Université catholique de Louvain
Ecole polytechnique de Louvain
Institute of Information and Communication
Technologies, Electronics and
Applied Mathematics

Decision making in large stochastic and adversarial environments

A complexity analysis of Policy Iteration

Romain Hollanders

Thesis submitted in partial fulfillment of the requirements
for the Ph.D. Degree in Engineering Sciences

Thesis Committee

Advisors:	Raphaël Jungers, UCL
	Jean-Charles Delvenne, UCL
Jury:	Philippe Chevalier, UCL
	Thomas D. Hansen, Aarhus University, Denmark
	Bruno Scherrer, Centre de recherche INRIA & Université de Lorraine, France
Chairman:	Philippe Lefèvre, UCL

Louvain-la-Neuve, December 7, 2015.

Qu'il est important pour réaliser un grand projet d'être

formé, guidé, soutenu, entouré.

Merci intensément à vous tous qui m'avez accompagné tout au long de mon doctorat.

Romain

Abstract

How to make the best decision in a complex environment is a question that has haunted many generations of researchers and practitioners. It has given rise to the field of Operations Research which is all about optimized decision making. If moreover, the environment in which decisions need to be made is stochastic, then one is probably trying to solve a Markov Decision Process. If above that, an adversary is to be taken into account, then we enter the framework of Two-Player Turn-Based Stochastic Games.

Solving these problems is of critical importance in a huge variety of domains. With the constant growth in problem sizes, efficiency is a main focus. One of the best practical algorithms out there to solve these problems is Policy Iteration. However, the analysis of its performance is admittedly a complex task which is the one we undertake in this thesis. We take as starting point a recent breakthrough from Fearnley showing that Policy Iteration may require an exponential number of steps for two of the three classical objective functions. Despite this result, the gap between upper and lower bounds on the complexity of Policy Iteration is still huge and needs to be tightened.

We analyze Policy Iteration through the angle of Unique Sink Orientations, an abstract framework that generalizes Markov Decision Processes and Two-Player Turn-Based Stochastic Games but also Linear Programming for instance. In our tools, we also exploit the Order-Regularity structure, a new line of ideas that has not yet been exploited.

Our results include tighter bounds on the complexity of Policy Iteration, both from above and below, and they invalidate a conjectured upper bound related to the Fibonacci sequence. We also show the limits of the classical approaches to obtain new bounds. Finally, we extend Fearnley's result and show that Policy Iteration may exhibit exponential complexity for all three classical objective functions. Today with the recent results regarding its complexity, the full portrait of Policy Iteration is closer to completion than it ever was.

Dear reader,

In this thesis, you will find the results of five years of my life as a researcher. The two first chapters are introductory. The Prelude portrays (or should I say fictionalizes?) the main concepts of the thesis in a tutorial way. It is meant as an optional introduction for anyone with a minimal scientific knowledge. Chapter 1 is a more advanced summary of everything we know about the Policy Iteration algorithm. It also summarizes our contributions to the field.

The chapters that follow contain all the details about the main technical contributions of the thesis. Except for a few add-ons, Chapters 2, 3 and 4 are essentially drawn from Hollanders *et al.* (2012), Hollanders *et al.* (2015) and Gerencsér *et al.* (2015) respectively. The last chapter is novel, unpublished material. Finally, the conclusion brings a different perspective on our results. It narrates my journey all the way from February 2010 when I first started studying Policy Iteration in the context of my master's thesis.

A summary of the acronyms we use across the thesis is available at the end of the manuscript.

I wish you a pleasant reading.

Romain

Contents

Prelude

Jesse recently created the website of KeepEatCool, a small-budget company specialized in the creation of frozen meals—small but nevertheless with a cutting edge expertise.[1] Even though the company was freshly ranked by the New York Times as one of the top companies in its category, Jesse has been quite disappointed to see that his website was hardly visible on the web. For instance, on Google, when typing keywords such as "frozen meals", users barely end up on the website after the third page of search results. When Jesse went to see his boss with a suggestion to make things right, it did not go so well.

"No, I will not be paying for ads on Google," his boss said. *"Do you realize how much money it costs? And what about Bing users? Should I pay for them as well? Besides, everybody knows that no one ever clicks on these links!"*

"You are aware that visibility is essential if we want to attract new customers, right?", Jesse argued.

"Well, then, find another way. But I am not committing any money to web ads."

Another way? Easier said than done... Jesse knew how valuable it is to be visible on search engines. Despite this setback, he undertook some research, starting from the basics: *how does Google even rank the web pages?*

Of course, the answer was easily found: *"it is a secret"*. Yet, even so, it quickly appeared to Jesse that one of the central criteria that search engines like Google use in their recipe is *PageRank*. The idea is quite simple: imagine a user randomly clicking on links as they come, a very large—if not infinite—number of times, starting from a random website. This user—that is referred to as the *agent*—will thereby perform what we call a *random walk* on the web. The frequency at which the agent visits each web page is defined as its PageRank. The higher, the better in Google's ranking.

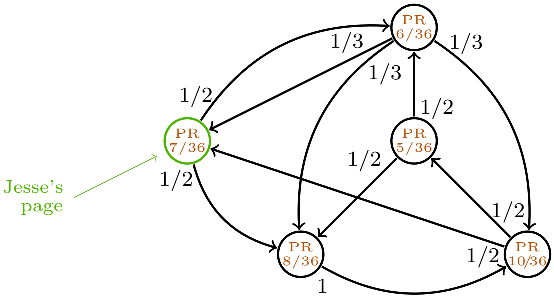
In fact, some additional tricks are needed for the above methodology

[1] Any resemblance to actual persons or companies is purely coincidental.

to work. For instance to make sure that the visited pages are focused on the desired keywords, it appears that Google first restricts the search of the agent to some predefined set of pages that we will call the *focused network*. Only links pointing inside this network are considered. Furthermore, to avoid a situation where the agent would be stuck on some web page(s) with no links to the outside world, the agent will restart the random walk from time to time from a random page.

Markov chains

The random walk originating from the computation of PageRank is usually described mathematically as a Markov chain. We represent each website as a *state*. In each state, we add a *directed edge* for each link it possesses, leading to the target state of the link. We thereby create a network of states that corresponds to the fraction of the web we are interested in. A random walk describes the process of the agent visiting states by following the edges *randomly*, starting from some initial state. “Randomly” here means that each possible link is chosen with some probability—by default, the same probability for each possible choice. These *transition probabilities* can be summarized in a matrix P where P_{ij} is the probability of jumping to state j when being in state i. It is then possible to compute the desired frequency of visit of each state after an infinite number of steps, which we call their PageRank.



A random walk on the focused network

Mathematically, we may represent the initial state as a probability distribution vector $u = \frac{1}{n}\mathbb{1}$ that is uniform over all states (here $\mathbb{1}$ is a vector of ones). Then, after one step of the agent in the Markov chain, the new probability distribution over the states become $P^\top u$. Applying $P^\top$ repeatedly, the vector eventually converges to the stationary distribution μ. To compute μ, we may observe that once we converged, applying $P^\top$ once more to μ should not change anything and therefore,

▶

Markov chains (continued)

we must have $P^\top\mu = \mu$. Additionally, the elements of μ are frequencies and should sum to 1, so $\mathbb{1}^\top\mu = 1$. Solving this system of equations yields the PageRank vector we aim for. A solution always exists provided the agent cannot get stuck (there are solutions to ensure that).

Many other situations can be efficiently modeled by a Markov chain, provided they can be represented through a set of states with transitions between them, where the next state only depends on the current one. Transition costs can be added as well if relevant to the situation, as we will see in the next box.

After the first period of excitement had passed, Jesse realized that exploiting the concept of PageRank to increase the visibility of his website was not as easy as he first thought it would be. But he would not give up and organized a meeting with his good friend Leonhard, a mathematician.

"I first realized that adding links towards my own web page would surely increase its PageRank," Jesse said when Leonhard asked him what he had tried so far. *"I therefore asked a few friends to add a link towards me on their blog and so on."*

"And did it work?"

"A little. But not well enough. I am still not on the first page of Google's search results."

"OK, let's recap then. You say that the higher the frequency at which a random walker visits your website, the higher its PageRank, right?"

"Absolutely."

"So it means that the average time between two visits of your website by the agent should be as low as possible, correct?"

"Oh! You mean that to increase my PageRank, I should reduce the average time-length of the path between me and myself? My average return time *in a sense..."*

"Exactly! Although I believe that this also goes through reducing the average time between a visit of any other page and a visit of yours."

This simple fact came as a revelation to Jesse. His average return time was something he could apprehend: look at every possible path from his page to itself that the random walker might choose and weigh their time-length by the probability that they are actually chosen. The total is then the desired average return time. Then he should exploit this fact by... But Leonhard interrupted his thoughts.

"This is all very nice, but I am not sure yet about how you could actually

influence your average return time. First, you would need to identify the focused network of KeepEatCool, which seems quite tough, yet not impossible I guess. But then, you would need to somehow modify the network. How do you plan to do that?"

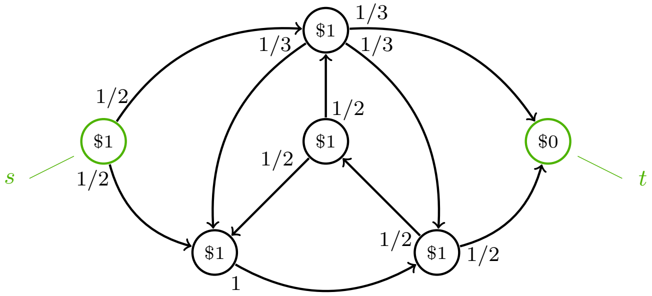
"I have a number of friends in my community. I am sure that they would agree to add some new links on their websites if I asked them to. Several of them already agreed to add links towards our website so why not a few more? If I manage to gather enough of these potential *links, I should be able to create shortcuts for the random walker and thereby reduce my average return time, don't you think?"*

"I can't see why not! Now the trick will probably be to choose which subset of these potential links to activate in order to maximize your PageRank."

The network of contacts Jesse has maintained over the years would now become a great asset!

Markov Decision Processes

The above problem of finding Jesse's average return time for the random walker can be represented with a Markov chain. First we reuse the same model as described in the previous box and we split the state corresponding to Jesse's page into two states: the starting state s and the target state t. To s, we give all the out-edges and to t, all the in-edges (hence t is an absorbing state). Additionally, since we want to count the number of transitions before reaching t, we now append a cost of 1 for each visit of a state, representing the time spent there, except for t which is cost-free. We may assume that every path will eventually end up in t in a finite number of steps, possibly with the use of some tricks as mentioned above. Then, the average return time is given by the sum, over all possible paths, of their time-length weighted by their probability of happening (which is simply given by the product of the probabilities of each transition that yield this path).



▶

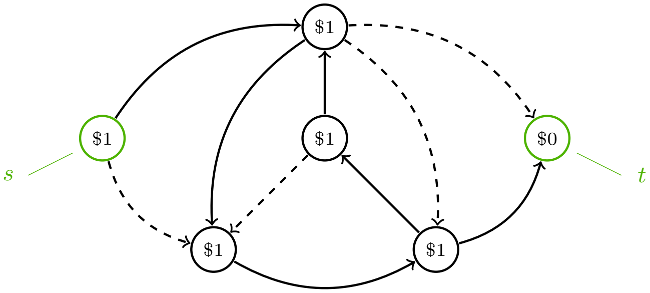
Markov Decision Processes (continued)

The above computation can be formulated mathematically. Let P be the transition matrix of the Markov chain (with all rows summing to 1 except for the row corresponding to t that sums to 0) and let c be its cost vector (where $c_i = 1$ for all states i except for state t where $c_t = 0$). We can compute the average time needed to reach t starting from any of the states by:

$$x = \sum_{k=0}^{\infty} P^k c = (I - P)^{-1} c,$$

where I is the identity matrix. In this expression, x_i corresponds to the average time needed to reach t starting from i, and thus x_s is the average return time we are looking for. The vector x is often referred to as the *value* vector.

In the situation discussed by Jesse and Leonhard, an additional element appears: the potential links. If there is a potential link in a state i, we have the choice between two sets of transition probabilities, depending on whether we activate the link or not. We call these choices *actions* and the choice of one or the other locally modifies the dynamics of the random walker. Now if there were, say, k potential links in a single state, there would be 2^k possible actions—one for each set of activation decision (e.g., activate all, activate none, activate the first one but not the others, etc.)—and this is annoying. However, it is possible to reformulate the problem in an equivalent way where we add a few auxiliary states such that each state has the choice between at most two actions, thereby reducing the complexity of the problem (Csáji *et al.*, 2014). Of course, the final goal is to find the set of actions that yields the best value for our objective function—here the average return time. The framework we just described is that of *Markov Decision Processes.*



Markov Decision Processes (continued)

Markov Decision Processes can model many situations that can be formulated with states and actions, including applications where a more general framework than in Jesse and Leonhard's problem is required. Indeed, in general, an action of a state may hide any transition probability distribution and any cost to be paid for visiting the state. The objective function to optimize may also vary depending on the situation. In our case, we chose to minimize the sum of costs for an infinite number of steps of the random walker (which is usually referred to as the *total-cost criterion*), and it was fine to do so because we were sure to eventually reach the absorbing state t. However, in general, without the right circumstances or without imposing an arbitrary number of steps (a *horizon*), this way of summing the costs may not yield a finite value. Therefore, in some cases, and depending on the problem studied, it may be useful to consider other objective functions. Among them, the two most widespread are the *discounted-cost criterion* where future costs become less important (they are multiplied by a *discount factor* less than one to allow the infinite sum to converge), or the *average-cost criterion* where only the long-run average cost of each step matters.

In practice, Markov Decision Processes are a very powerful tool to model decision problems in stochastic environments with many applications for instance in engineering. See for instance Puterman (1994) for a trusted and broad reference on the topic.

Gathering and negotiating all the potential links took some time but eventually, Jesse managed to collect about a hundred links that he would be allowed to activate or deactivate at will—all located on websites of the focused network of his company (figuring out this network did require some advanced investigations). But he would soon realize that extracting the *solution* from this mess of data was not as easy as anticipated.

"I need an algorithm but I can't figure it out", Jesse said when he met again with Leonhard.

"I'll do my best to help then. First, tell me what your solution should look like."

"This part is quite clear: I must decide for each potential link whether I should activate it or not. I naively thought that I could simply check each possibility separately, but there are billions of them, each inducing a different network for the random walk."

"But once a candidate solution is chosen, it is rather easy to evaluate *it—I mean to compute the PageRank you would get with it—correct?"*

"Yes, my computer does that reasonably fast. But as you mentioned last time, what I would actually compute is neither only nor exactly my PageRank, but rather the average time-distance between all the pages of the focused network (including mine) and my own website."

"Good to know. Now, you say that there are tons of possibilities but some of them are not so different from each other, right? For instance, is it hard to compare two candidate solutions that are identical except for one potential link?"

*"No, you're right. Once I evaluated one of them, I simply need to check whether switching (I mean, changing the activation state of) the potential link at which they differ creates a "*probabilistic shortcut*" for the random walker or not. If it does, then switching it will improve my PageRank. And this is indeed almost free to check computationally."*

"That is interesting! What do you think about the following then: you first take any candidate solution, regardless of how good it is, and evaluate it. Then, for every potential link independently, you determine whether it is better to switch it or not by checking if it creates a probabilistic shortcut. And then you do all the good switches at once and hopefully, you should obtain a much better candidate solution now!"

"Yes it might work! And then we should iterate this procedure, for instance until there is no link to switch anymore."

"Or until your PageRank decreases compared to the previous iterate, if it can even happen. We must check how well it works!"

"Wait! We need to give a name to our algorithm."

"What do you think about... PageRank Iteration?"

What Jesse and Leonhard did not yet know at the time was that PageRank Iteration would always terminate with the optimal solution. And quite fast at that too. In fact, the same algorithm was already long known for a more general framework.

Policy Iteration

PageRank Iteration is more widely known as *Policy Iteration* and is, in practice, one of the fastest known algorithms to solve Markov Decision Processes. The word *policy* refers to what Jesse and Leonhard call a "candidate solution": It is a choice of one action in each state. Selecting a policy fixes a Markov chain, so, a possible network for the random walker. Once a policy has been chosen, we may evaluate it to obtain the corresponding value of the objective function—here the average time to get from any state to the target state t. The final aim is the optimal policy that yields the lowest value, regardless of the starting state, and

Policy Iteration (continued)

therefore corresponds to the most favorable dynamics.

The way Policy Iteration works is similar to what Jesse and Leonhard describe. It first starts from a policy, say π, and using the chosen objective function, it evaluates the value x^π of the corresponding Markov chain. Now, let us consider an action a that is available in state i but that is not chosen by π. Suppose that choosing a makes us pay c^a and implies a vector p^a of transition probabilities. Then it is easy to determine whether changing the action of π in state i to a would improve its value (that is, decrease all the entries of x^π): simply ask whether $c^a + p^a \cdot x^\pi < x^\pi_i$ (this formula may vary depending on the objective function). If it is the case, then a is what we call an *improving action* over π, or a shortcut in some sense. Then, the way Policy Iteration works is by identifying all the improving actions over π and by switching them all together to obtain a new policy—the next iterate (if several improving actions exist in a state, it chooses the one that creates the "best shortcut"). This procedure is then repeated until reaching a policy where there are no more improving actions.

Regardless of its speed, there are some nice guarantees backing up Policy Iteration. First, it always considers strictly better policies from one iteration to the next and therefore, it always terminates in a finite number of steps (since there are only a finite number of them—even though a possibly large one). Moreover, it always finds a globally optimal policy.

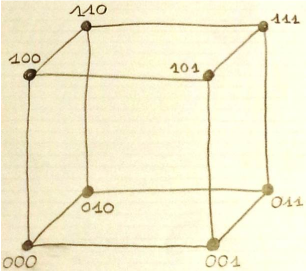
Promises of growth have been held for KeepEatCool which has become the leading group in its field. Jesse has been promoted and is now leading the IT department. A growth in size also brought a handful of new partners from which to acquire potential links. Unfortunately, PageRank Iteration's performance gradually started to drop in terms of computation time. Jesse identified that the number of iterations needed by the algorithm was to be blamed, probably due to the increasing number of potential links to handle. After trying a number of fixes, Jesse decided to call his good friend Leonhard again.

"PageRank Iteration has always worked great to optimize our PageRank", he explained. *"Until lately. I have tried a number of other algorithms and none of them ever surpassed ours. Nevertheless, the situation as it is now is not satisfactory. We are forced to use suboptimal solutions with no guarantee of quality and we recently lost our first position in Google's ranking. Do you have any idea how we could improve our algorithm?"*

“I think that what you need is to analyze the complexity *of PageRank Iteration. I mean, how fast does the execution time increase when you add new potential links to your problem? If you better understood when and why the algorithm requires more steps, then maybe you could figure out the right fix.”*

“I see. But how can we identify this complexity?”

“Well, first, you need to properly identify the structure of your problem. You agree that if you have 3 *potential links for instance, then you have* 2^3 *candidate solutions, right? And if you give a label* 1 *to* 3 *to each potential link, then each solution can be represented as a binary vector with* 3 *entries: assign* 1 *to the link* i *if it is activated,* 0 *otherwise. You probably see that these* 2^3 *vectors correspond to the vertices of a (hyper-)cube. Let me sketch this for you.”*



“Then what does PageRank Iteration do?”, Leonhard continued. *“Well, it starts from an initial vertex—a candidate solution—and then jumps from vertex to vertex on this cube. What we must understand is* how *it makes the jumps with the way we update the solutions at each step of the algorithm. Maybe the cube contains some special structure that could help us as well.”*

“Wow, this sounds like really advanced stuff!”

“It sure is.”

Back home, Jesse immediately set to work. A few days later, Leonhard received an unexpected visit.

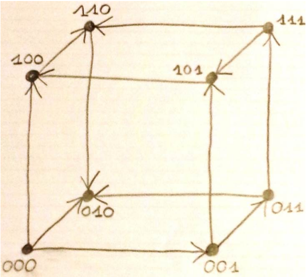
“I think I got it! The structure!”, Jesse said with an unusually thrilled tone. *“Do you have a minute?”*

“Sure! Come in. Tell me everything.”

“OK. First, you remember the trick we used earlier when we observed that once a candidate solution was evaluated, then it is easy and always possible to decide whether switching a potential link is a good thing or not, right?”

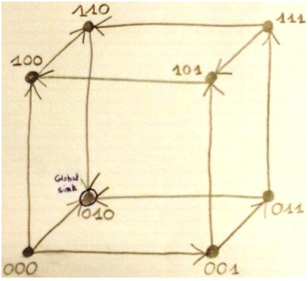
“Yes.”

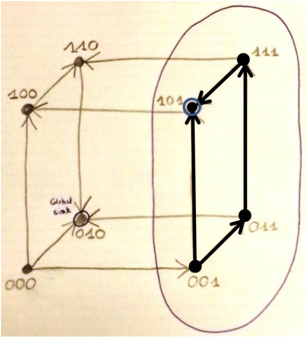
"Well, the candidate solution and the solution you obtain by switching a single link are exactly neighbors in the cube you drew the other day." Jesse said while showing the sketch in question. *"This basically means that you can always give an orientation to the edges, pointing toward the better solution of the two. And you do that for every edge, like this for instance."*



"Indeed. And what structure does it reveal?"

"It took me a while to figure it out but I think I got it. The first key observation is that the best solution we are looking for should be unique and it has only incoming edges in the cube, so it is some kind of sink. *Now, assume I had one less potential link to deal with. I would end up with a sub-cube of my original cube: with one less dimension but the same orientations for the remaining edges. This sub-cube would represent my problem with one less potential link and in this respect, it would also have a unique optimal solution, that is, a unique sink of its own. Now if you repeat the argument recursively, this means that every sub-cube of my original cube must have a unique sink, which looks like a strong characteristic. Don't you think?"*



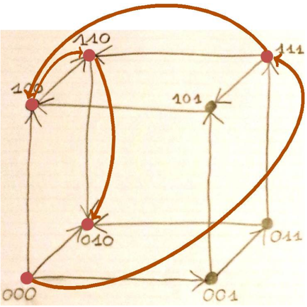


"Yes, I can believe that it must quite significantly reduce the number of oriented cubes that correspond to your PageRank problem."

“There is more. Since we know that PageRank Iteration never cycles, I assume that there must also be some kind of acyclicity in the cube.”

“I see... And what does this all mean for PageRank Iteration then?”

“Well, it is quite easy to picture how PageRank Iteration jumps from one candidate solution to another on the cube. Remember that, starting from some solution, it switches all the potential links that correspond to an improvement to that solution at once. This somehow means that, from the current vertex in the cube, we follow *all the outgoing edges at once to jump to the next iterate. On this picture for instance, starting from* 000*, this is how it would look like.”* Jesse picked his red pen.



“I chose an example where many steps are needed on purpose”, Jesse continued. *“I mean, here, the algorithm needs to explore five solutions before finding the best one, and there are only three potential links. Imagine what could happen with a thousand potential links. I believe this might explain the drop of performance.”*

“OK, but how bad can it really be? And how can we get around these bad cases? Can we somehow adapt the jumping rule *to ensure faster convergence?”*

“These are actually the central questions. I was not able to figure out their answers yet.”

“You know what? I am sure these questions would make a super-nice topic for a Ph.D. thesis!”

Acyclic Unique Sink Orientations

The way Jesse and Leonhard analyze the complexity of their algorithm using a (hyper-)cube can also be used to analyze Policy Iteration for solving a Markov Decision Process. The dimension of the cube corresponds to the number of states of the process and its vertices represent

►

Acyclic Unique Sink Orientations (continued)

the possible policies. Two policies that differ in only one action are *neighbors* in the cube and are connected by an edge. (Note that if there are states with more than two available actions, we would have a grid instead of a cube, but we can deal with them as well.) As mentioned in the previous box, it is always possible to determine which of two neighboring policies is best, therefore all the edges of the cube receive an orientation accordingly.

As Jesse and Leonhard rightfully observed, the resulting oriented cube has a strong structure: it is acyclic and every sub-cube has a unique sink. We call such a structure an *Acyclic Unique Sink Orientation.* One of the central goals of this thesis is to extract as much information as possible from this structure, and to use it to obtain new bounds on the number of steps of Policy Iteration.

Chapter 1

The epic story of Policy Iteration

In the prelude of this thesis, we had a first chance to grasp the idea of the Policy Iteration algorithm: how it works, how its complexity can be analyzed, how it generates fascinating research questions.

The aim of this introduction is to draw a complete portrait of Policy Iteration as we know it. To this end, we first describe the framework of Markov Decision Processes for which it was first designed. After exploring some links with Linear Programming and the Simplex algorithm, we describe Two-Player Turn-Based Stochastic Games for which Policy Iteration also applies. We then present Acyclic Unique Sink Orientations, a powerful framework to analyze the complexity of Policy Iteration as well as many other algorithms. Diving one step deeper in the analysis, we introduce the Order-Regularity condition that provides a tool specifically designed for the study of Policy Iteration.

During this chapter, we gradually introduce our different contributions to the field, in their context. Then in the last section, we detail the outline of the thesis and provide a summary of our results. The paragraphs with a "Good to know" label are of general interest but they are not required to understand the contributions of the thesis.

It should be noted that a reader familiar with the topic of this thesis should be able to read each chapter independently; each of them indeed includes all the necessary context and technical material, as well as reminders of previous chapters whenever necessary.

Solving Markov Decision Processes with Policy Iteration

Markov Decision Processes (MDPs) were introduced by Bellman (1957) to model sequential decision making in stochastic environments. MDPs proved a powerful modeling tool for the decision problems that arise daily in various domains of engineering such as control (Bertsekas, 2007), robotics (Mahadevan and Connell, 1992), maintenance (Dekker *et al.*, 2007), manufacturing (Chao, 2013), finance (Bäuerle and Rieder, 2011), communication networks (Altman, 2002), queuing systems (Chevalier and Wein, 1993; Meyn, 2008) or PageRank optimization as we saw in the Prelude of this thesis (Csáji *et al.*, 2014; Fercoq *et al.*, 2013). See White (1993) for a more complete list of successful applications.

> ***Good to know.*** *Markov Decision Processes are one of the successful fields of Operations Research. The name of the discipline originates from World War II when the necessity of improving the planning quality of military operations became obvious. Since then, it essentially refers to any field that has to do with optimal decision making. Today, Operations Research tools are widely used in almost all businesses and governments. They are still an active topic for research as well.*

Markov Decision Processes are described from the set of n states in which a system can be. When being in a state, the *controller* (or *decision maker*) of the system must chooses an available action in that state, each of which induces a reward and randomly moves the system to another state according to given transition probabilities. The process is repeated a possibly infinite number of times. The goal of the controller is then to choose the set of actions (or decisions) that rewards him with the highest "long term income", whose precise definition depends on the application. We then call this set of actions a *solution* to the MDP.

In this thesis, we focus on finite size MDPs (that is, with finite states and actions sets) with an infinite horizon and memoryless decisions, which is usually claimed to be the case where the theory is the most complete and elegant. With these hypotheses, an MDP always allows a solution where a single stationary action is selected in every state. Here, stationary means that the action does not change over time when we visit the same state again. Such a choice of one action in every state is what we call a (stationary deterministic) *policy* and we can restrict ourselves to this class of candidate solutions. Choosing a policy implies fixing a dynamics that corresponds to a Markov chain. To any policy, we can associate a *value vector* whose entries correspond to the "long term income" of an agent starting in each of the states of the MDP. (When following a policy, we refer to the *value of a state* to designate the corresponding entry of its value vector.) In fine, a solution of an MDP is a policy that maximizes the value of every state. We say that such a policy is *optimal*. Moreover, an MDP always allows an optimal policy. Depending on the application, we classically use a total-, discounted- or average-reward criterion to define the value function, that is, to define what we mean by the "long term income". Note that in all three cases, an optimal policy always exists. We give a more precise definition of MDPs and discuss the three classical reward criteria in Chapter 2, Section 2.1.1. For more, see, e.g., the excellent books of Puterman (1994) and Bertsekas (2007).

> ***Good to know.*** *Originating from their framework, MDPs have given rise to a number of specialized fields to capture always more elements from the problems encountered in applications. Some of the most important ones are Reinforcement Learning (Sutton and Barto, 1998) where the system is initially unknown to the controller, Partially Observable MDPs (Spaan, 2012) where the agent only has partial knowledge of his position in the system, Constrained MDPs (Altman, 1999; Chang, 2007) where the controller seeks to ensure a certain level of performance regarding some secondary variables or Continuous Time MDPs (Guo and Hernández-Lerma, 2009) which is suited for systems that naturally exhibit continuous time dynamics.*

One of the most practical ways to find an optimal policy for an MDP is to use the *Policy Iteration* algorithm (PI). Starting from an initial policy π_0, $i = 0$, this simple iterative scheme computes the value vector of π_i and, based on this evaluation, switches to a more "appealing" action in states where such an action exists to obtain the next iterate π_{i+1}. Here, an action in a state is called *appealing* if using this action once and then continuing with the current policy π_i improves the value of the state. It can be shown that the modification always ensures that the value vector of π_{i+1} improves on that of π_i in every entry. The process is then repeated until convergence to the optimal policy π^* for which there is no more appealing action. The same policy is never encountered twice and therefore, the process always terminates with the optimal policy in a finite number of steps (at most the total number of policies). See Chapter 2, Section 2.1.2 for a more precise definition.

Notice that there is some freedom in the choice of which appealing actions we choose to switch at every step. The most obvious and most studied choice is probably to switch to the most appealing action in every state. We usually refer to this version of PI as *Howard's* PI, by the name of its author (Howard, 1960). Unless stated otherwise, we will usually use the term PI to designate Howard's version which is our main focus in this thesis.

Every iteration of PI can be performed in polynomial time but the question of its number of iterations has remained a mystery for more than 30 years. The best known upper bound was due to Mansour and Singh (1999) with an $O\big(\frac{k^n}{n}\big)$ bound where n is the number of states of the MDP and k is the maximum number of actions per state. This bound may sound as a poor improvement over the trivial k^n bound that corresponds to the maximum number of policies. It is also disappointing given that in practice, PI mostly behaves as a polynomial time algorithm. Nevertheless, it holds for the three classical reward criteria and improving it appears to be a difficult challenge as we develop below.

It is only a few years ago that two major results regarding the complexity of PI were found, roughly at the same time, and significantly enriched the picture. On the positive side first, building on Meister and Holzbaur (1986) and Tseng (1990), PI was shown to run in strongly polynomial time in the important particular case of discounted-reward MDPs with a fixed discount rate (Ye, 2011). The bound in this result was later improved by Hansen *et al.* (2013) and Scherrer (2013). It was also adapted to fit other special cases including average-reward MDPs modeling replacement and maintenance problems (Feinberg and Huang, 2013), Mean-Payoff Games with bounded first return times (Akian and Gaubert, 2013) and deterministic MDPs (Post and Ye, 2015), although for the latter case, the strongly polynomial time bound only holds for a close variant of PI known as Simplex-PI. In a recent paper, Feinberg and Huang (2015) also provide a set of conditions under which a total- or average-reward MDP can be reduced to a discounted-reward MDP.

On the negative side, Fearnley (2010) showed that PI requires at least $\Omega(2^{n/7})$ steps to converge in the worst case for the total- and average-reward criteria. His result is principally based on the work of Friedmann (2009) on Parity Games. Forty years after the first exponential time proofs for pivoting rules of the Simplex method for Linear Programming (Klee and Minty, 1970), Fearnley's result revives the message that behaving like a polynomial time algorithm does not necessarily make you one. Besides, together with Friedmann's, his result did have a major impact on the study of the remaining polynomial time candidate pivoting rules for the Simplex method as well, as we mention below.

