 2^{h(X^{\mathrm{D},18})\cdot km + k\cdot o(m)},$$

     where the function $o(m)$ is related to the fact that we used 18 lines[5]. This implies that

     $$\lim_{n,m} \frac{\log_2(D_{n,m})}{nm} = \lim_{n,k} \frac{\log_2(D_{18k,m})}{18km} \geq h(X^{\mathrm{D},18}).$$

  3. This number is equal to $h(\varphi(X^{D,18}))$, which is computed using Section 3.5.1 and Section 3.5.3. The lower bound follows: for any value of n we may compute the transfer matrix T_n counting $D_{n,m}$. The n-th root of the largest eigenvalue of T_n is a lower bound for ν_D. We could compute T_n for at most 18 lines given the computing resources at our disposal.

[5]We could not do the computations with more lines out of lack of RAM.

- **Upper bound:**

  1. For all n, m, let us denote by $D^*_{n,m}$ the number of sets of vertices of $G_{n,m}$ which dominate the middle $n-2$ lines (i.e. cells of the first and last lines might not be dominated). We have a direct inequality

  $$D_{n,m} \leq D^*_{n,m}.$$

  2. For a reason similar as the one in the first point of the proof of the lower bound, for all n_1, n_2, $D^*_{n_1+n_2,m} \leq D^*_{n_1,m} \cdot D^*_{n_2,m}$.
  3. For all $k \geq 0$ and $m \geq 0$,

  $$D^*_{18k,m} \leq (D^*_{18,m})^k = 2^{h(X_*^{\mathrm{D},18})\cdot km + k\cdot o(m)}.$$

  As a consequence

  $$\nu_{\mathrm{D}} \leq h(X_*^{\mathrm{D},18}).$$

  With the same method as for the lower bound, we obtain the upper bound.

□

Remark 3.4. With further numerical manipulations, we notice that the lower bound and the upper bound seem to get closer to each other rather slowly. To speed up the convergence, we had the idea of using the sequences of ratios $2^{h(X^{\mathrm{D},n+1})}/2^{h(X^{\mathrm{D},n})}$ and $2^{h(X_*^{\mathrm{D},n+1})}/2^{h(X_*^{\mathrm{D},n})}$. This seems to offer a much better convergence speed. Indeed, for both sequences, from $h = 11$ on, the ratio seem to be stabilised around 1.954751195. This ratio is not a bound on ν_{D} but it is guaranteed to converge towards this quantity.

Conjecture 3 (Domination)**.** $\nu_{\mathrm{D}} \approx 1.954751$

Using the same method, we provide bounds for the other problems. For the total domination, we can make the computations up to $n = 17$. However, for the other problems (the ones with the minimality constraint) the number of patterns we enumerate grows (exponentially) at a much faster rate than for the non-minimal problems, thus the bounds are less good. We cannot go further than about $n = 10$ for these problems. This comes from the fact that we need to encode information about more columns than from their non-minimal counterparts.

Theorem 3.21 (Total domination)**.** $1.904220376 \leq \nu_{\mathrm{T}} \leq 1.923434191$. $\nu_{\mathrm{T}} \approx 1.9153$

Remark 3.5. As in Remark 3.4, we can observe that the ratios $2^{h(X^{\mathrm{T},n+1})}/2^{h(X^{\mathrm{T},n})}$ and $2^{h(X_*^{\mathrm{T},n+1})}/2^{h(X_*^{\mathrm{T},n})}$ offer a much better convergence speed. Indeed, from $n = 10$ they seem to stabilise, both around 1.915316.

Conjecture 4 (Total domination)**.** $\nu_{\mathrm{T}} \approx 1.9153$

Theorem 3.22 (Minimal domination)**.** $1.315870482 \leq \nu_{\mathrm{M}} \leq 1.550332154$.

Theorem 3.23 (Minimal total domination)**.** $1.275805204 \leq \nu_{\mathrm{MT}} \leq 1.524476040$.

3.6 A $(2k+3)$-block-gluing family: the minimal meta-k-domination

When we spoke about the block-gluing property, we only defined the *constant* version of it. This property may be declined in finer-grained versions than choosing between being block gluing or not being block gluing.[6]

Definition 3.30. Let X be a subshift and $f : \mathbb{N} \to \mathbb{N}$ be a function. We say that X is f-block-gluing when any two globally-admissible patterns q and q of size $n \times n$ may be glued when they are separated by at least $f(n)$ columns.

This definition encompasses more SFTs because now the gap depends on the size of the pattern. This leads to interesting questions, such as:

Question 1. *What functions can f be? For instance, can f be a non-constant sub-linear function, or a super-linear function?*

We will answer partly these questions by recalling some known results in Section 3.7.2.

We propose here an extension of the domination problems. The standard domination consists of two sets $S_1 = D$ and $S_0 = V \setminus D$: the members of S_0 need to be dominated by a vertex in S_1 while the members of S_1 do not need to be dominated. We introduce a new family of problems where we colour the vertices with $k+1$ colours, which denotes a hierarchy of who may dominate whom. We will then show that each SFT associated to a problem of this family is block gluing, but the gluing constant linearly depends on the parameter k. We will also give some hints as to why it is difficult to find problems which are f-block-gluing with f being something other than an affine function.

Definition 3.31. $S = (S_0, S_1, \cdots, S_k)$ is a **meta-k-dominating** tuple if it is a partition of V and every $v \in V_i$ such that $i < k$ has a neighbour in some V_j with $j > i$.

Notation 3.32. If S is a meta-k-dominating tuple and $x \in V$, we denote by $\mathbf{arg}_S(x)$ the integer $0 \leq i \leq k$ such that $x \in S_i$.

Definition 3.33. Similarly as in the other domination problems, we say that $u \in G_{n,m}$ is **dominated** by a neighbour v or that v **dominates** u when $\arg_S(u) < \arg_S(v)$. u is a **private neighbour** of v if, in addition to being dominated by v, u is not dominated by another neighbour.

Now that we have generalised the notion of domination and total domination, we attack the minimal and minimal total domination. Since we no longer have a natural inclusion order, we choose one which generalises the inclusion on sets, for tuples of sets.

Definition 3.34. Let S and S' be two meta-k-dominating tuples. We say that $S' \leq S$ if for every $x \in V$, $\arg_{S'}(x) \leq \arg_S(x)$. We say that $S' < S$ if $S' \leq S$ and if there exists at least one $x \in V$ such that $\arg_{S'}(x) < \arg_S(x)$.
We say that a meta-k-dominating tuple S is **minimal** if there is no meta-k-dominating tuple S' such that $S' < S$.

[6] 'To be or not to be, that is the question.' said a famous writer. Maybe he was thinking about block-gluing SFTs.

In this definition, $S' \leq S$ if we can reduce the labels of a set of vertices altogether simultaneously. We could define another order allowing the reduction of the label of a unique vertex at each step. We show that the two definitions are equivalent.

Definition 3.35. If S and S' are two meta-k-dominating tuples, we write $S' <_1 S$ if there exists a unique $x \in V$ such that $\arg_{S'}(x) < \arg_S(x)$ and $\arg_{S'}(u) = \arg_S(u)$ for every $u \neq x$.

Proposition 3.24. *Let S be a meta-k-dominating tuple. S is minimal for the relation $<$ if and only if it is minimal for the relation $<_1$.*

Proof. It suffices to show that if a tuple S is reducible for one order, it is also reducible for the other, and vice versa.

It is clear that if S is reducible for $<_1$, then it is reducible for $<$.

Now, let S be such that there exists some $S' < S$. We show that S is reducible for $<_1$. Let $D = \{x \in V \mid \arg_{S'}(x) < \arg_S(x)\}$ and $x_0 \in D$ with minimal $\arg_{S'}$ value among the elements of D. Let S'' be such that $\arg_{S''}(x_0) = \arg_{S'}(x_0)$ and $\arg_{S''}(x) = \arg_S(x)$ for every $x \neq x_0$. Let us show that S'' is a meta-k-dominating tuple.

First, any vertex in $V \setminus N[x_0]$ (see Definition 1.3 for the definition of the closed neighbourhood $N[v]$) is dominated because itself and its neighbours have the same label as in S. x_0 is also dominated because its label decreased while the others stayed constant. Finally, let $v \in N(x_0)$. If v belongs to S_k, then it still does not need to be dominated. Else, if v is dominated by some vertex other than x_0 in S', it is also the case in S''. Else, v is only dominated by x_0 in S'. The definition of x_0 implies that $\arg_S(v) = \arg_{S'}(v)$. Since v is dominated in S', it implies that $\arg_{S''}(x_0) = \arg_{S'}(x_0) > \arg_{S'}(v) = \arg_S(v) = \arg_{S''}(v)$, hence v is dominated in S''.

Hence the minimal elements for $<$ are the same as the minimal elements for $<_1$. □

Definition 3.36. If $u \in S_m$ with $m > 0$, we say that any private neighbour of u belonging to S_{m-1} is a **good** private neighbour.

Proposition 3.25. *Let $(S_0, \cdots, S_k)$ be a meta-k-dominating tuple. Then it is minimal meta-k-dominating if and only if every $x \in S_i$ with $i < k$ has a good private neighbour, and any $x \in S_k$ with a neighbour in S_k has a good private neighbour.*

Proof.

If a meta-k-dominating tuple has the private-neighbour property, then it is minimal.

Let S be a meta-k-dominating tuple having the private neighbour property. By Proposition 3.24 it is sufficient to show that there is no S' such that $S' <_1 S$. We proceed by contradiction and assume that we have some $S' <_1 S$. Let x_0 be the vertex such that $\arg_{S'}(x_0) < \arg_S(x_0)$. There are two cases, the first one being that x_0 has label k in S. Since x is dominated in S', it has a neighbour of label k in S' which has the same label in S: x_0 is not isolated in S. Since by hypothesis it has, in S, a private neighbour y of label $k-1$, this implies that y is not dominated in S'. Thus S' is not meta-k-dominating, a contradiction. The other case is when $\arg_S(x) < k$. Let y be a good private neighbour of x_0 in S: $\arg_S(y) = \arg_S(x_0) - 1$ and no neighbours of y dominates it in S. Since $\arg'_S(x_0) < \arg_S(x_0)$ and the labels of the others vertices are the same, y is not dominated in S', which is also a contradiction. This proves that S is minimal.

If a meta-k-dominating tuple is minimal then it has the private-neighbour property.

We show the contrapositive implication: let us consider a meta-k-dominating tuple S which does not have the private-neighbour property. We will prove that it is not minimal. Let x_0 be such a vertex with label $p > 0$ which does not have any good private neighbour, and if $p = k$ then has a neighbour in S_k. Let S' be such that $\arg_{S'}(x_0) = p-1$ and $\arg_{S'}(x) = \arg_S(x)$ for every $x \neq x_0$. In S', x_0 is dominated by the same vertex it is dominated by in S, or by its neighbour of label k if $p = k$. Any neighbour which is dominated by x_0 in S has either label at most $p-2$ or is also dominated by another vertex in S, hence it is dominated in S'. Therefore S' is a meta-k-dominating tuple and $S' < S$, which concludes the proof. □

Theorem 3.26. *The SFT associated to the minimal meta-k-domination problem is $(2k+3)$-block-gluing.*

We can notice that this result coincides with the one about the minimal domination, for which $k = 1$. We can also see easily that the meta-k-domination subshift is 1-block-gluing for all k (we put label k for any cell in the gap column).

Proof. Let X_k be the SFT associated to the meta-k-domination, and X_k^{M} its minimal version.

We give here a proof similar to the one in the proof of Theorem 3.16. The argument is more general and can be used to show that the minimal domination is 5-block-gluing. We give an algorithm to fill the middle columns, providing there are at least $2k+3$ of them, and then prove that the result is indeed minimal meta-k-dominating.

- **Filling the intermediate columns between two half-plane patterns.**

  Let p and q be two patterns respectively on, without loss of generality, $\mathbb{Z}_- \times \mathbb{Z}$ and $\mathbb{Z}_+ \times \mathbb{Z}$. Let us determine a configuration of $\mathcal{A}^{\mathbb{Z}^2}$ such that $x_{|\mathbb{Z}_- \times \mathbb{Z}} = p$ and $x_{|(k,0)+\mathbb{Z}_+ \times \mathbb{Z}} = q$. The intermediate columns $C_1, ..., C_n$ are split, like before, into the 'left part' of the middle columns (C_1 to C_{n-k-3}, the central column $_{n-k-2}$ and the 'right part' of the middle columns (C_{n-k-2} to C_n). We determine their values by the following algorithm. For concision, we introduce the following definition:

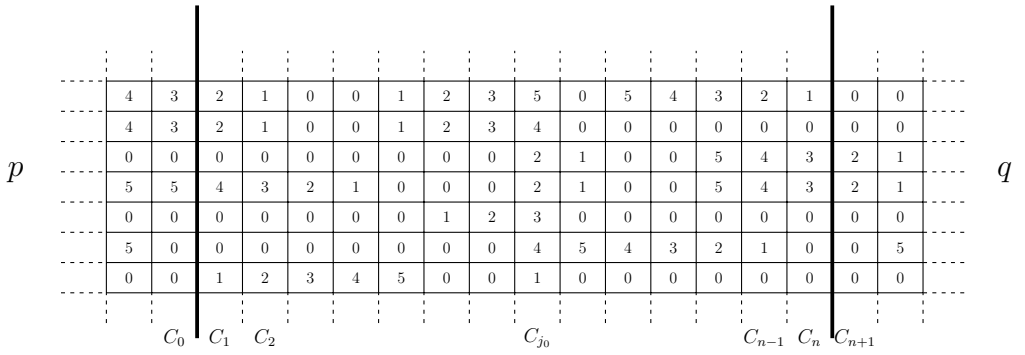


Figure 3.13: Illustration of the algorithm filling the middle columns for the minimal meta-k-domination for $k = 5$. We assume the line above the values and the one below them are filled with zeroes. Here $n = 14 = 2k + 4$, although a gap of 13 is enough.