In Chapter 2, Section 2.2, we complete the above picture by showing that Fearnley's exponential complexity result extends to discounted-reward MDPs when the discount factor approaches 1 sufficiently fast. We achieve this using perturbation analysis.

Note that a similar analysis was performed by Andersson and Miltersen (2009) to show polynomial time equivalence between solving Discounted Games and Mean-Payoff Games that allow discounted-reward and average-reward MDPs as special cases. This result was however focused on the complexity of problems rather than a particular algorithm like PI. Moreover, it has no impact on MDPs since the latter problems are already known to be solved in (weakly) polynomial time using their connection with Linear Programming.

> ***Good to know.*** *Many algorithms exist to solve MDPs. A noticeable competitor to PI is the well known Value Iteration (VI). To understand how it works, assume we knew the optimal value vector x^k of an agent who has k steps left to perform in the MDP. Applying dynamic programming, we can use x^k to efficiently compute the optimal value x^{k+1} if there were $k+1$ remaining steps. After a sufficient number of steps of this procedure, starting from $k = 0$, we are able to extract the optimal policy from the obtained value vector. Allegedly, VI iterates on values rather than policies, hence the name. The advantage is for cheaper steps than PI but with the drawback of a slower convergence. Modified PI (MPI) provides a middle ground between the two algorithms by performing iterations both on policies and values (Puterman and Shin, 1978). Doing so, MPI allies cheaper iterations than PI with a faster convergence than VI. Note however that the strongly polynomial time result from Ye (2011) does not apply to VI and MPI (Feinberg and Huang, 2014; Feinberg* et al.*, 2014). See, e.g., Bertsekas (1996), Scherrer (2014) or Chang* et al. *(2005) for more variants of PI and VI suited for large-scale problems. In addition to the aforecited iterative methods, one can also formulate an MDP as a Linear Program and thereby, solve it in (weakly) polynomial time. This last idea performs however quite poorly in practice.*

The link with Linear Programming and the Simplex method

Interestingly, MDPs with the three classical reward criteria can be written as a Linear Program (LP) where the goal is to optimize a linear objective function while satisfying a set of linear constraints that define a polytope of feasible solutions. In that respect, MDPs are a particular case of LP. Using this connection, MDPs can be solved in *weakly polynomial time* using either the *Ellipsoid* method by Khachiyan (1980) or the Interior Point method by Karmarkar (1984), the latter being actually efficient in practice. (Other weakly polynomial time algorithms have been developed as well, see Bertsimas and Vempala (2004), Dunagan and Vempala (2008) or Kelner and Spielman (2006).) A weakly polynomial time algorithm takes into account the bit-size of the input values and not only the number of values. So, as the values become larger, the runtime scales polynomially with the bit size of the input values too. In the case of LP, this means that a weakly polynomial time algorithm depends not only in the number of variables and constraints but also in the bit-size of the coefficients of the problem. Conversely, the complexity of a *strongly polynomial time* algorithm does not depend on the bit-size of the input.

Today, LP and MDPs are some of the few problems that allow a weakly polynomial time algorithm but for which we do not know any strongly polynomial time one. A well known candidate for the job would be the Simplex method for LP proposed by Dantzig (1948). This iterative scheme evolves along the edges of the polytope representing the feasible solutions of the LP, repeatedly jumping from a vertex to its neighbor, while heading towards better objective values. The convergence of the algorithm, and especially the number of steps it takes, depends on a *pivoting rule*, that is, a rule to decide which improving neighbor to visit next when there are several of them. Since the introduction of the method, the quest for a pivoting rule that would ensure its (strongly) polynomial time convergence has received a lot of attention (Todd, 2002), and continues to do so. In the hope of achieving this goal, many pivoting rules have been proposed in the literature. Unfortunately, a series of negative results followed a result of Klee and Minty (1970) who showed that, despite its practical efficiency, Dantzig's original rule could lead to an exponential number of steps of the method in the worst case. Building on this result, many other pivoting rules suffered the same fate (see Amenta and Ziegler (1999), Avis and Chvátal (1978), Goldfarb and Sit (1979) or Jeroslow (1973)). Only a few—especially randomized rules—survived the attacks for many years. Until they were also recently shown to run in super-polynomial time by Friedmann (2011) and Friedmann *et al.* (2011). It should be noted that these breakthrough results originated from the results on PI by Friedmann (2009) for Parity Games and Fearnley (2010) for MDPs, as mentioned above. Despite the negative results, the Simplex method was named one of the top 10 most influential algorithms of the 20th century in a special issue of the renowned journal of Computing in Science & Engineering (Cipra, 2000). Moreover, the question of finding a strongly polynomial time algorithm to solve LP was included in Smale's list of great unsolved problems of the 21st century (Smale, 1998).

> ***Good to know.*** *The Simplex method jumps along the edges of a polytope—from neighbor to neighbor—until it reaches an optimal solution. It is of natural interest to investigate the minimum number of hops it can ever hope for in the worst case. A simple lower bound to this quantity is the* diameter *of the polytope, that is, the minimum number of hops between any pair of vertices. It is well known that the diameter of a polytope of dimension n with m facets can be at least $m - n$ and Hirsch (1957) conjectured that this bound was also an upper bound (See Dantzig, 1963). Hirsch's conjecture remained open for almost 50 years until it was recently disproved by Santos (2012) using a counter-example with $m = 86$, $n = 43$ and a diameter of 44. Santos thereby slightly*

improved the lower bound on the diameter, but the question of the upper bound remains still widely open and the true worst case behavior could range from linear to subexponential (see Todd (2014) for a recent improvement of the classical bound by Kalai and Kleitman (1992)). Conversely, the Hirsch conjecture is true for Policy Iteration. Indeed, it can be easily shown that an ideal switching scheme for PI would be able to reach an optimal policy in at most n steps.

Policy Iteration is in many aspects the pendant of the Simplex method for MDPs and this connection was of key importance to the results of Friedmann (2011) and Friedmann *et al.* (2011). The main difference between the two algorithms is that "pivoting rules" for PI are allowed to perform multiple pivot steps in a single iteration. It is therefore more powerful in the sense that it has more possibilities to allow a pivoting rule that would lead to a polynomial number of steps. This is not surprising given that MDPs are a particular case of LP. Below, we will compare more in details the intrinsic structure of both algorithms with the unified view of Unique Sink Orientations. But before that, let us mention another well-known model to which PI also applies.

Solving Turn-Based Stochastic Games with Policy Iteration

Another generalization of MDPs is its two-player variant known as *Two-Player Turn-Based Stochastic Games* (2TBSGs). The framework of Stochastic Games was introduced by Shapley (1953) a few years before MDPs were introduced by Bellman (1957). In 2TBSGs, we make the assumption of perfect information, which is equivalent to requiring that players do not play simultaneously. The resulting game can be interpreted as an MDP in which the states are distributed among two players with opposite objectives. Conversely, MDPs can be seen as a single-player variant of 2TBSGs. More precisely in 2TBSGs, the states and their corresponding actions are divided into two sets, one of which belongs to Player 1 who is the *minimizer* (that is, he tries to minimize the value function), the other one belonging to Player 2 who is the *maximizer*. Such a game can be seen as a *zero-sum* game in which Player 1 would pay the perceived rewards to Player 2. The notion of policy for MDPs naturally extends to 2TBSGs where it is usually called a *strategy* for the players. The progress of the game then happens as in an MDP, except that we can no longer talk about an optimal strategy for the players. Instead, to solve a 2TBSG, the goal is to find a (Nash) *equilibrium* strategy, that is, a strategy for which no player can benefit from deviating alone by switching some of his actions. Because the game is zero-sum, this equilibrium always exists and it corresponds to the minimax strategy.

Despite the paradigm shift, it is still possible to solve a 2TBSG in finite time using *Strategy Iteration* (SI), an algorithm inspired by PI (Hoffman and Karp, 1966; Rao *et al.*, 1973; Vöge and Jurdziński, 2000). The idea is the following. Let us fix an initial strategy. Then, without loss of generality, let us consider the game under the point of view of Player 2, the maximizer. If we applied PI while keeping the strategy of Player 1 fixed, we would obtain an *optimal counter-strategy*, or *best-response*, against Player 1. (Here notice that a best-response against the other player can easily be found using PI.) Instead, what we do is that after each PI step for Player 2, we change the strategy of Player 1 to his best-response against Player 2 and keep repeating this procedure. It is possible to show that the value of all the states must increase at each iteration. As for PI, we therefore never encounter the same strategy twice and the process terminates in finite time. The above process basically corresponds to two imbricated PI algorithms. As a result, any upper bound on the number of steps of PI easily extends to SI as well. This is why in this thesis, we really focus on the former case. We give a more precise definition of 2TBSGs and SI in Chapter 2, Section 2.1.3, and we exhibit the equivalence between SI and PI in Chapter 3, Section 3.5.

The status of 2TBSGs in many aspects resembles that of LP forty years ago. Unlike MDPs, we do not yet know of any polynomial time algorithms to solve them, not even weakly polynomial time ones, unless using a discounted-reward criterion with a fixed discount factor (Hansen *et al.*, 2013) or related special cases (Akian and Gaubert, 2013). Yet, PI and its variants, a bit like the Simplex for LP, could possibly lead to a strongly polynomial time algorithm. In the mean time, even exponential time algorithms, and bounds on their complexity, are of interest.

Good to know. *Since 2TBSGs are so hard to solve, several—hopefully simpler—particular cases that appear in applications have been identified over the years. We mention for instance, in decreasing order of generality,* Simple Stochastic Games *(Condon, 1992),* Mean-Payoff Games *(Ehrenfeucht and Mycielski, 1979) or* Parity Games *(Emerson and Jutla, 1991; Grädel* et al.*, 2002) that often appear in the literature. These problems are in* $\mathrm{NP} \cap \mathrm{coNP}$ *as well as in* $\mathrm{PLS} \cap \mathrm{PPAD}$*; two reasons to believe that they are unlikely to be* $\mathrm{NP}-\mathrm{complete}$ *(Condon, 1992; Daskalakis and Papadimitriou, 2011). However, even though they are particular cases of 2TBSGs and thus presumably simpler, designing a fully polynomial time algorithm to solve them is still a major open problem (Hansen, 2012; Jurdzinski* et al.*, 2008; Zwick and Paterson, 1996). Tellingly, recall that Parity Games are also the original framework for which SI was first proved to run in exponential time by Friedmann (2009).*

In order to design new switching rules for PI, SI and the Simplex method, it is crucial to understand why existing ones fail to provide the polynomial bounds we aim for. The answer is partly given by a recent line of work showing that problems related to the complexity of these unsuccessful algorithms are PSPACE*-complete (Adler* et al.*, 2014; Fearnley and Savani, 2015a,b). For instance, considering Howard's PI starting from a given policy, it is* PSPACE*-complete to decide whether the algorithm will ever switch an action in some state chosen in advance (Fearnley and Savani, 2015a).* PSPACE *is the class of problems that can be solved using a polynomial amount of space. The* PSPACE*-complete problems are the hardest among them because they can be used to solve any problem in* PSPACE*, which also includes* P *and* NP*. The* PSPACE*-completeness of the above-mentioned problems indicates that the corresponding switching rules are capable of solving any problem in* PSPACE *which is why they fail running in polynomial time. In that respect, unless* P = NP*, it is for instance pointless to design a switching rule if the problem of deciding whether or not there will be a switch in a given state is* PSPACE*-complete.*

Analyzing Policy Iteration with Acyclic Unique Sink Orientations

At each iteration, PI looks for the appealing actions that exist. Doing so, it actually compares the current policy π_i with its neighbors, that is, with the policies that differ from π_i in exactly one state. This comparison is done with respect to the value vectors of the policies. But vectors are not always *comparable*, that is, there is not always a clear entry-wise domination. Therefore, it is not always possible to claim that some policy is better than another because the policies are only *partially ordered*.

It is well-known that the policies of an MDP can be embedded in a (hyper-)grid where each state is assigned a dimension along which we organize the available actions. Each policy then corresponds to a vertex of the grid. Neighboring policies (which are in fact always comparable in the partial order) are then connected with an oriented edge, pointing towards the better policy of the two. This construction yields an oriented graph with the shape of a grid that has the particular properties of being *acyclic* and *unique sink* (see Figure 1.1 for an example). The unique sink property requires *every* subgrid to have a unique vertex with only incoming edges, that is, a unique *sink*. As a simple consequence of this condition, each subgrid must also have a unique *source*, that is, a unique vertex with only outgoing links. We call the resulting orientation an *Acyclic Unique Sink Orientation* of a grid (or Grid AUSO). Note that this grid reduces to a (hyper-)cube in the case of an MDP with two actions per state. We then talk about a *Cube AUSO*. We often restrict ourselves to this simpler—yet rich—case to simplify the analysis with a negligible loss of generality.

Finding the unique sink of a Grid AUSO that corresponds to an MDP also means finding the solution of that MDP. The problem of finding an optimal policy to an MDP can therefore be formulated in an abstract way as the problem of finding the sink of a Grid AUSOs. Moreover, using a similar construction, the problem of finding an equilibrium strategy for a 2TBSGs can also be formulated as such, see Chapter 3, Section 3.5.

The unique sink structure can also be found in other well known problems such as Linear Programming. Indeed, in the polytope defined by the constraints of the problem, if we orient every edge towards the most rewarding objective values, we ensure that every facet of the polytope has a unique sink, this time without the guarantee of acyclicity. Generally speaking, we call the result a Unique Sink Orientation (USO).

> ***Good to know.*** *USOs were first introduced by Stickney and Watson (1978) as digraph models for P-matrix Linear Complementarity Problems. They have then been disregarded for a number of years before they were reintroduced by Szabó and Welzl (2001) and Morris Jr (2002b).*
>
> *Interestingly, LP-type problems, introduced by Sharir and Welzl (1992) as a generalization of LP, offer a unified abstract framework for 2TBSGs, MDPs, LP and USOs. They can be solved in subexponential time using the Random-Facet algorithm from Matoušek* et al. *(1996). LP-type problems can be even further generalized into the framework of Violator Spaces introduced by Gärtner* et al. *(2008). Likewise, abstract objective functions introduced by Kalai (1997) particularize USOs while still generalizing LP and AUSOs. The main motivation behind these formulations is to provide a common framework and an alternative, abstract view for the design of new algorithms to solve the aforementioned problems.*

A major goal in the study of (A)USOs is to find a polynomial time algorithm to find the sink or to show that any algorithm requires an exponential number of queries. More precisely, such an algorithm should make no more than a polynomial number of vertex evaluations in the dimension and the number of facets of the (A)USO. By "vertex evaluation", we mean a request of the orientation of the edges adjacent to the vertex. (Note that Schurr and Szabó (2004) showed that any algorithm requires at least $\Omega\big(\frac{n^2}{\log n}\big)$ such evaluations.) In the case of LP, answering this question for Cube USOs would imply the first, long awaited, strongly polynomial time algorithm (this fact results from a technique of Gärtner and Schurr (2006) to cast any LP into a sink finding problem in a Cube USO). Regarding MDPs and 2TBSGs, it would be enough to find a polynomial time algorithm for Cube AUSOs to obtain the same consequence. Note that an MDP can always be formulated as an LP so it is not surprising that the problem at hand is simpler. It is more surprising for 2TBSGs though, as today no polynomial time algorithms are known to solve them in general.

Most algorithms to find the sink of a Cube or Grid (A)USO can be categorized along two axes:

- they can be either *deterministic* or *randomized* when choosing the next vertex to query;
- at two successive steps, they can perform *local hops* (like the Simplex method for LP, from neighbor to neighbor) or *multiple hops* in the cube.

A successful example of a multi-hop deterministic algorithm is the so-called *Fibonacci Seesaw* from Szabó and Welzl (2001) that applies to Cube USOs, and thus also to Cube AUSOs, and is guaranteed to converge in at most $O(1.61^n)$ steps, n being the dimension of the cube and 1.61 being a slightly smaller constant than the golden ratio. This is today the best known upper bound for deterministic (A)USO algorithms. On the other hand, *Random-Facet* for AUSOs from Gärtner (2002) is a local-hop randomized algorithm that is currently the only known method to solve AUSOs in (expected) subexponential time, namely in $e^{O(\sqrt{n})}$ steps. However, this bound was shown to be tight, up to a constant factor in the exponent, by Matoušek (1994) who obtained an $e^{\Omega(\sqrt{n})}$ lower bound. Apart from these two, many other methods have received a lot of attention, such as the *Product* Algorithm, *Random-Edge* or *Random-Jump* to mention only a few (Gärtner *et al.*, 1998; Hansen *et al.*, 2014; Mansour and Singh, 1999; Morris Jr, 2002b; Szabó and Welzl, 2001).

> ***Good to know.*** *Random-Facet was originally proposed as a pivoting rule for the Simplex method by Kalai (1992) and Matoušek* et al. *(1996). For LP as for AUSOs, it used to be the fastest known combinatorial algorithm with its expected subexponential worst-case running time. Interestingly, Random-Facet has recently been slightly improved by Hansen and Zwick (2015).*

Like Fibonacci Seesaw, Policy Iteration is a deterministic multi-hop algorithm specialized for AUSOs. Despite the negative complexity results cited earlier, it appears as an interesting competitor to the former algorithm. Indeed, as we will see below, we have reasons to believe that $F_{n+2}\big(= O(1.62^n)\big)$, the $(n+2)$nd Fibonacci number, is a possible upper

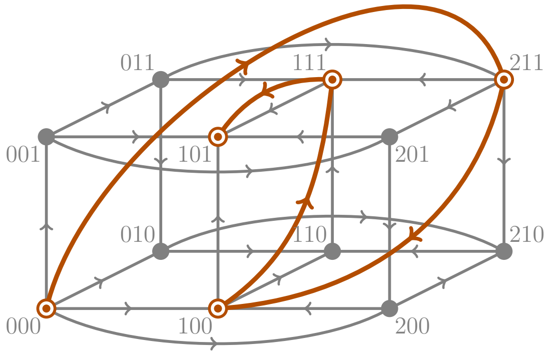


Figure 1.1: A possible run of PI on an example Grid AUSO, starting from the vertex/policy labeled 000. Here, 101 is the global sink that corresponds to the optimal policy. The PI jumps are represented in red.

bound on the number of steps of PI. This bound is quasi-identical to the one from Fibonacci Seesaw mentioned above. Moreover, there are many ways of modifying PI's update rule while maintaining the convergence guarantees. It is thus a flexible scheme.

PI's update rule can be interpreted in the grid as follows: let π be a vertex of an AUSO (typically the current iterate of PI) and let us consider the subgrid rooted in π and spanned by the outlinks of π. In the literature, π is sometimes referred to as the *bottom* or the *source* of this subgrid. It can be shown that there is always a directed path from π to any vertex μ in this subgrid. Intuitively, because of the acyclicity of the orientation, this means that any such vertex μ is "closer" to the sink than π. From π, choosing any vertex $\mu \neq \pi$ in the subgrid as the next iterate therefore provides an update rule that is guaranteed to converge to the global sink. PI's choice is to jump from π to an *antipodal* vertex μ in the subgrid, that is, a vertex that is as far away from π as possible (in a grid, there can be several). From there, it iterates until it finds the global sink, as illustrated in Figure 1.1. Because of its dynamics, Howard's PI is sometimes referred to as *Bottom-Antipodal*, *Jump* or *Switch-All*. Note that Random-Jump mentioned above is based on the same principle except that it chooses any vertex of the subgrid with uniform probability (Mansour and Singh, 1999).

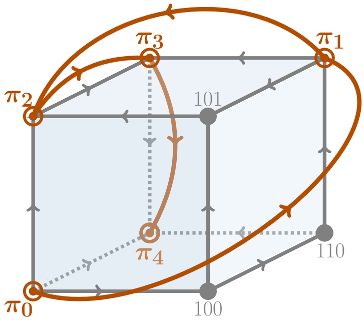
From the Grid AUSO structure, we can extract a number of useful properties to study the complexity of PI. For instance, we can show that every subsequent iterates must be connected by a path in the grid. Or we can use the fact that all vertices visited by PI must have a different *outmap*, that is, a different "set of outgoing edges" (indexed by directions). Using these properties, Mansour and Singh (1999) derived a $13 \cdot \frac{k^n}{n}$ upper bound on the number of steps of PI in the most general setting (that is, for the three classical reward criteria), where k is the maximum number of actions per state in the MDP.

In Chapter 3, we extend the set of properties to obtain a constant factor improvement over Mansour and Singh's bound, namely a $\frac{k}{k-1} \cdot \frac{k^n}{n} + o\left(\frac{k^n}{n}\right)$ upper bound. It is of natural interest to explore which of the properties we use are likely to be further exploitable to improve the bound and which ones are not. We show that all the properties we actually use are "fully exploited" and thus cannot lead to further improvements to the bound. Therefore, the bound is optimal for a relaxation of the complexity problem of PI considered by Mansour and Singh where it is only allowed to use some subset of properties. On the other hand, our bound remains far away from the state of the art $\Omega\big(\sqrt{2}^n\big)$ lower bound for PI on Cube AUSOs, due to Schurr and Szabó (2005). These last two observations indicate that some progress, either regarding the lower or the upper bounds, is still to be achieved, but that we therefore need to develop new—possibly more specific—properties on PI.

> ***Good to know.*** *When we decide which vertex to visit next in PI, choosing an antipodal vertex to the current vertex π is usually considered a fair choice because it agrees with all the outgoing—and thus improving—links at π. However, jumping directly to the local sink—also denoted the top—of the subgrid rooted in π would appear to be an even better idea. Schurr and Szabó (2005) called this fictional algorithm* Bottom-Top*. It is fictional because it assumes access to an oracle that reveals the local sink of the subgrid, which is not available in practice. Nonetheless, Schurr and Szabó showed the same $\sqrt{2}^n$ lower bound for Bottom-Top in Cube AUSOs as for Bottom-Antipodal (that is, Howard's PI). Legitimately, it may be asked whether a polynomial update rule for PI is possible, even theoretically. As it happens, it can easily be shown that from any vertex of an n-dimensional Cube AUSO, there always exists a directed path of length at most n leading to the global sink. Therefore, there must theoretically exist an ideal switching scheme ensuring no more than n iterations, although in that case, the design of an oracle would probably require a global knowledge of the AUSO.*

Analyzing Policy Iteration with Order-Regular matrices

Since improving the upper bound on PI requires exploring more properties specific to AUSOs, we investigate a new relaxation of the complexity problem of PI in a Cube AUSO known as the *Order-Regularity* problem (OR). First introduced by Hansen (2012), the idea of this formulation is



$$\begin{array}{c} \pi_0 : \\ \pi_1 : \\ \pi_2 : \\ \pi_3 : \\ \pi_4 : \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 1.2: The extremal example for Policy Iteration in a Cube AUSO of dimension 3. The PI jumps are represented in red and the corresponding OR matrix is represented on the right.

the following. Consider a sequence of vertices (or policies) $\pi_0, \pi_1, ..., \pi_{m-1}$ explored by PI in a Cube AUSO. Since we are in an n-dimensional cube, these vertices can be represented by n-dimensional binary (row) vectors. In the OR formulation, we record all these vectors into an $m \times n$ binary matrix so that each row corresponds to an iteration, as illustrated in Figure 1.2. We then translate the AUSO property into a combinatorial condition on this matrix: the OR condition, as defined in Section 4.1, Definition 4.1. We say that an AUSO *realizes* an OR matrix if the matrix can be obtained by a PI run on this AUSO. Note that any matrix realized by an AUSO satisfies the OR condition but it is not necessarily true that any OR matrix can be realized by some AUSO. The OR condition is therefore a relaxation of the AUSO structure.

Using the OR formulation, Hansen and Zwick performed an exhaustive search on all OR matrices with up to 6 columns and reported the maximum number of rows (that is, iterations for PI) each time: $2, 3, 5, 8, 13, 21$. Based on these empirical observations, they conjectured that the maximum number of rows of an OR matrix should follow the *Fibonacci sequence* (Hansen, 2012). Confirming this conjecture for $n = 7$ was a hard computational challenge. It was introduced as January 2014's *IBM Ponder This* Challenge. Proving the conjecture in general would provide an $O(1.618^n)$ upper bound on the number of iterations of PI, a quasi-identical bound as that of Fibonacci Seesaw. Regarding lower bounds, nothing better than the one from Schurr and Szabó for the AUSO framework was known prior to our work.

It should be noted that except for the above conjecture by Hansen and Zwick, the OR condition is still uncharted territory: we are not aware of any related work other than ours[1] and everything remains possible. Notably, the study of this condition has proved surprisingly challenging. On the other hand, it has led, in my eyes, to the most elegant contributions of this thesis. We now report our findings regarding the OR condition.

In Chapter 4, our first contribution is to disprove Hansen and Zwick's conjecture by performing an exhaustive search for $n = 7$. We obtained a maximum number of rows that is lower than the expected ninth Fibonacci number, which does not rule out the hope for the $O(1.618^n)$ bound to be a possible—loose—upper bound. Our second contribution is to (exponentially) improve Schurr and Szabó's $\Omega(1.414^n)$ lower bound to $\Omega(1.427^n)$, yet only in the framework of OR matrices. The key ideas behind both our results rely on the construction of large matrices satisfying OR-like conditions, which required substantial computational refinements.

Finding an AUSO that realizes a given OR matrix is not an obvious task to undertake. In Chapter 5, we push the analysis of the OR condition a step further and make the link back with AUSOs. More precisely, we identify the Odd-and-Even-Free Order-Regular (or OEF-OR) condition, a restriction of the OR condition that produces matrices allowing $\sqrt{2}^{n+2} - 1 = \Theta(1.414^n)$ as a tight upper bound. Although this bound does not improve our lower bound for OR matrices from Chapter 4, we show that it does extend to AUSOs where it improves Schurr and Szabó's $\sqrt{2}^n$ lower bound by a factor of 2. In this last chapter, we also develop techniques to recover an AUSO realizing a given OR matrix whenever possible.

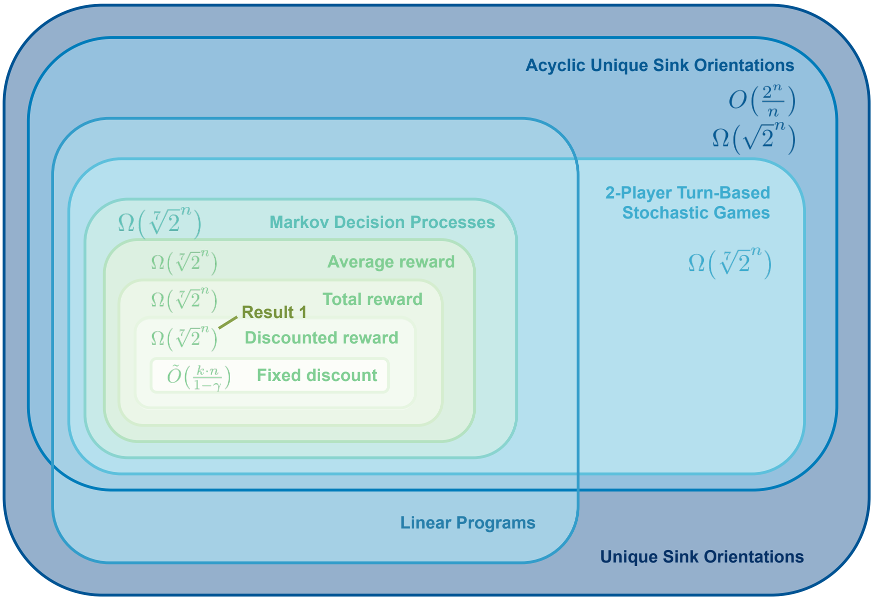
Now that we have a better idea of which OR matrices can be realized by an AUSO, an interesting perspective would be to investigate whether these matrices can also be realized by an MDP or a 2TBSG. In particular, we would like to know whether our extremal OEF-OR matrices of size $\Theta(1.414^n)$ are realizable in order to extend the corresponding lower bound on the complexity of PI to these frameworks as well.

Good to know. *It is of natural interest to ask which (A)USOs are realizable by an MDP, a 2TBSG or an LP. A first step in that direction was to identify the Holt-Klee condition which is a necessary condition for an (A)USO to be realizable by an LP or an MDP (Gärtner* et al.*, 2005; Holt and Klee, 1999). This condition requires that for any d-dimensional subcube of an AUSO, there exist exactly d edge-disjoint paths between the unique source and the unique sink of the subcube. This condition was used to show that some AUSOs cannot be realized by an LP or an MDP (Gärtner* et al.*, 2005). The starting point was a family of AUSOs defined by Matoušek (1994) on which he showed that the Random-Facet*

[1] We should mention that we had a number of private communications with Thomas Hansen and Uri Zwick who also worked on the topic on their own.

algorithm may require a subexponential number of steps to converge. Then, Gärtner (1998) showed a quadratic upper bound for Random-Facet on the specific instances of Matoušek's family that satisfy the Holt-Klee condition. Therefore, there must exist some AUSO of that family—for instance one on which Random-Facet behaves badly—that violates the Holt-Klee condition and that is thus neither realizable by an LP, nor by an MDP. However, the Holt-Klee condition is not sufficient either. Indeed, Morris Jr (2002a) identified a 4-dimensional AUSO that is not realizable by an LP but that still satisfies the Holt-Klee condition.

The figure below summarizes how the main problem classes that are mentionned in this introduction relate to each other and what are the state of the art bounds regarding PI's complexity for each problem class that can be solved using PI.


Acyclic Unique Sink Orientations
$O\left(\frac{2^n}{n}\right)$
$\Omega\left(\sqrt{2}^n\right)$
2-Player Turn-Based Stochastic Games
$\Omega\left(\sqrt[7]{2}^n\right)$
$\Omega\left(\sqrt[7]{2}^n\right)$
Markov Decision Processes
$\Omega\left(\sqrt[7]{2}^n\right)$
Average reward
$\Omega\left(\sqrt[7]{2}^n\right)$
Total reward
Result 1
$\Omega\left(\sqrt[7]{2}^n\right)$
Discounted reward
$\tilde{O}\left(\frac{k\cdot n}{1-\gamma}\right)$
Fixed discount
Linear Programs
Unique Sink Orientations

Outline of the thesis

The flow of the thesis goes from more applied and linear views to more abstract and combinatorial ones. The chapters are organized as follows.

- Chapter 2 is devoted to Markov Decision Processes and Two-Player Turn-Based Stochastic Games. We first provide a precise definition of the two frameworks and define Policy and Strategy Iteration to solve them. Then we detail how we extended Fearnley's example to discounted-reward MDPs (Result 1).

  Result 1 was published in the Proceedings of the 51st IEEE Conference on Decision and Control (CDC) (Hollanders *et al.*, 2012).

- In Chapter 3, we move to the Acyclic Unique Sink Orientations point of view. We make the link with MDPs and extract a number of properties from AUSOs about the partial order of the policies. We then improve Mansour and Singh's upper bound and show that our bound is tight for a relaxation of the complexity problem of PI (Result 2).