1. **Filling the middle left columns, from C_1 to C_{n-k-2}.**
   The following algorithm, to fill the values of the middle columns, is illustrated through an example in Figure 3.13. Successively, for all j from 1 to $n-k-2$, we determine the column C_j according to the following rule: for each $i \in \mathbb{Z}$,

$$C_j[i] = \begin{cases} C_{j-1}[i]+1 & \text{if } C_{j-1}[i] \text{ is not dominated;} \\ C_{j-1}[i]-1 & \text{if } C_{j-1}[i] \neq 0 \text{ and has no private neighbours in } S_{C_{j-1}[i]-1}; \\ 0 & \text{otherwise.} \end{cases}$$

   Note that since p is globally admissible, no cells in C_0 (or C_{n+1}) can need both a good private neighbour and some neighbour to be dominated. For the same reason, if some $C_0[i]$ needs a good private neighbour then $C_0[i-1]$ must, if it needs a good private neighbour, be equal to $C_0[i]$: otherwise its neighbour in $C_1[i-1]$ (resp. $C_1[i+1]$) would either dominate or be dominated by $C_1[i]$, hence one of the two would not play its role of good private neighbour. It must also be dominated, either by $C_0[i]$ if lesser, or by another cell in p if greater or equal (otherwise its neighbour in $C_1[i-1]$ would dominate $C_1[i]$. The same applies to $C_0[i+1]$, so that the algorithm is well defined for the first step. These properties propagate to each new column filled, so that the algorithm is completely well defined.
   Filling the middle right columns, from C_n to C_{n-k-1}.
   We apply the symmetric rule: instead of considering the value of $C_{j-1}[i]$ to determine the one of $C_j[i]$, we use the one of $C_{j+1}[i]$.
2. **The central column C_{n-k-1}.**
   We first let $j_0 = n-k-1$ be the index of the central column. We first set the value $C_{j_0}[0]$, then the values of $C_{j_0}[-1]$ and $C_{j_0}[1]$. After this, we give the way to fill the values of the column from index 2 to infinity, and in parallel from index -2 to minus infinity. We define $\max(\emptyset) = -1$.
   First let R_0 be the set of values among $C_{j_0-1}[0]$ and $C_{j_0+1}[0]$ which are different from k and are not dominated so far. Then $C_{j_0}[0] = \max(R_0)+1$.
   Now let[7] R'_1 (resp. R'_{-1}) be the set of values among $C_{j_0-1}[1]$ and $C_{j_0+1}[1]$ (resp. $C_{j_0-1}[-1]$ and $C_{j_0+1}[-1]$) which are different from k and are not dominated so far. Up to symmetry, we may assume that $\max(R'_1) \geq \max(R'_{-1})$. We then set $R"_1 = R'_1$ if $C_{j_0}[0]$ is already dominated, or $R"_1 = R'_1 \cup \{C_{j_0}[0]\}$ otherwise. We define $C_{j_0}[1] = \max(R"_1) + 1$ and $C_{j_0}[-1] = \max(R'_{-1}) + 1$. We do the same thing in symmetric if $\max(R'_{-1}) > \max(R'_1)$.
   Now for $i = 2$ to infinity, we define R_i to be the set of values among $C_{j_0-1}[i]$, $C_{j_0+1}[i]$ and $C_{j_0}[i-1]$ which are different from k and not dominated so far and we set $C_{j_0}[i] = \max(R_i)+1$. In parallel downwards for i from -2 to minus infinity, we define similarly R_i considering $C_{j_0}[i+1]$ instead of $C_{j_0}[i-1]$ and we set $C_{j_0}[i] = \max(R_i)+1$ as well.

- **The obtained configuration is in X_k^{M}.**

  We have to check that the configuration x we constructed satisfies the local rules of the minimal-domination subshift. We divide this part of the proof according to whether or not the cells belong to p or q, or lie outside.

[7] We named it R'_1 instead of R_1 here because it is not exactly how the others R_is are defined.

1. **The local rules are satisfied inside the half planes.**

   By hypothesis, the patterns p and q are globally admissible in X_k^{M}. As a consequence, all symbols in p or q except the columns C_0 and C_{n+1} are dominated, and all symbols in p or q except the columns C_{-1}, C_0, C_{n+1} and C_{n+2} have a private neighbour with the right value. We prove that this is also the case for C_{-1} and C_0. The cases for C_{n+1} and C_{n+2} are symmetric. Let $i \in \mathbb{Z}$. If $C_0[i] \neq k$ is not dominated by an element inside p then the algorithm sets $C_1[i] = C_0[i] + 1$, which dominates $C_0[i]$. Remember that $C_0[i]$ cannot both need being dominated and a good private neighbour, for p would not be globally admissible if this were the case. Now we know that every cell in C_0 is dominated, and has a potential good private neighbour in C_1. It remains to show that this neighbour is not dominated by C_2. Let $i \in \mathbb{Z}$ such that $C_0[i]$ does not have a good private neighbour in p. We showed that both $C_1[i-1]$ and $C_1[i+1]$ are less than or equal to $C_1[i]$. $C_1[i]$ is dominated by $C_0[i]$, hence $C_2[i] < C_1[i]$ and $C_1[i]$ is indeed a good private neighbour for $C_0[i]$. If $C_{-1}[i]$ needs a good private neighbour, either it has it in C_{-2} or C_{-1} and we are done, or it needs to be $C_0[i]$. But then $C_0[i]$ is dominated by $C_{-1}[i]$ so that $C_1[i] < C_0[i]$ and $C_0[i]$ stays a good private neighbour for $C_{-1}[i]$.

2. **Every position outside $\mathbf{supp}(p) \cup \mathbf{supp}(q)$ with label less than k is dominated.**

   We prove the case for columns C_1 to C_{n-k-1} (the central column), the proof is identical for the columns at the right. At the left of the central column, this is true by the definition of the algorithm: if $C_j[i]$ is not dominated, then $C_{j+1}[i]$ is equal to $C_j[i]+1$, or has at least this value if C_{j+1} is the central column.

   For the central column it is also true for the same reason by the definition of the algorithm for this column.

3. **Every dominant position outside $\mathbf{supp}(p) \cup \mathbf{supp}(q)$ with label greater than 0 has a good private neighbour.**

   We recall that $j_0 = n - k - 1$ is the index of the central column. Once again we only prove this for columns C_1 to C_{j_0}. We first state an intermediate result before proving this.

   Claim 3.1. *Let $0 \leq j < j_0$ and $i \in \mathbb{Z}$. If $C_j[i]$ needs a good private neighbour in C_{j+1} then $C_j[i]$ is the unique good private neighbour of $C_{j-1}[i]$, i.e. its predecessor was in the same situation.*

   Proof. Let $0 \leq j < j_0$ and $i \in \mathbb{Z}$ be (if any) such that $C_j[i]$ needs a good private neighbour in C_{j+1}. This iplies, by the definition of the algorithm, that $C_{j-1}[i]$ did not need to be dominated by $C_j[i]$ (otherwise $C_{j-1}[i]$ would have been a good private neighbour). Since $C_j[i] \neq 0$ (a position with value 0 does not need a good private neighbour) then the definition of the algorithm implies that we are in the case $C_j[i] = C_{j-1}[i] - 1$, hence C_{j-1} needed a private neighbour. □

   We now go back to the proof of Theorem 3.26. Let $0 \leq j < j_0 - 1$ and $i \in \mathbb{Z}$. Let us show that $C_j[i]$ has a good private neighbour if it is different from 0.

From Claim 3.1 and an easy induction, we know that $C_0[i]$ also needed a good private neighbour in $C_1[i]$. We mentioned that each of $C_0[i-1]$ and $C_0[i+1]$ is dominated in p, and if any needed a good private neighbour in C_1 then this neighbour would have the same value as $C_0[i]$. By an easy induction, we can see that $C_j[i-1] = 0$ or $C_j[i-1] = C_j[i]$ and the same applies to $C_j[i+1]$. For the same reasons that $C_0[i]$ was guaranteed to have a good private neighbour in C_1, $C_j[i]$ has a good private neighbour in $C_{j+1}[i]$.

If now $j = j_0 - 1$, then $j = n - k - 1 - 1 \geq 2k + 3 - k - 2 \geq k + 1$. Let us show that C_{j_0-1} does not need a good private neighbour in C_{j_0}. Let us assume that this is false: it does need one. Then using the same argument as before, we know that each $C_u[i]$ needed a good private neighbour in $C_{u+1}[i]$ for $-1 \leq u < j$. This implies that $C_1[i] = C_0[i] - 1$, and so on, so that $C_j[i] \leq C_{k+1}[i] = C_0[i] - k - 1$. However, $C_0[i] \leq k$, which implies that this is not possible. Therefore, at most column C_{k-1} may have cells which need a good private neighbour.

It only remains to show that positions in C_{j_0}, if containing values greater than 0, have a good private neighbour. This is true thanks to the definition of the algorithm for the central column: every set S_i only contains values of undominated neighbours. This implies that any new defined value in the central column is one above its maximum undominated neighbour, which will not have any new neighbour to also dominate it. The exception is with the cells at lines 1 and -1. However, the way their values are defined in the central column ensures that at most one of them may have $C_{j_0}[0]$ as a good private neighbour, and the other one does not dominate it. This proves that both have a good private neighbour in column C_{j_0}, C_{j_0-1} or C_{j_0+1}.

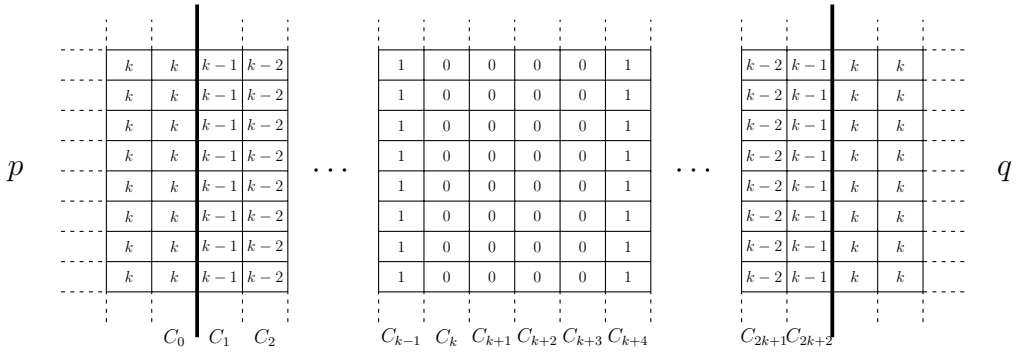


Figure 3.14: Illustration of the fact that X_k^M is not $2k+2$-block-gluing: when attempting to glue p and q, the values of the next $k+1$ columns are forced, and leave the cell of the middle columns C_{k+1} and C_{k+2} undominated.

- **X_k^{M} is not $2k$+2-block-gluing.**

  We take a globally-admissible half-plane pattern p of $\mathbb{Z}_- \times \mathbb{Z}$ such that its rightmost two columns are filled with value k. It is globally admissible because the preceding column may be filled with values $k-1$, the one before by $k-2$, and so on until a column of 0s. The column preceding it must also be filled with 0s and the one

at its left can be filled by ones, then by twos, and so on. We take q as the vertical symmetric of p.

By Proposition 3.25 since the cells of C_0 (see Figure 3.14) have no neighbours of value $k-1$ in p, column C_1 must be filled with value $k-1$. In fact for the same reason, for every $1 \leq i \leq k$, the column C_i must be filled with the value $k-i$. Now, to guarantee for each cell of column C_{k-1} that its neighbour in C_k is its private neighbour, column C_{k+1} must not dominate column C_k, hence it must be filled with zeroes. A symmetric reasoning forces the values of the cells of the other columns between p and q. Now columns C_{k+1} and C_{k+2} are not dominated. This proves that the minimal-meta-k-dominating subshift S_k^{M} is not $(2k+2)$-block-gluing. Hence it is block gluing with $c = 2k+3$.

□

We can notice, for the minimal meta-k-domination, that there is some linear decrease on the values of the forced cells. Indeed, in Figure 3.14 for instance, we see that when the label is k, we managed to need roughly k steps to get down to 0, so that a minimum gap cannot be less than around $2k$. We could easily have only needed a gap of k if we had slightly modified the rules, so that a private neighbour of a vertex with label i may have either label $i-1$ or $i-2$. We may even implement a roughly $2k/q$-block-gluing SFT if a vertex of label i needs a private neighbour of label between $i-q$ and $i-1$. This behaviour is similar to the one in Remark 3.3 when we mentioned the sagging of the set of forced cells. In the example showing that the minimal-meta-k-domination subshift is not $(2k+2)$-block-gluing, we took half-plane patterns to avoid some technicalities. Had we not done that, at each new column one cell at the top (and one at the bottom) would not have been forced, since they have upper (and lower) neighbours outside the band of the same height of the pattern. These extra cell give some liberty for the symbol in the top and bottom cells, so that with each new column we may lose two lines which were forced but which are independent now. This constitutes, as in Remark 3.3, a linear sagging. Hopefully, here it does not prevent the SFT from being block-gluing because the values decrease by themselves.

3.7 Conclusions

3.7.1 Counting dominating sets

Here we both proved the existence and the computability of an asymptotic growth rate for four variants of the domination problem in grid graphs. We gave some bounds and some values we think approximates rather accurately each of these growth rates. The bounds for the domination and total-domination problems are rather good: respectively 0.5% and 1% of the computed value. However, the ones for their minimal counterparts are looser: the gaps between the upper and the lower bounds are around 20% of the lower bound.

Theorem 3.22 improves the bound given by Fomin et al. [15], when the graph is a grid. They provide an algorithm enumerating the minimal dominating sets of a graph. By analysing its complexity, they show that there are at most 1.7159^n minimal dominating

sets for a graph on n vertices. We reduce this bound by approximately 10% in the case of grids.

As for the minimal domination and minimal total domination, the associated bounds could be improved by using a more powerful computer (mainly one with more than 1.5TB of memory), or by optimising the technique or finding one more efficient.

Also, the bounds we give are not numerically certified because of two reasons. First, the computations are done in floating-point arithmetic, hence some rounding errors may propagate. However, we only do additions and one division, so this should not occur. Also, we checked one value using arbitrary-precision numbers and it gave the exact same results. Second, we compute the largest eigenvalue of the matrix by using the power iteration method: start with a vector V, compute the iterates $(M^k V)$ and see how the norm of the vector evolves. We observe that this method converges rather quickly (around 20 iterations), but it does not certify any digit of the value as being the right one: we have no guarantee on the precision of the numerical values. However, when we used the arbitrary-precision numbers, we computed $M^{150}V$ and there seemed to be 85 digits which stabilised, so it really looks like it is converging here. One direction of work could be to find a way it to certify the digits of the computed eigenvalues and use it.

3.7.2 Around the block-gluing property

In the proofs of the block-gluing property, we mentioned some phenomenon: the sagging of the values in the middle cells between two pattern we try to glue, or the one about the number of lines which are still forced when the patterns are finite (instead of half planes). We discuss here the matter further, for there exist subshifts for which this sagging prevents the block-gluing property: the values or number of forced cells decrease too slowly, in a linear manner. This prevents two patterns to be glued with a gap less than something linear in the heights of the patterns, as we shall see.

Among others (and stronger) results, Gangloff and Sablik introduced the concept of linearly-block-gluing SFTs in [16] and showed that some SFTs indeed have this property. One example they show kinds of encodes an integer by having a pattern with a column of k black cells. By forbidding a list of three small patterns, they force the next column to be a column of $k-2$ black cells, the black cells being centred compared to the ones of the previous column. The next column must be a column of $(k-2)-2 = k-4$ black cells also centred, and so on. This means that if we begin with a column pattern of height k full of black cells we need a gap of size around $k/2$ to glue it to a column of white cells. Hence no constant block-gluing gaps are possible, but only a gap the size of which is linear in the height of the pattern. We can see this as if the height of the pattern encodes a number, which then decreases is a linear way.

This reminds us of the Question 1 to know if there exists subshifts, or even SFTs which are $f(n)$-block-gluing, but where f is not an affine nor a constant function. Let us now think about a potential SFT which would $\log(n)$-block-gluing for instance. We can think that each new column we build between two patterns should have halved the number of forced cells from the number in the previous column. For instance, with the previous example we would need, beginning with a column with k adjacent black cells,

that only $k/2$ cells are black in the next columns, then $k/4$, and so on. This could be done by several ways. We could, for instance in one dimension, encode an arbitrarily large alphabet with a finite alphabet but considering arbitrarily large patterns: if the alphabet is $\{0, 1\}$ a word 1111 could represent 15. For instance, we can define a subshift which would allow any word which forbids 01^k0^q1 if $q < \log_2 k$. This subshift is not an SFT and is not constant block gluing, but it is $\log(n)$-block-gluing.

However, for a SFT to be $\log(n)$-block-gluing appears to be a much more difficult problem: we have a finite alphabet and only a finite number of finite patterns which we forbid. For a subshift to have a sagging other than linear, it seems to need to be able to encode arbitrarily large values: if the sagging is logarithmic, it should work for patterns arbitrarily large so that it can effectively 'compute' the logarithm of arbitrarily large numbers, but only with local rules of fixed radius. This seems rather difficult to achieve in the world of SFTs, but Gangloff and Sablik also showed in [16] that this was possible by constructing a $\log(n)$-block-gluing SFT. They basically implement an $+1$ adder component (like in CPUs) with a system of carry. The number of digits only increases by one when we reach a new power of two, that is after 2^k iterations if the pattern is of size 2^k. This settles one case of the question we asked in Question 1, but all the spectrum of other subliner functions (except for the logarithm) or superlinear ones remains open. For instance, are there SFTs or subshifts which are $\sqrt{n}$-block-gluing?

Chapter 4

Tiling rectangles with polyominoes

Even though we can wonder how we let this possible and even happen, some parts of the Pacific ocean are currently covered by 80 000 tons of plastic, tiling around three times the surface of France. Some other people are untiling big and ancient forests right now: over the past year, for instance, the Amazon rainforest in Brasil suffered the loss of between 500 000 and one million football pitches. The rate of deforestation of this area has increased eight times as much it was before Bolsonaro was elected.

Fortunately here, while we try to tile, it is not with plastic but instead with objects called polyominoes, looking like the inoffensive Tetris pieces. We try to fill a rectangle with such pieces, a bit like in a jigsaw puzzle, except that all the pieces are copies of the same one.

Informally, a polyomino is a connected finite set of unit squares. This means that the polyomino is made of only 'one part': see Figure 4.1. We already talked about them in Chapter 2. We showed there how to encode a domination problem into a polyomino. A dominating set of the grid is then a *covering* of the rectangle with the right polyomino: placing copies of the polyomino so that each cell is covered by at least one polyomino. A minimum dominating set is a covering of the rectangle with as few polyominoes as possible. The number of polyominoes used in such a covering gives the domination number of the grid.

In this chapter, we are interested in a notion close to the one of coverings: the notion of *tiling*. Tiling a surface with a shape means covering the surface with copies of the shape such that no copy go over the surface, and no two copies overlap. We want to know, given one particular polyomino, if we can tile a rectangle with it. There are many questions in the topic of tilings with polyominoes. The natural question is to wonder which polyominoes can tile the plane. In the case when we allow only translations, that is when we choose one orientation for each polyomino and all its copies are obtained from it by translation only, the case is settled. Indeed, there exists an algorithm to decide if a given polyomino can tile the plane $\mathbb{Z}^2$ by translations. However, in the case when we allow translations, rotations and mirrors of the polyomino, we have little knowledge. We do not know whether or not this problem of tiling is decidable or whether there exists a polyomino which tiles only the plane in an aperiodic way. However, we know that the problem of tiling a plane with a set of polyominoes is not decidable. We will develop a bit more on these results in Section 4.1. Unless written otherwise, when we speak of

tiling, we mean a tiling by copies allowing translations, rotations and vertical mirrors.

A funny concept about the polyominoes is the one of **reptiles**.[1] It stands for auto-replicating tiles. This means that we can assemble several copies of a polyomino P in such a way that the obtained shape is P zoomed in. This implies that the polyomino can tile the infinite plane $\mathbb{Z}^2$. Indeed, just take your shape, assemble copies of it so that you obtain that shape again but bigger in size. This way, you tile a bigger and bigger connected area by extending step by step your partial tiling, and never changing what is already done. Hence you will tile the whole plane if you can wait indefinitely, with no need for the axiom of choice.

This chapter is focused on a stronger property a polyomino can have: to be *rectifiable*, i.e. to be able to tile (allowing rotations, translations and mirrors) some rectangle. This property is stronger than tiling the plane since when a polyomino tiles a rectangle then we can put together copies of this arrangement to tile the plane. We can also assemble the rectangles to obtain a big square, with which we can obtain a bigger shape of the polyomino, hence any rectifiable polyomino is also a reptile. If a polyomino is rectifiable, we want to find the smallest rectangle which can be tiled by this polyomino, or in fact the smallest number of copies we need to tile a rectangle. This minimum number of copies is called the **order** of a polyomino, if it exists. This problem looks rather simple, or at least can be expressed very easily, however there are big gaps in our knowledge of the subject. For instance, we do not know if there exists a polyomino the order of which is five, that is one which can tile a rectangle with five copies but cannot tile a rectangle with a smaller number of copies.

In 1989, in a paper studying polyominoes, their tileability and their rectifiability, Golomb [20] asked a question which is still unresolved. It was our goal for studying polyominoes.

Question ([20])**.** *Is there any polyomino with odd order greater than one?*

In the first chapter, we first give basic formal definitions and a bit of history on the subject of tilings using polyominoes. We will for instance give more details of the hierarchy Golomb made about the tiling properties polyominoes can have (tiling the plane, being rectifiable, being a reptile for instance). We will also say a bit about tilings with sets of polyominoes. In Section 4.2, we will give the methods which can be used to find the order of polyominoes. We give there simple versions of the algorithm, to be refined in Section 4.3. It begins with Section 4.2.1 where we explain how we enumerate the polyominoes, which is a first step before testing them for rectifiability. We then focus on how to test if a rectangle can be tiled by some polyomino. We present two methods: one looking like a DFS in Section 4.2.2 which is rather slow, and another one which looks like a breadth-first search (BFS) in Section 4.2.3 which is much faster. We even mention in Section 4.2.4 an approach using integer linear programming, which is slower than the two previous methods. In Section 4.3 we explain the methods we used to detect polyominoes which are not rectifiable, in order to rule them out. We also provide some optimisations mainly for the BFS algorithm. In Section 4.4, we give some statistics on the number of polyominoes which are rectifiable, and on their orders. We also give some ideas for further work in order to tackle this problem.

[1] Don't worry, they don't bite!

4.1 Definition and some history

Definition 4.1. A **polyomino** P is a finite and connected[2] union of unit cells of the $\mathbb{Z}^2$ lattice.

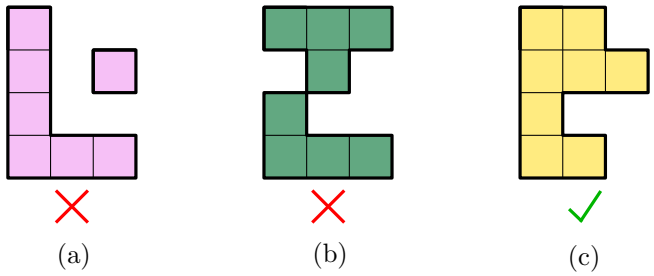


Figure 4.1: Illustration of the polyomino definition. (a) is not a polyomino because it is not connected. The same goes for (b): connectivity is done through edges and not corners. (c) is a valid polyomino.

Definition 4.2. An **isometry** of the plane is a geometric transformation which preserves the distance between any pair of points. If a and b were at distance d, their images must be at distance d as well.

Fact 4.1. *Including itself, a polyomino may have up to 8 isometric copies in total (see Figure 4.2). All isometric copies of P are polyominoes obtained from P by a succession of rotations by $\pi/2$ and vertical symmetries.*

Definition 4.3. Given a (finite or not) surface $X \subset \mathbb{Z}^2$ and a polyomino P we say that P **tiles** X when X can be decomposed into isometric copies of P.

In Definition 4.3 X may be finite of infinite. In this chapter, we will mostly consider finite rectangles to be tiled, but sometimes we speak of tiling the plane $\mathbb{Z}^2$.