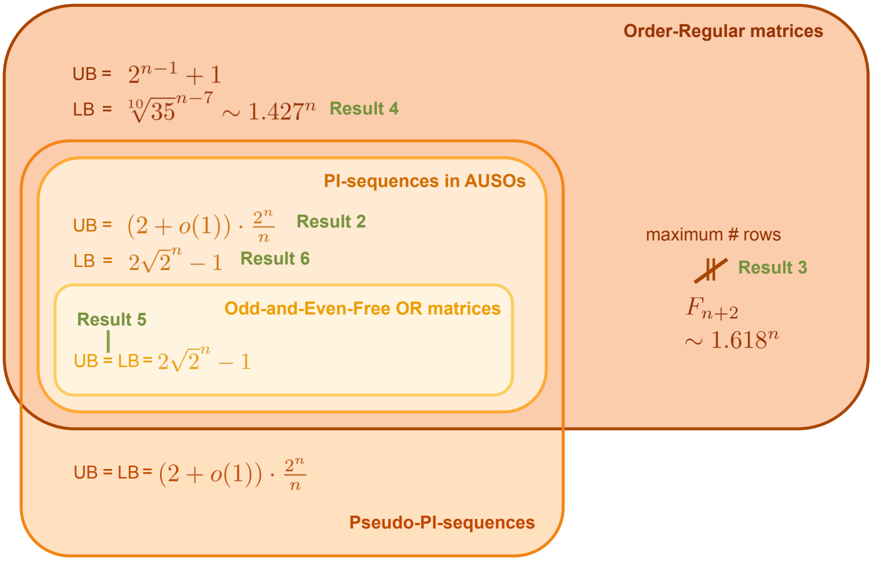
  Result 2 was accepted for pulication to Operations Research Letters (Hollanders *et al.*, 2015).

- In Chapter 4, we introduce Order-Regular matrices and the Fibonacci conjecture by Hansen and Zwick. We provide a computational refutation of this conjecture (Result 3). We then improve the state of the art lower bound on the size of OR matrices (Result 4). Both results rely on computational methods that we also describe in the chapter. We close by two approaches that we propose as perspective for further analysis.

  Results 3 and 4 were submitted to the Journal of Discrete Algorithms (Gerencsér *et al.*, 2015).

- Finally, Chapter 5 comes back to the AUSO framework. We first give a precise definition of Cube AUSOs. Then, we describe the Odd-and-Even-Free Order-Regular condition, a restriction of the OR condition for which we provide tight bounds (Result 5). We next show that any OEF-OR matrix can be realized by an AUSO and we provide an algorithm to recover this AUSO. With this result in hands, we extend the lower bound we had for OEF-OR matrices to AUSOs, thereby slightly improving the state of the art lower bound (Result 6). A general OR matrix can also often be realized by an AUSO. We end the chapter with a technique that recovers such an AUSO whenever possible and discuss its consequences (Result 7).

Each chapter comes with the necessary context, details and reminders to allow them being read independently of one another.


Order-Regular matrices
UB = $2^{n-1}+1$
LB = $\sqrt[10]{35}^{\,n-7} \sim 1.427^n$ Result 4
PI-sequences in AUSOs
UB = $(2+o(1)) \cdot \frac{2^n}{n}$ Result 2
LB = $2\sqrt{2}^n - 1$ Result 6
Odd-and-Even-Free OR matrices
Result 5
UB = LB = $2\sqrt{2}^n - 1$
maximum # rows
$\not\parallel$ Result 3
F_{n+2}
$\sim 1.618^n$
UB = LB = $(2+o(1)) \cdot \frac{2^n}{n}$
Pseudo-PI-sequences

Chapter 2

An exponential lower bound for Policy Iteration

Markov Decision Processes were introduced by Bellman (1957) to model sequential decision problems under uncertainty. Many variants of MDPs have been declined since then, with application in a wide range of areas, including Operations Research, Control, Economics, Finance and many more.

It is well known that an optimal solution to an MDP can be found in weakly polynomial time using Linear Programming. However, a more efficient way to solve these problems in practice is to use an appropriate iterative algorithm. Among them, *Policy Iteration* (PI) is one of the most renowned. It usually converges in a few iterations and is guaranteed to find the optimal solution in finite time. It can be viewed as a Simplex algorithm in which several pivoting steps are performed simultaneously.

The study of PI has been an active topic of research since its introduction by Howard (1960). However, until recently, its algorithmic complexity was not well understood. Things changed though with the recent negative result of Fearnley (2010) that showed that PI may require an exponential number of steps in the worst case for the total- and average-reward criteria. Moreover, on the positive side, another recent result of Ye (2011) showed that PI runs in strongly polynomial time in the case of discounted-reward MDPs where the discount rate is fixed to a constant. This last assumption is indeed often made in applications that make use of the discounted-reward criterion.

In this chapter, we extend Fearnley's result to the discounted-reward case when the discount rate is part of the input, showing that PI may also require an exponential number of steps in that case, provided that the discount factor approaches one sufficiently fast. Our result combined with the ones of Ye (2011) and Fearnley (2010) completes the characterization of the worst case complexity of PI for MDPs: it is strongly polynomial for discounted-reward MDPs with a fixed discount rate but exponential for total-reward, average-reward and discounted-reward MDPs in general.

The chapter is divided in two sections. In Section 2.1, we properly define MDPs and PI. We also define Two-Player Turn-Based Stochastic Games (2TBSGs), a natural two-player generalization of MDPs for which our result also applies. Then, in Section 2.2, we apply perturbation analysis to extend the result of Fearnley to general discounted-reward MDPs.

2.1 Markov Decision Processes, Stochastic Games and algorithms

Markov Decision Processes (MDPs) model intrinsically dynamic decision making in stochastic environments. Nevertheless, if we assume that the system evolves for an infinite amount of time and that decisions are made independently of the history of choices, MDPs can be formulated as static optimization problems. In this section, we start by precisely defining MDPs. We focus on the finite states and actions case with infinite horizon valuations which is the case for which the theory is the most elegant and complete. We then formulate *Policy Iteration* (PI), one of the most practical algorithms to solve MDPs, and the focus of this thesis. After that, we extend the MDP model to its natural two-player generalization known as *Two-Player Turn-Based Stochastic Games* (2TBSGs). In this setting, we formulate the *Strategy Iteration* algorithm (SI), the 2TBSG equivalent for PI. Since MDPs are a particular case of 2TBSGs, many of our results on PI can be extended to SI as well, including the lower bound from the next section.

2.1.1 Markov Decision Processes

Markov Decision Processes (MDPs) describe the discrete-time behavior of an *agent* evolving on a (finite) set of *states*. His evolution is determined by the decisions of the *controller* (or *decision maker*, or *player*) that chooses an *action* from some (finite) set of actions allowed in the current state. Each possible action is associated with some *transition probabilities* that determine the state to be visited in the next time-step and yields some *reward*. We assume that the controller has a perfect knowledge of the systems. Here is a more precise definition.

Definition 2.1 (Markov Decision Process)**.** A *Markov Decision Process* (MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{U}, \mathcal{P}, \mathcal{R})$ where

- $\mathcal{S} = \{1, ..., n\}$ is the finite set of *states*;

- $\mathcal{U}$ is the finite set of all *actions*, and $\mathcal{U}_s$ is the (non-empty) set of actions available to the agent in state s;
- $\mathcal{P} = \{ P^u_{s,s'} \mid s, s' \in \mathcal{S}, u \in \mathcal{U}_s \}$ is the set of *transition probabilities* that represents uncertainty. For any action $u \in \mathcal{U}_s$ available in s, $P^u_{s,s'}$ is the probability of going from state s to state s' when the action $u \in \mathcal{U}_s$ is chosen;
- $\mathcal{R} = \{ r^u_s \mid s \in \mathcal{S}, u \in \mathcal{U}_s \}$ is the set of *rewards* collected by the agent at any state s when using action $u \in \mathcal{U}_s$.

An example is given in Figure 2.1.

Using the control actions $u_0, u_1, ...$, suppose that the agent visits a sequence of states $s_0, s_1, ...$. Such a sequence is observed with probability $\prod_{t\geq 0} P^{u_t}_{s_t,s_{t+1}}$ and it offers the agent a reward sequence $\{ r^{u_t}_{s_t} \}_{t\geq 0}$. The expected reward sequence is obtained from the index-wise sum of all possible reward sequences weighted by their probability of occurrence. The goal of the agent is to maximize the "*utility*" of this expected reward sequence by choosing the right sequence of actions. (We will clarify below what we here mean by the "utility" of a sequence.)

As a matter of fact, the dynamic nature of the above decision problem makes it look challenging. However, if we assume that the agent evolves in this system for an infinite amount of time and that the actions are chosen independently of past decisions, there is no reason for the controller to choose a different action when visiting the same state. This leads to the idea of *policy* (or *strategy*).

Definition 2.2 (Policy)**.** We say that the agent follows a *deterministic* and *stationary* (or *positional*) *policy* $\pi : \mathcal{S} \to \mathcal{U}$ if every time he visits state $s \in \mathcal{S}$, he moves to the next state with respect to the same action $\pi(s) \in \mathcal{U}_s$. In this context, the dynamics of the agent comes back to that of Markov chain where we let P^π be the (row-stochastic) transition probability matrix and r^π be the reward vector corresponding to the policy π such that for any pair of states s and s', $P^\pi_{s,s'} = P^{\pi(s)}_{s,s'}$ and $r^\pi_s = r^{\pi(s)}_s$. An example policy is indicated in red in Figure 2.1.

Once a policy π is chosen, the probability of being in the different states after t time steps, starting from state s, is simply given by $(e^s)^\top (P^\pi)^t$, where e^s is the $s-th$ base vector ($e^s_{s'} = 1$ if $s = s'$, 0 otherwise). From there on, we can compute the expected reward sequence received by the agent, starting from s, as $\{ (e^s)^\top (P^\pi)^t r^\pi \}_{t\geq 0}$. The goal of the agent is to maximize the "utility" of this sequence, which we call the *value* (or *value vector*) x^π of the policy π. The entry x^π_s can be thought of as the overall gain of the agent if he follows policy π starting from state s. However, we cannot simply

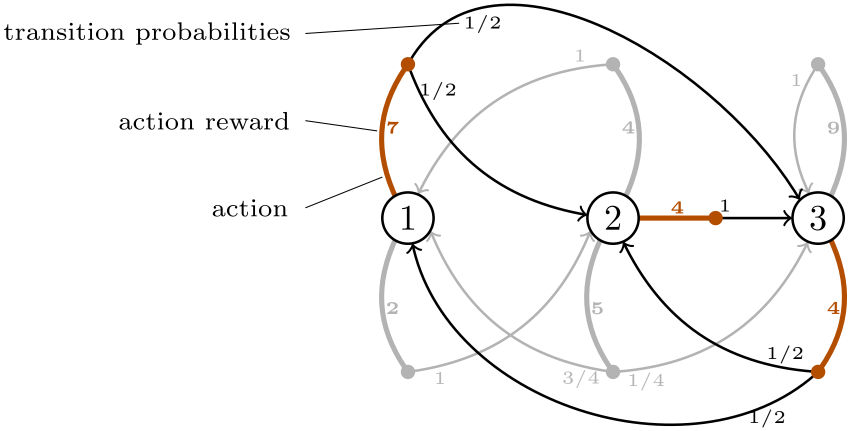


Figure 2.1: An example Markov Decision Process with three states. The thick edges represent the possible actions (the controller has the choice between two actions in states 1 and 3, and three actions in state 2). The red edges indicate the actions that are chosen by some policy. Removing the gray edges leaves exactly a Markov chain.

sum up the obtained rewards since this generally leads to a divergent series. Instead, we now present the three classical ways of defining the utility of an infinite sequence of rewards. Which criterion one should use really depends on the application.

Definition 2.3 (Discounted-reward)**.** In the *discounted-reward criterion*, we make a discounted sum of the rewards, that is, the reward perceived by the agent at step t is weighted by a factor γ^t where $0 < \gamma < 1$ is some fixed constant that we call the *discount factor*. Then, the value vector is given by

$$x^\pi = \sum_{t \geq 0} (\gamma P^\pi)^t \, r^\pi.$$

Since P^π is stochastic, the Perron-Frobenius theorem ensures that γP^π has a spectral radius lower than 1 and therefore that $(I - \gamma P^\pi)$ is non-singular and that $\sum_{t \geq 0} (\gamma P^\pi)^t = (I - \gamma P^\pi)^{-1}$, with I the identity matrix. As a result, x^π can also be computed as the unique solution of the following linear system:

$$(I - \gamma P^\pi) \, x^\pi = r^\pi. \tag{2.1}$$

The discount factor can be interpreted in two ways. It can be seen as a devaluation rate for future costs which are considered less valuable than immediate ones (because we do not entirely trust our model for instance). Or, $1 - \gamma$ can be seen as a probability of stopping the process at any time, hence γ^t is the probability that the process is still going on at time t.

Definition 2.4 (Total-reward)**.** In the *total-reward criterion*, we define x^π as the sum rewards perceived by the agent:

$$x^\pi = \sum_{t \geq 0} (P^\pi)^t \, r^\pi .$$

Since the above sum does not converge in general, we make the additional assumption that there exists some reward-free absorbing state τ such that $P^u_{\tau,\tau} = 1$ and $r^u_\tau = 0$ for all $u \in \mathcal{U}_\tau$. We then say that a policy π is *proper* if τ is reachable from any starting state, that is, if for each starting state s, there exists a sequence of states $\{s_0 = s, s_1, ..., s_\mathrm{T} = \tau\}$ such that $P^{\pi(s_t)}_{s_t, s_{t+1}} > 0$ for all $t = 0, 1, ..., \mathrm{T} - 1$, $\mathrm{T} < \infty$. Note that if such a sequence exists, then there exists a sequence of length at most $n + 1$. We usually assume that every policy is proper, but the problem can be solved under weaker assumptions too.

In the total-reward framework, we assume that τ is given implicitly in the formulation of the MDP. So, we do not consider it as part of the set $\mathcal{S}$ and we let P^π and r^π be respectively the transition probability matrix and the reward vector corresponding to policy π where we removed the rows and columns corresponding to τ. Thus, the matrix P^π is row-substochastic, that is, every entry is positive and every row sums to at most 1 with one of them that sums to less than 1. From there, similarly to the discounted-reward criterion, it is possible to show that if π is proper, then $(I - P^\pi)$ is non-singular (see, e.g., Bertsekas (2007, Volume 2, Proposition 2.2.1) for a proof). Therefore, we can compute x^π from the following linear system:

$$(I - P^\pi) \, x^\pi = r^\pi . \tag{2.2}$$

Note that, in a given state s, the probability of reaching τ in the next time-step can be interpreted as a probability of stopping the process. In that sense, in the discounted case, $1 - \gamma$ can be seen as a probability of visiting τ next in every state, while the other transition probabilities are being scaled by γ. This is a natural way of viewing the discounted-reward criterion as a particular case of the total-reward criterion.

We mention the following average-reward criterion for completeness, even though we do not specifically analyze it in this thesis.

Definition 2.5 (Average-reward)**.** In the *average-reward criterion*, we aim to maximize the long run average reward at each time-step, that is:

$$x^\pi = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} (P^\pi)^t\, r^\pi.$$

However, the value for a given starting state will be essentially determined by the final recurrent class of states it ends up in. Therefore, we could have many policies that share the same value, which may not be convenient. This is why we usually define an additional variable h^π to represent the *potential* of π for moving to better recurrence classes. Here h^π can be seen as the expected reward accumulated before reaching the limit cycle, in a similar fashion as with the total-reward criterion, with the absorbing reward-free state being the limit cycle in that case. Policies are then compared using x^π as a first criterion, then h^π in case of a tie. For more about the average-reward criterion, we refer to Puterman (1994).

Remark 2.1. It should be noted that the three above criteria are closely related. A discounted-reward MDP can always be formulated as a total-reward one by allowing every state to go to the absorbing state with probability γ. Moreover, a total-reward MDP can be solved using the average-reward criterion. This is because every starting state results in the same—reward-free—limit cycle and therefore, the value is entirely determined by the potentials, which in this case turn out to be equal to the total-reward values. Finally, a total-reward MDP can also be solved using a discounted-reward criterion if the chosen discount factor is close enough to 1. We will develop this last fact in Section 2.2 to show that Fearnley's example, which is defined for the total-reward criterion, also holds for discounted-reward MDPs. See Puterman (1994), in particular Section 10.4, for more about these links.

Regardless of the chosen criterion, the goal of the controller is to find an *optimal policy* π^* such that $x_s^{\pi^*} \geq x_s^\pi$ for all starting states $s \in \mathcal{S}$ and all policies π. By *solving* an MDP, we mean finding such an optimal policy.

The trick here is that, since x^π is a vector, it is not always possible to compare policies. This is why ordering the policies according to their value yields a partial order. (This partial order will play an important role in the analysis of the next chapters.) It is therefore unclear whether an optimal policy should always exist. Fortunately, the following classical result by Bellman (1957) saves the day. Proofs can be found, e.g., in Puterman (1994).

Theorem 2.1. *There always exists an optimal policy π^* for the total-, discounted- and average-reward criteria such that $x_s^{\pi^*} \geq x_s^\pi$ for all states $s \in \mathcal{S}$ and all policies π.*

We are now ready to formulate Policy Iteration, one of the most efficient schemes to solve MDPs and the main focus of this thesis.

2.1.2 Policy Iteration

The idea of *Policy Iteration* (PI) is to jump from one policy to the next, always going "forward" in the partial order of the policies, until converging to the global optimum. The new iterate π_{i+1} is obtained from the current policy π_i by "*switching*" in every state to some locally "*appealing*" action, based on the value of π_i. In this section, we formulate the *Policy Iteration* algorithm (PI) in a similar fashion as Fearnley (2010), to help relating the results from this chapter to his analysis. In the next chapter, we will provide an alternative version in terms of improving switches. The following definition formalizes the notions of switch and of appealing action.

Definition 2.6 (Appeal and improving switch)**.** For every state s, following Fearnley (2010), we define the *appeal* $a^{\pi}_{s\to u}$ of an action $u \in \mathcal{U}_s$ with respect to π as the value that the agent would gain if, starting from state s, he used action u instead of $\pi(s)$ once and then continued with the policy π. For the discounted-reward criterion, it is defined as:

$$a^{\pi}_{s\to u} \triangleq r^u_s + \sum_{s'\in\mathcal{S}} \gamma P^u_{s,s'}\, x^{\pi}_{s'}, \tag{2.3}$$

where x^{π} is solution of (2.1). Similarly for the total-reward criterion, it is defined as:

$$a^{\pi}_{s\to u} \triangleq r^u_s + \sum_{s'\in\mathcal{S}} P^u_{s,s'}\, x^{\pi}_{s'}, \tag{2.4}$$

where x^{π} is solution of (2.2). At some state s, an action $u \in \mathcal{U}_s$ is said to be *appealing* with respect to π if $a^{\pi}_{s\to u} > x^{\pi}_s$. Changing the action $\pi(s)$ to an appealing action u is called making an *improving switch* to π.

Interestingly, making any non-empty subset of improving switches to a policy π yields a strictly better policy π', that is, $x^{\pi'}_s \geq x^{\pi}_s$ for all states s, with a strict inequality for at least one state. Of course, we also have $a^{\pi}_{s\to\pi(s)} = x^{\pi}_s$. Moreover, if there is no improving switch to π, then π is optimal. We formalize these two facts in the following propositions. Proofs can be found, e.g., in Puterman (1994); see Theorems 7.2.15 and 6.2.1. Alternatively, we refer to Hansen (2012) for an analysis that is more in line with the present work; see Theorems 2.2.12, 2.3.9 and 2.4.6 for the discounted-, average- and total-reward respectively.

Algorithm 1: Howard's Policy Iteration

```
Input: An arbitrary policy $\pi_0$, $i = 0$.
Output: The optimal policy $\pi^*$.
while $\pi_i \neq \pi_{i-1}$ do
    $\pi_{i+1}(s) = \underset{u \in \mathcal{U}_s}{\operatorname{argmax}}\ a^{\pi_i}_{s \to u}$ for all states $s \in \mathcal{S}$.
    $i \leftarrow i + 1$.
return $\pi_i$.
```

Theorem 2.2. *Let π and π' be two policies such that $a^{\pi}_{s \to \pi'(s)} \geq x^{\pi}_s$ for every state s and such that this inequality is strict for some states. Then $x^{\pi'}_s \geq x^{\pi}_s$ for every state $s \in \mathcal{S}$ and this inequality is strict for at least one state.*

Theorem 2.3. *For all sub-optimal π, there exists $s \in \mathcal{S}$ and $u \in \mathcal{U}_s$ such that $a^{\pi}_{s \to u} > x^{\pi}_s$.*

PI heavily relies on these two facts. Here is how it works: starting from some initial policy, it simply makes some improving switches at each step to obtain the next iterate until no more improving switch is available. Since there is only a finite number of policies and since each new policy is strictly better than the previous one, PI is guaranteed to converge in a finite number of steps. However, Proposition 2.3 does not guarantee convergence in polynomial time.

We can think of many variations to define a Policy Iteration algorithm, depending on which rule we choose to select the improving switches to make at each step. For instance, one could go for a Simplex-like rule that switches only one action (typically the most appealing one) at each step. The most classical update rule is the so-called *Howard's* one (named after its original author) in which the most appealing action in each state is switched at each iteration.

We present Howard's PI in Algorithm 1. Every iteration requires us to compute the value of the current iterate x^{π_i}, which can be done in polynomial time for the three criteria presented above. The main issue is the number of iterations needed to converge which can be exponential in the number of states n, as shown by Fearnley (2010). Nevertheless, there still exists a huge gap between lower and upper bounds. Reducing this gap will be the main focus of this thesis. As we will see, the simplicity of the formulation of PI is inversely proportional to the complexity of its analysis.

2.1.3 Two-Player Turn-Based Stochastic Games

A *Two-Player Turn-Based Stochastic Game* (2TBSGs) is the natural extension of MDPs to a setting where two players with opposite objectives compete in a stochastic environment. In this context, we split the set of states $\mathcal{S}$ into two disjoint sets $\mathcal{S}^1$ and $\mathcal{S}^2$. All the actions in $\mathcal{S}^1$ are controlled by Player 1 whereas the actions in $\mathcal{S}^2$ are controlled by Player 2. Player 1 aims to minimize the rewards while Player 2 tries to maximize them, according to the discounted-, total- or average-reward criterion described above. The resulting game is a zero-sum game: all rewards that are being collected are actually paid by Player 1 to Player 2. The rest of the dynamics (transition between the states, collection of reward, etc.) happens exactly as in an MDP. Except that the solution we aim for is no longer the most rewarding policy but rather an equilibrium policy, that is, a policy that the players have no interest to deviate from. Note that in the 2TBSG setting, we usually refer to *strategies* rather than policies.

It is convenient to split a strategy π into π^1 and π^2 that are respectively controlled by Player 1 and Player 2. If we fix the strategy of one of the players, e.g., π^2, what remains is an MDP whose decision states are the ones of $\mathcal{S}^1$ and where Player 1 aims to minimize the value vector x^π. (Note that solving an MDP with a reward-minimization objective can be achieved by solving the usual reward-maximization problem after negating the sign of each reward.) We call a strategy of Player 1 that is a solution of this MDP an *optimal counter-strategy* or *best-response* to Player 2's strategy π^2. We let $\mathrm{br}_1(\pi^2)$ be such a strategy. Algorithmically speaking, this strategy can be computed using for instance PI in the MDP where the actions of the states in $\mathcal{S}^2$ have been frozen to π^2. A best-response strategy $\mathrm{br}_2(\pi^1)$ can be defined in the same way for player 2. Then, a strategy $\pi = (\pi^1, \pi^2)$ is called a (Nash) *equilibrium* iff the strategy of both players are a best-response against each other, that is, $\pi^1 = \mathrm{br}_1(\pi^2)$ and $\pi^2 = \mathrm{br}_2(\pi^1)$.

Similar results as the ones from Propositions 2.1, 2.2 and 2.3 can be stated for 2TBSGs. We refer to Hansen (2012) for proofs, see Theorems 3.1.7 and 3.2.5[1] and Corollary 3.2.3. First, we know that an equilibrium always exists.

Theorem 2.4. *There always exists an equilibrium strategy for the total-, discounted- and average-reward criteria.*

Suppose that Player 1 always chooses the best-response strategy against Player 2's strategy π^2. Then, making improving switches to π^2 can only improve the value vector for Player 2. This fact generalizes Theorem 2.2 for

[1] Note that the proof of Theorem 3.2.5 in Hansen (2012) does not hold as such for the average-reward criterion for 2TBSGs. To make it work, a more careful definition of the criterion is needed, as developed in Akian *et al.* (2012).

```
Algorithm 2: Howard's Strategy Iteration
  Input: An arbitrary policy $\pi_0 = (\pi_0^1, \pi_0^2)$, $i = 0$.
  Output: An equilibrium policy $\pi^*$.
1 while $\pi_i \neq \pi_{i-1}$ do
2     $\pi_{i+1}^2(s) \leftarrow \operatorname{argmax}_{u \in \mathcal{U}_s} a_{s \to u}^{\pi_i}$ for all states $s \in \mathcal{S}^2$.
3     $\pi_{i+1}^1 \leftarrow \mathrm{br}_1(\pi_{i+1}^2)$.
4     $\pi_{i+1} \leftarrow (\pi_{i+1}^1, \pi_{i+1}^2)$.
5     $i \leftarrow i + 1$.
6 return $\pi_i$.
```

2TBSGs. A similar statement holds for Player 1 as well. Note that in that case, an action $u \in \mathcal{U}_s$ is appealing to Player 1 iff $a_{s \to u}^{\pi} < x_s^{\pi}$.

Theorem 2.5. *Let $\pi = (\mathrm{br}_1(\pi^2), \pi^2)$ and $\overline{\pi} = (\mathrm{br}_1(\pi^2), \overline{\pi}^2)$ be obtained from π by making improving switches for Player 2 such that $a_{s \to \overline{\pi}(s)}^{\pi} \geq x_s^{\pi}$ for every state $s \in \mathcal{S}^2$ and such that this inequality is strict for at least one state. Then $x_s^{(\mathrm{br}_1(\overline{\pi}^2), \overline{\pi}^2)} \geq x_s^{(\mathrm{br}_1(\pi^2), \pi^2)}$ for every state $s \in \mathcal{S}$ and this inequality is strict for at least one state.*

As Theorem 2.3 did for MDPs, the next proposition makes the link between the non-existence of any appealing action with respect to a strategy π and the fact that π is an equilibrium.

Theorem 2.6. *A strategy π is an equilibrium iff no player has an improving switch, that is, for all states $s \in \mathcal{S}$ and all actions $u \in \mathcal{U}_s$, we have $a_{s \to u}^{\pi} \geq x_s^{\pi}$ if $s \in \mathcal{S}^1$ and $a_{s \to u}^{\pi} \leq x_s^{\pi}$ if $s \in \mathcal{S}^2$.*

In Algorithm 2, we present the *Strategy Iteration* algorithm (SI) to solve 2TBSGs, using Howard's switching rule. Step 2 is similar to step 2 in Algorithm 1. Then, step 3 of the algorithm can be thought of as an inner loop where we compute the best-response for Player 1 against Player 2. It can be achieved in finite time using PI in the MDP where the actions of Player 2 are frozen to π_{i+1}^2, or possibly by any other method. When counting the iterations of SI, we only account for the outer loop. Theorem 2.5 guarantees that the value vector strictly improves for Player 2 at each iteration, hence we never encounter the same strategy twice and the algorithm terminates in finite time. Theorem 2.6 provides the stopping criterion that ensures finding an equilibrium strategy.

Interestingly, in the eyes of Player 2, SI is completely analogous to PI: each step consists in switching to the most appealing actions in every state to obtain the next iterate, which always improves on the previous one. In fact, it is not difficult to show that both algorithms are equivalent when abstracted to Acyclic Unique Sink Orientation, as already discussed in Chapter 1. Consequently, the bounds that we will derive on PI in the next chapters will extend to SI as well. See Section 3.5 for a more detailed discussion of this equivalence.

2.2 An exponential lower bound for discounted Markov Decision Processes

A few years ago, based on a result from Friedmann (2009) on parity games, Fearnley (2010) introduced a family of Markov Decision Processes (MDPs) on which Policy Iteration (PI) requires an exponential number of steps to converge for the total- and average-reward criteria. Roughly at the same time, Ye (2011) showed that PI actually runs in strongly polynomial time for the discounted-reward criterion provided that the discount factor is fixed to a constant. These two results marked a milestone after more than 25 years of research on the question of the complexity of Policy Iteration.

In this section, we complete the picture by showing that Fearnley's result also applies to discounted-reward MDPs if the discount factor is part of the input. We also identify that choosing the discount rate γ to be $1 - 2^{-\Omega(n^2)}$, where n is the number of states, is enough to observe the exponential growth of the number of steps of PI with the family of MDPs defined by Fearnley.

Our starting point is the construction of Fearnley. Then, using perturbation analysis, we provide an adequate value of the discount factor such that adding discount to Fearnley's family of examples does not change the choices made by PI. Hence, it takes the same number of steps with or without discount and therefore, it requires an exponential number of steps to converge in both cases.

2.2.1 Fearnley's exponential complexity example for the total-cost criterion

Following the idea of Friedmann (2009), Fearnley (2010) provides a family of examples for a total-reward MDP with n states on which PI requires an exponential number of steps to converge. For that purpose, he implements an MDP in which some actions correspond to the bits of a binary counter. The policies explored by PI then gradually increment this binary counter until every configuration has been explored. Figure 2.2 illustrates the construction for a binary counter with 2 bits.

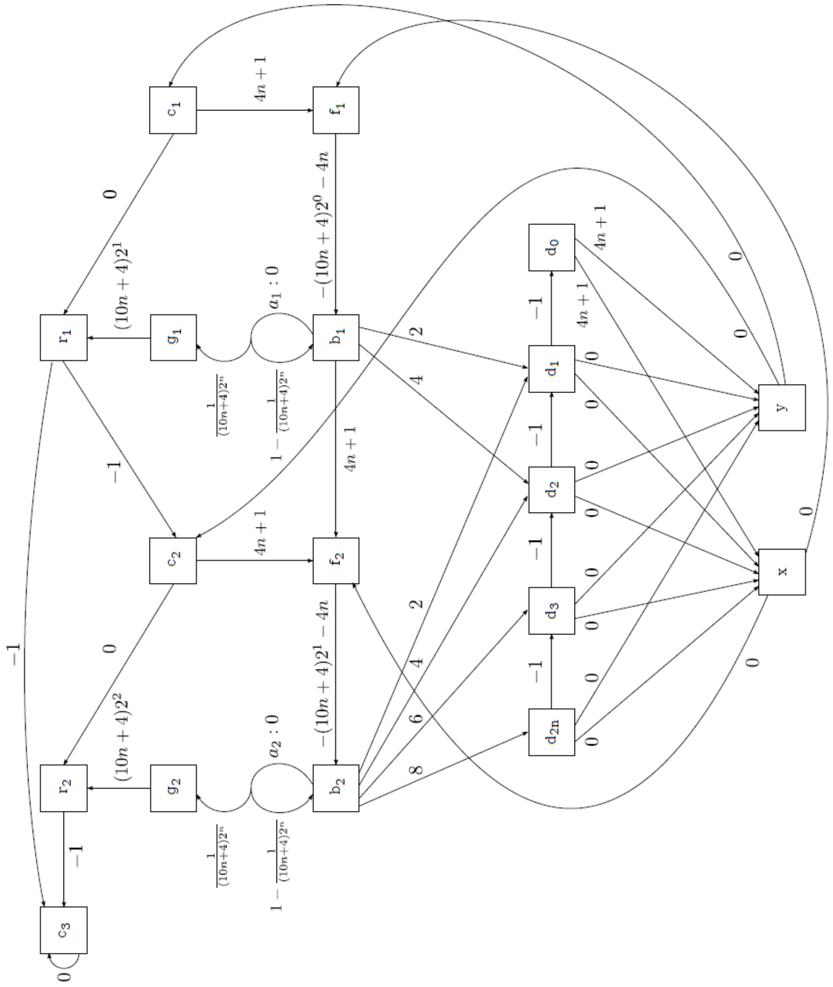


Figure 2.2: Fearnley's construction with a two-bits binary counter. Here, the actions a_1 and a_2 represent the bits of the binary counter. Except for these two actions, notice that all actions are deterministic.