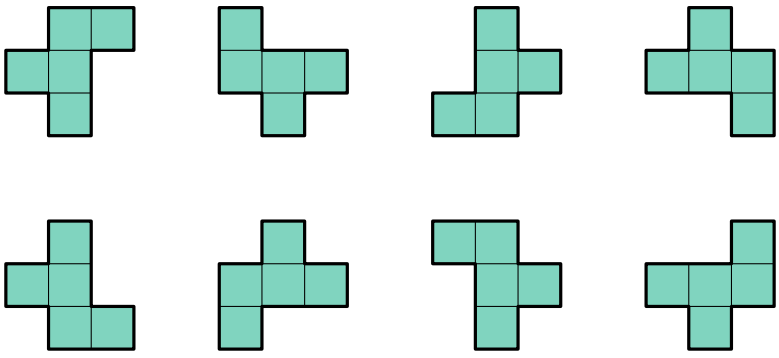


Figure 4.2: The eight copies of one heptomino. The second line is the vertical flip of the first one.

Definition 4.4. A polyomino P is **rectifiable** when there exists a finite rectangle which can be tiled with copies of P. The **order** of a polyomino is the minimum number of copies with which we can tile a rectangle, or $+\infty$ if it is not rectifiable.

[2]Two unit cells of $\mathbb{Z}^2$ are **connected** when they share an edge.

Golomb [18] established in 1966 a hierarchy of polyominoes according to their ability to tile some parts of the planes. For instance, we mentioned that reptiles can tile the plane, thus being a reptile is a stronger property than tiling the plane. Also, being rectifiable implies that a polyomino can tile an infinite strip, which implies it can tile a half-plane, which in turn implies that it can tile the whole plane. We mentioned in the introduction that a rectifiable polyomino can tile a big square by combining the rectangles, hence it is a reptile: a bigger version of the polyomino can be constructed with copies of this big squares. In fact, the rectifiability is the strongest class of the hierarchy Golomb defined because of this ability to tile a square: when you tile a square you can tile any shape decomposable into squares. The full set of categories and implications can be found in [18]. One big question remains open in this domain:

Question. *Is tiling the plane with one polyomino decidable?*

Golomb also studied the tiling of different surfaces when, instead doing it with just one polyomino, we are allowed to use several polyominoes from a given set. In 1970, he gave in [19] a hierarchy of the different tiling problems when we tile with a set of polyominoes. He also answered the above question in this context.

Theorem 4.1 (Golomb, [19])**.** *The problem of tiling the plane with a set of polyominoes is equivalent to Wang's domino problem and is therefore undecidable.*

This means that there are no Turing machines which, on every possible set of polyominoes given as input, would output, in finite time, 1 if we can tile $\mathbb{Z}^2$ with the polyominoes in this set, and 0 otherwise. Theorem 4.1 was further improved in 2008:

Theorem 4.2 (Ollinger, [37])**.** *The problem of tiling the plane with a set of five polyominoes is undecidable.*

Ollinger also proved this result by reducing the domino problem to the problem of tiling the plane with a set of five polyominoes. This narrows the gap between the undecidability of tiling the plane by a set of polyomino and the question about doing so with a single polyomino. Maybe tiling the plane with a set of four polyominoes would turn out to be decidable, hence implying the same for sets of size less than four, including tiling with one polyomino.

In 1991, Beauquier and Nivat showed in [2] an interesting characterisation of the polyominoes which can tile the plane by translation (forbidding copies obtained by rotation or mirror). This gives an algorithm to check if a polyomino can tile $\mathbb{Z}^2$ by translation: all we needed is to look at its frontier and use some combinatorics on words. However this does not help us here since we consider also rotations and symmetries of the polyominoes. Recently, Nitica investigated how to translate Golomb's hierarchies of polyominoes and sets of polyominoes when only translation is allowed. This resulted in different classes and some inclusion relations between these classes, as it can be seen in [35] and [36].

Shortly after, in 1992, Stewart and Wormstein [48] put a first stone towards the resolution of this question: they answered the question about a polyomino of odd order for the smallest odd integer greater than one. We still do not know the answer for any other odd number.

Theorem 4.3 ([48])**.** *No polyominoes of order 3 exist.*

This theorem can be restated in the following way: if a polyomino can tile a rectangle with three copies then this polyomino is necessarily a rectangle. However, there are no strong reasons for ruling out any odd order greater than three. Indeed, Golomb has shown that there are infinitely many polyominoes which can tile some rectangle with an odd number of copies; the problem is that it might not be the minimum number of copies needed, hence not their orders in that case. Let us call such a polyomino, which can tile some rectangles with an odd number of copies, an **odd polyomino**. In 1997, Reid [44] showed that the minimum number of odd copies needed to tile a rectangle with a polyomino can be arbitrarily large. He showed this by giving a way to construct, for each prime number p, a polyomino of odd-order $3(p+2)$: a polyomino which can tile a rectangle with $3(p+2)$ copies but cannot tile a rectangle with an odd number of copies less than that.

The funny thing is that while we have no clues about half of the possible orders, namely the odd numbers, we have knowledge for half of the even orders.

Theorem 4.4 ([20])**.** *Each positive multiple of 4 is the order of some polyomino.*

Golomb showed this theorem by providing, for each number $4s$, a polyomino of order precisely this number. In 1989, Dahlke [12] gave the order of a polyomino mentioned by Golomb: it was 92 (see Figure 4.3). He also answered in another paper, with the same program, that another polyomino the ordered of which Golomb had also asked, had order 76. To summarise, regarding which numbers are the order of some polyomino, we know the answer for a quarter of $\mathbb{N}$ plus a few isolated cases. So there is still much space for discoveries. We also do not know the answer for small numbers: are there polyominoes of order 5? Of order 6, 14 or 22?

4.2 Finding the order of a polyomino: the basic algorithms

We explain in this section how to look for rectifiable polyominoes. We first show how to enumerate them, or in fact, enumerate the ones which have no holes since any polyomino with a hole cannot tile the plane. After this we present two methods, given a rectifiable polyomino, to find its order. This section will be completed by Section 4.3, which provides some optimisations of our algorithms, as well as techniques to show that some polyominoes are not rectifiable. In all this section we assume that we have a polyomino P and try to find its order, by trying to tile rectangles of size $n \times m$ with isometric copies of P.

4.2.1 Enumerating all the polyominoes

Counting polyominoes is not a new topic at all. Several tables or sequences giving the number of a certain type of polyominoes according to their sizes can easily be found on some papers and on the Internet. The On-Line Encyclopedia of Integer Sequences (`oeis.org`) lists such tables: see for instance the sequences A000105, A001168, A000988. There are several sequences because there are several ways to list polyominoes, according for instance to whether or not we count isometric copies as different polyominoes. The

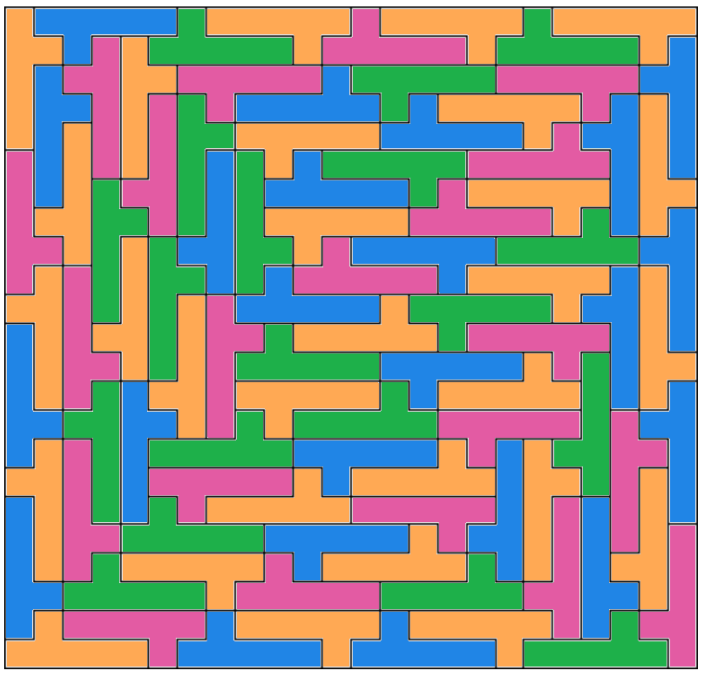


Figure 4.3: A tiling with 92 copies of a polyomino of order 92. It also illustrates the four-colour theorem: two polyominoes sharing an edge always have different colours.

sequences respectively focus on free, one-sided and fixed polyominoes. The first one considers that two isometric polyominoes are the same; the second one considers that two polyominoes are the same if one can be obtained from the other by combining rotations and translations; to the third one two polyominoes are the same only if they differ by translation.

In this chapter, we are interested in enumerating all the polyominoes. We consider, since we authorise isometric copies of a polyomino for our tilings, that two isometric copies of a polyomino are the same polyomino, and we enumerate just one of them. As mentioned in Fact 4.1, a polyomino can have up to 8 'different' forms: we may obtain them for instance by applying from zero to three rotations by $\pi/2$ and, on top of this, also applying zero or one vertical flip[3] (see Figure 4.2).

Definition 4.5. We say that X is a set of **free polyominoes** when no two elements are isometric.[4]

Definition 4.6 (see Figure 4.4)**.** Let P be a polyomino placed on a grid. Any finite connected region of cells not in P which is disconnected from the rest of the grid by P is called a **hole**.

Notation 4.7. Let us denote by $\boldsymbol{\mathcal{P}_n}$ the set of free polyominoes of size n with no holes.

[3]also called horizontal symmetry

[4]We keep this phrase because it is widespread, though we are not quite satisfied with it: the polyominoes are not free. Saying that a polyomino is free means nothing. It would be better to speak of a free set of polyominoes: two sets of polyominoes are equivalent if any polyomino in one has an isometric copy in the second, and vice versa.

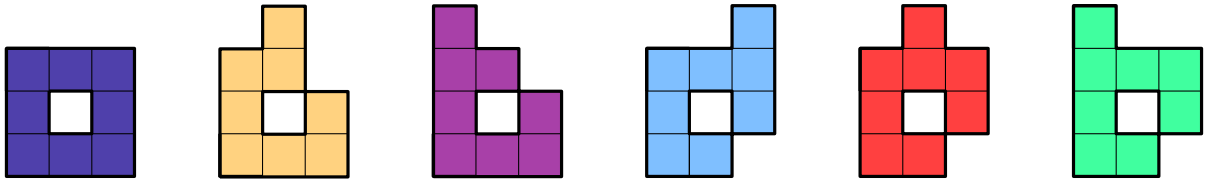


Figure 4.4: The six octominoes containing a hole.

We now introduce two new notions we will use to show how to choose a *representative* for equivalent polyominoes. As we will see, doing so will improve the running time of the enumeration.

Definition 4.8. Let L be the list of coordinates of a copy of a polyomino P. We say that L, or the copy of P it represents, is **lifted** when L is sorted in lexicographic order and the first coordinate of the list is $(0, 0)$.
The copy represented by L is the **representative** of P when L is lexicographically smaller than the lifted list of coordinates of the other copies of P.

To enumerate the polyominoes of size n, we use a rather simple method. It is based upon the fact that in any polyomino of size $n > 1$ there is at least one square we can remove to obtain a polyomino of size $n-1$. This fact is trivial, but here we deal with hole-free polyominoes, which is trickier. The proof of Lemma 4.5 will explain why this assertion is still true in the context of hole-free polyominoes. Our algorithm (see Algorithm 2 on next page) takes as input the set $\mathcal{P}_{n-1}$ of polyominoes of size $n-1$: for each $P \in \mathcal{P}_{n-1}$, we add a square at every possible location (maintaining the connectivity of the polyomino). We will explain the hole detection, done with a *flood-and-fill* algorithm, in Section 4.3.

In order to avoid storing several copies of the same polyomino, we use Definition 4.8 to keep a unique representative for each copy we enumerate. Let us reformulate the definition: a copy is *lifted* when the first element of its list of cells is $(0, 0)$ and this list is sorted by increasing x-values, and in case of equality, increasing order of y-values. A lifted polyomino does not have negative x-coordinates, but may have negative y-coordinates. This notion is central to compare two polyominoes: any copy of a polyomino has a unique lifted list of coordinates, which is a representative of this copy. Two copies with the same lifted list of coordinates are in fact the same up to translation. The *representative* copy of a polyomino is the copy which, once lifted, has the list of coordinates which is minimum for the lexicographic order. For instance, in Figure 4.2 the leftmost cell (and topmost in case there are several) of every copy will have coordinate $(0, 0)$ once the (coordinate list of the) copy is lifted. Then the two copies of the second column will have two cells of x-coordinate 0 so they are smaller than the other ones. The third cell in the (sorted) list will have x-coordinate 0 for both copies, but y-coordinate -1 for the bottom one, whereas the copy on the first row will have $y = 0$. Hence the copy of the second row and second column is the smallest copy of Figure 4.2. Again, this concept of representative of a polyomino is crucial: one can detect if two copies of polyominoes are copies of the same polyomino by comparing their representatives. We will detail in Section 4.3.3 the *hash table* data structure we used to achieve a fast insertion of new polyominoes without having duplicates of them.

Algorithm 2: Computing $\mathcal{P}_n$ from $\mathcal{P}_{n-1}$

Input: An integer n and the set $\mathcal{P}_{n-1}$
Output: $\mathcal{P}_n$

$X \leftarrow \emptyset$
foreach $P \in \mathcal{P}_{n-1}$ **do**
  foreach $(x, y) \in P$ **do**
    foreach (x', y') neighbour of (x, y) **do**
      $Q \leftarrow P \cup (x', y')$
      if Q has no holes **then**
        $Q' \leftarrow$ representative of Q `// the smallest lifted copy of` Q
        Insert Q' into X
      end
    end
  end
end
return X

4.2.2 Tiling a rectangle with a DFS (inefficient)

We present here a classical backtracking algorithm, described in Algorithm 3, which features the characteristics of a depth-first search (DFS). The idea is, when examining one specific (partial) tiling, to continue it as far as we can. When we find a contradiction we rollback one step before, try another choice, continue, and so on. DFS stands for 'depth-first search': this means that when we have several options, we first examine the first one as far as we can go before considering the second one, and so on. By doing this, we explore all possible tilings, hence if there exists a tiling we will find it. If we exhaust all the possible choices for where to put the next copy of P without managing to tile the rectangle, this constitutes a proof of the fact that P does not tile the rectangle.

Algorithm 3: Trying to tile an $n \times m$ rectangle, in a DFS fashion.

Input: The dimensions n and m of the rectangle, the polyomino P
Output: True if the $n \times m$ rectangle can be tiled by P, False otherwise

Function `DFS_tile(`*grid, P, nbLeft*`):`
  if $nbLeft = 0$ **then**
    return True
  end
  $(x, y) \leftarrow$ `choose_free(`*grid*`)`
  foreach partial tiling *grid'* extending *grid* by exactly one copy of P covering (x, y) **do**
    if `DFS_tile(`*grid', P, nbLeft-1*`)` **then**
      return True
    end
  end
  return False
return `DFS_tile(`*empty_grid, P,* $nm/|P|$`)`

One important point in the algorithm is not to enumerate the same partial tiling several times: we want to avoid putting a copy in P at cell $(0, 0)$ then one copy at cell $(3, 3)$, realise that there is a contradiction... and then try to put a copy at $(3, 3)$ and then one at $(0, 0)$ and come up with the same conclusion. To enforce this, at each depth we choose a unique cell to be tiled at that step (the role of the function `choose_free`). We then force the chosen cell (x, y) to be covered at this step backtrack directly, should the one we chose fail to be covered. We are sure to still enumerate all possible tilings: the chosen cell must be covered at some point so it might as well be covered now, and we try every possible way to cover it. When we go back because we fail to cover the cell we chose at the current step, it is possible that another option for a choice made at an earlier step will lead to a tiling of the rectangle.

In our program, we chose the function `choose_free` to return the leftmost free cell, and in case of ties, the topmost one. However, it could be chosen differently as we mention below. The choice for this function `choose_free` is crucial. Indeed, choosing judiciously the next cell to be covered can impact the performances a lot. There is one main approach. It consists in trying one cell which has very few possibilities to be covered, so that we do not have to branch a lot. We hope that we can find successive positions for which the number of ways to cover them is indeed very small: ideally only one possibility to cover it, or zero so that we backtrack directly. If we are lucky, by doing so we only have one choice at each time and the running time would in fact be 'linear': the first tiling we explore would work, with no need to backtrack. However, nothing guarantees that by choosing a spot with only a few possibilities we do not double (or even more) the number of possibilities for the next spots. The only sure thing is: if there is only one way to cover one tile, we may as well cover it now so that we realise some contradictions sooner, and we cannot make things worse since there was a unique choice to cover the cell here no matter when it is done. Other approaches based on other heuristics can be tried, as for instance tiling in 'spiral': prioritise cells according to their distance to the closest edge of the rectangle, and in clockwise direction if there are ties. This choice was less efficient than the one prioritising the cell with the lest number of possibilities. Another approach is to prioritise cells to be tiled according to their distance from the top-left corner, that is tiling kinds of diagonal waves one after the other.

This approach is very fast for polyominoes of small order, for instance it takes 0.08s for a specific polyomino of order 50. However it led to very long running times for greater orders: it takes one minute on a polyomino of order 76, and 24 minutes for a polyomino of order 96. This is why we describe a second more efficient method in Section 4.2.3: the BFS approach. This other approach is outperformed by the DFS for small orders: it takes one second for the polyomino of order 50. However, for the other two, the BFS method (with the optimisations of Section 4.3.2) takes respectively 6.5s and 4.8s.

4.2.3 BFS on the frontiers

We present here an approach which may seem slower at first: we try iteratively and 'simultaneously' all the partial tilings with k copies. Basically, we try to fill the rectangle column by column and enumerate all the partial tilings with one tile, then with two tiles, and so on. We do it on a BFS fashion: all the partial tilings with k copies of P are tried

before trying the ones with $k+1$ copies. This seems slower because we enumerate all the partial tilings sequentially whereas a good heuristic might have discarded a lot of them early and explored a promising tiling first. However, if the rectangle cannot be tiled, we are forced to try all possible tilings, hence the BFS approach might not be more costly

Definition 4.9. Let T be a partial tiling of a rectangle. Let C_l (resp. C_r) be the leftmost (resp. rightmost) non-empty yet non-completely filled column. The **frontier** is the set of cells of the columns $C_l, C_{l+1}, \cdots, C_r$ covered by the polyomino (shown in red boxes in Figure 4.5).

Remark 4.1. Assume that at each step the next cell to be covered is chosen to be the leftmost one still uncovered, and in case of ties, the topmost one. The surface covered by any partial tiling T obtained in this fashion is connected.

This is true because of the design of the algorithm. Apart from the first one, every new copy we put in the current partial tiling T necessarily shares an edge with T. In case the obtained partial tiling does not contain a hole, the surface it covers constitutes a polyomino.

We now introduce the basic algorithm without the technical details and without some optimisations: we enumerate the frontiers resulting of the partial tilings with first one tile, then two, and so on.

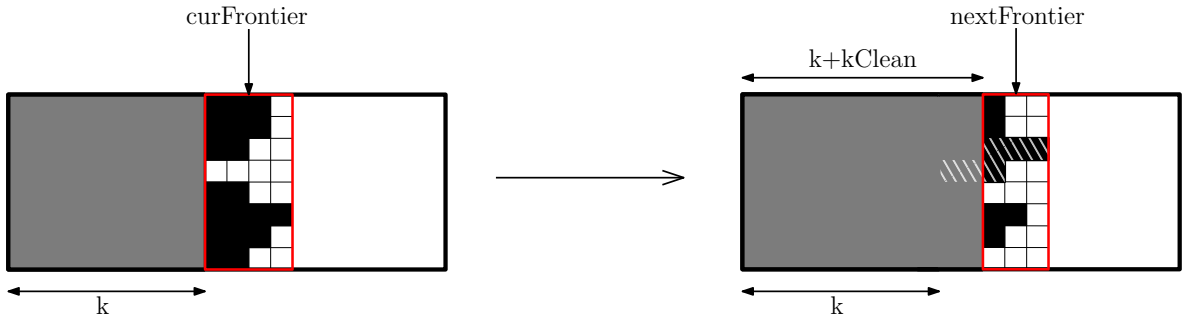


Figure 4.5: Illustration of the creation of a new frontier from en existing one (Line 8 of Algorithm 4). The cells hatched with white lines represent the copy of P which was added at this step.

Algorithm 4: Trying to tile a rectangle with n lines and at most $m_{\max}$ columns, in a BFS fashion.

Input: The dimension n and $m_{\max}$ of the rectangle, the polyomino P
Output: (True, m) if the $n \times m$ rectangle can be tiled by P, False otherwise

```
1  nbColumnsTiledRectangle ← +∞ // No tiled rectangle so far.
2  newFrontiers ← ∅   // This is a queue.
3  emptyColumn ← [0, ··· , 0] // n zeroes:  an empty column; covered cells
       contain 1's.
4  newFrontiers.push((0, emptyColumn)) // 0 means there were no filled
       columns at the left
5  while newFrontiers is not empty do
6  |   (k,curFrontier) ← newFrontiers.pop() // curFrontier was seen with k
   |      filled columns at its left
7  |   (x,y) ← choose_leftmost_free(curFrontier) // In case of ties, the
   |      topmost one.
8  |   foreach nextFrontier extending curFrontier by a copy of P covering (x,y)
   |      // See Figure 4.5.
9  |    do
10 |   |   nextFrontierClean ← nextFrontier stripped of its fully filled columns
11 |   |   kClean ← number of removed filled columns
12 |   |   if nextFrontierClean = [ ] then
   |   |   |   // We tiled a rectangle.
13 |   |   |   nbColumnsTiledRectangle ← k+kClean
14 |   |   |   break // We get out of the loop.
15 |   |   end
16 |   |   if nextFrontierClean was not seen yet then
17 |   |   |   Mark nextFrontierClean as seen
18 |   |   |   newFrontiers.push((k+kClean, nextFrontierClean))
19 |   |   end
20 |   end
21 end
22 return nbColumnsTiledRectangle
```

The general principle of this algorithm was already used by Karl Dahlke. In this algorithm, we first explore all frontiers consisting of one copy of P, then the ones with two copies, then the ones with three copies, and so on. Hence this algorithm has the characteristic of a BFS. Indeed, BFS stands for 'Breadth-first search': we explore step by step all possible solutions, as opposed to the DFS which explores fully each one after the other.

We can observe that any frontier may consists of at most w columns, where w is the maximum width of the copies of the polyomino. We speak here of the number of columns which are not completely filled. Indeed, since we always try to fill the leftmost free cell, any copy we add must fill this cell. The result follows since it spans at most w columns. This means that the number of frontiers is bounded by 2^{nw}. However, since we forbid a lot of configurations, we have fewer than 2^{nw} possible configurations: see Section 4.3.2 to see how we discard some frontiers, apart from the obvious cases when

the configuration contains a hole. Since the maximum width of a polyomino is fixed, it is better, when trying to tile a $n \times m$ rectangle, to choose $\min(n, m)$ as the number of lines of the rectangle we actually try to tile. The method works well for rectangles with disproportionate widths and heights, and less with the ones close to squares.

Keeping the frontiers in memory enables us to reduce the running time of the program (see Line 16). Indeed, we may obtain a frontier by several different ways. By keeping them in memory we know that when we find some frontier F which we had already seen (necessarily with fewer filled columns since we enumerate the frontiers in a BFS way, with fewer copies needed first), we may skip it now instead of investigating again what other frontiers it leads to. To save memory and for easier access, we only remember the frontiers without the completely filled columns, and instead store this number and link the frontier to it: in Figure 4.5 the four (on the left) and three (on the right) columns, surrounded by a red rectangle, are the real frontiers we store. For the left one we associate the number k of filled columns, and the number k+kClean for the right one. By doing so, if we find a frontier F with 42 preceding filled columns and see again later F with 44 filled column, we overlook it. We used a *hash table* to perform fast lookups for already seen frontiers. We detail this data structure in Section 4.3.3.

4.2.4 Another approach: solving a linear program

At the beginning, we decided to give a chance to some solvers of (integer) linear programs. Indeed, the problem of knowing if a polyomino tiles a given rectangle can be expressed as an integer linear program. The idea is that we encode the locations of the copies of the polyomino and ensure that each cell is covered exactly once. To so do, we first consider the 8 different orientations of our polyomino P (applying rotations and horizontal symmetry): $P_1, \ldots, P_8$. For each P_i and each cell (x, y) of the rectangle, we denote by $C(P_i, x, y)$ the set of coordinates covered by placing the top-left cell of P_i on cell (x, y) or $\varnothing$ if doing so causes the polyomino to go over the rectangle. Our integer linear program can then be defined as:

$$\text{For each } x_0, y_0 : \sum_{(x_0,y_0)\in C(P_i,x,y)} A_{i,x,y} = 1.$$

Each sum is done for a fixed value (x_0, y_0): we are summing over i, x and y. To each (x_0, y_0) corresponds an equation for our linear program. If the set of feasible solutions is not empty, this means that a tiling exists, and the variables set to one give this tiling. If, on the contrary, it is empty, this is a proof of the fact that the given polyomino does not tile the rectangle.

However, even by using Gurobi, one of the best solvers available, we did not obtain any interesting results with this method. The program using Gurobi was much too slow: 4 minutes for the polyomino of order 76, and 24 minutes for another one of order 96. The method in Section 4.2.3 (with the optimisations of Section 4.3.2) took respectively 6.5s for the polyomino of order 76 and 4.8s for the one of order 96. In addition to this, Gurobi used the 24 cores available in the test machine whereas the time for the BFS method was achieved using only one core. Gurobi was even (slightly) outperformed by the slow DFS algorithm!