Example 2.1 (Fearnley (2010)). Let $\mathcal{M}_{\text{exp}}$ be the MDP instance with n states proposed by Fearnley (2010). There exists an initial policy π_0 such that Policy Iteration needs to explore the sequence of policies

$$\{\, \pi_0, \pi_1, ..., \pi_{m-1} \,\} \tag{2.5}$$

to converge, with $m \geq 2^b$, where b is the number of bits of the binary counter and where the number of states is $n = 7b + 4$.

We now state three important properties of Example 2.1 that will be useful to our analysis. These features all follow from the developments made by Fearnley (2010).

Property 2.7. *In Example 2.1, every step of Policy Iteration, starting at π_0, is made in a non-ambiguous way, that is, at every step $i = 0, ..., m - 1$ of sequence (2.5) and for every state s, $\operatorname*{argmax}_{u \in \mathcal{U}_s} a^{\pi_i}_{s \to u}$ is unique.*

Property 2.8. *Every policy of Example 2.1 is proper, that is, whatever the chosen policy π and the starting state s, there exists a positive-probability path from s to the final state τ.*

Property 2.9. *In Example 2.1, $P^\pi \in \mathbb{Q}^{n \times n}$ and $r^\pi \in \mathbb{Z}^{n \times n}$ for every policy π and there exist values $\delta(n) \in \mathbb{N}$, $\delta(n) \leq (10b+4)2^b$ and $\kappa(n) \in \mathbb{N}$, $\kappa(n) \leq (10b + 4)2^b$, with $n = 7b + 4$, such that $\delta(n) \cdot P^\pi \in \mathbb{N}^{n \times n}$ for all π and that $|r^\pi_s| \leq \kappa(n)$ for all s, π*[2].

Property 2.8 makes sure that the value of the total-reward MDP from Example 2.1 is finite for any policy and any starting state, i.e., that the linear system (2.2) always has a unique solution (Bertsekas and Tsitsiklis, 1991). Property 2.9 guarantees that the considered MDP has reasonable size.

Note that Properties 2.8 and 2.9 summarize the restrictive assumptions that are made in our results at the next sections.

2.2.2 Elements of perturbation analysis

In this section, we study how a small perturbation of the total-reward MDP instance affects the value of the states. We expect that a small enough perturbation should not affect the behavior of Policy Iteration. This section builds up the basis on which our main result will rely. It may be useful to have a glance at Figure 2.3 at this point to have a better idea of the procedure that leads towards Theorem 2.18.

[2] To keep notations simple, we will write δ and κ and temporarily forget about the dependence in n.

Let π be some policy in a total-reward MDP and let x be the value of all states when using policy π, which can be computed by solving the linear system (2.2). Our goal is to determine how much x is perturbed from a perturbation of the system's matrix.

Lemma 2.10. *Let x be the solution of*

$$A\,x = b \tag{2.6}$$

and $\tilde{x}$ be the solution of the perturbed system

$$\tilde{A}\,\tilde{x} = b, \tag{2.7}$$

where A is an invertible matrix and let $\Delta x \triangleq \tilde{x} - x$ and $\Delta A \triangleq \tilde{A} - A$. Then the following bound holds for any subordinate norm:

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \cdot \|\Delta A\|}{1 - \|A^{-1}\| \cdot \|\Delta A\|}, \tag{2.8}$$

whenever

$$\|A^{-1}\| \cdot \|\Delta A\| < 1. \tag{2.9}$$

Proof. See, e.g., Higham (1996, Theorem 7.2). □

Note that (2.6) can be identified to (2.2) by taking $A = (I - P^{\pi})$ and $b = r^{\pi}$. We now bound the different norms that appear in (2.8) to obtain a usable bound on $\|\Delta x\|$. For that purpose, we will use the norm $\|\cdot\|_{\infty}$. Recall that for any matrix M, $\|M\|_{\infty} = \max_i \sum_j |M_{i,j}|$. Our analysis will make use of Hadamard's determinant inequality.

Theorem 2.11 (Hadamard)**.** *Let M be an $n \times n$ matrix such that $|M_{i,j}| \leq \beta$ for all i, j. Then:*

$$|\det(M)| \leq \beta^n \, n^{n/2}.$$

The next lemma gives us an upper bound on $\|A^{-1}\|_{\infty}$.

Lemma 2.12. *Let us assume that $A \in \mathbb{Q}^{n \times n}$ is an invertible matrix such that $|A_{i,j}| \leq 1$ and $\delta \cdot A_{i,j} \in \mathbb{Z}$ for all i, j. Then:*

$$\|A^{-1}\|_{\infty} \leq \delta^n \, n^{(n+1)/2}.$$

Proof. Since A is invertible, we may use Cramer's rule and express A^{-1} as:

$$A^{-1} = \delta\,(\delta \cdot A)^{-1} = \delta\,\frac{\operatorname{adj}(\delta \cdot A)}{\det(\delta \cdot A)} \tag{2.10}$$

where $\operatorname{adj}(\delta \cdot A)$ is the adjugate matrix of $\delta \cdot A$ in which every entry is the determinant of an $(n-1) \times (n-1)$ sub-matrix of the integer matrix $\delta \cdot A$ (possibly with a minus sign). We know that every entry of $|\delta \cdot A|$ is less than δ and that $|\det(\delta \cdot A)| \geq 1$. Hence, using (2.10) and Theorem 2.11, we have:

$$\begin{aligned}\|A^{-1}\|_\infty &\leq \delta \cdot \|\operatorname{adj}(\delta \cdot A)\|_\infty \\ &\leq \delta \cdot \max_{1 \leq j \leq n} \sum_{i=1}^{n} \delta^{n-1} (n-1)^{(n-1)/2} \\ &\leq \delta^n n^{(n+1)/2}.\end{aligned}$$ □

It now remains to find an upper bound on $\|x\|_\infty$.

Lemma 2.13. *Let x be the solution of (2.6), and assume that $A \in \mathbb{Q}^{n \times n}$ is an invertible matrix such that $|A_{i,j}| \leq 1$ and $\delta \cdot A_{i,j} \in \mathbb{N}$ for all i, j and that $b \in \mathbb{Z}^n$ with $|b_i| \leq \kappa$ for all i. Then:*

$$\|x\|_\infty \leq \kappa \, \delta^n \, n^{(n+1)/2}.$$

Proof. Using Lemma 2.12, we have $\|x\|_\infty \leq \|A^{-1}\|_\infty \cdot \|b\|_\infty \leq \delta^n n^{(n+1)/2} \kappa$. □

The next theorem uses the bounds from Lemmas 2.10 to 2.13 to obtain an upper bound on $\|\Delta x\|_\infty$.

Theorem 2.14. *Let x be the solution of (2.6) and $\tilde{x}$ be the solution of the perturbed system (2.7), and let $\Delta x = \tilde{x} - x$ and $\Delta A = \tilde{A} - A$. Let us further assume that $A \in \mathbb{Q}^{n \times n}$ is an invertible matrix such that $|A_{i,j}| \leq 1$ and $\delta \cdot A_{i,j} \in \mathbb{N}$ for all i, j and that $b \in \mathbb{Z}^n$ with $|b_i| \leq \kappa$ for all i. Then, provided that $\|\Delta A\|_\infty$ satisfies:*

$$\|\Delta A\|_\infty \leq 1/2 \cdot \delta^{-n} \, n^{-(n+1)/2}, \tag{2.11}$$

we have:

$$\|\Delta x\|_\infty \leq 2 \, \kappa \, \delta^{2n} \, n^{n+1} \cdot \|\Delta A\|_\infty.$$

Proof. We know from Lemma 2.12 that $\|A^{-1}\|_\infty \leq \delta^n n^{(n+1)/2}$. So if we impose $\|\Delta A\|_\infty$ to satisfy (2.11), then Assumption (2.9) from Lemma 2.10 is satisfied and the denominator in (2.8) is at least $1/2$. Substituting the other available bounds from Lemmas 2.12 and 2.13 into (2.8) gives the result. □

2.2.3 Perturbing Fearnley's example

In this section, we show that adding a discount factor $\gamma \triangleq 1-\varepsilon$ close enough to 1 to the originally discount-free total-reward MDP of Example 2.1 does not change the behavior of Policy Iteration and thus that the latter requires an exponential number of steps to converge on discounted-reward MDPs. We show this by providing a value of ε such that the same choices are made by Algorithm 1 on both problems at each improvement step. We proceed in three steps:

1. first, we identify the minimum possible difference between the appeal of the best action and the appeal of the other actions in a state;
2. then, we characterize the perturbation induced by adding a discount factor γ;
3. finally, we provide a value for ε that induces a small enough perturbation so that the action with the best appeal does not change in each state, and this at every step of PI.

The minimum difference between the appeals of actions

First, let us observe that the value of the states of an MDP can be expressed as a fraction with bounded denominator.

Lemma 2.15. *Let x be the solution of (2.6) and let us assume that $A \in \mathbb{Q}^{n\times n}$ is an invertible matrix such that $|A_{i,j}| \leq 1$ and $\delta \cdot A_{i,j} \in \mathbb{Z}$ for all i, j. Then the vector x can be expressed as:*

$$x = \frac{v}{d}$$

where v is an integer vector of the same dimension as x and d is a positive integer satisfying $d \leq \delta^n\, n^{n/2}$.

Proof. The linear system (2.6) can be rewritten as $(\delta \cdot A)\, x = \delta \cdot b$, where both $\delta \cdot A$ and $\delta \cdot b$ are integer valued. Hence, using Cramer's rule for linear systems, x can be expressed as:

$$x = \frac{v}{|\det(\delta \cdot A)|}$$

where v is an integer vector with the same dimension as x and $|\det(\delta \cdot A)|$ is a positive integer, say d. Since $|\delta \cdot A_{i,j}| \leq \delta$ for every i, j, Theorem 2.11 enables us to conclude. □

The following bound makes use of the fact that every step of PI is made in a non-ambiguous way.

Theorem 2.16. *For any state s and any step i of Policy Iteration applied to Example 2.1, let $u^* = \underset{u \in \mathcal{U}_s}{\operatorname{argmax}}\, a^{\pi_i}_{s \to u}$ and let $u' \neq u^*$ be any other action in $\mathcal{U}_s$. Then:*

$$a^{\pi_i}_{s \to u^*} - a^{\pi_i}_{s \to u'} \geq \frac{1}{\delta^{n+1}\, n^{n/2}}.$$

Proof. From the definition of appeal (2.4) and from Property 2.7, we know that

$$a^{\pi_i}_{s \to u^*} - a^{\pi_i}_{s \to u'} = (r^{u^*}_s - r^{u'}_s) + \sum_{s' \in \mathcal{S}} (P^{u^*}_{s,s'} - P^{u'}_{s,s'})\, x^{\pi_i}_{s'} \;\; > \;\; 0.$$

Since Example 2.1 has Properties 2.8 and 2.9, $\delta \cdot (P^{u^*}_{s,s'} - P^{u'}_{s,s'})$ is an integer and we can use Lemma 2.15 and write:

$$a^{\pi_i}_{s \to u^*} - a^{\pi_i}_{s \to u'} = \frac{w}{\delta \cdot d},$$

where w is an integer strictly greater than 0 and d is less than $\delta^n\, n^{n/2}$. □

The perturbation induced by the discount

Let $\mathcal{M}$ be a total-reward MDP and let $\mathcal{M}_\gamma$ be the corresponding discounted-reward MDP, i.e., the MDP with same states- and actions space, transition probabilities and rewards but with an additional discount factor $\gamma = 1 - \varepsilon$. For recall, the value $\tilde{x}^\pi$ of the states of $\mathcal{M}_\gamma$ under policy π is obtained by solving the following linear system:

$$(I - \gamma\, P^\pi)\, \tilde{x}^\pi = r^\pi.$$

We can identify this system to (2.7), where $\tilde{A} = I - \gamma\, P^\pi$. Furthermore, if we define $A = I - P^\pi$ as in (2.6), then $\tilde{A}$ can be expressed as a perturbation ΔA of A, namely $\Delta A = \tilde{A} - A = \varepsilon\, P^\pi$. Hence, $\|\Delta A\|_\infty \leq \varepsilon$. For recall, given a state s of the discounted-reward MDP $\mathcal{M}_\gamma$, we define the appeal $\tilde{a}^\pi_{s \to u}$ of an action $u \in \mathcal{U}_s$ with respect to some policy π as:

$$\tilde{a}^\pi_{s \to u} \triangleq r^u_s + \sum_{s' \in \mathcal{S}} \gamma\, P^u_{s,s'}\, \tilde{x}^\pi_{s'},$$

according to equation 2.3. Let us now quantify the perturbation on the appeals incurred from the discount.

Theorem 2.17. *For any state s, any action $u \in \mathcal{U}_s$ and any step i of Policy Iteration applied to Example 2.1, we have:*

$$|\, a^{\pi_i}_{s \to u} - \tilde{a}^{\pi_i}_{s \to u}\,| \leq 4\, \kappa\, \delta^{2n}\, n^{n+2}\, \varepsilon,$$

where $a^{\pi_i}_{s \to u}$ and $\tilde{a}^{\pi_i}_{s \to u}$ are defined by (2.4) and (2.3) respectively and $\gamma = 1 - \varepsilon$ is the discount factor.

Proof. In the definition (2.3) of $\tilde{a}^{\pi_i}_{s\to u}$, we may write $\tilde{x}^{\pi_i}$ as a perturbation of x^{π_i} using the same notation as in Lemma 2.10: $\tilde{x}^{\pi_i} \triangleq x^{\pi_i}+\Delta x^{\pi_i}$. From (2.4) and (2.3), we have:

$$|\, a^{\pi_i}_{s\to u} - \tilde{a}^{\pi_i}_{s\to u} \,| \leq \sum_{s'\in\mathcal{S}} P^u_{s,s'} \;|\, \varepsilon\, x^{\pi_i}_{s'} - (1-\varepsilon)\,\Delta x^{\pi_i}_{s'} \,|$$

Since $P^u_{s,s'} \leq 1$ and $|v_j| \leq \|v\|_\infty$ for any j and any vector v, we have:

$$|\, a^{\pi_i}_{s\to u} - \tilde{a}^{\pi_i}_{s\to u} \,| \leq n \,\cdot\, (\,\varepsilon\, \|x^{\pi_i}\|_\infty + (1-\varepsilon)\, \|\Delta x^{\pi_i}\|_\infty\,).$$

Using the fact that $1-\varepsilon < 1$ and the bounds from Lemma 2.13 and Theorem 2.14 with a perturbation $\|\Delta A\|_\infty \leq \varepsilon$ gives the result. □

An exponential lower bound for discounted MDPs

Let us now combine the results from this section to show that the choices made by PI at every improvement step do not change when applied to the discounted or the undiscounted version of Example 2.1. This is the main result from this chapter.

Theorem 2.18. *There exists an infinite family of discounted-reward MDPs with a particular starting policy and a discount factor $\gamma = 1 - 2^{-\Omega(n^2)}$ on which the number of iterations that Policy Iteration takes is lower bounded by $\Omega\big(\sqrt[7]{2}^{\,n}\big)$, an exponential function of the size n of the MDP.*

Proof. Let $\mathcal{M}$ be the total-reward MDP from example 2.1 on which PI explores the exponential size sequence of policies (2.5) and let $\mathcal{M}_\gamma$ be the corresponding discounted-reward MDP with $\gamma = 1-\varepsilon$ defined above. We show that PI also explores sequence (2.5) on $\mathcal{M}_\gamma$, of size at least $2^b = 2^{(n-4)/7} = \Omega\big(\sqrt[7]{2}^{\,n}\big)$, provided ε is small enough. Figure 2.3 sketches the idea of the proof.

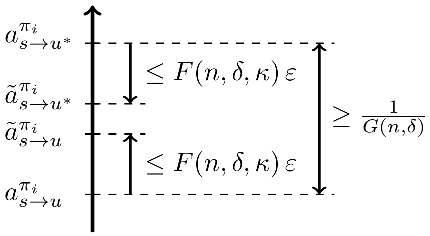

$a^{\pi_i}_{s\to u^*}$
$\tilde{a}^{\pi_i}_{s\to u^*}$
$\tilde{a}^{\pi_i}_{s\to u}$
$a^{\pi_i}_{s\to u}$
$\leq F(n,\delta,\kappa)\,\varepsilon$
$\leq F(n,\delta,\kappa)\,\varepsilon$
$\geq \frac{1}{G(n,\delta)}$

Figure 2.3: The idea of the proof of Theorem 2.18 is to bound the parameter ε in order to make sure that $|a^{\pi_i}_{s\to u^*} - \tilde{a}^{\pi_i}_{s\to u^*}| + |a^{\pi_i}_{s\to u} - \tilde{a}^{\pi_i}_{s\to u}| \leq a^{\pi_i}_{s\to u^*} - a^{\pi_i}_{s\to u}$.

Let $F(n, \delta, \kappa) \triangleq 4\,\kappa\,\delta^{2n}\,n^{n+2}$ and $G(n, \delta) \triangleq \delta^{n+1}\,n^{n/2}$. Theorem 2.17 tells us that

$$|\,a^{\pi_i}_{s\to u} - \tilde{a}^{\pi_i}_{s\to u}\,| \leq F(n, \delta, \kappa) \cdot \varepsilon,$$

for every state s, action $u \in \mathcal{U}_s$ and policy π_i of sequence (2.5), where $a^{\pi_i}_{s\to u}$ and $\tilde{a}^{\pi_i}_{s\to u}$ are respectively defined by (2.4) and (2.3). Similarly, Theorem 2.16 tells us that for every state s and policy π_i from sequence (2.5),

$$a^{\pi_i}_{s\to u^*} - a^{\pi_i}_{s\to u} \geq \frac{1}{G(n, \delta)},$$

where $u^* = \operatorname{argmax}_{u'\in\mathcal{U}_s} a^{\pi_i}_{s\to u'}$ and u is any action in $\mathcal{U}_s$ different from u^*. Since $|y - x| \geq y - x \geq -|y - x|$ for any $x, y \in \mathbb{R}$, the following relations are true:

$$\begin{aligned} a^{\pi_i}_{s\to u} \;\; - \tilde{a}^{\pi_i}_{s\to u} \;\; &\geq -|a^{\pi_i}_{s\to u} \;\; - \tilde{a}^{\pi_i}_{s\to u} \;\;| \\ &\geq -F(n, \delta, \kappa)\,\varepsilon, \qquad (2.12) \\ a^{\pi_i}_{s\to u^*} - \tilde{a}^{\pi_i}_{s\to u^*} &\leq \quad |a^{\pi_i}_{s\to u^*} - \tilde{a}^{\pi_i}_{s\to u^*}| \\ &\leq \quad F(n, \delta, \kappa)\,\varepsilon. \qquad (2.13) \end{aligned}$$

Subtracting (2.13) to (2.12), we obtain

$$\tilde{a}^{\pi_i}_{s\to u^*} - \tilde{a}^{\pi_i}_{s\to u} \geq a^{\pi_i}_{s\to u^*} - a^{\pi_i}_{s\to u} - 2\,F(n, \delta, \kappa)\,\varepsilon.$$

From Theorem 2.16, we know that

$$\tilde{a}^{\pi_i}_{s\to u^*} - \tilde{a}^{\pi_i}_{s\to u} \geq \frac{1}{G(n, \delta)} - 2\,F(n, \delta, \kappa)\,\varepsilon.$$

If we take ε to satisfy

$$\varepsilon < \frac{1}{2\,F(n, \delta, \kappa)\,G(n, \delta)} = \frac{1}{8\,\kappa\,\delta^{3n+1}\,n^{3/2\,n+2}}, \qquad (2.14)$$

we have $\tilde{a}^{\pi_i}_{s\to u^*} > \tilde{a}^{\pi_i}_{s\to u}$ for every $u \neq u^*$ and hence, $\operatorname{argmax}_{u\in\mathcal{U}_s} \tilde{a}^{\pi_i}_{s\to u} = u^* = \operatorname{argmax}_{u'\in\mathcal{U}_s} a^{\pi_i}_{s\to u'}$. Therefore, by induction, PI makes the same choice on both $\mathcal{M}$ and $\mathcal{M}_\gamma$ at every step i and sequence (2.5) is observed on both problems.

Note that $\mathcal{M}_\gamma$ has the same size as $\mathcal{M}$ and recall that $\delta(n) = \kappa(n) = (10b+4)2^b$, where $n = 7b+4 > b$. Hence, an ε that satisfies condition (2.14) can be written with a polynomial number of bits since:

$$\begin{aligned} \varepsilon &< \frac{1}{2^{3+\log_2\kappa+(3n+1)\,\log_2\delta+(1.5n+2)\,\log_2 n}} \\ &= \frac{1}{2^{3+b+(3n+1)n+(3n+2)\log_2(10n+4)+(1.5n+2)\,\log_2 n}} \\ &< \frac{1}{2^{\Omega(n^2)}}, \end{aligned}$$

Finally, observe that condition (2.14) implies condition (2.11) from Theorem 2.14. □

2.2.4 Conclusions on Fearnley's example

In Theorem 2.18, we identified that it is enough to choose $\gamma = 1 - 2^{-\Omega(n^2)}$ to observe Fearnley's result with the discounted-reward criterion. In contrast, we know from Ye (2011) that it is necessary to have γ approaching 1 as n grows to observe the same phenomenon. The minimum rate of growth of γ that allows to observe the exponential time behavior of PI is still unknown at this point and it would be interesting to further explore the question. Our guess is that Fearnley's example still applies if we choose $\gamma = 1 - 2^{-\Omega(n)}$, which is what we observe using our practical implementation of the construction. A possible way to show this could come from refining the estimates in our proof.

Example 2.1 rules out hope for greedy Policy Iteration to be a strongly polynomial time algorithm to solve Markov Decision Processes, even though it was one of the best candidates. Nevertheless, this example is artificial and it is unlikely to be encountered in practical applications. In Figure 2.4, we attempt to challenge the robustness of Example 2.1 by adding a small perturbation to PI; we therefore make *all but one* (instead of *all*) improving switches at every step of the algorithm. We observe that the number of iterations seems to grow at a polynomial rate in that case. This observation supports the idea that applying smoothed analysis, developed by Spielman and Teng (2004) for the Simplex method, to PI could help to explain its practical efficiency. In such an event, we expect the multi-switch nature of PI to be the main challenge in successfully applying these techniques.

Finally, we note that Fearnley's example requires almost no randomization as only b actions are non-deterministic with transition probabilities close to 0 or 1. It would be interesting to determine whether this randomization is actually required to observe the exponential time behavior. As a comparison, Simplex PI—the single-switch variant of PI—has been shown to run in strongly polynomial time on deterministic MDPs by Post and Ye (2015), even when the discount factor is part of the input. However the arguments leading to their result seem unlikely to apply to PI because of its multi-switch nature and therefore, to the best of our knowledge, the question remains widely open for PI.

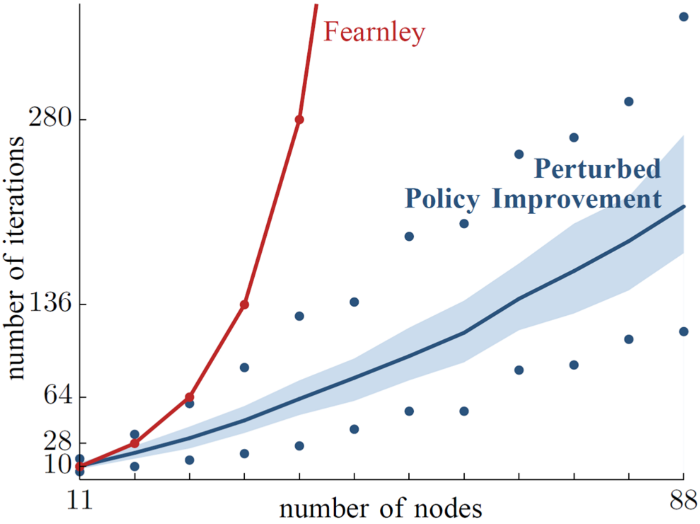


Figure 2.4: (Red) Policy Iteration run on instances of Example 2.1 of increasing size. The exponential increase of the number of iterations is observed. (Blue) At each step of the algorithm, instead of making every improving switch, we choose one at random that is not switched. 200 trials have been made for every problem size and the number of iterations achieved has been recorded: the straight line is the average number of steps observed, the blue shadow contains 2/3 of the points and the blue dots are the extreme values for each problem size. The exponential behavior seems to have disappeared.

Chapter 3

A new upper bound for Policy Iteration

In the previous chapter, we introduced the classical view of *Markov Decision Processes* (MDPs) and of *Policy Iteration* (PI) to solve them. On that occasion, we saw the notion of *policy* which is fundamental to the latter algorithm and we saw that policies were characterized by a value vector. Because of the vector form, policies cannot always be compared with one another with respect to their value since there is not always a clear domination. However, we have observed in Theorem 2.2 that switching any subset of appealing actions from a policy leads to a strictly improved policy. This leads to the idea of an underlying partial order among the policies. PI is designed to jump from policies to policies in the partial order while always going "forward", that is, improving all the entries of the value vector at each step, until it converges to the optimal policy.

In fact, the partial order hidden by an MDP has the well-known structure of an *Acyclic Unique Sink Orientation* of a grid (or a cube if every state of the MDP has at most two available actions). In this chapter, we take advantage of this structure to extract as many relevant properties as possible about the partial order. In the end, we use these properties to improve the existing upper bound on the number of iterations of PI in the most general setting, that is, for the three classical reward criteria.

Prior to our work, the best known upper bound on the number of iterations of PI was given by Mansour and Singh (1999) with a $13 \cdot \frac{k^n}{n}$ bound, where n is the number of states of the MDP and k is the maximum number of actions per state. In this chapter, we modestly improve the constant of this bound to $\frac{k}{k-1}$. Our improvement also holds for the Strategy Iteration algorithm described in Section 2.1.3 to solve Two-Player Turn-Based

Stochastic Games for which no polynomial time algorithm is known.

To obtain our bound, we use a number of properties of the PI-sequence, that is, of the sequence of policies actually evaluated by PI. In order to investigate the tightness of the bound, it is of natural interest to explore which of these properties could be further exploited to improve the bound and which ones cannot. It turns out that the properties we actually use cannot lead to further improvements, that is, they are "fully exploited". To formally prove this fact, we introduce the notion of *pseudo-PI-sequence* in Section 3.1 to describe any sequence of policies satisfying only the properties that we use to obtain our bound from Theorem 3.9. We then show in Theorem 3.12, Section 3.3, that there always exists a pseudo-PI-sequence whose size matches the upper bound of Theorem 3.9. This confirms that the bound is sharp for pseudo-PI-sequences. Therefore, obtaining new bounds on the number of steps of PI would require exploiting stronger properties, like the Order-Regularity condition that we explore in the next chapters.

This chapter is organized as follows. First in Section 3.1 we reintroduce PI under the perspective of the underlying partial order of the policies, and we formulate a number of important properties about the latter. In Section 3.2, we prove our new upper bound for PI-sequences. Then in Section 3.3, we show that the upper bound is tight for pseudo-PI-sequences. In Section 3.4, we put the whole analysis of this chapter in perspective by linking it with the structure of Acyclic Unique Sink Orientations of grids. In Section 3.5 we argue that our upper bound also holds for the number of steps of Strategy Iteration to solve Two-Player Turn-Based Stochastic Games. Finally in Section 3.6, we mention some aspects of the analysis that become significantly simpler when we restrict ourselves to the case where each state of the considered MDPs has at most two available actions. In the next chapters, we will focus exclusively on that case.

3.1 Policy Iteration revisited

For recall, a Markov Decision Process is made of four elements: a (finite) set of *states* $\mathcal{S} = \{1, \dots, n\}$, a (finite) set of *actions* $\mathcal{U}$ where $\mathcal{U}_s$ is the set of actions available in state s, a set of *transition probabilities* for the next state to visit and a set of *rewards* associated with each action. For simplicity, we use a common numbering for the actions, that is, $\mathcal{U}_s \triangleq \mathcal{U} = \{1, \dots, k\}$ for all $s \in \mathcal{S}$. With this notation, for every pair $(s, u) \in \mathcal{S} \times \mathcal{U}$, the transition probability and reward functions are uniquely defined. We call a *policy* $\pi \in \{1, \dots, k\}^n$ the stationary choice of one action for every state. Every policy induces a given transition probability matrix P^π and a reward vector r^π corresponding to some Markov chain.

We may ask how rewarding a policy π is in the long run. This is represented by its *value* vector $x^\pi \in \mathbb{R}^n$ whose $s-th$ entry corresponds to the long term expected reward obtained from starting in state s and following the policy π thereafter. It can be computed by solving a system whose definition depends on the problem studied. See Section 2.1.1 for a discussion about the three classical reward criteria that are usually used to define the value of a policy. In this chapter, the bounds that we derive do not depend on the chosen reward criterion. By *solving* an MDP, we mean finding the optimal policy π^* such that for any other policy π, $x^{\pi^*} \geq x^\pi$, that is, $x^{\pi^*}(s) \geq x^\pi(s)$ for all states s. The existence of such a policy is guaranteed, see Theorem 2.1.

In this section, we propose an alternative view of MDPs compared to the previous chapter, focused on the policies and how they relate to each other through a partial order. Thereby, we are going to build a deeper understanding of how the policies of an MDP are structured, which is a key step to better understand Policy Iteration.

Definition 3.1 (Domination)**.** Given two policies π and π', if $x^{\pi'}(s) \geq x^\pi(s)$ for all states $s \in \mathcal{S}$, then we say that π' *dominates* π and we write $\pi' \succeq \pi$. If moreover $x^{\pi'}(s) > x^\pi(s)$ for at least one state, then the domination is strict and we write $\pi' \succ \pi$.

Definition 3.2 (Switching)**.** Let U be a collection of state-action pairs (s, u). We say that U is *well-defined* if it contains every state $s \in \mathcal{S}$ at most once. In that case, we define $\pi' = \pi \oplus U$ to be the policy obtained from π by *switching* the action $\pi(s)$ to u for each (s, u)-pair in U. In the case of MDPs with exactly two actions per state, we allow U to contain only the switching states, the action to switch to in each state being implicit.

Definition 3.3 (Improvement set)**.** We define the *improvement set* of a policy π as:

$$T^\pi = \big\{(s, u) \mid \pi \oplus \{(s, u)\} \succ \pi\big\},$$

and the set of *improvement states* S^π of π as the set of states that appear in T^π.

Using the above definitions, the Theorems 2.2 and 2.3 from Chapter 2 allow the following, simpler, expression. Proofs can be found, e.g., in Hansen (2012); see Theorems 2.2.12, 2.3.9 and 2.4.6 for the discounted-, average- and total-reward criteria respectively. Alternate statements can also be found in Puterman (1994).

Proposition 3.1. *Let π be a policy and $U \neq \emptyset$ be any well-defined subset of its improvement set T^π. Then $\pi \oplus U \succ \pi$.*

Proposition 3.2. *For a given policy π, if $T^\pi = \emptyset$, then π is optimal.*

Algorithm 3: Policy Iteration

```
Initialization: $\pi_0$, $i = 0$
while $T^{\pi_i} \neq \emptyset$ do
    Select a non-empty and well-defined $U_i \subseteq T^{\pi_i}$
    $\pi_{i+1} = \pi_i \oplus U_i$
    $i \leftarrow i + 1$
end
return $\pi_i$
```

We may also propose an alternative formulation for *Policy Iteration* compared to the previous chapter, in terms of improving switches. The rest of our analysis in this chapter relies solely on these two results which thus holds for the three classical reward criteria.