4.3 Refinements of the algorithms and other optimisations

We present here two main things. On the one hand we introduce other techniques to improve our search. We begin by giving and explaining methods to show that a polyomino is not rectifiable, so that we may avoid to lose time by not attempting to find its order. We then present optimisations both in the design and the implementation of the algorithms of the Section 4.2, which we kept at the time as simple as possible for pedagogical purposes.

4.3.1 Ruling out non rectifiable polyominoes

One crucial point is to find which polyominoes can be discarded because they are not rectifiable. Indeed, it would take an infinite amount of time to try to tile all possible rectangles with a polyomino which is not rectifiable. This is why we need efficient methods which can discard as many non-rectifiable polyominoes as possible. Some of the techniques were used by Karl Dahlke (see [11]), and some are 'new' or were improved by us.

Playing chess.

This method does not, properly speaking, detect non-rectifiable polyominoes, but rather shows that some classes of rectangles with specific properties cannot be tiled by some polyominoes. This well-known parity argument consists in overlaying a checkerboard on the rectangle and deducing some properties of the tiling. For instance, let us assume that the polyomino P is formed out of a rectangle with 2 lines and three columns by removing the cell of the middle column in the first row. Wherever on the checkerboard we place it, horizontally or vertically, it consumes either three white cells and one black, or the contrary, once put on the checkerboard. Since P has an even number of cells, the rectangle it might tile must have, for instance an even number of lines, therefore it contains as many black cells as white cells. We deduce that there should be an even number of copies of P since each one consumes an odd number of either black or white cells: the set of copies covering 3 black cells must be of even size, as must its white counterpart. This shows that our polyomino cannot tile any rectangle of size $n \times l$ when kl is not a multiple of 10. For instance tiling a rectangle of size 5×5 would require 5 copies of the polyomino, hence either the number of white cells covered or the one of black cells covered would be odd. This argument implies that the polyomino is less interesting: its possible order cannot be odd. We also tried to obtain other equations showing that some other types of rectangles are impossible to be tiled by some polyominoes. We tried to put alternating columns of white and black cells, or even different moduli: instead of having white and black cells, that is reasoning modulo 2, we can try with other prime numbers. We managed to discard some rectangles which could not be discarded with the classical checkerboard argument, but not many more.

The checkerboard argument can also be exploited in another way: when tiling any rectangle there must be as many pieces covering three black cells as pieces covering three white cells since there are as many black cells as white cells. This could be used to improve the DFS algorithm: if, out of the $k = nm/|P|$ copies, we already placed more than $k/2$ pieces covering three white cells out of the k pieces to put, we may backtrack.

Tiling a quarter of the plane.

As we mentioned in Section 4.1, being rectifiable implies tiling a half-plane. Using the same argument, it is easy to prove that in fact being rectifiable also implies tiling a quarter of the plane ($\{(x, y) \mid x \geq 0, y \leq 0\}$ for instance). We use this property to show the non-rectifiability of some polyominoes: if they do not tile a quarter of the plane then they are not rectifiable. We also use another similar property which we will define just below.

We now present one property which can be tested as soon as when we enumerate the polyominoes and helps us enumerating fewer of them, hence speeding up the process.

Definition 4.10. We say that a polyomino P is **corner compatible** when there is a way to place it such that it covers the corner cell of an arbitrarily large rectangle without disconnecting the set of empty cells of the rectangle.

Informally, P is not corner compatible when, however the way we place it, it separates the set of empty cells into several connecting components. This implies that the polyomino cannot tile a quarter of the plane, thus we can overlook them. Note that if we cannot cover the top-left corner, it implies that the polyomino is not corner compatible.

Notation 4.11. Let us denote by $\boldsymbol{\mathcal{P}_n^*}$ the set of free polyominoes of size n, with no holes, which are corner compatible.

The cross with 5 squares (see Figure 4.6) is an example of a polyomino which is not corner compatible: because it creates a hole which cannot be filled so that the corner cannot be tiled. Our program uses a modified version of Algorithm 2 to enumerate elements of $\mathcal{P}_n^*$: it discards any polyomino which is not corner compatible. This is done in the same way as the elimination polyominoes with holes: we use a flood-and-fill algorithm[5]. We assume (up to rotating the copy of the polyomino) that we want to tile the top-left corner. We put each copy of the polyomino into the smallest rectangle hull and add a column at the right and one line at the bottom. We then compute the set of connected components of the free cells (the ones which do not belong to the polyomino), considering that the polyomino is a 'wall': two cells are connected if they are connected in the grid and none of them belong to the polyomino. The polyomino contains a hole or is not corner compatible if and only if there are several components.

We must now ensure that any element of $\mathcal{P}_n^*$ can be obtained from an element $\mathcal{P}_{n-1}^*$ to which we add a square, so that computing $\mathcal{P}_n^*$ from $\mathcal{P}_{n-1}^*$ indeed enumerates all the polyominoes we are interested in.

Lemma 4.5. *Let $n > 1$ and P be a polyomino of size n which is corner compatible and contains no holes. Then there exists a polyomino Q of size $n-1$ with the same properties such that $P \setminus Q$ is a single square.*

What follows also proves that the algorithm we explained in Section 4.2 for the enumeration of hole-free polyominoes is valid. Indeed, the lemma remains true if we remove

[5]The flood-and-fill algorithm is a classical graph algorithm. It starts from one vertex and fills it with one colour, along with any vertex accessible: it colours its connected component with the same colour. Repeat this, with a new colour each time, for any vertex not yet coloured so that we compute the different connected components of the graph, in linear time.

the corner-compatibility requirement and only keep the hole-free one, as we mention at the end of the proof.

Proof. Let us place P on the top-left corner of a smallest rectangle containing it and augmented by a column on the right and a line below, in a way such that the flood-and-fill algorithm would find a single component outside P. We first observe that if the polyomino is corner compatible, this implies that there exists some x_0 such that every cell of the top row with $x \leq x_0$ belongs to P. Otherwise, this would create a component of free cells disjoint from the one containing the rightmost cell of the bottom line. The same goes for the leftmost column: there exists some y_0 such that every cell of the first column with $y \leq y_0$ belongs to P. Note that both sets we defined contain the top-left cell, so they are connected.

We know, since $n > 1$, that $\max(x_0, y_0) > 0$. We assume without loss of generality that $x_0 > 0$: at least two cells of the top row are covered by P. We define a procedure which will terminate, and find the a cell we can remove. We first choose the cell $(x_0, 0)$ and try to remove it. We know that we are neither creating a hole nor obtaining a non-corner-compatible polyomino since this cell has a free neighbour at its right. If the polyomino we obtain is still connected, we set $Q = P \setminus (x_0, 0)$.

Otherwise, we consider the polyomino P_1' defined as the component disconnected from P by removing $(x_0, 0)$, and we set $P_1 = P_1' \cup (x_0, 0)$. Note that P_1' does not contain any cell of the form $(0, y)$ or $(x, 0)$: we mentioned that all the cells of this shape which belong to P are connected to $(x_0, 0)$, hence not in P_1. This means that any Q we will define by removing a cell from P_1 will necessarily be corner compatible. If P_1' is reduced to a single cell, we may remove it to obtain Q with the desired properties. Otherwise, we choose a cell $(x_1, y_1) \in P_1 \setminus (x_0, 0)$ with fewer than four neighbours in P_1. Since it has fewer than 4 neighbours, removing it does not create a hole in the polyomino. If removing it does not disconnect P_1, then it does not disconnect P: the only way to disconnect P_1 from P is to remove (x_0, y_0) and we define $Q = P \setminus (x_1, y_1)$. Otherwise, let P_2' be a connected component of $P \setminus (x_1, y_1)$ which does not contain (x_0, y_0). We set $P_2 = P_2' \cup (x_1, y_1)$. P_2 is smaller than P_1 so that our procedure selects smaller and smaller parts of P. This means that it will terminate at some point, finding some cell (x, y) with fewer than four neighbours which fulfils our conditions.

If we only want a cell which does not create a hole or disconnects the polyomino, to show that our generation of hole-free polyominoes in Section 4.2.1 works, it suffices to choose for (x_0, y_0) any cell with fewer than four neighbours. The rest of the procedure is not modified, and comes up with some cell such that $P \setminus (x, y)$ is a connected hole-free polyomino. □

Trying to 'fully' tile a corner.

This method looks a bit like the method we have just explained, to test if the polyomino is corner compatible. In fact, it is an extension of the corner-compatibility concept. Trying to fully tile a corner consists in taking the corner of a quarter of a plane. We then try to tile this corner, by tiling the k cells which are the closest to the corner point. If we want to cover all cells at distance at most d, we will have to cover $(d+1)(d+2)/2$ cells. We do this using our DFS algorithm, covering cells according to the distance to the corner. If at any step it turns out that it is impossible to cover these k cells, then we know

that the polyomino will not be able to tile any rectangle (but it might tile the plane like the one in Figure 4.6). This method extends the one of the corner compatibility because by tiling cells at distance at most $|P| + 1$ for instance we would have realised that the position of the first copy of the polyomino induces an unfillable hole. To see the relative efficiency of this method, see Table 4.2 where we can observe that almost every polyomino ruled out by this method also does not 'tile' a bottom band of a certain size. One could ask why introduce two concepts if tiling a corner implies being corner compatible. The answer is that the corner-compatibility test can be used during the enumeration phase, discarding a lot of polyominoes (see the data of Section 4.4.1). This saves us a lot of time generating the polyominoes.

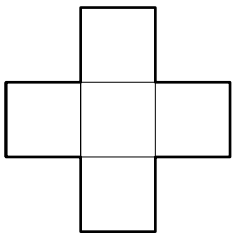


Figure 4.6: The cross polyomino. It cannot tile the rectangle of a rectangle but can tile the plane $\mathbb{Z}^2$.

Trying to 'tile' the bottom band.

The most efficient technique to rule-out polyominoes which are not rectifiable is to try to tile the bottom band of a rectangle. Indeed, it may not only show that a particular rectangle cannot be tiled by a polyomino, but also show that the polyomino is not rectifiable at all. The scheme is simple: we try to tile a band of height h but we allow the tiling to go over h. However, it must not exceed the strip neither by the left or by the bottom. This way, we simulate the bottommost h lines of any rectangle to be tiled by the polyomino. We enumerate all the possible frontiers in a BFS manner, like the one for tiling a rectangle. We keep all the frontiers in memory and stop when either we found a 'full frontier': all the columns are filled, or when we no longer enumerate new frontiers. In the latter case, this means that any rectangle with a height greater than h cannot be tiled: indeed this outcome of the algorithm shows that in any possible tiling the bottom-right corner cannot be covered. It is easy to show, by using this technique with $h = 2$, that the z-shaped pieces in Tetris cannot tile any rectangle. Sometimes we have no contradictions with $h = 2$ or $h = 3$, but when we increase the height we find one. For the efficiency of this method, see Table 4.2.

4.3.2 Improving the BFS approach

After presenting some ways to mark a polyomino as not rectifiable or showing that it cannot tile some family of rectangles, we focus here on the BFS algorithm described in Algorithm 4 for which we present some optimisations.

Looking for a complementary frontier.

In addition to using the BFS approach, Dahlke used a clever idea: when trying to tile a rectangle, we may at each step look for the **complement** of the current frontier among the ones we have seen. If we have already enumerated the complement (the rectangle deprived of the current tiling), then assembling the two frontiers yield to a tiling of the rectangle and we are finished. Dahlke uses this idea to stop his search at roughly half the width of the desired rectangle: if a tiling exists, then it can be split into two parts of almost equal sizes, the biggest of the two being roughly half of the width of the rectangle. Therefore it is possible to look for this biggest part, and to stop the search when we no longer have any chance to find it. However, due to the way we generate the frontiers, we cannot guarantee that we would enumerate this biggest part of the particular decomposition we spoke of. Hence we always look for a complementary frontier, but we do not stop the search at half the width of the rectangle. Yet, in our runs, we could always find the order of the known rectifiable polyominoes by stopping halfway of when trying to tile a rectangle.

Using symmetries.

The notion of symmetries can generally lead to large reductions in the running time of some algorithms. Here we use it to reduce the running time of Algorithm 4 but also its memory usage. Let us say that two frontiers F and F' are **equivalent** if F' is obtained from F by applying a horizontal symmetry. For each frontier, we maintain only one representative for each equivalence class[6]. In our case, we choose the lexicographically smallest (see the paragraph after Notation 4.7). This way, we approximately halve (some states are symmetric) the number of frontiers we store. Since we also process half the number of states, hence also the running time. When we consider a new frontier, we now also check if we have seen the symmetric of the representative, or the complement of its symmetric.

Forbidden relative positions.

Some copies of a polyomino P may need to avoid some particular cells. For instance, let us consider a square of width three from which we remove the top cell of the middle column. It is obvious that if we place it in this orientation at the top of the rectangle, the middle free cell will never be covered. If we reverse the orientation (by applying a horizontal symmetry), the same problem happens: it cannot be placed at the bottom of a rectangle for the same reason. To avoid exploring and storing useless frontiers leading to or containing such contradictions, we compute at the beginning, for each copy of P, the set of cells on which we may place the copy such that that the frontier will not be trivially not extendible. To test this, we may for instance place the copy on a location and then try to cover the next k cells for some value of k. If this is not possible, we forbid this position for the given copy. The bigger k is, the more time it takes to do this pre-computation but the most efficient it will turn out later so we have to choose a good trade-off. One could worry that if we place the copy near the last column of the rectangle we want to tile, then there is no need for some k other copies to be placed. This could lead us to some false forbidden positions. However this is not true: let us assume P tiles

[6]They are of size two, or one if the frontier is invariant by symmetry.

a $n \times m$ rectangle. Then it also tiles a $2n \times 2m$ rectangle, so this is not a problem at all because with a wide enough rectangle, we would be able to place k copies of P for any value of k.

4.3.3 Other optimisations

We begin by giving an optimisation for the DFS approach, and then speak of more technical optimisations, which are a matter of implementation.

Forbidden pairs.

In order to avoid placing one copy of P and realise later on that there is no way to cover some neighbouring cells, we do some other pre-computations. We trade again some extra time at the beginning of the program for benefits each time some particular function is run. This time the idea is to know, for every two copies P_1, P_2 of our polyomino, at which relative positions they may be put from each other to guarantee that the obtained partial tiling is extendible. Let us assume that we put P_1 at (x_1, y_1). For each free position (x_2, y_2) we check whether, if P_2 is put at (x_2, y_2), there is a way to cover all cells neighbouring $P_1 \cup P_2$. If it is not possible we forbid P_1 and P_2 to be placed at these positions relatively to each other. For instance, if putting P_1 at $(2, 3)$ and P_2 at $(6, 3)$ would necessarily cause one neighbouring cell to be uncovered however the way we extend the partial tiling, we will forbid P_2 to be separated by the vector $(6, 3) - (2, 3) = (4, 0)$ from where any P_1 is placed. This is very useful because it saves us time (we realise sooner that placing P_2 there leads nowhere) and memory (we store fewer frontiers). This optimisation is much more suited to the DFS approach, because we explore one partial tiling at a time. For the BFS approach, we would need to store a list of forbidden cells for specific copies, and detect when it is time to remove elements from this list.

Data structures used.

In order to have good performance, one needs to use the right data structures. Some of them may be asymptotically optimal but have poor performance when used with few elements. For instance, a priority queue achieves insertion and deletion in $\mathcal{O}(\log(n))$ while keeping the elements ordered but there is a constant in the $\mathcal{O}$ which can make it worse than using an array and inserting or deleting (at an arbitrary position) in time $\mathcal{O}(n)$, when n is not big enough. Here, we used a hash table (`std::unordered_map` in the C++ code) to store and remember the frontiers. This allows us to make operations (search, insertion for instance) in time $\mathcal{O}(1)$ on average. To further optimise, since the complement of any frontier F must have the same number of partially filled columns to be able to match F, we may store the frontiers according to their sizes. We create an array of size w of hash tables, one for every possible number of partially filled columns, and store and look for a complement in the right hash table. When we look up for a frontier of width k, we look it up in the hash table of index k. This way, each hash table has fewer elements than a single one would have. This makes the insertions and lookups empirically faster.

Let us describe a bit more this data structure. Each element is assigned a *hash*, for instance a 64-bit value computed from the value of the element. The hash is then some

index to access the element in the hash table. Since there are many 64-bit integers, we cannot allocate the memory for a full array, so different hashes may be attributed the same *bucket*. This also happens for two elements which would have the same hash. In case, when we look for the presence in the hashtable of some element x, two scenarii can occur. If the hash of x is not present, we know that x is necessarily absent from the table. If the hash exists, all the elements with the same hash (the point in the structure is to choose a good hash function so that there are few collisions) are checked until one equal to x is found, if any. This makes almost all operations on a hashtable to take $\mathcal{O}(n)$ in the worst case (all elements are put in the same bucket) but an average $\mathcal{O}(1)$ practical complexity. In C++, this structure corresponds to `std::unordered_set<>` and `std::unordered_map<>`.

For the frontiers, we use a vector of boolean values: each cell is either occupied (`true`) or empty (`false`). In C++, the corresponding `std::vector<bool>` is in fact a *bitset*: instead of having each bool element take its usual one byte size, here one byte stores eight elements. This is saving a lot of memory, but may or may not slow down the program depending on the operations made.

Too many polyominoes.
When enumerating polyominoes, at some point there are just too many of them. In this case, even if they can fit in the very large RAM we had at our disposal (around 1.5TiB), we may not want to write such a file to the disk. Therefore we decided to compress the files we were writing. One simple way was to use the gzip format and the gzstream library[7] which enables us to directly write in the compressed format with very little modification of the C++ code. We achieved a good gain. For instance, the list of the about 5 million of free hole-free and corner-compatible polyominoes of size 16 takes 358Mib uncompressed versus 45Mib if compressed. If we compare the other sizes, it seems that we reduce the size of the file by a factor around 9.

4.4 Statistics and perspectives

4.4.1 Statistics on the polyominoes and their orders

We give here statistics about the orders of polyominoes, and also details about polyominoes we could discard as not rectifiable. Given the way we decided to enumerate only polyominoes which had a chance to tile a rectangle, the statistics we give are to be read as data about polyominoes in $\mathcal{P}_n^*$: **with no holes** and which **are corner compatible**. Polyominoes with holes, or which would split the set of free cells of a rectangle into several connected components when put on a corner (however the way we put them) are not counted here.

We recall that $\mathcal{P}_n$ is the set of hole-free free polyominoes, that is no two polyominoes in this set are isometric; $\mathcal{P}_n^*$ is the set of polyominoes we enumerate: they contain no holes, are corner compatible (see Definition 4.10). In Section 4.3.1 we explained two methods to rule out a polyomino as not rectifiable, after they are enumerated. We show

[7] `https://www.cs.unc.edu/Research/compgeom/gzstream/`

here the efficiency two methods: the *band method* which consists in trying to tile the bottom band of a rectangle and the *corner method* which tries to tile a corner with some number k of pieces, covering cells by prioritising at each step the ones closest to the corner.

As indicated in the caption of Table 4.2, for lines 17 and 18 the percentage of ruled out polyominoes is 100.00. In fact there remain a few polyominoes, but very very few: 18 out of the 18 637 273 we generated (using the corner-compatibility optimisation) for size 17, for instance. This illustrates completely the scarcity of rectifiable polyominoes. Another interesting property is that when the size is a prime number there seems to be much fewer possibly rectifiable polyominoes: the number of remaining polyominoes is less than 20 for orders 11, 13, 17 when it is at least 97 for orders 12, 14, 15 and 16. The numbers for 15 and 16 are even 210 and 385, before dropping down to 18 for size 17... and it increases again to 686 for size 18.

n	$\#\mathcal{P}_n$	$\#\mathcal{P}_n^*$	Maybe	Order ≤ 10	$10 <$ Order ≤ 100	$100 <$ Order ≤ 300	? (> 150)	? (> 300)
1	1	1	1	1	0	0	0	0
2	1	1	1	1	0	0	0	0
3	2	2	2	2	0	0	0	0
4	5	5	4	4	0	0	0	0
5	12	11	4	4	0	0	0	0
6	35	32	10	8	2	0	0	0
7	107	91	7	4	2	0	1	0
8	363	288	16	10	1	2	2	1
9	1248	923	36	33	0	0	3	0
10	4460	3062	33	26	1	1	2	3
11	16094	10296	13	6	1	1	4	1
12	58937	35175	97	79	0	1	8	9
13	217117	121349	10	7	0	0	1	2
14	805475	422665	101	84	0	0	9	8
15	3001127	1483274	210	192	1	0	6	11
16	11230003	5241856	385	370	0	0	4	11
17	42161529	18637273	18	9	0	0	0	9
18	158781106	66635182	686	650	0	0	18	18

Table 4.1: Statistics on the order of polyominoes up to size 18.
$\mathcal{P}_n$ is the number of free polyominoes of size n with no holes. $\mathcal{P}_n^* \subset \mathcal{P}_n$ contains the polyominoes without holes which are corner-compatible (see Definition 4.10).
'Maybe' indicates the number of polyominoes that we could not discard with the corner method with 50 copies and the bottom 'tiling' method of height up to 11 (one more than in table 4.2.
The '? $> x$' columns mean that we proved the polyomino either is not rectifiable or has order greater than x. These columns are not cumulative but mutually exclusive: the column '? (> 100)' does not include the polyominoes of the column '? (> 200)'.

n	Ruled out	C	B	C & B	B2	B3	B4	B5	B6	B7	B8	B9	B10
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	20.00	0.00	20.00	0.00	20.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	63.64	45.45	63.64	45.45	18.18	36.36	9.09	0.00	0.00	0.00	0.00	0.00	0.00
6	68.75	50.00	68.75	50.00	18.75	31.25	9.38	6.25	0.00	0.00	0.00	3.12	0.00
7	92.31	86.81	91.21	85.71	19.78	45.05	15.38	7.69	1.10	1.10	1.10	0.00	0.00
8	94.44	88.54	94.44	88.54	21.53	44.44	16.32	7.99	3.12	0.69	0.00	0.00	0.35
9	96.10	93.17	96.10	93.17	24.05	46.05	17.77	5.31	1.84	0.76	0.00	0.22	0.11
10	98.92	97.26	98.92	97.26	26.09	47.68	17.41	5.23	1.60	0.56	0.23	0.07	0.07
11	99.87	99.78	99.87	99.78	28.20	48.47	17.03	4.24	1.31	0.38	0.16	0.07	0.03
12	99.72	99.36	99.71	99.35	30.00	48.51	15.97	3.73	1.01	0.33	0.11	0.04	0.01
13	99.99	99.98	99.99	99.97	31.71	48.93	14.91	3.19	0.86	0.26	0.09	0.03	0.01
14	99.98	99.94	99.97	99.93	33.21	48.84	14.01	2.89	0.71	0.21	0.07	0.02	0.01
15	99.99	99.97	99.98	99.96	34.58	48.68	13.19	2.64	0.63	0.18	0.06	0.02	0.01
16	99.99	99.98	99.99	99.98	35.78	48.50	12.46	2.45	0.58	0.16	0.05	0.02	0.01
17	100.00	100.00	100.00	100.00	36.86	48.25	11.85	2.29	0.54	0.15	0.04	0.01	0.00
18	100.00	100.00	100.00	100.00	37.83	47.98	11.33	2.17	0.50	0.14	0.04	0.01	0.00

Table 4.2: How the polyominoes are ruled out as not rectifiable.
'C' means the percentages of polyominoes (out of $|\mathcal{P}_n^*|$) ruled out as not being able to put 50 copies to tile a corner.
'Bi' means the ones which cannot 'tile' a band of size i but can tile a band of size $i-1$.
'C & B' means both the band and the corner methods rule out these polyominoes.
The methods for ruling out polyominoes are explained in Section 4.3.1. For $n = 17$ and $n = 18$ the 100.00 and 0 .00 are rounded: some polyominoes are rectifiable, and the band method for $h = 10$ is useful: it rules out 2464 polyominoes for $n = 18$ for example.

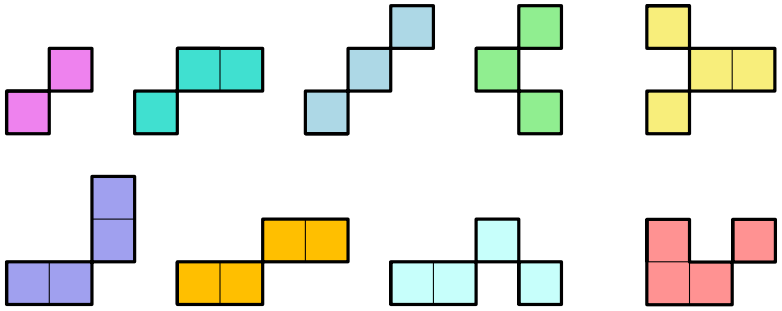


Figure 4.7: Example of extended polyominoes which are not classical polyominoes. Six of them have order two.

4.4.2 Perspectives for future works

Trying other types of polyominoes.

Since we could not find a polyomino of odd order, we thought about more general types of objects. First we thought about 3D polyominoes, which would have required quite some work to rewrite the program to test them. A less costly approach, which we tried, was to test the **extended polyominoes** (see Figure 4.7): sets of unit squares of $\mathbb{Z}^2$ connected either by edges or by corners. This means that the cells of an extended polyomino need not be connected by edges but can be connected through their corners. We did not have more success with extended polyominoes: we found none with an odd order, despite the fact that there are much more of them than of regular polyominoes. We explored extended polyominoes of size up to 11 (there are around 1.6 millions of them) but found none of odd order either.