Definition 3.4 (Policy Iteration)**.** Algorithm 7 describes *Policy Iteration* (PI). The standard way of choosing $U_i \subseteq T^{\pi_i}$ is the greedy update rule, namely choose any U_i with maximal cardinality $|S^{\pi_i}|$, which yields Howard's PI algorithm of the previous chapter. Howard's PI remains the main focus of this thesis.

Definition 3.5 (Comparable)**.** We say that two policies π and π' are *comparable* if either $\pi \preceq \pi'$ or $\pi \succeq \pi'$. We call two policies *neighbors* if they differ in only one state. Neighbors are always comparable (Mansour and Singh, 1999, Lemma 3).

Definition 3.6 (Partial order)**.** For a given MDP, we consider the *partial order* PO of the policies defined by the domination relation. A set of policies $\pi^{(1)}, \dots, \pi^{(k)}$ is called a *sequence* if $\pi^{(1)} \preceq \dots \preceq \pi^{(k)}$.

Definition 3.7 (PI-sequence)**.** We refer to the sequence of policies $\pi_0, \dots, \pi_{m-1}$ explored by greedy PI as a *PI-sequence* of length m.

Problem 3.1. Find the longest possible PI-sequence.

Lemma 3.3 (Mansour and Singh (1999, Lemma 4))**.** *For any two policies π, π' such that $\pi'(s) = \pi(s)$ for all improvement states $s \in S^\pi$, we have $\pi' \preceq \pi$.*

Proof. Consider an MDP M' where the only action that is available in the states $s \in S^\pi$ is $\pi(s)$. Clearly π and π' are valid policies for M' and their value does not change. On the other hand, the improvement set of π is empty in M', so by Proposition 3.2, π is optimal for M' and we must have $\pi' \preceq \pi$. □

The next property indicates how the improvement set of a policy is constrained by the dominated policies and by their own improvement sets.

Proposition 3.4 (Extended from Lemma 12 in (Mansour and Singh, 1999)). *For any two policies $\pi \prec \pi'$, there exists an improvement state $s \in S^\pi$ such that $\pi(s) \neq \pi'(s)$ and $(s, \pi(s)) \notin T^{\pi'}$.*

Proof. Suppose on the contrary that it is not the case. Then for all states $s \in S^\pi$, either $\pi(s) = \pi'(s)$ or $(s, \pi(s)) \in T^{\pi'}$. Let $U \triangleq \left\{ (s, \pi(s)) : s \in S^\pi \cap S^{\pi'} \text{ and } \pi(s) \neq \pi'(s) \right\}$, then we have $U \subseteq T^{\pi'}$. Therefore, Proposition 3.1 tells us that $\pi'' \triangleq \pi' \oplus U \succeq \pi'$.

Now, let us consider any $s \in S^\pi$. If $\pi'(s) = \pi(s)$, then for any $u \in \mathcal{U}$, we have $(s, u) \notin U$ and $\pi''(s) = \pi(s)$. On the other hand, if $\pi'(s) \neq \pi(s)$, then $s \in S^{\pi'}$, hence $(s, \pi(s)) \in U$ and $\pi''(s) = \pi(s)$ again. Therefore $\pi''(s) = \pi(s)$ for all $s \in S^\pi$ and from Lemma 3.3, $\pi'' \preceq \pi$ $(\prec \pi')$ which is a contradiction. □

Note that for $k = 2$, the statement of Proposition 3.4 can be simplified and implies that for any two policies $\pi \prec \pi'$, it holds that $S^\pi \not\subseteq S^{\pi'}$.

When performing a PI step, we jump from the current policy to some policy that can be quite different (in terms of number of different entries). However, we now show that there always exists a path of small steps in the partial order connecting the two, that is, from neighbor to neighbor.

Proposition 3.5 (Extended from Lemma 6 in (Mansour and Singh, 1999)). *Let π and π' be two policies such that $\pi' = \pi \oplus U$ for some well-defined $U \subseteq T^\pi$ of cardinality d. Then there exist at least d policies $\pi^{(1)}, \ldots, \pi^{(d)}$ such that $\pi \prec \pi^{(1)} \preceq \cdots \preceq \pi^{(d)} = \pi'$ and such that $\pi^{(i)}$ and $\pi^{(i+1)}$ are neighbors for all $1 \leq i < d$.*

Proof. If $d = 1$, simply take $\pi^{(d)} = \pi'$. Suppose that the result is true for $d - 1 \geq 1$ and let us show it for d. From Proposition 3.4, there exists a state $s \in S^\pi$ such that $(s, \pi(s)) \notin T^{\pi'}$, that is, such that $\pi' \oplus (s, \pi(s)) \not\succ \pi'$. Since neighbors are always comparable, it means that $\pi'' \triangleq \pi' \oplus (s, \pi(s)) \preceq \pi'$. By definition of π', we have $(s, \pi'(s)) \in U$ and $U' \triangleq U \setminus (s, \pi'(s)) \subseteq U \subseteq T^\pi$. We can then recursively apply the statement of Proposition 3.5 with:

$$\begin{aligned} \pi' &\longmapsto \pi'' = \pi' \oplus (s, \pi(s)), \\ U &\longmapsto U' = U \setminus (s, \pi'(s)), \end{aligned}$$

since $\pi'' = \pi \oplus U'$ and $|U'| = d - 1$. In that case, $\pi^{(d-1)} = \pi''$, and we can choose $\pi^{(d)} = \pi'$ which is indeed a neighbor of $\pi^{(d-1)}$. □

Note that the above proof simplifies Mansour and Singh's original argument.

Definition 3.8 (Subsequence and supersequence)**.** Let O be a sequence. We call *subsequence* of O any ordered subset of elements of O. We call *supersequence* of O any sequence that contains O as a subsequence.

The following property is perhaps the most important consequence of Proposition 3.5.

Corollary 3.6 (Jumping)**.** *Let π_i be a policy of a PI-sequence. Then the partial order of policies contains a supersequence of the PI-sequence with at least $|S^{\pi_i}|$ different policies between π_i and π_{i+1}, that is, $|S^{\pi_i}|$ policies π such that $\pi_i \prec \pi \preceq \pi_{i+1}$. When we step from π_i to π_{i+1}, we say that we* jump *$|S^{\pi_i}|$ policies of the supersequence.*

Proof. The result is a direct consequence of Proposition 3.5. Recall that with Howard's PI, $|U_i|$ always equals $|S^{\pi_i}|$. □

We now introduce an object that is similar to a PI-sequence in that it describes a sequence of policies embedded into a partial order. However, we will forget about some of the structure that originates from MDPs and only require Proposition 3.4 and Corollary 3.6 to be ensured by the sequence and the partial order. This will allow us to show that these two properties—that will be the two milestones in the proof of our upper bound in Theorem 3.9—can actually not help to further improve the bound.

Definition 3.9 (Pseudo-PI-sequence)**.** We call *pseudo-PI-sequence* of size m a triple $(\Pi, O, \mathcal{T})$ where:

- $\Pi = \pi_0, \pi_1, \ldots, \pi_{m-1}$ is a sequence of policies. We define the abstract ordering $\prec$ on the elements of the sequence Π by the ordering of their indices.
- O is a sequence of policies of $\{1, \ldots, k\}^n$ that is a supersequence of Π.
- $\mathcal{T}$ is a collection of abstract improvement sets T^π for every policy π appearing in O.

We require the claim from Proposition 3.4 to hold for O and we require Π to satisfy Corollary 3.6 as a subsequence of O.

Definition 3.9 leads to a relaxation of Problem 3.1, and therefore any upper bound on the size of pseudo-PI-sequences also holds for PI-sequences. Note that there is a natural way of constructing a pseudo-PI-sequence from any PI-sequence. Of course, Proposition 3.4 and Corollary 3.6, that are the key results towards our upper bound in Theorem 3.9, still hold for pseudo-PI-sequences by design. Furthermore, as we will show in Theorem 3.12, our upper bound is tight for the relaxation.

Problem 3.2. Find the longest possible pseudo-PI-sequence.

To close this section, we propose a simple consequence of Proposition 3.1 that was not expressed by Mansour and Singh (1999). Although we were not able to exploit this consequence to improve the upper bound on PI's complexity, we will see in Chapter 4 how it leads to the Order-Regularity condition—a powerful tool for the analysis of PI that we will develop in the next chapters. The remainder of this section is not mandatory for readers only interested with the results of the present chapter.

To obtain this last condition, we first need to extend the notion of improvement sets and generalize Proposition 3.1

Definition 3.10 (Generalized improvement sets)**.** We define the *generalized improvement set* of a policy π as:

$$T^\pi_\propto = \big\{(s,u) \mid \pi \oplus \{(s,u)\} \propto \pi\big\},$$

where $\propto \in \{\succ, \succeq, \prec, \preceq\}$, and let $S^\pi_\propto$ be the set of states that appear in $T^\pi_\propto$. In particular, with this notation, $T^\pi = T^\pi_\succ$ and $S^\pi = S^\pi_\succ$.

Using this notation, we can generalize Proposition 3.1 in a natural way.

Corollary 3.7 (Generalized version of Proposition 3.1)**.** *Let π be a policy and $U \neq \emptyset$ be any well-defined subset of its generalized improvement set $T^\pi_\propto$, with $\propto \in \{\succ, \succeq, \prec, \preceq\}$. Then $\pi \oplus U \propto \pi$.*

Proof. Proposition 3.1 proves the case where $\propto$ is $\succ$. To show the case where $\propto$ is $\succeq$, we consider the MDP $\mathcal{M}'$ which is identical to the original MDP $\mathcal{M}$ except in the states $s \in S^\pi_\precsim$ (that is, the states $s \notin S^\pi_\succeq$) where the action is fixed to $\pi(s)$. Clearly, $T^\pi_\succeq$ remains unchanged in this MDP, as do the value vectors of the policies in $\mathcal{M}'$ compared to their counterparts in $\mathcal{M}$. This new MDP ensures that for any policy π' in $\mathcal{M}'$, we have that $\pi'(s) = \pi(s)$ for all $s \in S^\pi_\precsim$. It follows from a variant of Lemma 3.3 with a reversed objective function of the MDP (that is, where rewards are turned into costs) that we must have $\pi' \succeq \pi$. In particular, for any well-defined subset $U \subseteq T^\pi_\succeq$, we must have $\pi \oplus U \succeq \pi$, both in $\mathcal{M}'$ and in $\mathcal{M}$. Finally, the cases where $\propto$ is $\prec$ and $\preceq$ are easily solved from the ones where $\propto$ is $\succ$ and $\succeq$ respectively, by reversing the objective function of the MDP. □

We now state the desired condition on the partial order of the policies.

Theorem 3.8. *Let $\pi \prec \pi'$ be two policies. Then for all $U \subseteq T^\pi_\preceq$ and all $U' \subseteq T^{\pi'}_\succeq$, it holds that $\pi \oplus U \neq \pi' \oplus U'$.*

Proof. For all $U \subseteq T^{\pi}_{\preceq}$ and all $U' \subseteq T^{\pi'}_{\succeq}$, we have:

$$\pi \oplus U \preceq \pi \prec \pi' \preceq \pi' \oplus U',$$

where the first and last relations come from Corollary 3.7. This implies that $\pi \oplus U$ cannot be equal to $\pi' \oplus U'$. □

Note that Theorem 3.8 remains true if we replace $T^{\pi}_{\preceq}$ by $T^{\pi}_{\prec}$ or $T^{\pi}_{\succeq}$ by $T^{\pi}_{\succ}$.

3.2 A new upper bound for Policy Iteration

In order to precisely solve Problem 3.2, we need to both provide a lower and an upper bound on the length of pseudo-PI-sequences. We start by showing the upper bound, which also holds for Problem 3.1 and therefore provides a new upper bound on the complexity of Policy Iteration in general.

Theorem 3.9. *The number of iterations of Policy Iteration is bounded above by* $\frac{k}{k-1} \cdot \frac{k^n}{n} + o\left(\frac{k^n}{n}\right)$.

Before we proceed to the proof of Theorem 3.9, we need to formulate two additional properties. First, we derive the following lemma from Proposition 3.4.

Lemma 3.10 (Adapted from Lemma 4 in Mansour and Singh (1999))**.** *Let* $(\Pi, O, \mathcal{T})$ *be a pseudo-PI-sequence. Then for any two policies* $\pi \prec \pi'$ *of* O *and any* $U \subseteq T^{\pi'}$, *we have* $\pi \neq \pi' \oplus U$.

Proof. Let $s \in S^{\pi}$ such that $\pi'(s) \neq \pi(s)$ and $(s, \pi(s)) \notin T^{\pi'}$ whose existence is guaranteed by Proposition 3.4. It is impossible to switch from $\pi'(s)$ to $\pi(s)$ hence the result. □

When $k = 2$, it is easy to see using Proposition 3.4 that two policies with exactly the same improvement states cannot exist. When $k > 2$, this is no longer the case. However, using Lemma 3.3, Mansour and Singh showed that there cannot be more than k^d policies with the same d improvement states in a PI-sequence (see Mansour and Singh (1999, Corollary 13)). In the following proposition, we use Proposition 3.4 to improve this bound to $(k-1)^d$. Note that this improvement is a crucial step to achieve tight bounds for Problem 3.2.

Proposition 3.11. *Given a pseudo-PI-sequence* $(\Pi, O, \mathcal{T})$ *and a set of states* $S \subseteq \mathcal{S}$ *of cardinality* d, *it holds that* O *contains at most* $(k-1)^d$ *policies* π *with* $S^{\pi} = S$.

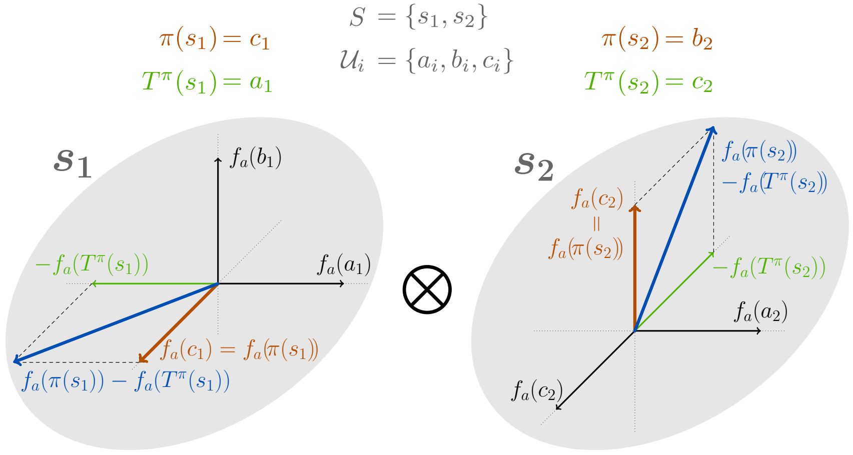


Figure 3.1: We illustrate the proof of Proposition 3.11 using an example MDP with $d = 2$ states in S and $k = 3$ actions per state. The actions at both states can be represented in a k-dimensional space V of which $f_a : \mathcal{U}_i \to \mathbb{R}^k$ defines the base vectors. Likewise, the policies can be represented in the (Kronecker) product of these spaces $W = V \otimes V$ of dimension k^n. Observe that the blue vectors $f_a(\pi(s_i)) - f_a(T^\pi(s_i))$ uniquely represent the pairs $(\pi(s_i), T^\pi(s_i))$ since $T^\pi(s_i)$ is always distinct from $\pi(s_i)$.

Proof. Given the supersequence O of the pseudo-PI-sequence, we consider its subsequence $\pi^{(1)} \preceq \cdots \preceq \pi^{(K)}$ such that $S^{\pi^{(i)}} = S \triangleq \{s_1, \ldots, s_d\}$ for all $1 \leq i \leq K$. We show that if the subsequence satisfies Proposition 3.4, then $K \leq (k-1)^d$. To this end, we first claim that the improvement sets of the policies of the subsequence can be assumed to be all well-defined. Indeed, for any policy $\pi^{(i)}$ of the subsequence, we can simplify its improvement set $T^{\pi^{(i)}}$ by keeping only a single (s, u) pair for every $s \in S^{\pi^{(i)}}$. This does not modify $S^{\pi^{(i)}}$ (that is, $\pi^{(i)}$ remains in the subsequence), nor does it imply the violation of Proposition 3.4.

Therefore, given a policy $\pi^{(i)}$ of the subsequence and a state $s \in S$, we can assume that there is exactly one action u such that $(s, u) \in T^{\pi^{(i)}}$, which we refer to as $T^{\pi^{(i)}}(s)$.

We represent an action $i \in \mathcal{U}$ as a k-dimensional base vector $f_a(i) \triangleq e_i$ of $V = \mathbb{R}^k$, where $e_i(j) = 1$ if $i = j$, 0 otherwise. Similarly, we represent policies as base vectors of the space $W = V^{\otimes d}$ of dimension k^d through the application:

$$f_p : \pi \longmapsto f_a(\pi(s_1)) \otimes \cdots \otimes f_a(\pi(s_d)),$$

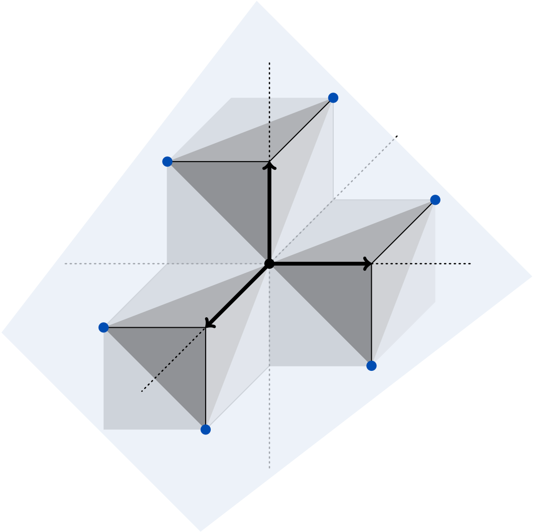


Figure 3.2: The light-blue plane intersects every blue dot and the origin. More precisely, in the k-dimensional space V (here represented with $k = 3$), all the vectors $f_a(\pi(s_i)) - f_a(T^\pi(s_i))$—the blue dots—that represent the possible pairs $(\pi(s_i), T^\pi(s_i))$ lie in the same $(k-1)$-dimensional subspace V_0—the light-blue plane.

where $\otimes$ stands for the Kronecker product. Finally, we represent pairs of policies and their improvement sets in a similar way in W through the application:

$$f_c : (\pi, T^\pi) \longmapsto \Big[f_a(\pi(s_1)) - f_a(T^\pi(s_1))\Big] \otimes \cdots \otimes \Big[f_a(\pi(s_d)) - f_a(T^\pi(s_d))\Big],$$
$$= f_p(\pi) \quad + \sum_{\substack{U \subseteq T^\pi \\ U \neq \emptyset}} (-1)^{|U|} \cdot f_p(\pi \oplus U).$$

These applications are illustrated on Figure 3.1.

We claim that the vectors $f_c\big(\pi^{(i)}, T^{\pi^{(i)}}\big)$ are linearly independent. Assume on the contrary that we have:

$$\sum_{i=1}^{K} \lambda_i \, f_c\left(\pi^{(i)}, T^{\pi^{(i)}}\right) = 0, \tag{3.1}$$

with not all λ_i being 0. Choose the first index i with non-zero λ_i. The corresponding term gives a non-zero coefficient to the base vector $f_p\big(\pi^{(i)}\big)$. But from Lemma 3.10, for all $j > i$ and all $U \subseteq T^{\pi^{(j)}}$, $\pi^{(i)} \neq \pi^{(j)} \oplus U$. Thus the base vector $f_p\big(\pi^{(i)}\big)$ never appears later in the series in (3.1) which can therefore not be null.

Additionally, the coordinates of $f_a(\pi(s_i)) - f_a(T^\pi(s_i)) \in V$ sum to 0 (in the standard base) for all $1 \le i \le d$ which means they lie in a subspace V_0 of V of dimension $k-1$, as illustrated by Figure 3.2. As a result,

$$f_c\left(\pi^{(i)}, T^{\pi^{(i)}}\right) \in W_0 = V_0^{\otimes d}.$$

The dimension of W_0 is $(k-1)^d$ implying this is the maximum number of linearly independent vectors $f_c\big(\pi^{(i)}, T^{\pi^{(i)}}\big)$. This translates to the desired upper bound. □

Note that the above result also holds for usual PI-sequences.

Proof of Theorem 3.9. The proof proceeds in two steps. First, we consider "small" improvement sets and show that there are at most $o\left(\frac{k^n}{n}\right)$ of them. Then we consider "large" improvement sets and show that PI explores at most $\frac{k}{k-1} \cdot \frac{k^n}{n} + o\left(\frac{k^n}{n}\right)$ of them because they jump many policies on the way.

Small improvement sets. We consider the small improvement sets T^π such that $|S^\pi| \le \frac{k-1}{k} \cdot n - f(n)$ with:

$$f(n) \triangleq \sqrt{n \log n}.$$

From Proposition 3.11, policies with the same set of improvement states S of cardinality d can appear at most $(k-1)^d$ times in a (pseudo-)PI-sequence, hence the number of small improvement sets can be expressed as follows:

$$\begin{aligned}\sum_{d=0}^{\left\lfloor \frac{k-1}{k} \cdot n - f(n) \right\rfloor} \binom{n}{d} (k-1)^d &= k^n \sum_{d=0}^{\left\lfloor \frac{k-1}{k} \cdot n - f(n) \right\rfloor} \binom{n}{d} \left(\frac{k-1}{k}\right)^d \left(\frac{1}{k}\right)^{n-d}, \\ &= k^n \cdot P\left[X \le \frac{k-1}{k} \cdot n - f(n)\right],\end{aligned}$$

where $X \sim \text{Bin}\left(n, \frac{k-1}{k}\right)$ follows a binomial distribution. Using Hoeffding's inequality (1963, Theorem 1), we have:

$$P\left[X \le n \cdot \left(\frac{k-1}{k} - \frac{f(n)}{n}\right)\right] \le e^{-2 \cdot \left(\frac{f(n)}{n}\right)^2 \cdot n} = \frac{1}{n^2}.$$

Therefore we have:

$$\sum_{d=0}^{\left\lfloor \frac{k-1}{k} \cdot n - f(n) \right\rfloor} \binom{n}{d} (k-1)^d \le k^n \cdot \frac{1}{n^2} = o\left(\frac{k^n}{n}\right).$$

Large improvement sets. We now consider the improvement sets T^π with the set of improvement states satisfying $|S^\pi| > \frac{k-1}{k} \cdot n - f(n)$. We show that these sets jump many policies on the way and hence we cannot have many of them in the (pseudo-)PI-sequence. Suppose that we have K such improvement sets in the sequence. Then, from Corollary 3.6, we jump at least $K \cdot \left(\frac{k-1}{k} \cdot n - f(n)\right)$ distinct policies. Since we cannot jump more that k^n policies, we have the following condition on K:

$$\begin{aligned} K &\leq \frac{k^n}{\frac{k-1}{k}n - f(n)} \\ &= \frac{k}{k-1} \cdot \frac{k^n}{n} \cdot \frac{1}{1 - \frac{k-1}{k}\sqrt{\frac{\log n}{n}}}, \\ &= \frac{k}{k-1} \cdot \frac{k^n}{n} \cdot \left(1 + O\left(\sqrt{\frac{\log n}{n}}\right)\right), \end{aligned}$$

hence $K \leq \frac{k}{k-1} \cdot \frac{k^n}{n} + o\left(\frac{k^n}{n}\right)$. □

To the best of our knowledge, the bound from Theorem 3.9 is the best known upper bound on the number of steps of a general run of PI, including the case where $k = 2$. An interesting perspective would be to extend the bound to cases where every state of the MDP have a different number of actions $k_1, ..., k_n$. Looking at our bound, we can guess that such a bound should be close to

$$2 \cdot \frac{\prod_{i=1}^n k_i}{n} + o\left(\frac{\prod_{i=1}^n k_i}{n}\right).$$

Note that the above developments have their counterpart in the framework of Two-Player Turn-Based Stochastic Games (2TBSGs) defined in Section 2.1.3. In fact, the policies explored by the main loop of the Strategy Iteration algorithm described in Algorithm 2 can be interpreted as a PI-sequence. All the properties about PI-sequences also hold in that case. In particular, the bound of Theorem 3.9 also holds to bound the number of steps of the main loop of Strategy Iteration.

3.3 The bound is tight for Problem 3.2

The following theorem shows that the upper bound from Theorem 3.9 is tight for Problem 3.2.

Theorem 3.12. *There exists a pseudo-PI-sequence of size* $\frac{k}{k-1} \cdot \frac{k^n}{n} + \omega\left(\frac{k^n}{n}\right)$.

Proof. We first build a sequence containing all the k^n policies that will play the role of the supersequence O for the pseudo-PI-sequence. Preliminarily, given any policy π of O, we define its (well-defined) improvement set T^π

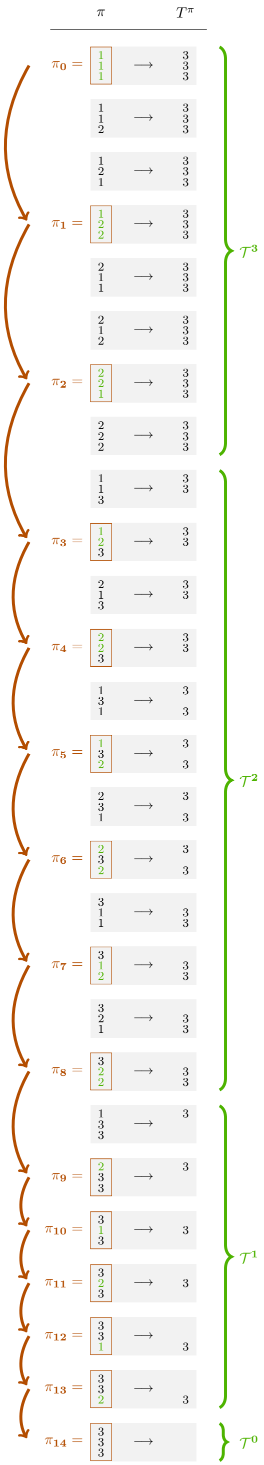


Figure 3.3: An example of a pseudo-PI-Sequence of size $\frac{k}{k-1} \cdot \frac{k^n}{n} + o\left(\frac{k^n}{n}\right)$ with its supersequence O for $n = k = 3$. Each gray box corresponds to a policy of the supersequence. We represent the improvement sets only through the prospective improving action for each state (action 3 for state s if $\pi(s) \neq 3$ or nothing, according to the construction). The red policies are the ones from the sequence Π from Definition 3.9. It can be checked that if some policy π_i is in $\mathcal{T}^d$, then d policies of the supersequence are jumped from π_i to π_{i+1} and it can be observed that the supersequence contains k^n elements and satisfies the claim of Proposition 3.4.

such that $(s, u) \in T^\pi$ iff $\pi(s) \neq k$ and $u = k$. Here action k can be thought of as some special action. Let $\mathcal{T}^d$ be the set of all policies π such that $|T^\pi| = d$. By definition, $\mathcal{T}^d$ contains all policies π such that $\pi(s) \neq k$ for exactly d different states s, hence $\binom{n}{d} \cdot (k-1)^d$ elements. We now order all k^n policies as a sequence by decreasing order of cardinality of their improvement sets, hence the policies in $\mathcal{T}^d$-sets with a large d come first in the sequence. The (total) ordering inside a given $\mathcal{T}^d$-set can be arbitrarily chosen. Given this ordering, notice that for any $\pi \prec \pi'$, if $S^\pi \subseteq S^{\pi'}$, then $S^\pi = S^{\pi'}$.

The sequence O obtained with the above construction satisfies the claim of Proposition 3.4. Indeed, let us choose any two policies of the sequence $\pi \prec \pi'$. First assume that $S^\pi \setminus S^{\pi'} \neq \emptyset$ and let $t \in S^\pi \setminus S^{\pi'}$. Then by construction, $\pi(t) \neq k = \pi'(t)$ and $(t, \pi(t)) \notin T^{\pi'}$ since $t \notin S^{\pi'}$, hence Proposition 3.4 is true in that case. If now $S^\pi \setminus S^{\pi'} = \emptyset$, then the ordering of the policies imposes that $S^\pi = S^{\pi'}$, as observed above. In that case, by construction $\pi(s) \neq k$ for all $s \in S^\pi$ and $\pi(s) = \pi'(s) = k$ for all $s \notin S^\pi$. Since $\pi \neq \pi'$, there must exist some state $t \in S^\pi$ such that $\pi(t) \neq \pi'(t)$. Furthermore by definition of $T^{\pi'}$, $(t, \pi(t)) \notin T^{\pi'}$ because $\pi(t) \neq k$, and the claim of Proposition 3.4 is true again.

At this point, we have built a supersequence for our PI-sequence that satisfies the claim of Proposition 3.4. Let us now select a subsequence Π of O while ensuring Corollary 3.6 as follows: we start from the first policy of the supersequence π_0, $i = 0$. Then at each step i, we jump $|T^{\pi_i}|$ elements in the sequence to select π_{i+1}. With this greedy procedure, we clearly ensure Corollary 3.6 and we pick at least $\frac{1}{d+1}|\mathcal{T}^d|$ policies from each $\mathcal{T}^d$-set, for a total number of hypothetical PI-steps of at least:

$$\begin{aligned}
&\sum_{d=0}^{n} \frac{1}{d+1} |\mathcal{T}^d|, \\
=&\sum_{d=0}^{n} \frac{1}{d+1} \binom{n}{d} (k-1)^d, \\
=&\frac{1}{n+1} \cdot \sum_{d=0}^{n} \binom{n+1}{d+1} \cdot (k-1)^d \cdot 1^{n-d}, \\
=&\frac{1}{k-1} \cdot \frac{1}{n+1} \cdot \left[\underbrace{\sum_{d=0}^{n+1} \binom{n+1}{d} \cdot (k-1)^d \cdot 1^{(n+1)-d}}_{=k^{n+1}} - 1 \right], \\
=&\frac{k}{k-1} \cdot \frac{k^n}{n} \cdot \left(1 - \frac{1}{k^{n+1}}\right) \cdot \left(1 - \frac{1}{n+1}\right), \\
=&\frac{k}{k-1} \cdot \frac{k^n}{n} \cdot (1 + \omega(1)),
\end{aligned}$$

which corresponds to our claim and matches the upper bound from Theorem 3.9. An example of a pseudo-PI-sequence constructed from the above procedure with $n = k = 3$ is given in Figure 3.3. □

Of course, the lower bound from Theorem 3.12 only holds for pseudo-PI-sequences which are less constrained than usual PI-sequences. Indeed, it can for instance be observed that the pseudo-PI-sequence constructed in Figure 3.3 cannot correspond to a real PI-run since for instance its supersequence does not satisfy Proposition 3.1. Therefore, obtaining better bounds than the one from Theorem 3.9 will require a more advanced analysis, as attempted in the next chapter.

3.4 The Acyclic Unique Sink Orientation of grids representation

Theorem 3.12 revealed that future improvements of our bound will require to take into account more of the combinatorial structure of PI-sequences. In this section, we describe how Problem 3.1 can be formulated in the framework of Acyclic Unique Sink Orientations of grids.

The idea of this approach is to represent the partial order of the policies of an MDP as an oriented graph whose nodes—the policies—are embedded in an n-dimensional grid and whose directed edges—that translate the domination relation—only connect neighboring policies (that is, that differ in only one state). (We here assume that there is never a tie, so we either have $\pi \prec \pi'$ or $\pi \succ \pi'$. This can be ensured by making infinitesimal perturbations in the initial problem or using a suitable tie-breaking rule.) It can be shown that the obtained graph must be *acyclic* and *unique sink*, that is, any sub-grid of dimension $d \leq n$ contains a unique vertex of maximum in-degree d (Gärtner *et al.*, 2005). This is why we call it an Acyclic Unique Sink Orientation of a grid[1] (or Grid AUSOs). Grid AUSOs were introduced by Gärtner *et al.* (2005) as a generalization of Acyclic Unique Sink Orientations of Cubes (Szabó and Welzl, 2001) when $k > 2$. They accurately characterize the structure of the partial order of any MDP and essentially all necessary conditions we know on PI-sequences can be derived from this framework, including Propositions 3.1 and 3.2 and their consequences described in this chapter.

More precisely, Grid AUSOs can be described as follows: take a Cartesian grid of dimension n, the number of states of the MDP. A policy can be represented by its actions at each state as a vector in $\{1, ..., k\}^n$ and it

[1] One could strengthen the conditions on the graph even a bit further by requiring the Holt-Klee condition as well (Gärtner *et al.*, 2005; Holt and Klee, 1999).

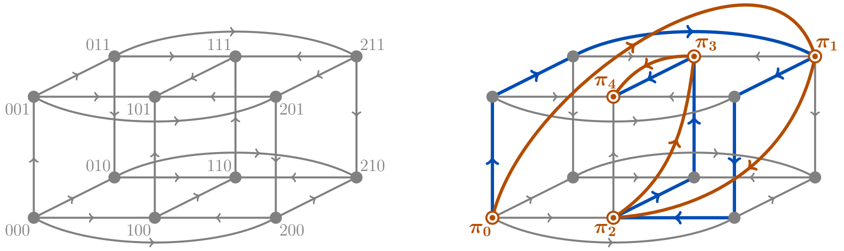


Figure 3.4: (Left) An example of a Grid AUSO that we could obtain for an MDP with 3 actions for the first state and 2 actions for the two other states. Here, 101 is the global sink that corresponds to the optimal policy. (Right) A possible run of PI on this Grid AUSO, starting from 000. The PI jumps are represented in red. Proposition 3.5 guarantees that there exists a directed path from π_{i-1} to π_i in the grid for all $0 < i < m$. This path is here depicted in blue.

thereby corresponds to a vertex of the grid. For every neighboring policies π, π', we draw a directed edge from π to π' if $\pi \prec \pi'$ (recall that neighboring policies are always comparable and that we assumed the absence of ties). We refer to the set of outgoing edges of the vertices (or policies) as their *outmap*. The obtained directed graph on the grid is guaranteed to be acyclic and unique sink. An example of a Grid AUSO is given in Figure 3.4A.

With this structure, PI-steps can be viewed as jumps in the grid as follows: from a policy π_i of the PI-sequence, the outgoing links at the corresponding vertex span a sub-grid. In general, the next vertex π_{i+1} chosen by PI can be any vertex of this sub-grid (except π_i), but in Howard's version, some antipodal vertex to π_i is chosen. A possible run of Howard's PI is illustrated on Figure 3.4B. This algorithm is also known as Bottom-Antipodal in the AUSO framework. Note that it is possible to design Cube AUSOs for which PI takes $\sqrt{2}^n$ steps (Schurr and Szabó, 2005) but to the best of our knowledge, it is unknown if this lower bound can be adapted to MDPs or 2TBSGs.

Interestingly, Propositions 3.5 and 3.4 allow a simple interpretation in Grid AUSOs. Proposition 3.5 requires that there exists a directed path in the grid connecting all pairs of policies π_{i-1} and π_i of the PI-sequence. Such a path is depicted in blue in Figure 3.4B. Regarding Proposition 3.4, it requires that any two comparable policies (that is, vertices of the grid that can be connected by a path) never share the same outmap. In fact, it is even a bit stronger as it requires the outmap of the dominated policy not to be "contained" in that of the dominating policy.

3.5 The relationship between Policy and Strategy Iteration

Recall that in Chapter 2, Section 2.1.3, we defined *Two-Player Turn-Based Stochastic Games* (2TBSGs) as a two-player generalization of MDPs. Player 1 is the minimizer (he tries to minimize the value vector) and is granted control over the states in $\mathcal{S}^1 \subseteq \mathcal{S}$, while Player 2 is the maximizer and controls the remaining states $\mathcal{S}^2 = \mathcal{S} \backslash \mathcal{S}^1$. The dynamics of the game is the same as an MDP except that the two players have opposite objectives regarding their policy (that we usually call *strategy*). To solve the 2TBSG, we aim to find an *equilibrium* strategy, that is, a strategy for which no player has the ability to improve his value vector by changing some of his actions. To find such an equilibrium, one can apply Strategy Iteration (SI), an algorithm that generalizes PI to 2TBSGs. Each iteration of SI is composed of two steps. First we apply a PI step to the strategy of Player 2 thereby increasing his value vector. Then we update Player 1's strategy to his best-response against Player 2. The process is then repeated until convergence to an equilibrium strategy. Theorem 2.5 guarantees that the value vector of Player 2 strictly increases after each iteration so that the process always terminates. Theorem 2.6 provides the stopping condition.

The solution of a 2TBSG can also be obtained by applying Policy Iteration on some appropriately designed AUSO. The way to construct this AUSO is based on how SI works. We label the vertices of the AUSO by the strategies of Player 2. For each strategy, we consider that Player 1 is playing his best-response. Therefore, to each vertex of the AUSO corresponds a single strategy $(\mathrm{br}_1(\pi^2), \pi^2)$ for the players, where $\mathrm{br}_1(\pi^2)$ is the best-response of Player 1 against Player 2's strategy π^2. Adjacent vertices, those for which the strategy of Player 2 differs by exactly one action, are connected by an edge oriented towards the most rewarding strategy of the two for Player 2. (We assume the absence of ties and otherwise make infinitesimal perturbations to the problem.) Theorem 2.5 guarantees that these orientations are well-defined and satisfy a similar statement as the one of Proposition 3.1. Theorem 2.6 ensures that a vertex is a sink iff it is an equilibrium strategy of the 2TBSG. From there, it is not hard to see that the resulting orientation must be AUSO. Moreover, PI applied to this AUSO explores exactly the same strategies as SI would explore if applied to the initial 2TBSG with the same starting strategy of Player 2, which can be stated as follows:

> *Strategy Iteration applied on a 2TBSG explores the same strategies as Policy Iteration does when applied on a Grid AUSO describing the partial order of the strategies of Player 2 in the 2TBSG when Player 1 always plays a best-response against him.*

A formal proof of this statement can be found, for instance, in Hunter (2007, Theorem 3.4). As a consequence, the bound of Theorem 3.9 also holds for SI.

Corollary 3.13. *The number of iterations of the outer loop of Strategy Iteration is bounded above by* $\frac{k}{k-1} \cdot \frac{k^n}{n} + o\left(\frac{k^n}{n}\right)$.

3.6 A simple case: MDPs with two actions per state

Analyzing the case where an MDP has only two actions per state is simpler than the general k-actions case in the same way that analyzing the structure of a cube is simpler than analyzing that of a grid. In the next three chapters, we restrict ourselves to that case where $k = 2$ and we sometimes reuse some of the results developed above. Therefore, identifying which parts of the above analysis become simpler is of interest.

The main simplification that comes with $k = 2$ is that there is only one way to switch an action in a state. Therefore, there is no need to specify to which action we switch, and we no longer need to distinguish the improvement set T^π of a policy π from its improvement states S^π since they both contain the same information. Moreover, subsets of T^π can only ever be well-defined since for every state $s \in S^\pi$, there is only one action other than $\pi(s)$.

Thanks to these observations, we can restate Proposition 3.4 as a kind of non-inclusion property of the improvement sets.

Proposition 3.4 (revisited with $k = 2$)**.** *If* $\pi \prec \pi'$, *then* $S^\pi \not\subseteq S^{\pi'}$.

In particular, this simplification implies that we cannot have comparable policies that share the same improvement states. Therefore, for $k = 2$, the statement of Proposition 3.11 becomes trivial. This means that without Proposition 3.11, we would have been able to show the bound from Theorem 3.9 for $k = 2$ but we would have obtained a higher constant factor for $k > 2$. In that case, we would not have been able to match the upper bound with the lower bound from Theorem 3.12.

Chapter 4

Order-Regular matrices: a powerful tool for the analysis of Policy Iteration

In the previous chapter, we explored the structure of the partial order of the policies of a Markov Decision Process (MDP) and of the strategies of a Two-Player Turn-Based Stochastic Game (2TBSG) through the angle of Acyclic Unique Sink Orientations (AUSOs). This fairly general structure provides an alternative framework to formulate and study a number of MDP and 2TBSG algorithms, including Policy Iteration (PI)[1]. In our quest for new bounds on the number of steps of PI in the previous chapter, we extracted a number of properties of AUSOs (in particular Propositions 3.4 and 3.5) and saw that these properties alone were not enough to obtain new bounds. In this chapter, we investigate the *Order-Regularity* condition (OR), a property that is specific to the sequence of policies explored by PI—the so-called PI-sequence—that disregards all the policies outside from this sequence. We will see that this specificity opens new possibilities. Like AUSOs were an abstraction of MDPs and 2TBSGs, the OR condition can be seen as an abstraction of the progress of PI in AUSOs.

First introduced by Hansen (2012), the idea of the OR condition is the following. Suppose we record the policies $\pi_0, \pi_1, ..., \pi_{m-1}$ of a PI-sequence as the rows of an $m \times n$ binary matrix. (If n is the number of states of the MDP and there are two available actions in each state, we can always write the policies as binary vectors of dimension n.) We can then translate the

[1] Because it is equivalent to PI, we also cover the Strategy Iteration algorithm for 2TBSGs, see Section 3.5.

AUSO property into a necessary combinatorial condition on this matrix—the OR condition—that we will precisely define in the next section. Since matrices that originate from AUSOs are guaranteedly OR, upper bounds on the maximum number of rows of an n-columns OR matrix directly extend to the number of steps of PI in an n-dimensional AUSO (and thus also in an n-states MDP or 2TBSG).

Using the OR formulation, Hansen and Zwick performed an exhaustive search on all OR matrices with up to $n = 6$ columns and reported the maximum number of rows (that is, the number of abstract iterations for PI) each time: $2, 3, 5, 8, 13, 21$. Based on these empirical observation, they conjectured that the maximum number of steps of PI should follow the *Fibonacci sequence* (Hansen, 2012). Confirming this conjecture for $n = 7$ has been claimed to be a hard computational challenge. It was introduced as January 2014's *IBM Ponder This* Challenge. Proving the conjecture in general would provide an $O(1.618^n)$ upper bound on the number of iterations of PI, a quasi-identical bound as that of the Fibonacci Seesaw algorithm introduced by Szabó and Welzl (2001). Regarding lower bounds, nothing better than the $\Omega(1.4142^n)$ bound from Schurr and Szabó (2005) originating from AUSOs was known prior to our work.

In this chapter, our first contribution is to disprove Hansen and Zwick's conjecture by performing an exhaustive search for $n = 7$. We obtained a maximum number of rows that is lower than the expected Fibonacci number, which still allows the Fibonacci sequence as a possible upper bound. Our second contribution is to (exponentially) improve Schurr and Szabó's $\Omega(1.4142^n)$ lower bound to $\Omega(1.4269^n)$, yet only in the framework of OR matrices, thereby raising the question whether this bound also holds for AUSOs. Finally, the key ideas behind our two results relied on our ability to build large matrices satisfying OR-like conditions, which required substantial computational refinements.

The chapter is organized as follows. In Section 4.1 we formulate the OR condition together with some key elements for its analysis. In Section 4.2, we discuss Hansen and Zwick's conjecture as well as an upper bound on the size of OR matrices. Section 4.3 establishes step by step our new lower bound on the number of rows of OR matrices, starting from Schurr and Szabó's construction. Then, since our results heavily rely on our ability to build large matrices, we describe in Section 4.4 the different ideas that we combined in order to speed up the existing computational methods. We close this chapter in Section 4.5 with two intriguing approaches that could inspire new techniques for the analysis.

In the two last chapters, we will investigate how we can extend our lower bounds to Acyclic Unique Sink Orientations as well.

4.1 Definitions and preliminaries

Let $\{\pi_0, \pi_1, ..., \pi_{m-1}\}$ be a PI-sequence generated by an n-dimensional Cube AUSO (which includes MDPs and 2TBSGs with n states and two actions per state). We can write π_i as a vector of $\{0,1\}^n$. Let $A \in \{0,1\}^{m \times n}$ be the matrix whose rows correspond to the ordered policies of the PI-sequence. From Theorem 3.8 that regulates the structure of the PI-sequence, or using Hansen's original argument (2012), we can extract the following necessary condition on A.

Definition 4.1 (Order-Regularity)**.** We say that $A \in \{0,1\}^{m \times n}$ is Order-Regular (OR) whenever for every pair of rows i, j of A with $1 \leq i < j \leq m$, there exists a column k such that

$$A_{i,k} \neq A_{i+1,k} = A_{j,k} = A_{j+1,k}. \tag{4.1}$$

We may have $j + 1 = m + 1$. In that case, we use the convention that $A_{m+1,k} = A_{m,k}$.

With other words, for all pairs (i, j), there exists a column k in A such that at the entries $i, i+1, j, j+1$ in this column we see either $0, 1, 1, 1$ or $1, 0, 0, 0$. Another possible reading of the OR condition in terms of Policy Iteration is as follows: at any iteration, some changes are made to the entries of the current vertex. It must always be assumed in future iterations that at least one of these changes was "right". Note that a variant of the OR condition was already formulated by Madani in his Ph.D. thesis, see Madani (2000, Section 4.6).

Using Theorem 3.8, we now show that any matrix A that corresponds to a PI-sequence must satisfy the OR condition, and therefore that an upper bound on its number of rows yields an upper bound on the number of steps of PI.

Theorem 4.1. *Let $A \in \{0,1\}^{m \times n}$ be a matrix whose rows correspond to the ordered policies $\pi_0, \pi_1, ..., \pi_{m-1}$ of a PI-sequence generated by an n-dimensional Cube AUSO. Then A must be Order-Regular.*

Proof. We use an appropriate variation[2] of Theorem 3.8 that can be stated as follows. Let $\pi \prec \pi'$. Then for all $U \in S^{\pi}_{\preceq}$ and all $U' \in S^{\pi'}_{\succ}$, it holds that $\pi \oplus U \neq \pi' \oplus U'$.

[2] It is a variation regarding two aspects. First we use the improvement states instead of the improvement sets that contain state-action pairs. This is allowed for MDPs with two actions per state because the action to switch to in a state is implicitly defined as the other available action. Therefore, the switching operator "$\oplus$" remains well-defined. The other variation is that we use $S^{\pi'}_{\succ}$ instead of $S^{\pi'}_{\succeq}$ for which Theorem 3.8 still applies.

Let $1 \leq i \leq m$ be a row index of A. By definition of PI and the matrix A, we can write the sets $S_{\succ}^{\pi_{i-1}}$ and $S_{\preceq}^{\pi_{i-1}}$ as:

$$\begin{aligned} S_{\succ}^{\pi_{i-1}} &= \{k : \pi_{i-1}(k) \neq \pi_i(k)\} = \{k : A_{i,k} \neq A_{i+1,k}\} \\ S_{\preceq}^{\pi_{i-1}} &= \{k : \pi_{i-1}(k) = \pi_i(k)\} = \{k : A_{i,k} = A_{i+1,k}\}. \end{aligned}$$

(Here note that the i-th row of A corresponds to the policy with index $i-1$ in the PI-sequence. Also note that we now use k to designate states.)

Let us choose any row indices i and j satisfying $1 \leq i < j \leq m$. Theorem 3.1 ensures that $\pi_{i-1} \prec \pi_{j-1}$. Now suppose by contradiction that $\pi_{i-1}(k) = \pi_{j-1}(k)$ for all $k \notin (S_{\preceq}^{\pi_{i-1}} \cup S_{\succ}^{\pi_{j-1}})$. Let $K = \{k : \pi_{i-1}(k) \neq \pi_{j-1}(k)\}$. With our assumption, we must have $K \subseteq (S_{\preceq}^{\pi_{i-1}} \cup S_{\succ}^{\pi_{j-1}})$. Let U and U' form a partition of K such that $U \subseteq S_{\preceq}^{\pi_{i-1}}$ and $U' \subseteq S_{\succ}^{\pi_{j-1}}$. In that case, $\pi_{i-1} \oplus U = \pi_{j-1} \oplus U'$, which violates Theorem 3.8. Therefore, there must exist a state k for which (1) $k \notin S_{\preceq}^{\pi_{i-1}}$, (2) $k \notin S_{\succ}^{\pi_{j-1}}$ and (3) $\pi_{i-1}(k) \neq \pi_{j-1}(k)$. In terms of the matrix A, this implies that there exists a column k such that (1) $A_{i,k} \neq A_{i+1,k}$, (2) $A_{j,k} = A_{j+1,k}$ and (3) $A_{i,k} \neq A_{j,k}$. These three conditions yield the Order-Regularity condition. □

Note that the OR condition is invariant under permutation or negation of matrix columns. By *negating* a column, we mean changing all 0 entries to 1 and vice versa. Also observe that each pair (i, j) can be considered as a constraint to be satisfied by some column of the matrix. We will say that A *satisfies* a constraint (i, j) if it satisfies condition (4.1) for some column k.

Our aim is to find bounds on the number of rows of Order-Regular matrices. It can easily be seen that all the rows must be different, leading to a trivial 2^n upper bound. The following tool will help us reach better bounds.

Definition 4.2 (Constraint space)**.** We introduce the *constraint space* as a visualization tool to relate a binary matrix with the Order-Regularity condition. To any pair (i, j) for which condition (4.1) is required (i.e. for all i, j such that $1 \leq i < j \leq m$), we associate a unit square of the Cartesian grid centered at coordinate (i, j). Whenever we want to emphasize which part of the matrix satisfies which part of the constraint space, we use a matching coloring on some subset of entries of the matrix and the corresponding squares of the constraint space.

The constraint space can be used in several ways. When considering a matrix that is not Order-Regular, it allows us to visualize which constraints (i, j) are satisfied by the matrix and which ones are not, and possibly detect patterns. It also allows to visualize how each column of a matrix (or even any given part of it) contributes in achieving Order-Regularity as illustrated by Figure 4.1.

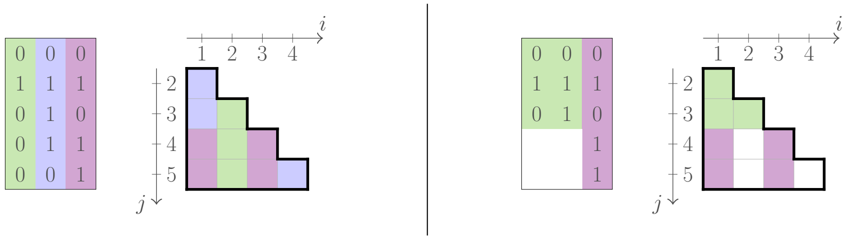


Figure 4.1: (Left) Each column of the matrix and the constraints it satisfies are associated with a color. (Right) The contribution of the red and blue blocks are indicated while the unfilled part of the matrix is disregarded.

Most binary vectors encountered in our constructions are repetitions of simple patterns, for which we introduce a compact notation.

Definition 4.3 (Patterns)**.** A *pattern* $\left\{ {A \atop B} \right\}_M$ is a matrix composed of M copies of A or B put below each other in alternation, starting from A. Here M is called the size of the pattern. The matrices A and B are assumed to have the same number of columns. For example, $\left\{ {0 \atop 1} \right\}_5 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T$. We sometimes omit the size of the pattern if it is clear from the context.

The following operation is also frequently used in our constructions.

Definition 4.4 (Extension)**.** Let A be a binary matrix and let $\widetilde{A}$ be obtained from A by negating some of its columns such that the first row of $\widetilde{A}$ is identical to the last row of A. We call "the M-*extension* of A" the construction of a matrix A' composed of M alternating copies of A and $\widetilde{A}$ such that:

$$A' = \left\{ \begin{matrix} A \\ \widetilde{A} \end{matrix} \right\}_M = \begin{bmatrix} A \\ \widetilde{A} \\ \vdots \\ A \\ \widetilde{A} \\ A \end{bmatrix}.$$

Notice that if A is OR, then $\widetilde{A}$ is OR too. Furthermore, the first row of A is also identical to the last row of $\widetilde{A}$. The effect of an extension is illustrated in Figure 4.2 with A an OR matrix.

The following straightforward lemma will be of key importance for our constructions.

Lemma 4.2. *Let A' be the M-extension of an OR matrix A with m rows. Then any constraint (i, j) such that $(s-1) \cdot m < i < j \leq s \cdot m$ for some integer $1 \leq s \leq M$ (that is, i and j are within one block) is satisfied by A'.*

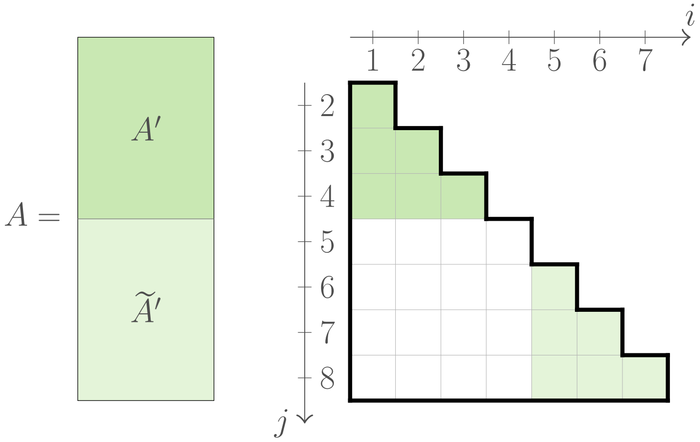


Figure 4.2: An illustration of the effect of a 2-extension with a 4-rows matrix A. If the last row of A had been different from the first row of $\widetilde{A}$, then the row of $j = 4$ of the constraint space would not have been completely colored.

Notice that if the appropriate columns of $A \in \{0,1\}^{m\times n}$ had not been negated to obtain $\widetilde{A}$, then Lemma 4.2 would not have been guaranteed when $j = s \cdot m$ for any $1 \leq s < M$. This is because the convention $A'_{s\cdot m,k} = A'_{s\cdot m+1,k}$ from Definition 4.1 must be ensured for all columns at the connection between the different A and $\widetilde{A}$ blocks.

4.2 Upper bounds and the Fibonacci conjecture

In their works, Hansen and Zwick have performed an exhaustive search on every possible Order-Regular matrix with up to $n = 6$ columns and proposed the following conjecture that matches their observations perfectly.

Conjecture 4.3 (Hansen and Zwick, (2012))**.** *The maximum number of rows of an n-column Order-Regular matrix is given by F_{n+2}, the $(n+2)$nd Fibonacci number.*

For recall, the Fibonacci numbers are related to the golden ratio $\varphi = (\sqrt{5}+1)/2 \approx 1.618$ in such a way that $F_n = \left[\frac{\varphi^n}{\sqrt{5}}\right]$, where $[\cdot]$ stands for "the closest integer to". Therefore, $F_n \sim \varphi^n \approx 1.618^n$. Note that except for $n = 5$, the extremal matrix found (that is, the matrix with the most rows) was always unique, up to symmetry[3]. In Table 4.1, we show these extremal matrices for $n = 3$ and 4.

0	0	0
1	1	1
0	0	1
0	1	1
0	1	0

0	0	0	0
1	1	1	1
0	0	0	1
0	1	1	1
0	0	1	0
0	1	1	0
0	1	0	0
1	1	0	0

Table 4.1: The unique extremal Order-Regular matrices for 3 and 4 columns.

In Section 4.4, we develop an algorithm to search for large OR matrices, with a special attention given to speed. Using this algorithm, we were able to perform an exhaustive search on all OR matrices with $n = 7$ columns. The largest matrices we found had only 33 rows, hence the following Theorem.

Theorem 4.4. *For $n = 7$, there exist no Order-Regular matrix with $34 = F_{n+2}$ rows and therefore Conjecture 4.3 is false.*

Theorem 4.4 indicates that the Fibonacci sequence provides at best an upper bound on the number of rows of an OR matrix. Its proof is given by the computer-aided exhaustive search for $n = 7$. Our implementation of the algorithm described in Section 4.4 is available at `http://sites.uclouvain.be/ORsearch`.

The proof of Theorem 4.4 relies on the ability of our code to actually perform an exhaustive search in the space of all 7-columns OR matrices. To allow verifying this statement, we provide in Section 4.4 a detailed description of the design of the code and of the underlying algorithms. Note that, as a sanity check, we verified that our program correctly recovers all the extremal OR matrices for $n = 1$ to 6. To go even further, it would be interesting to find a correctness certificate for the code or, if not possible, to use a formal proof language like Coq (Bertot and Castéran, 2013) to assess its correctness.

Although Theorem 4.4 denies the Fibonacci sequence as an exact fit on the number of rows of extremal OR matrices, it still suggests $F_{n+2} = O(\varphi^n)$ as a possible upper bound. Note that for small values of n, this bound is surprisingly close to our $(2 + o(1)) \cdot \frac{2^n}{n}$ upper bound from Theorem 3.9 with $k = 2$[4], although it is asymptotically much lower. Moreover, such a bound would be a significant improvement over the best proven upper bound on the number of rows of OR matrices which is given by the following proposition.

[3] Symmetric matrices can be obtained through column permutations and/or the negation of some columns.

[4] For $n = 1, ..., 7$, compare $2, 3, 5, 8, 13, 21, 34$ with $4, 4, 5.3, 8, 12.8, 21.3, 36.6$.

Proposition 4.5. *Let $A \in \{0,1\}^{m\times n}$ be an Order-Regular matrix. Then $m \leq 2^{n-1}+1$.*

To derive the bound of Proposition 4.5, let us relax the OR condition.

Definition 4.5. We say that a matrix $A \in \{0,1\}^{m\times n}$ is Order-Regular w.r.t. the relaxed Condition (4.2) whenever for every pair of rows i, j of A with $1 \leq i < j \leq m$, there exists a column k such that

$$A_{i,k} \neq A_{i+1,k} = A_{j,k}. \tag{4.2}$$

Of course, an OR matrix must also be OR w.r.t. the relaxed Condition (4.2). Let us now prove Proposition 4.5.

Proof of Proposition 4.5. First observe that further relaxing Condition 4.2 to $A_{i,k} \neq A_{j,k}$ expresses the fact that every row of A must be distinct. Furthermore, from Condition 4.2, it follows that for every pair of rows i, j with $2 \leq i < j \leq m$, we can never have $A_{i,k} \neq A_{j,k}$ for all columns k. This means that, except for the first row, any row r excludes some other potential row $\bar{r}$ to be in the matrix. Therefore, A can at most contain its first row and $\frac{2^n}{2}$ other rows. □

Interestingly, we will now see for the rest of this section that the bound from Proposition 4.5 can be attained for matrices that are OR w.r.t. the relaxed Condition (4.2). Even though insightful, the reading of the following developments is not required to follow the rest of this chapter.

Definition 4.6 (Fused extension)**.** Let A be a binary matrix and let $\widetilde{A}$ be obtained from A by negating (that is, "switching all entries from 0 to 1 or vice versa") some of its columns such that the first row of $\widetilde{A}$ is identical to the last row of A. We call "the *fused extension* of A" a matrix A' defined as:

$$A' = \begin{bmatrix} A \\ {}^{-}\widetilde{A} \end{bmatrix}$$

where ${}^{-}\widetilde{A}$ is the matrix $\widetilde{A}$ without its first row. We can understand a fused extension as a usual extension where the last row of A is somehow fused with the first row of $\widetilde{A}$. The effect of a fused extension is illustrated in Figure 4.3 with A an OR matrix w.r.t. Condition (4.2).

Fused extensions are the natural way of transposing the concept of extensions to OR matrices w.r.t. Condition (4.2). The result of Lemma 4.2 becomes the following.

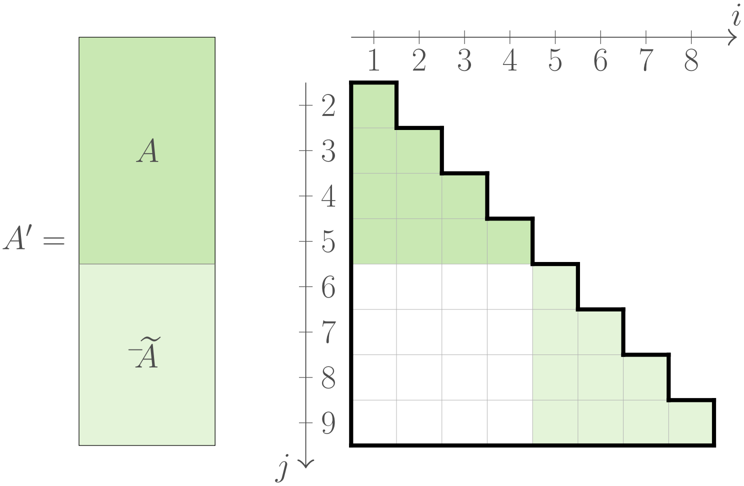


Figure 4.3: An illustration of the effect of a fused extension with a 5-rows matrix A that is OR w.r.t. Condition (4.2).

Lemma 4.6. *Let A' be the fused extension of an OR matrix A with m rows, w.r.t. Condition (4.2). Then any constraint (i, j) such that $1 \leq i < j \leq m$ or $m \leq i < j \leq 2m - 1$ is satisfied by A'.*

The following construction describes a family of matrices that attains the bound from Proposition 4.5.

Construction 4.1. Let $A^{(1)} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ and $A^{(2)} = \left[\begin{smallmatrix}0&0\\1&1\\0&1\end{smallmatrix}\right]$. We inductively build a matrix $A^{(n)} \in \{0,1\}^{m_n \times n}$, $n \geq 3$, as follows:

$$A^{(n)} = \left[\begin{array}{c|c} A^{(n-1)} & \left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}_{m_{n-1}} \\ \hline -\widetilde{A}^{(n-1)} & \left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}_{m_{n-1}-1} \end{array}\right]$$

The first $n - 1$ columns of $A^{(n)}$ correspond to a fused extension of $A^{(n-1)}$. Note that for $n \geq 2$, m_n is always odd.

Example 4.1. We illustrate Construction 4.1 for $n = 4$. We start from the matrix $A^{(3)} \in \{0,1\}^{m_3 \times 3}$, with $m_3 = 2^{3-1} + 1 = 5$, which can be checked to be OR w.r.t. Condition (4.2) and we define $\widetilde{A}^{(3)}$ to be $A^{(3)}$ where the last

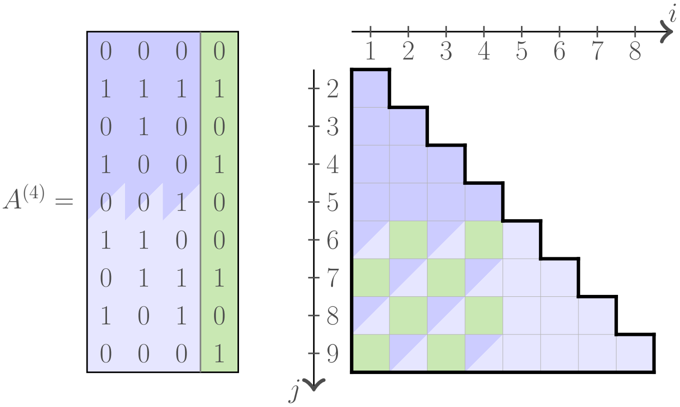

$$A^{(4)} = \begin{array}{|ccc|c|}\hline 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline\end{array}$$
i
1
2
3
4
5
6
7
8
j
2
3
4
5
6
7
8
9

Figure 4.4: (Left) To build the three first columns of $A^{(4)}$, we make a fused extension of $A^{(3)}$. We then add a fourth column to fill the constraints that remained to be satisfied. (Right) The constraint space of $A^{(4)}$. The blue squares are satisfied by the first three columns of $A^{(4)}$ and the green squares are satisfied by its last column.

column has been negated so that the first row of $\widetilde{A}^{(3)}$ equals the last row of $A^{(3)}$.