Finding other impossible orders.

So far, we only know that the number three cannot be the order of a polyomino. It would be interesting to check if this fact still holds for greater odd number, like five, seven and so on. The problem with the argument in [48] is that it is an ad-hoc geometric argument which does not seem, as the authors write, to be generalisable. It would be nice to simplify their argument so that showing the same for five, should it be true, would be less tedious.

Conclusions and perspectives

The main goal of this thesis was to use the power of computers to solve some problems of graph theory. We attacked four main problems, two of which are much related: the dominations numbers and the growth rates of various dominating sets. We solved totally some problems we attacked, some others partially, and did not make much raw progress in the polyomino problem. We recall in what follows our contribution to each problem we tackled, chapter by chapter.

In Chapter 1 we reproduced the proof of the four-colour theorem by Robertson et al. [45]. It did not need verification since it was proved in Coq. However, this gave us an occasion to give another explanation of their proof, in a more abstracted way, but keeping some technical details. A piece of software we developed was released to the community to make it easier for anyone to investigate a problem using the discharging method. We gave small hints as to how the search for the discharging rules could be automated. Achieving some kind of automation on this part would be a very good improvement to the method, and would contribute to make it more widespread and performing.

Chapter 2 provides an alternate explanation of the method of the loss introduced by Gonçalves et al. [21]. We reuse this method for the first time to solve the 2-domination and the Roman-domination numbers on grid graphs: we give closed formulas computing these numbers for any size of grids. This proves that the lower bound for the Roman domination of grids given by Currò [10] was tight. We also give values for the total-domination and distance-2-domination numbers for small number of lines on grids. This confirms the first formulas of the work of Crevals and Ostergård [9]. We also give bounds for arbitrary grids for the total domination, as well as a conjecture on the real formula. According to our conjectures, the total domination number is out of reach of the loss method. What method will be found to solve this problem? For all the problems, we may also wonder how to solve the problems on cylinders: Cartesian products between a path and a cycle.

In Chapter 3, we study the counting problem for various dominations problems: the domination, total domination, and their minimal counterparts. To solve it, we link it to the study of SFTs. We show some properties on the associated subshifts: they are block gluing. We prove that each number of dominating sets grows exponentially, at a specific growth rate. We show that these growths rate are computable, and give numerical bounds on each of them. The bounds on the domination and total domination are quite good, and we are able to conjecture the actual value of their growth rates. Will these conjecture be proved someday? Also, we may resort to other methods to improve the bounds for the minimal domination and the minimal total domination.

Concerning the polyominoes, in Chapter 4, we do our little part in tackling the question of whether or not a polyomino of odd order exists. We pursue notably the work done by Dahlke to find new orders empirically. We recall some algorithmical techniques to discard a polyomino as being not rectifiable, and to find its order. We also improve some of them. We compare the effectiveness of the main methods to show that a polyomino is not rectifiable. We also sum up some data about the orders we could find for polyominos of size up to 18. If there exists one polyomino of odd order, the only proof needed is this polyomino and the reasons why it does not tile a smaller rectangle. This involves checking a finite number of cases. It seems much harder to prove that no odd number greater than one is the order of some polyomino, given that so far we only proved it for three. Also, the fact that some polyominoes can tile some rectangle with an odd number of copies (but have a smaller even order) rather seems to give more credit to the existence of an odd-order polyomino than to its impossibility. Also, due to parity reasons there are many polyominoes which cannot tile any rectangle with an odd number of copies. This could explain their possible scarcity and our difficulties in finding one.

This PhD was also an occasion to think about the proofs using results from some computer program. Some doubt them because a bug could occur, or even some bit in the memory of the computer could flip because of cosmic rays[8]. One first answer to this objection is to ask if a very long mathematical proof (with no use of a computer) can truly be verified with a high degree of scrutiny. The more the proof is long and complicated, the more small (or even bigger ones) flaws can be contained in it. Also, a source code may be hard to verify, because it requires simulating it somehow. It is not necessarily more prone to risks than a long proof. Yet, we could get some inspiration from experimental sciences to convince sceptics: it is possible to strengthen the confidence in some program by having other people program it their own way, independently. Should one or more teams be able to independently make a program giving the same results, we could consider the risk of bugs or errors is negligible.

Something else we may wonder is how much computers will help us solve problems in the future. Or, rather, which problems will be within reach, and which could remain forever out of reach of computer solving. In 2010, Google put a lot of computing power to solve a problem on Rubik's cube. They showed that, from any of the $43 \cdot 10^{18}$ starting configurations, one could solve the cube with 20 moves or less. Other problems seem out of reach of today's computers, like finding a winning strategy for chess, because the combinatorial complexity of this game is huge: there are an enormous amount of configurations of the game. However, it seems impossible from today's knowledge. Maybe the next years will see a shift on the computing models: some people for instance think that quantum computers will help us solve some problems which were previously thought out of reach. On the contrary, we do not know if the computing resources will continue to grow forever. For instance, the frequencies of processors has stopped to grow for some years, because of limits from physics. This forces people to use parallelism instead of raw power, but comes with limitations: the more units you put together, the more time you spend in communications between them. So the question remains: which problems will

[8] yes, it happens, but not that much

be solved in 100 years[9] which were not possible to solve today?

Finally, each chapter was the occasion, in this manuscript, to sensitize the readers to some problems of our world. The major one is ecological: global warming and the huge loss of biodiversity. Maybe it is also time to reconsider everyday life, and some choices made a long ago when we were not informed. It should require us to reconsider and update our values and lives accordingly.

[9] if we are still here trying to solve these kind of abstract problems, and with computers...

Bibliography

[1] Appel, K. I., & Haken, W. (1989). Every planar map is four colorable. American Mathematical Society, 98.

[2] Beauquier, D., & Nivat, M. (1991). On translating one polyomino to tile the plane. Discrete & Computational Geometry, 6(4), 575-592.

[3] Birkhoff, G. D. (1913). The reducibility of maps. American Journal of Mathematics, 35(2), 115-128.

[4] Bonomo, F., Brešar, B., Grippo, L. N., Milanič, M., & Safe, M. D. (2018). Domination parameters with number 2: Interrelations and algorithmic consequences. Discrete Applied Mathematics, 235, 23-50.

[5] Bouznif, M., Moncel, J., & Preissmann, M. (2016). A constant time algorithm for some optimization problems in rotagraphs and fasciagraphs. Discrete Applied Mathematics, 208, 27-40.

[6] Chang, T. Y., Clark, W. E., & Hare, E. O. (1994). Domination numbers of complete grid graphs. I. Ars Combin, 38(1), 994.

[7] Chartrand, G., & Zhang, P. (2008). Chromatic graph theory. Chapman and Hall/CRC.

[8] Courcelle, B. (1990). The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and computation, 85(1), 12-75.

[9] Crevals, S., & Ostergård, P. R. Total Domination of Grid Graphs. (2017). Journal of Combinatorial Mathematics and Combinatorial Computing, 101, 175-192.

[10] Currò, V. (2014). The Roman domination problem on grid graphs, PhD Thesis.

[11] Dahlke, K., `http://www.eklhad.net/polyomino/`

[12] Dahlke, K. A. (1989). The Y-hexomino has order 92. Journal of Combinatorial Theory Series A, 51(1), 125-126.

[13] Farina, M., & Grez, A. (2016). New Upper Bounds on the Distance Domination Numbers of Grids. Rose-Hulman Undergraduate Mathematics Journal, 17(2), 7.

[14] Fisher, D. C. (1993). The domination number of complete grid graphs. Manuscript.

[15] Fomin, F. V., Grandoni, F., Pyatkin, A. V., & Stepanov, A. A. (2008). Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Transactions on Algorithms (TALG), 5(1), 9.

[16] Gangloff, S., & Sablik, M. (2017). Quantified block gluing, aperiodicity and entropy of multidimensional SFT. arXiv preprint arXiv:1706.01627.

[17] Gangloff, S., & Talon, A. (2019). Asymptotic growth rate of square grids dominating sets: a symbolic dynamics approach. arXiv preprint arXiv:1906.10779.

[18] Golomb, S. W. (1966). Tiling with polyominoes. Journal of Combinatorial Theory, 1(2), 280-296.

[19] Golomb, S. W. (1970). Tiling with sets of polyominoes. Journal of Combinatorial Theory, 9(1), 60-71.

[20] Golomb, S. W. (1989). Polyominoes which tile rectangles. Journal of Combinatorial Theory, Series A, 51(1), 117-124.

[21] Gonçalves, D., Pinlou, A., Rao, M., & Thomassé, S. (2011). The domination number of grids. SIAM Journal on Discrete Mathematics, 25(3), 1443-1453.

[22] Gonthier, G. (2008). Formal proof–the four-color theorem. Notices of the AMS, 55(11), 1382-1393.

[23] Gravier, S. (2002). Total domination number of grid graphs. Discrete Applied Mathematics, 121(1-3), 119-128.

[24] Guichard, D. R. (2004). A lower bound for the domination number of complete grid graphs. Journal of Combinatorial Mathematics and Combinatorial Computing, 49, 215-220.

[25] Klavžar, S., & Seifter, N. (1995). Dominating Cartesian products of cycles. Discrete Applied Mathematics, 59(2), 129-136.

[26] Hales, T. C., Lagarias, J. C., & Ferguson, S. P. (2011). The Kepler Conjecture: The Hales-Ferguson Proof. Spinger.

[27] Hales, T., Adams, M., Bauer, G., Dang, T. D., Harrison, J., Le Truong, H., ... & Nguyen, Q. T. (2017). A formal proof of the Kepler conjecture. Forum of Mathematics, Pi, 5.

[28] Heawood, P. J. (1949). Map-Colour Theorem. Proceedings of the London Mathematical Society, 2(1), 161-175.

[29] Heinrich, I., & Tittmann, P. (2018). Neighborhood and Domination Polynomials of Graphs. Graphs and Combinatorics, 34(6), 1203-1216.

[30] Henning, M. A. (2009). A survey of selected recent results on total domination in graphs. Discrete Mathematics, 309(1), 32-63.

[31] M. Hochman and T. Meyerovitch Hochman, M., & Meyerovitch, T. (2010). A characterization of the entropies of multidimensional shifts of finite type. Annals of Mathematics, 171(3), 2011-2038.

[32] Kempe, A. B. (1879). On the geographical problem of the four colours. American journal of mathematics, 2(3), 193-200.

[33] Lind, D., Marcus, B., Douglas, L., & Brian, M. (1995). An introduction to symbolic dynamics and coding. Cambridge university press.

[34] Lu, Y., & Xu, J. M. (2012). The 2-domination and 2-bondage numbers of grid graphs. arXiv preprint arXiv:1204.4514.

[35] Nitica, V. (2018). Revisiting a Tiling Hierarchy. IEEE Transactions on Information Theory, 64(4), 3162-3169.

[36] Nitica, V. (2018). Revisiting a Tiling Hierarchy (II). Open Journal of Discrete Mathematics, 8(02), 48.

[37] Ollinger, N. (2009). Tiling the Plane with a Fixed Number of Polyominoes. Language and Automata Theory and Applications, 638-647.

[38] Pavlič, P., & Žerovnik, J. (2012). Roman domination number of the Cartesian products of paths and cycles. the electronic journal of combinatorics, P19-P19.

[39] Pavlov, R. (2012). Approximating the hard square entropy constant with probabilistic methods. The Annals of Probability, 40(6), 2362-2399.

[40] Pavlov, R., & Schraudner, M. (2015). Entropies realizable by block gluing $\mathbb{Z}^d$ shifts of finite type. Journal d'Analyse Mathématique, 126(1), 113-174.

[41] Rao, M. (2017). Exhaustive search of convex pentagons which tile the plane. arXiv preprint arXiv:1708.00274.

[42] Rao, M. & Talon, A., (2019). The 2-domination and Roman domination numbers of grid graphs. Discrete Mathematics & Theoretical Computer Science, 21.

[43] Rauzy, G. (1982). Suites à termes dans un alphabet fini. Séminaire de théorie des nombres de Bordeaux, 1-16.

[44] Reid, M. (1997). Tiling rectangles and half strips with congruent polyominoes. Journal of Combinatorial Theory Series A, 80, 106-123.

[45] Robertson, N., Sanders, D., Seymour, P., & Thomas, R. (1997). The four-colour theorem. Journal of combinatorial theory, Series B, 70(1), 2-44.

[46] Robertson, N., Sanders, D. P., Seymour, P., & Thomas, R. (2014). Discharging cartwheels. arXiv preprint arXiv:1401.6485.

[47] Shaheen, R., Mahfud, S., & Almanea, K. (2016). On the 2-domination number of complete grid graphs. Open Journal of Discrete Mathematics, 7(1), 32-50.

[48] Stewart, I. N., & Wormstein, A. (1992). Polyominoes of order 3 do not exist. Journal of Combinatorial Theory, Series A, 61(1), 130-136.