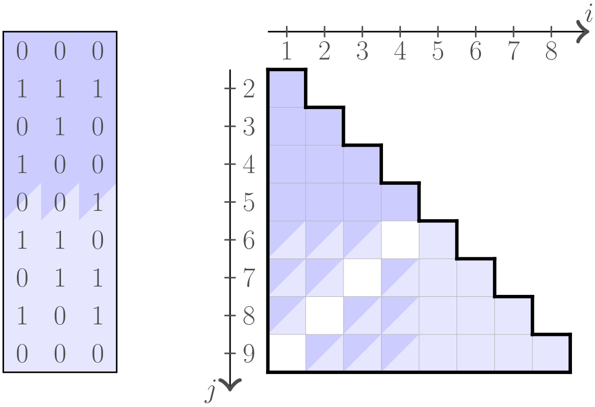
$$A^{(3)} = \begin{array}{|ccc|}\hline 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \hline\end{array} \qquad \widetilde{A}^{(3)} = \begin{array}{|ccc|}\hline 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \\ \hline\end{array}$$

We then build $A^{(4)}$ as shown in Figure 4.4. Using the constraint space on the right of the figure, it is easy to check that $A^{(4)}$ is indeed OR w.r.t. Condition (4.2).

Interestingly, even if we do not add the fourth column, we already cover almost the whole constraint space, as Figure 4.5 illustrates. The only constraints that remain unfilled are the ones such that $i + j = m_4 + 1$. This fact remains true for other values of n.

We now give a proof that the matrices defined in Construction 4.1 are indeed Order-Regular.

Lemma 4.7. *The matrices $A^{(n)}$ obtained from Construction 4.1 are OR w.r.t. Condition (4.2).*



$$\begin{array}{ccc} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{array}$$

Figure 4.5: The first three columns of $A^{(4)}$ almost cover the whole constraint space by themselves.

Proof. $A^{(1)}$ and $A^{(2)}$ can easily be checked to be OR w.r.t. Condition (4.2). Let us show that $A^{(n)}$ is also OR w.r.t. Condition (4.2), assuming that $A^{(n-1)}$ is so. We decompose the set of constraints that must be satisfied in three cases.

Claim 1. *The first $n-1$ columns of $A^{(n)}$ satisfy every constraint (i,j) where $1 \leq i < j \leq m_{n-1}$ or $m_{n-1} \leq i < j \leq 2m_{n-1}-1$.*

Claim 1 is a direct consequence from Lemma 4.6 since the first $n-1$ columns of $A^{(n)}$ are simply a fused extension of the matrix $A^{(n-1)}$ which has m_{n-1} rows by definition.

Claim 2. *The first column of $A^{(n)}$ satisfies every constraint (i,j) where $1 \leq i < m_{n-1} < j \leq 2m_{n-1}-1$ such that $i+j$ is an* odd *number.*

First, we show that the first column of $A^{(n)}$ is the simple pattern $\left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}_{m_n}$. It is indeed true for $n = 1$. Assume it is true for $n-1$. From Construction 4.1, the first column of $A^{(n)}$ is the fused extension of the first column of $A^{(n-1)}$, namely the fused extension of $\left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}_{m_{n-1}}$, that is $\left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}_{m_n}$.

Let (i,j) be one of the constraints of interest. From the pattern we identified, we have $A^{(n)}_{i,1} \neq A^{(n)}_{i+1,1}$ and since $i+j$ is odd, we also have $A^{(n)}_{i+1,1} = A^{(n)}_{j,1}$ which ensures Condition (4.2).

Claim 3. *The last column of $A^{(n)}$ satisfies every constraint (i,j) where $1 \leq i < m_{n-1} < j \leq 2m_{n-1}-1$ such that $i+j$ is an* even *number.*

The last column of $A^{(n)}$ is designed such that for all i, $1 \leq i < m_{n-1}$, we have $A^{(n)}_{i,n} \neq A^{(n)}_{i+1,n}$ and such that for all i,j, $1 \leq i < m_{n-1} < j \leq$

$2m_{n-1} - 1$, we have $A_{i,n}^{(n)} = A_{j,n}^{(n)}$ whenever $i + j$ is even. Therefore, under the assumptions made in the claim, Condition (4.2) is satisfied.

With these three cases, we have covered all possible pairs (i, j) and hence, $A^{(n)}$ is indeed OR w.r.t. Condition (4.2). □

Proposition 4.8. *For any number of columns n, there always exists a matrix A with $2^{n-1} + 1$ rows that is OR w.r.t. Condition (4.2).*

Proof. Such a matrix is given for instance by $A^{(n)}$ in Construction 4.1. Indeed, from Lemma 4.7, $A^{(n)}$ is OR w.r.t. Condition (4.2) and its number of rows satisfies $m_n = 2m_{n-1} - 1$ and $m_1 = 2$ which yields $m_n = 2^{n-1} + 1$. □

Remark 4.1. Using the same model as the relaxation from Definition 4.5, we may think of two other meaningful relaxations where we replace Condition (4.2) by $A_{i,k} \neq A_{j,k} = A_{j+1,k}$ or by $A_{i,k} \neq A_{i+1,k}$ and $A_{j,k} = A_{j+1,k}$. Interestingly, the second case is equivalent to requiring Proposition 3.4 as only constraint on the PI-sequence. In both cases it is possible to show that 2^n stands as a tight upper bound.

4.3 A new lower bound

This section is organized as follows. First, we show a simple construction that allows to build OR matrices with n columns and $\Omega(\sqrt{2}^n)$ rows. Then we detail our construction to beat this bound as follows:

1. we introduce Strongly Order-Regular matrices, a refinement of OR matrices that we will need for our construction and improve on the $\sqrt{2}$ growth rate of the number of rows from the simple construction;

2. we describe and prove the heart of our construction and obtain a first improvement over Schurr and Szabó's bound in the setting of OR matrices;

3. we finally add an additional refinement to our construction that allows us to improve our bound even a bit further to our final bound.

4.3.1 A family of Order-Regular matrices with $\Omega(\sqrt{2}^n)$ rows

The following construction provides Order-Regular matrices for arbitrarily large n with $m = \Omega\big(\sqrt{2}^n\big)$.

Construction 4.2. We recursively build a matrix $A^{(\ell)}$ as follows:

$$A^{(\ell)} = \left[\begin{array}{c|cc} A^{(\ell-1)} & \left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\} & \left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\} \\ \hline \widetilde{A}^{(\ell-1)} & \left\{\begin{smallmatrix}1\\1\end{smallmatrix}\right\} & \left\{\begin{smallmatrix}0\\0\end{smallmatrix}\right\} \end{array}\right]$$

where $A^{(1)} = [\begin{smallmatrix}0\\1\end{smallmatrix}]$ and where $\widetilde{A}^{(\ell-1)}$ is obtained from $A^{(\ell-1)}$ by negating some of its columns such that the first row of $\widetilde{A}^{(\ell-1)}$ is identical to the last row of $A^{(\ell-1)}$. The size of all the patterns used in the construction is $2^{\ell-1}$ and the resulting matrix $A^{(\ell)}$ has $n_\ell = 2\ell - 1$ columns and $m_\ell = 2^\ell$ rows. A matching construction is given by Schurr and Szabó (2005) in the framework of Acyclic Unique Sink Orientations.

Lemma 4.9. *Matrices obtained from Construction 4.2 are Order-Regular and satisfy $m = \Omega(\sqrt{2}^n)$ with m and n its number of rows and columns respectively.*

Proof. We prove the lemma by induction on ℓ. Clearly $A^{(1)}$ is OR (only one (i, j) pair to check). We show that if $A^{(\ell-1)}$ is OR, then Order-Regularity follows for $A^{(\ell)}$.

First, observe that the left part of $A^{(\ell)}$ is a 2-extension of $A^{(\ell-1)}$. Using Lemma 4.2, we get all constraints (i, j) satisfied when either $1 \leq i < j \leq 2^{\ell-1}$ or $2^{\ell-1} + 1 \leq i < j \leq 2^\ell$. The remaining constraints, i.e. those such that $1 \leq i \leq 2^{\ell-1}$ and $2^{\ell-1}+1 \leq j \leq 2^\ell$, are satisfied by the two last columns of $A^{(\ell)}$. Indeed, if i is odd, then choosing k to be the first of the two extra columns ensures condition (4.1) for all $2^{\ell-1} + 1 \leq j \leq 2^\ell$. The same goes with even i's and the second of the two columns. This reasoning is illustrated by Figure 4.6. Furthermore, the matrix $A^{(\ell)}$ satisfies $m_\ell = \sqrt{2}^{n_\ell+1}$. □

Using Construction 4.2, we have a way of building OR matrices satisfying $m = \Omega(\sqrt{2}^n)$. In the next subsections, we show how we can improve this bound.

4.3.2 Building blocks

Similarly to the above Construction 4.2, our construction starts with a building block that we will use to trigger the recursion. We require the following Strong Order-Regularity condition on the building block which is a restriction of the Order-Regularity condition.

Definition 4.7 (Strong Order-Regularity)**.** We say that $B \in \{0, 1\}^{m\times n}$ is Strongly Order-Regular (SOR) whenever

$$A^{(3)} = \begin{array}{|ccc|cc|} \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

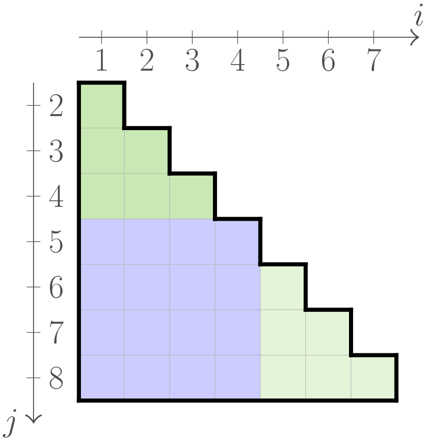


Figure 4.6: One step from Construction 4.2 is illustrated. The green matrices are glued together and ensure the top and right constraints in the constraint space. The two additional blue columns take care of the remaining square.

(1) for every pair of rows i, j of B with $1 \leq i < j \leq m$, there exists a column k_1 such that:

$$B_{i,k_1} \neq B_{i+1,k_1} = B_{j,k_1} = B_{j+1,k_1} \tag{4.3}$$

(the original Order-Regularity condition);

(2) for every pair of rows i, j of B with $1 \leq i$ and $i+1 < j \leq m$, there exists a column k_2 (necessarily different from k_1) such that:

$$B_{i,k_2} \neq B_{i+1,k_2} \neq B_{j,k_2} = B_{j+1,k_2}. \tag{4.4}$$

Again, we choose the convention that $B_{m+1,k} = B_{m,k}$.

In other words, at the entries $i, i+1, j, j+1$ we now ask for one column k_1 at which we observe either $0, 1, 1, 1$ or $1, 0, 0, 0$ and for another column k_2 at which we observe either $0, 1, 0, 0$ or $1, 0, 1, 1$. Clearly, this second column cannot exist when $j = i+1$, hence we do not ask for its existence in that case. We say that a matrix *doubly-satisfies* a constraint (i, j) if there exists a column k_1 that verifies (4.3) and a column k_2 that verifies (4.4) for that constraint. (For an SOR matrix, every constraints such that $1 \leq i$ and $i+1 < j \leq m$ is doubly-satisfied.) An SOR matrix with 8 columns and 33 rows is given in Figure 4.7.

$$B = B^{(1)} = \begin{array}{|cccccccc|}
\hline
0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1\\
0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&0\\
1&0&1&1&1&1&1&1\\
0&0&0&0&0&1&0&0\\
1&0&0&1&1&1&1&1\\
0&1&0&0&1&0&0&0\\
1&0&0&1&1&1&1&0\\
0&0&1&0&1&0&0&0\\
1&0&0&1&1&0&1&0\\
0&1&1&1&1&0&0&0\\
1&0&0&1&0&0&0&1\\
0&1&0&1&1&1&0&0\\
1&0&0&1&0&0&0&0\\
0&1&0&1&1&0&0&0\\
1&0&0&1&1&0&0&1\\
0&0&0&0&1&1&0&0\\
0&0&0&1&1&0&1&1\\
0&0&1&1&0&1&0&0\\
0&0&0&1&1&0&1&0\\
0&0&1&1&1&1&0&0\\
0&0&0&1&1&0&0&1\\
0&0&0&1&0&1&0&0\\
0&0&0&1&1&1&1&1\\
0&0&0&0&0&1&0&1\\
0&1&0&1&0&1&1&1\\
0&0&1&0&0&1&1&1\\
0&1&0&0&0&1&1&1\\
1&1&1&0&0&1&1&1\\
0&1&1&0&0&0&1&1\\
0&1&1&0&0&1&1&1\\
\hline
\end{array}$$

Figure 4.7: The 33×8 Strongly Order-Regular building block that we use to obtain our first improvement to the $\Omega\big(\sqrt{2}^n\big)$ lower bound.

4.3.3 Blowing up

We now provide our main construction that enables us to improve the bound. We start by describing the components of each iterate of the construction. Then we show that it indeed generates Order-Regular matrices and conclude with the resulting new lower bound.

Construction 4.3. Let $B = B^{(1)} \in \{0,1\}^{M \times N}$ be the building block for the construction. We inductively build a matrix $B^{(\ell)} \in \{0,1\}^{m_\ell \times n_\ell}$ as the merging of three blocks:

$$B^{(\ell)} = \begin{bmatrix} C^{(\ell)} & D^{(\ell)} & E^{(\ell)} \end{bmatrix}.$$

The blocks $C^{(\ell)}, D^{(\ell)}$ and $E^{(\ell)}$, $\ell \geq 2$, are defined as follows.

- The $C^{(\ell)}$ block is composed of M copies of the previous iterate glued together:

$$C^{(\ell)} \triangleq \left\{\begin{matrix} B^{(\ell-1)} \\ \widetilde{B}^{(\ell-1)} \end{matrix}\right\}_M = \begin{bmatrix} B^{(\ell-1)} \\ \widetilde{B}^{(\ell-1)} \\ \vdots \\ B^{(\ell-1)} \\ \widetilde{B}^{(\ell-1)} \\ B^{(\ell-1)} \end{bmatrix}.$$

- The $D^{(\ell)}$ block expands the building block B in the following way:

$$D^{(\ell)} \triangleq \begin{bmatrix} d^{1,1} & d^{1,2} & \dots & d^{1,N} \\ d^{2,1} & d^{2,2} & \dots & d^{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ d^{M,1} & d^{M,2} & \dots & d^{M,N} \end{bmatrix}$$

  with:

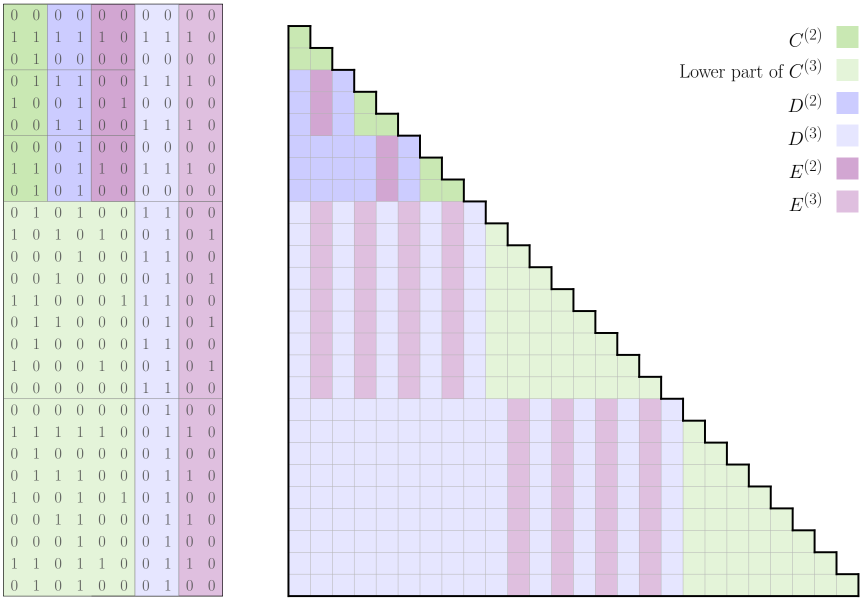
$$d^{i,k} \triangleq \left\{\begin{matrix} B_{i,k} \\ B_{i+1,k} \end{matrix}\right\}_{m_{\ell-1}} = \begin{cases} \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} & \text{if } B_{i,k} = 0 \text{ and } B_{i+1,k} = 0 \\ \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} & \text{if } B_{i,k} = 0 \text{ and } B_{i+1,k} = 1 \\ \left\{\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} & \text{if } B_{i,k} = 1 \text{ and } B_{i+1,k} = 0 \\ \left\{\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} & \text{if } B_{i,k} = 1 \text{ and } B_{i+1,k} = 1 \end{cases}$$

  again with the convention that $B_{M+1,k} = B_{M,k}$ for all k.

- The $E^{(\ell)}$ block is composed of two extra columns that will ensure the Order-Regularity of the whole:

$$E^{(\ell)} \triangleq \left[\left\{\begin{matrix} \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} \\ \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} \end{matrix}\right\}_M \quad \left\{\begin{matrix} \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} \\ \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} \end{matrix}\right\}_M \right] = \begin{bmatrix} \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} & \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} \\ \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} & \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} \\ \vdots & \vdots \\ \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} & \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} \\ \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} & \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} \\ \left\{\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right\}_{m_{\ell-1}} & \left\{\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right\}_{m_{\ell-1}} \end{bmatrix}.$$

  We call $e^{i,k}$ the $i-th$ pattern encountered in the $k-th$ columns of $E^{(\ell)}$.



$$\left[\begin{array}{cc|cc|cc|cc|cc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0
\end{array}\right]$$

Figure 4.8: An example of two blowup steps of Construction 4.3 with a 3 × 2 SOR building block. Notice how each part of the construction contributes in filling the constraint space. The C blocks (green) fill in triangles of the size of the previous iterate along the diagonal. The D blocks (blue) almost fill in the rest of the space. The two last columns (violet) of each step of the construction aim to fill in the remaining holes.

Given this construction, it follows that $m_\ell = M \cdot m_{\ell-1} = M^\ell$ and that $n_\ell = n_{\ell-1} + N + 2 = \ell \cdot N + 2(\ell - 1)$. Figure 4.8 helps visualizing the role of each block.

Definition 4.8 (Slices)**.** All three blocks $C^{(\ell)}, D^{(\ell)}$ and $E^{(\ell)}$ from construction 4.3 are divided into M *slices* (that is, blocks of consecutive rows) of size $m_{\ell-1}$ each. We say that a row index i *belongs to* a slice s, $1 \leq s \leq M$, if $(s-1) \cdot m_{\ell-1} < i \leq s \cdot m_{\ell-1}$. We also say that i corresponds to an *odd* (or *even*) index of s if its relative index within s is odd (or even).

We now prove the central lemma of this section.

Lemma 4.10. *Matrices $B^{(\ell)}$, $\ell = 1, 2, \ldots$, obtained from Construction 4.3 using a Strongly Order-Regular matrix B with an odd number of rows as building block are Order-Regular.*

Proof. Clearly, $B^{(1)} = B$ is OR because it is also Strongly OR. Assuming that $B^{(\ell-1)}$ is OR, let us show that $B^{(\ell)}$ is also OR. Therefore, we show that each block of the construction is designed to satisfy complementary subsets of the constraints space. Figure 4.8 graphically illustrates on a particular case how each block contributes in filling in the constraint space.

Claim 1. *The $C^{(\ell)}$ block satisfies every constraint (i, j) where i and j belong to the same slice s of $B^{(\ell)}$.*

Claim 1 follows directly from Lemma 4.2 since $C^{(\ell)}$ is simply an M-extension of $B^{(\ell-1)}$ which has $m_{\ell-1}$ rows by definition.

Claim 2. *The $D^{(\ell)}$ block satisfies every constraint (i, j) where i and j belong to two different and non-adjacent slices s_i and s_j.*

Let (i, j) be such a constraint for some integers s_i and s_j. From the Strong Order-Regularity of B and the fact that s_i and s_j are non adjacent (that is, $s_i + 1 < s_j$), we know that B doubly-satisfies the constraint (s_i, s_j). Therefore, from the definition of $D^{(\ell)}$, we know that there exist two columns k_1 and k_2 of $D^{(\ell)}$ such that the patterns that appear in the slices s_i and s_j for these columns are of the form:

$$d^{s_i,k_1} = \left\{ \begin{smallmatrix} \overline{\alpha} \\ \alpha \end{smallmatrix} \right\} \qquad d^{s_i,k_2} = \left\{ \begin{smallmatrix} \beta \\ \overline{\beta} \end{smallmatrix} \right\}$$
$$d^{s_j,k_1} = \left\{ \begin{smallmatrix} \alpha \\ \alpha \end{smallmatrix} \right\} \qquad d^{s_j,k_2} = \left\{ \begin{smallmatrix} \beta \\ \beta \end{smallmatrix} \right\}$$

for some $\alpha, \beta \in \{0, 1\}$ where $\overline{\alpha} = 1 - \alpha$ and $\overline{\beta} = 1 - \beta$. Let $I(i, j) \triangleq \begin{bmatrix} i & i+1 & j & j+1 \end{bmatrix}$ be the vector of row indices needed when checking the OR condition (4.1) for the constraint (i, j). Then, using Matlab-like notations, two cases are possible depending on parity:

- either $D^{(\ell)}_{I(i,j),k_1} = \begin{bmatrix} \overline{\alpha} & \alpha & \alpha & \alpha \end{bmatrix}$ and $D^{(\ell)}_{I(i,j),k_2} = \begin{bmatrix} \beta & \overline{\beta} & \beta & \beta \end{bmatrix}$;
- or $D^{(\ell)}_{I(i,j),k_1} = \begin{bmatrix} \alpha & \overline{\alpha} & \alpha & \alpha \end{bmatrix}$ and $D^{(\ell)}_{I(i,j),k_2} = \begin{bmatrix} \overline{\beta} & \beta & \beta & \beta \end{bmatrix}$.

In one or the other case there will always be a column k, either k_1 or k_2, such that condition (4.1) is verified. This will be true even if $i+1$ or $j+1$ belong to the next slice (respectively s_i+1 or s_j+1) thanks to the assumption that the building block B, and therefore also every iterate of Construction 4.3, have an odd number of rows. Indeed, because of this parity, any pattern $d^{i,k}$, of the form $\left\{ \begin{smallmatrix} \alpha \\ \beta \end{smallmatrix} \right\}$, ends with an α and the next pattern below it starts over with a β, thereby continuing the alternation of α and β for one more row.

Claim 3. *The $D^{(\ell)}$ block also satisfies every constraint (i, j) where i and j belong to two adjacent slices s and $s+1$ and i corresponds to an* odd *index of s.*

In the case of adjacent slices, condition (4.4) is no longer ensured for B. However, the original Order-Regularity still holds and there exists a column

k of $D^{(\ell)}$ such that $d^{s,k} = \left\{ \begin{smallmatrix} \overline{\alpha} \\ \alpha \end{smallmatrix} \right\}$ and $d^{s+1,k} = \left\{ \begin{smallmatrix} \alpha \\ \alpha \end{smallmatrix} \right\}$ for some $\alpha \in \{0,1\}$. Since i corresponds to an odd index of s, we must have $D^{(\ell)}_{i,k} = \overline{\alpha}$ and therefore we have $D^{(\ell)}_{I(i,j),k} = \begin{bmatrix} \overline{\alpha} & \alpha & \alpha & \alpha \end{bmatrix}$ which confirms Claim 3.

Claim 4. *The $E^{(\ell)}$ block satisfies every constraint (i,j) where i and j belong to two adjacent slices s and $s+1$ and i corresponds to an* even *index of s.*

From the definition of $E^{(\ell)}$, we know that there is always one of the two columns, say k, such that $e^{s,k} = \left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}$ and $e^{s+1,k} = \left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\}$, where $e^{i,k}$ is the i-th pattern encountered in the $k - th$ columns of $E^{(\ell)}$. Since i corresponds to an even index of s, it means that $E^{(\ell)}_{i,k} = 1$ and therefore we have $E^{(\ell)}_{I(i,j),k} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$ which confirms Claim 4.

Summary. Given any constraint (i,j):

- if i and j belong to the same slice, then the Order-Regularity condition holds for the constraint from Claim 1;
- if they belong to different slices that are non-adjacent to each other, then the condition holds from Claim 2;
- if they belong to adjacent slices, then the condition holds from Claims 3 and 4 together;

Therefore, all constraints are satisfied by $B^{(\ell)}$. □

Proposition 4.11 (A first improvement on the lower bound)**.** *For all n there exists an n-column Order-Regular matrix with at least $m = \left(\sqrt[10]{33}\right)^{n-7} = \Omega(1.4186^n)$ rows.*

Proof. We use Construction 4.3 with the 33×8 building block from Figure 4.7. After ℓ steps of the construction, we get a matrix $B^{(\ell)}$ with $m = 33^\ell$ rows and $n = 10\ell - 2$ columns and therefore $m = 33^{(n+2)/10} = \Omega(\sqrt[10]{33}^n)$ when $n = 8, 18, 28, \ldots$ From Lemma 4.10, this matrix is Order-Regular. For a value of n such that $10\ell - 2 < n < 10(\ell+1) - 2$ for some integer ℓ, the same construction as for $n = 10\ell - 2$ applies (simply add up to 9 dummy columns to the construction to match the required number of columns). Clearly this does not change the rate of growth. □

4.3.4 One step further: modified building block constraints

Definition 4.9 (Partially-Strong Order-Regularity)**.** We say that $B \in \{0,1\}^{m \times n}$ is Partially-Strongly Order-Regular (PSOR) whenever

(1) for every pair of rows i, j of B with $1 \leq i < j \leq m$, there exists a column k_1 such that:

$$B_{i,k_1} \neq B_{i+1,k_1} = B_{j,k_1} = B_{j+1,k_1}$$

(the original Order-Regularity condition);

(2) for every pair of rows i, j of B with $1 < i < j < m$ and for which $j - i$ is even, there exists a column k_2 (necessarily different from k_1) such that:

$$B_{i,k_2} \neq B_{i+1,k_2} \neq B_{j,k_2} = B_{j+1,k_2}.$$

Once again, we choose the convention that $B_{m+1,k} = B_{m,k}$.

The difference with the Strong Order-Regularity lies in the second condition. We now no longer require the existence of the column k_2 when $i = 1$, when $j = m$ or when $j - i$ is an odd number, hence the constraints that are doubly-satisfied by a PSOR matrix are those such that $1 < i < j < m$ and $j - i$ is even. As illustrated by Figure 4.10, we are allowed to do this relaxation because the $E^{(\ell)}$ block from Construction 4.3 actually satisfies more constraints than the sole ones it was designed to satisfy initially (as referred to in Claim 4 of the proof of Lemma 4.10) making it possible to reduce the set of constraints that the $D^{(\ell)}$ block has to satisfy and hence to soften the SOR condition. The softened condition allows us to find a larger building block which in turn results in an improved lower bound.

Before we show why using PSOR building blocks results in OR matrices, we need to slightly adapt Construction 4.3, and more precisely the definition of the $E^{(\ell)}$ block.

Construction 4.3*. Let $\hat{B} = \hat{B}^{(1)} \in \{0,1\}^{M \times N}$ be a PSOR building block matrix. In the same spirit as Construction 4.3, we inductively build a matrix $\hat{B}^{(\ell)} \in \{0,1\}^{m_\ell \times n_\ell}$ as the merging of three blocks:

$$\hat{B}^{(\ell)} = \begin{bmatrix} \hat{C}^{(\ell)} & \hat{D}^{(\ell)} & \hat{E}^{(\ell)} \end{bmatrix},$$

where the definitions of $\hat{C}^{(\ell)}$ and $\hat{D}^{(\ell)}$ are the same as those of $C^{(\ell)}$ and $D^{(\ell)}$ in Construction 4.3 with $\hat{B}^{(\ell)}$ and $\hat{B}$ taking the role of $B^{(\ell)}$ and B respectively and where

$$\hat{E}^{(\ell)} = \begin{bmatrix} \hat{e}^{1,1} & \hat{e}^{1,2} \\ \hat{e}^{2,1} & \hat{e}^{2,2} \\ \vdots & \vdots \\ \hat{e}^{M,1} & \hat{e}^{M,2} \end{bmatrix}$$

$$\hat{B} = \hat{B}^{(1)} = \begin{array}{|cccccccc|}\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
\hline\end{array}$$

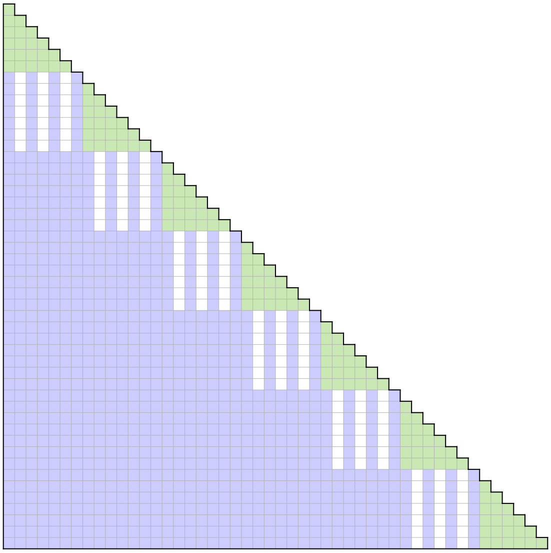
Figure 4.9: The 35 × 8 Partially-Strongly Order-Regular building block that we use to obtain our final lower bound.

is a slight modification of $E^{(\ell)}$ such that:

$$\hat{e}^{i,k} = \begin{cases} \left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\} & \text{if } i = 1 \text{ and } k = 2, \\ \left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\} & \text{if } i = M \text{ and } k = 1, \\ e^{i,k} & \text{otherwise.} \end{cases}$$

The only changes compared to Construction 4.3 is that the second column of $\hat{E}^{(\ell)}$ now starts with $\left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}$ instead of $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\}$ and its first column now ends with $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\}$ instead of $\left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}$. Clearly, the modification can only help to satisfy more constraints.

0	0	0	0	0	0	0	0		
1	1	1	1	1	1	1	1		
0	0	0	1	0	0	0	0		
0	1	1	1	1	1	1	1		
0	0	1	0	0	0	0	0		
1	0	1	1	1	1	1	1		
0	0	1	1	0	0	0	0		
0	0	1	1	1	1	1	1		
1	1	0	0	0	0	0	1		
0	0	1	0	1	1	1	1		
0	1	0	0	0	0	0	1		
0	0	0	1	1	1	1	1		
1	0	0	0	0	0	0	1		
0	0	0	0	1	1	1	1		
0	0	0	0	0	0	0	1		
1	1	1	1	0	1	1	1		
0	0	0	1	0	0	0	1		
0	1	1	1	0	1	1	1		
0	0	1	0	0	0	0	1		
1	0	1	1	0	1	1	1		
0	0	1	1	0	0	0	1		
0	0	1	1	0	1	1	1		
1	1	0	0	0	0	1	0		
0	0	1	0	0	1	1	1		
0	1	0	0	0	0	1	0		
0	0	0	1	0	1	1	1		
1	0	0	0	0	0	1	0		
0	0	0	0	0	1	1	1		
0	0	0	0	0	0	1	0		
1	1	1	1	1	0	1	1		
0	0	0	1	0	0	1	0		
0	1	1	1	1	0	1	1		
0	0	1	0	0	0	1	0		
1	0	1	1	1	0	1	1		
0	0	1	1	0	0	1	0		
0	0	1	1	1	0	1	1		
1	1	0	0	0	0	1	1		
0	0	1	0	1	0	1	1		
0	1	0	0	0	0	1	1		
0	0	0	1	1	0	1	1		
1	0	0	0	0	0	1	1		
0	0	0	0	1	0	1	1		
0	0	0	0	0	0	1	1		
1	1	1	1	0	0	1	1		
0	0	0	1	0	0	1	1		
0	1	1	1	0	0	1	1		
0	0	1	0	0	0	1	1		
1	0	1	1	0	0	1	1		
0	0	1	1	0	0	1	1		



0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1
0	0	1	1	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
1	1	0	0	0	0	0	1	1	0
0	0	1	0	1	1	1	1	0	0
0	1	0	0	0	0	0	1	1	0
0	0	0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1	1	0
0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	1	0	0
1	1	1	1	0	1	1	1	0	1
0	0	0	1	0	0	0	1	0	0
0	1	1	1	0	1	1	1	0	1
0	0	1	0	0	0	0	1	0	0
1	0	1	1	0	1	1	1	0	1
0	0	1	1	0	0	0	1	0	0
0	0	1	1	0	1	1	1	0	0
1	1	0	0	0	0	1	0	1	0
0	0	1	0	0	1	1	1	0	0
0	1	0	0	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	0
1	0	0	0	0	0	1	0	1	0
0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	1	0	0	0
1	1	1	1	1	0	1	1	0	1
0	0	0	1	0	0	1	0	0	0
0	1	1	1	1	0	1	1	0	1
0	0	1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	1	0	1
0	0	1	1	0	0	1	0	0	0
0	0	1	1	1	0	1	1	0	0
1	1	0	0	0	0	1	1	1	0
0	0	1	0	1	0	1	1	0	0
0	1	0	0	0	0	1	1	1	0
0	0	0	1	1	0	1	1	0	0
1	0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	1	1	0	0
0	0	0	0	0	0	1	1	0	0
1	1	1	1	0	0	1	1	0	0
0	0	0	1	0	0	1	1	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	0	0	1	1	0	0
1	0	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	0

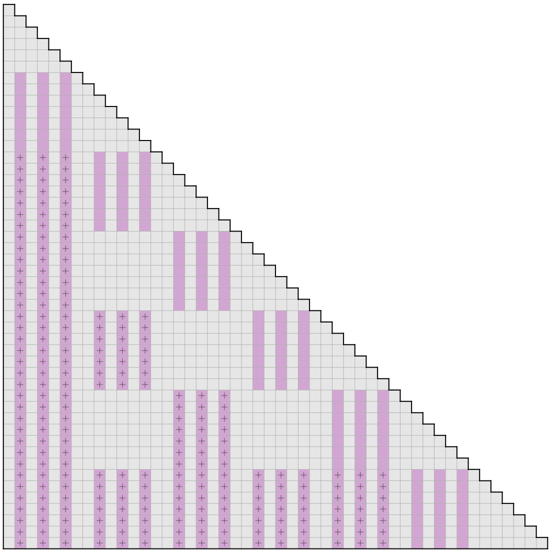


Figure 4.10: If we only used the $C^{(\ell)}$ and $D^{(\ell)}$ blocks (respectively in green and blue) in Construction 4.3 with an SOR building block, there would remain a few holes in the constraint space (top). The $E^{(\ell)}$ block is designed to fill these holes. But a slightly improved version of the $E^{(\ell)}$ block (violet) actually fills much more than just the required holes (bottom). This fact can be exploited to soften the constraints on the building block and further improve the lower bound to obtain our final bound.

Lemma 4.12. *Matrices obtained from Construction 4.3** *using Partially-Strongly Order-Regular building blocks with an odd number of rows are Order-Regular.*

Proof. The proof is inductive, in the same flavor as the proof of Lemma 4.10. Knowing that $\hat{B}^{(1)}$ is OR and assuming that $\hat{B}^{(\ell-1)}$ is OR too, we show that $\hat{B}^{(\ell)}$ must also be OR.

Claim 1. *The $\hat{C}^{(\ell)}$ block satisfies every constraint (i, j) where i and j belong to the same slice s of $\hat{B}^{(\ell)}$.*

The argument is the same as for Claim 1 in the proof of Lemma 4.10.

Claim 2. *The $\hat{D}^{(\ell)}$ block satisfies every constraint (i, j) where i and j belong to two different slices s_i and s_j such that $s_i \neq 1$, $s_j \neq M$ and $s_j - s_i$ is even.*

From the Partially-Strong Order-Regularity of the building block $\hat{B}$ and the conditions on s_i and s_j, we know that the constraints (s_i, s_j) is doubly-satisfied by $\hat{B}$. Therefore, the same reasoning as the one of Claim 2 in the proof of Lemma 4.10 applies.

Claim 3. *The $\hat{D}^{(\ell)}$ block also satisfies every constraint (i, j) where i and j belong to two different slices s_i and s_j such that $s_i = 1$, $s_j = M$ or $s_j - s_i$ is odd and such that i corresponds to an* odd *index of s_i.*

Here, the constraint (s_i, s_j) is not doubly-satisfied by $\hat{B}$ but i corresponds to an *odd* index of s_i. Again, the same argument as for Claim 3 in the proof of Lemma 4.10 applies here.

Claim 4. *The $\hat{E}^{(\ell)}$ block satisfies every constraint (i, j) where i and j belong to two different slices s_i and s_j such that $s_i = 1$, $s_j = M$ or $s_j - s_i$ is odd and such that i corresponds to an* even *index of s_i.*

We evaluate the three possible cases when i corresponds to an even index of s_i.

(1) If $s_i = 1$, we have $\hat{E}^{(\ell)}_{[i\ i+1],k} = \begin{bmatrix}1 & 0\end{bmatrix}$ for both columns $k = 1$ and 2 (since i corresponds to an even index of s_i), and we have $\hat{E}^{(\ell)}_{[j\ j+1],k} = \begin{bmatrix}0 & 0\end{bmatrix}$ for either $k = 1$ or $k = 2$.

(2) When $s_j = M$, we have $\hat{E}^{(\ell)}_{[j\ j+1],k} = \begin{bmatrix}0 & 0\end{bmatrix}$ for both $k = 1$ and 2, and we have $\hat{E}^{(\ell)}_{[i\ i+1],k} = \begin{bmatrix}1 & 0\end{bmatrix}$ for either $k = 1$ or $k = 2$.

(3) If $s_i \neq 1, s_j \neq M$ and $s_j - s_i$ is an odd number, then $\hat{e}^{s_i,k_1} = \left\{\begin{smallmatrix}0\\0\end{smallmatrix}\right\}$ and $\hat{e}^{s_i,k_2} = \left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}$ or vice versa. Furthermore, $\hat{e}^{s_i,k}$ and $\hat{e}^{s_j,k}$ are different patterns for both $k = 1$ and 2 (either $\left\{\begin{smallmatrix}0\\0\end{smallmatrix}\right\}$ and $\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}$ or vice versa). Therefore, there will always be one of the two columns, say

k', such that $\hat{e}^{s_i,k'} = \left\{ \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \right\}$ and $\hat{e}^{s_j,k'} = \left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\}$ and hence such that $\hat{E}^{(\ell)}_{I(i,j),k'} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$.

Summary. A constraint (i, j) such that i and j belong to the slices s_i and s_j is satisfied by:

- the $\hat{C}^{(\ell)}$ block if $s_i = s_j$;
- the $\hat{D}^{(\ell)}$ block if $s_i \neq s_j$ and the constraint (s_i, s_j) is doubly-satisfied by B;
- either the $\hat{D}^{(\ell)}$ block or the $\hat{E}^{(\ell)}$ block if $s_i \neq s_j$ and the constraint (s_i, s_j) is not doubly-satisfied by B (which is the case when $s_i = 1$, $s_j = M$ or $s_j - s_i$ is odd). □

Theorem 4.13. *Given a number of columns n, there exists an Order-Regular matrix with at least $m = \left(\sqrt[10]{35}\right)^{n-7} = \Omega(1.4269^n)$ rows.*

Proof. The proof is analog to the one of Proposition 4.11 with the PSOR building block from Figure 4.9 and using Lemma 4.12 to guarantee that the construction indeed provides OR matrices. □

4.4 The art of building large matrices

Our results heavily rely on our ability to work with large (PS)OR matrices efficiently. First, to disprove Conjecture 4.3, we performed an exhaustive search on the massive set of OR matrices with $n = 7$ and found no matrix with $34 = F_{n+2}$ rows. Then, to obtain our lower bounds in Proposition 4.11 and Theorem 4.13, we searched for large enough matrices in the even huger set of (P)SOR matrices with $n = 8$.

In this section, we describe the elements that make the algorithms efficient while also assessing their correctness. This is especially important for Theorem 4.4 where we need to ensure that the search for OR matrices was indeed exhaustive.

To emphasize why the efficiency of the algorithms matters, let us illustrate the size of the search space. First regarding the exhaustive search, 3×10^{11} is a conservative lower bound on the total number of OR matrices with $n = 7$, excluding symmetrical cases[5]. Therefore, we cannot afford to examine each of these matrices individually and performing an exhaustive search requires to come up with some additional tricks. Furthermore, the size of the search space grows doubly exponentially with n hence stepping

[5] We extrapolate the exact number to be around 3×10^{16} using a doubly exponential regression from the number of branches for $n = 1$ to 6.

from n to $n+1$ columns makes a big difference. Including all the tricks and optimization described below, we were able to reduce the execution time to 1 month for $n = 7$ (using 10 Intel® Xeon® X5670 cores). As a comparison, the final code took less than 10 seconds for $n = 6$. This time increase when incrementing n by one suggests that the exhaustive search for $n = 8$ is very challenging. Regarding the search for building blocks, the total number of (P)SOR matrices with $n = 8$ is significantly larger than that of OR matrices with $n = 7$. However in that case, we only need to find one matrix that is as large as possible, which we achieve through the design of an efficient search strategy.

In the rest of this section, we present the techniques that we used to search the space of OR matrices without having to scan every candidate matrix and provide a pseudo-code of our algorithm. We also present the specific ideas that we used to perform an exhaustive search on the space and describe our search strategy to look for large matrices when an exhaustive search is neither within reach, nor necessary. The source code for our algorithm is written in Go language and is available at http://sites.uclouvain.be/ORsearch. A help file explaining how to use the program and tune the parameters is also available there.

4.4.1 General principles

The steps below focus on OR matrices but an equivalent procedure applies for (P)SOR matrices as well.

Symmetry. OR matrices stay OR when a permutation or a negation is applied to some of their columns. Therefore we always assume that the columns follow each other in a lexicographical order and that the first row is composed of all 0 entries. We can also assume that the second row is composed of all 1 entries since starting a column with, e.g., 00, can only satisfy less constraints than the same (negated) column that would start with 01 instead. This way we remove redundancy in the search space.

Branching. If the first block of d rows of a matrix is infeasible itself there is no need to check the rest of the matrix. On the other hand, if the first d rows of several matrices are the same, it is unnecessary to recheck this part every time. We exploit these observations by using a depth first search on the matrices. If we have an initial block of d rows that is feasible, we try every extension to $d+1$ rows and only continue with those that do not violate the OR condition. We are thus exploring a huge search tree whose root is by default the empty matrix and for which any node at *depth* d, that is at distance d from the root, corresponds to an OR matrix of size $d \times n$.

Remark 4.2 (Order-Regularity*). In this section, we use a variation of the

OR condition which we refer to as OR*: we require that there exists a column k such that identity (4.1) is verified for all $1 \leq i < j < m$ but not for $j = m$. We thereby allow the last two rows to be equal. Yet, both conditions are equivalent. Indeed, from an OR* matrix, remove the last row and it becomes OR. On the other hand, take an OR matrix and copy its last row to obtain an OR* matrix. Therefore, there exists an OR matrix with m rows iff there also exists an OR* matrix with $m+1$ rows. Similarly, we refer to the same variation of the (P)SOR condition by (P)SOR*.

Filtering. During the branch search, we aim to avoid checking infeasible possibilities over and over again. We try to filter them out as soon as they become infeasible. Assume we are investigating a branch with the first d rows fixed. The order-regularity condition deals with rows by pairs, and in any extension, any pair of rows that we encounter later has to be compatible with the same first d rows. We can see these pairs as the rows labeled j and $j+1$ in the order-regularity condition, to be compared with the pairs labeled i and $i+1$ with $i < d$. Adding row $d+1$ causes new conditions to be imposed on the future pairs, thereby reducing the set of feasible ones. Since these conditions must hold as long as the same $d+1$ rows are there, this allows to filter the set of pairs to be considered for all the branches below. In the end, we record pairs of rows because of the nature of the OR condition and this allows to extend the matrix row by row without ever checking the same OR condition twice in the same branch. We now formalize how this filtering happens.

Definition 4.10 (Compatible pairs)**.** Let A be some $d \times n$ Order-Regular* matrix. We define P_A, the *set of compatible pairs* of A, as:

$$P_A = \Big\{(r,q) : r,q \in \{0,1\}^n \text{ and } \forall\, i,\ 1 \leq i < d, \exists\, k,\ 1 \leq k \leq n \qquad (4.5)$$
$$\text{such that } A_{i,k} \neq A_{i+1,k} = r_k = q_k\Big\}.$$

We also define R_A and Q_A, the projections of P_A on the set of rows that respectively appear as the first and second entry of a pair:

$$R_A = \Big\{r \in \{0,1\}^n : \exists\, q\ \in \{0,1\}^n \text{ for which } (r,q) \in P_A\Big\}, \qquad (4.6)$$
$$Q_A(r) = \Big\{q \in \{0,1\}^n : (r,q) \in P_A\Big\}. \qquad (4.7)$$

Figure 4.11 illustrates how P_A, R_A and Q_A relate to each other on an example matrix.

Given an OR* matrix A, the set of possible extension rows q such that $\left[\begin{smallmatrix} A \\ q \end{smallmatrix}\right]$ is OR* can be easily identified using P_A. Indeed, if r is the last row of A, then the set of rows q that are allowed to extend A are exactly the ones

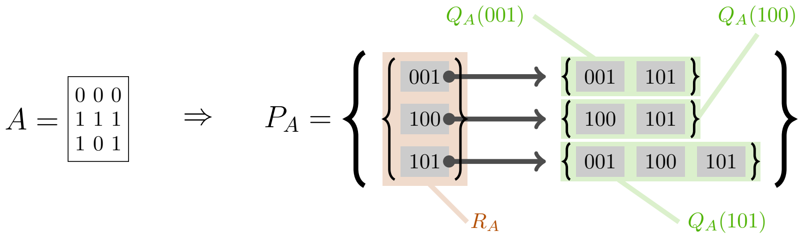


Figure 4.11: For this example matrix A, any pair of rows $(r, q) \notin P_A$ will never allow a valid extension of A. We encode the set of compatible pairs P_A as the set R_A where each entry r relates to a set $Q_A(r)$. In this example, notice that even though A is OR*, it does not satisfy the symmetry rules.

such that $(r, q) \in P_A$. Moreover, the P_A set can only shrink as we add rows to A hence the following lemma.

Lemma 4.14. *Let A be some $d \times n$ Order-Regular* matrix of the form $\left[\begin{smallmatrix} A^- \\ r \end{smallmatrix}\right]$ and let $A^+ = \left[\begin{smallmatrix} A \\ q \end{smallmatrix}\right]$ for some row q. Then A^+ is Order-Regular* iff $q \in Q_A(r)$. Furthermore for any r, q, it holds that $(r, q) \in P_{A^+}$ iff both $(r, q) \in P_A$ and there exists a column k such that $A^+_{d,k} \neq A^+_{d+1,k} = r_k = q_k$. Therefore $P_{A^+} \subseteq P_A$.*

Proof. First we observe that:

$$\begin{aligned} q \in Q_A(r) \Leftrightarrow (r, q) \in P_A, & \\ \Leftrightarrow \forall\, i, 1 \leq i < d, \exists\, k : \quad & A_{i,k} \neq A_{i+1,k} = r_k = q_k, \\ \Leftrightarrow \forall\, i, 1 \leq i < j = d, \exists\, k : & A^+_{i,k} \neq A^+_{i+1,k} = A^+_{j,k} = A^+_{j+1,k}, \end{aligned}$$

since $A_{i,k} = A^+_{i,k}$ for all $i, 1 \leq i \leq d$. Furthermore, using the fact that A is OR*, we also have that for all $i, j, 1 \leq i < j < d$, there exists a column k such that $A^+_{i,k} \neq A^+_{i+1,k} = A^+_{j,k} = A^+_{j+1,k}$. Therefore, condition (4.1) is verified for all $i, j, 1 \leq i < j < d + 1$ and we have that $q \in Q_A(r)$ iff A^+ is OR*.

The fact that $(r, q) \in P_{A^+}$ iff both $(r, q) \in P_A$ and there exists a column k such that $A^+_{d,k} \neq A^+_{d+1,k} = r_k = q_k$ follows directly from the definitions of P_A and P_{A^+}. □

Direct cutting. Storing and maintaining the sets of compatible pairs of rows during the search has an additional advantage. Assume we are looking at a branch corresponding to a $d \times n$ matrix A. Then we have $|R_A|$ distinct rows appearing as r in the set of compatible pairs of rows P_A. There is clearly no way of getting more than $d + |R_A| + 1$ rows by extending this particular branch. Consequently, when searching for an $(m^* + 1) \times n$ OR* matrix, if $d + |R_A| + 1 < m^* + 1$, then we discard the node right away and make a step back in the search tree. This idea is formalized by the following lemma.

Lemma 4.15. *Let A be some $d \times n$ Order-Regular* matrix, let R_A be defined by equation (4.6) and let $m^* = d + |R_A|$. Then there exists no Order-Regular* matrix with more than $m^* + 1$ rows such that the first d rows equal A.*

Proof. First we observe that $Q_A(r) \subseteq R_A$ for all r. Indeed:

$$\begin{aligned} q \in Q_A(r) &\Rightarrow (r,q) \in P_A && \text{(by definition of } Q_A(r)\text{)},\\ &\Rightarrow (q,q) \in P_A && \text{(because if the condition in the definition of } P_A\\ &&& \text{holds for some } r\text{, it must also hold when } r = q\text{)},\\ &\Rightarrow q \in R_A && \text{(by definition of } R_A\text{)}. \end{aligned}$$

Therefore, any row we may want to add to A at any point must come from R_A.

Moreover, an OR* matrix cannot contain twice the same row (except possibly its two last rows). Indeed, assume an $m \times n$ matrix A has its $i-th$ and $j-th$ rows identical, $1 \leq i < j < m$, then this matrix cannot satisfy the OR* constraint (i,j) since $A_{i,k} = A_{j,k}$ for all k. Furthermore, any row that we would add below A must come from R_A. Therefore, we cannot add more than $|R_A|$ different rows to A. Any extension of A must thus have at most $m^* + 1$ rows (the $+1$ comes from the fact that the two last rows of an OR* matrix are allowed to be the same). □

Using this trick, we are able to spot poor branches early on and hence to significantly reduce the size of the search tree without missing any OR* matrix with $34 + 1$ rows or more. To have a better idea of the amount of computations saved from direct cutting, we compared the total number of n-column OR matrices explored by our algorithm with or without cutting, for small values of n. We observed that the algorithm with direct cutting actually explores $8.3\%, 46.6\%, 81.3\%$ and 94.3% less matrices than the one without direct cutting for $n = 3, 4, 5$ and 6 respectively. Extrapolating from these ratios, we estimate that the direct cutting trick allows to reduce the search space by a factor of around 60 for $n = 7$.

4.4.2 General implementation

Combining the ideas from Section 4.4.1, we sketch the *branch search* strategy in Algorithm 4. Notice that the starting branch needs not necessarily be the empty matrix. Though, choosing a $d \times n$ OR* matrix A as the root in Algorithm 4 will result in an OR* matrix whose first d rows correspond to the rows of A. As we will see, this option will be useful later, but then this also means Algorithm 4 only performs an exhaustive search on a restricted portion of the tree.

Observe that in Algorithm 4, A^* always corresponds to the best matrix found so far. Indeed, step 3 is the only step where the current A^* is replaced by another, better, matrix. Therefore, at step 14, we always overwrite A^* with a matrix with at least as many rows.

Complexity issues. The steps 6 and 7 can be performed efficiently using, e.g., a two dimensional array to encode the $P_{A^{(\ell)}}$ sets. Moreover, the step 8 encodes the direct cutting according to Lemma 4.15. Regarding step 10, the rows q can be taken in any order. By adding randomness in the order, we allow the algorithm to randomly return any matrix with the target size. Finally, Lemma 4.14 ensures that step 13 requires at most $|P_{A^{(\ell)}}|$ OR*-checks which is still the most expensive operation of each step of the recursion. Observe that since $Q_A(r) \subseteq R_A$ (as shown in the proof of Lemma 4.15), it holds that $|P_A| \leq |R_A|^2$ and hence that the cardinality of both sets decrease together when ℓ increases.

4.4.3 For extremal matrices: we need exhaustive search

To further speed up our code in order to perform an exhaustive search on all OR* matrices with 7 columns, we develop a code capable of parallel processing.

Parallelization. In Algorithm 4, it is possible to perform the search in parallel on different branches of the tree. For this purpose, we first fix a depth d and precompute every possible non-symmetrical $d \times 7$ OR* matrix. These matrices act as the roots of several independent subtrees that together span the complete search tree. We then launch Algorithm 4 in parallel each time with a different root matrix as input. It finishes with the answer whenever every subtree has been completely searched.

In our case, we chose $d = 9$ which resulted in 106 million distinct subtrees of variable size. We obtained 35 subtrees that ended up with an OR* matrix of $33 + 1$ rows but none with an OR* matrix of $34 + 1$ rows, leading to the statement of Theorem 4.4 in Section 4.2.

4.4.4 For building blocks: we need an efficient search strategy

To search for (P)SOR building blocks with 8 columns, the strategy described in Section 4.4.1 still applies but the size of the search space does not allow to perform an exhaustive search. However, in this case, we only need to find a large matrix but not to prove that it is the largest (we found an SOR block with 33 rows and a PSOR block with 35 rows in our case, see Figures 4.7 and 4.9). To this end, based on the special structure of (P)SOR

Algorithm 4: Branch search

Input: A, the (PS)OR* matrix of size $d \times n$ at the root of the search tree (optional, [ ] by default).
m^{target}, the target number of rows for the solution A^* (optional, ∞ by default).

Initialization: Precompute P_A using equation (4.5).

Output: $A^* = \texttt{branchsearch}(d, A, P_A, A)$, a (PS)OR* matrix with n columns and the maximum (or the target) number of rows such that the first d rows of A^* are given by A.

```
Function branchsearch($\ell, A^{(\ell)}, P^{(\ell)}, A^*$)
    if $\ell > \#\text{rows}(A^*)$ then
        $A^* := A^{(\ell)}$.
    if $\#\text{rows}(A^*) = m^{\text{target}}$ then
        return $A^*$.
    Extract $R^{(\ell)} := R_{A^{(\ell)}}$ from $P^{(\ell)}$ using equation (4.6).
    Extract $Q^{(\ell)} := Q_{A^{(\ell)}}(r)$ from $P^{(\ell)}$ using equation (4.7) with $r$
    being the last row of $A^{(\ell)}$.
    if $\ell + |R^{(\ell)}| < m^{\text{target}}$ then
        return $A^*$.
    for $q \in Q^{(\ell)}$ do
        $A^{(\ell+1)} := \begin{bmatrix} A^{(\ell)} \\ q \end{bmatrix}$.
        if $A^{(\ell+1)}$ satisfies the symmetry rules then
            Compute $P^{(\ell+1)} := P_{A^{(\ell+1)}}$ using equation (4.5).
            $A^* := \texttt{branchsearch}(\ell + 1, A^{(\ell+1)}, P^{(\ell+1)}, A^*)$
    return $A^*$.
```

Symmetry rules: the columns of the matrix must be lexicographically sorted and the first and second rows must be respectively all zeros and all ones.

matrices, we designed an efficient search strategy to quickly find these large instances. We now develop this strategy for SOR matrices. An equivalent strategy exists for PSOR matrices as well but it requires us to introduce some nonessential details. As in Section 4.4.1, we here use the variation of the (P)SOR condition denoted by (P)SOR* and defined in Remark 4.2.

The search strategy is based on the reversing operation that reverses the order of the rows and negates the even rows.

Definition 4.11 (Reversing)**.** Let $A \in \{0,1\}^{m\times n}$. We define A^{rev}, the *reverse* of A, where for all $1 \le i \le m$, we have:

$$A^{\text{rev}}_{i,k} = \begin{cases} A_{m+1-i,k} & \text{if } i \text{ is odd,} \\ 1 - A_{m+1-i,k} & \text{if } i \text{ is even,} \end{cases}$$

for all columns k.

The key observation is that reversing an SOR* matrix preserves its Strong Order-Regularity*.

Lemma 4.16. *If $A \in \{0,1\}^{m\times n}$ is SOR*, then its reverse is also SOR*.*

Proof. For all $i, j, 1 \le i < j < m$, let $i' = m - j$ and $j' = m - i$ so that $1 \le i' < j' < m$. Let also $I(i,j) \triangleq \begin{bmatrix} i & i+1 & j & j+1 \end{bmatrix}$ and $I'(i,j) \triangleq m + 1 - I = \begin{bmatrix} j'+1 & j' & i'+1 & i' \end{bmatrix}$ using Matlab notations. From the Strong Order-Regularity* of A, there exist two columns k_1 and k_2 such that:

$$A_{I'(i,j),k_1} = \begin{bmatrix} \alpha & \alpha & \alpha & \overline{\alpha} \end{bmatrix} \quad \text{and} \quad A_{I'(i,j),k_2} = \begin{bmatrix} \beta & \beta & \overline{\beta} & \beta \end{bmatrix}$$

for some $\alpha, \beta \in \{0,1\}$. Then for A^{rev}, the reverse of A, we have:

$$\begin{cases} A^{\text{rev}}_{I(i,j),k_1} = \left[\, \alpha\ \overline{\alpha}\ \alpha\ \alpha \,\right] \text{ and } A^{\text{rev}}_{I(i,j),k_2} = \left[\, \beta\ \overline{\beta}\ \overline{\beta}\ \overline{\beta} \,\right] & \text{if } i \text{ is } \textit{odd} \text{ and } j \text{ is } \textit{odd}, \\ A^{\text{rev}}_{I(i,j),k_1} = \left[\, \alpha\ \overline{\alpha}\ \overline{\alpha}\ \overline{\alpha} \,\right] \text{ and } A^{\text{rev}}_{I(i,j),k_2} = \left[\, \beta\ \overline{\beta}\ \beta\ \beta \,\right] & \text{if } i \text{ is } \textit{odd} \text{ and } j \text{ is } \textit{even}, \\ A^{\text{rev}}_{I(i,j),k_1} = \left[\, \overline{\alpha}\ \alpha\ \alpha\ \alpha \,\right] \text{ and } A^{\text{rev}}_{I(i,j),k_2} = \left[\, \overline{\beta}\ \beta\ \overline{\beta}\ \overline{\beta} \,\right] & \text{if } i \text{ is } \textit{even} \text{ and } j \text{ is } \textit{odd}, \\ A^{\text{rev}}_{I(i,j),k_1} = \left[\, \overline{\alpha}\ \alpha\ \overline{\alpha}\ \overline{\alpha} \,\right] \text{ and } A^{\text{rev}}_{I(i,j),k_2} = \left[\, \overline{\beta}\ \beta\ \beta\ \beta \,\right] & \text{if } i \text{ is } \textit{even} \text{ and } j \text{ is } \textit{even}. \end{cases}$$

In every case, the Strong Order-Regularity* of A^{rev} is ensured. □

Based on Lemma 4.16, we can now formulate our *back-and-forth search* strategy to find large SOR* matrices as described by Algorithm 5. For that, we use an adapted version of Algorithm 4 to generate SOR* instead of OR* matrices. This is achieved by modifying the definition of the set of compatible pairs P_A in Definition 4.10 with respect to the SOR* condition.

Algorithm 5: Back-and-forth search

```
Input: $d$, $T$.
Output: An SOR$^*$ matrix.
Initialization: $A^{(0)}$, a random SOR$^*$ matrix with $d$ rows obtained
                from Algorithm 4 using input $m^{\text{target}} = d$.
                $t = 0$.
1 while stopping criterion do
2     Compute $B^{(t+1)}$ as the result of Algorithm 4 using input $A = A^{(t)}$.
3     Compute $B^{\text{rev}}$, the reverse of $B^{(t+1)}$.
4     $A^{(t+1)} \triangleq B^{\text{rev}}_{1:d,:}$, the first $d$ rows of $B^{\text{rev}}$.
5     $t \leftarrow t + 1$.
6 return $B^{(t)}$.
```

Stopping criterion: after at least T steps, stop whenever $B^{(t-T+1)}$ and $B^{(t)}$ have the same number of rows (stagnation in the last T steps).

In Algorithm 5, the parameter d is typically chosen so that applying Algorithm 4 at step 2 finishes in a reasonable time (so d should be large enough to ensure a manageable size of the search trees) while leaving as much room as possible for the optimization process (so d should not be too large either). When looking for SOR* matrices with 8 columns, we typically used $d = 14$. Also note that in Algorithm 5, it is important to avoid getting the same A over and over again. We rely on the randomness introduced at step 10 of Algorithm 4 to always get a random instance of the possible B matrices. Interestingly, Algorithm 5 is guaranteed to not make the solution worse at each iteration, as stated by the following proposition. However, we cannot guarantee that it will find a globally optimal solution. Therefore it may be useful to restart it until finding a matrix with a suitable number of rows.

Proposition 4.17. *In Algorithm 5, the number of rows of $B^{(t+1)}$ is always at least as large as that of $B^{(t)}$ for all $t \geq 1$.*

Proof. At step 2 of Algorithm 5, applying Algorithm 4 with $A^{(t)}$ as the root means performing a search in a subtree of the whole tree where the first d rows are fixed. In this subtree, the matrix B^{rev} computed at the step t is a feasible solution since from Lemma 4.16, it is SOR* and since from step 4 its first d rows match those of $A^{(t)}$. Therefore, the best SOR* matrix $B^{(t+1)}$ that can be found in the subtree must be at least as good (in terms of its number of rows) as $B^{(t)}$. □

4.5 Two approaches to be further explored

As it is often the case in research worth undertaking, there are many approaches to attack the same problem. Some of them succeed, others fail and most are left open. In this section, we briefly relate two approaches of the latter type. We believe that these approaches could inspire new ideas to attack the problem from a different angle.

Both approaches share the same premise, namely they look at a dual version of the OR problem. In the OR problem, we try to insert as many rows as possible in a matrix whose number of columns n is fixed while maintaining the OR condition. In its dual version though, we fix the number of rows m and try to satisfy every OR constraint (there is one for every pair (i, j), $1 \leq i < j \leq m$) with as few columns as possible. The problem at hands is thus a constraint satisfaction problem and we are looking for a lower bound in terms of m on the number of columns needed to solve it. Then, upper bounds on the number of rows of an OR matrix can be recovered by inverting the relation between m and n of this lower bound.

4.5.1 A balanced weighted constraint space

Suppose that every OR constraint of the constraint space is given a weight $w_{i,j}$, with $1 \leq i < j \leq m$. Any column ξ of an OR matrix satisfies a subset of these constraints. To ξ, we associate a value x_ξ defined as the sum of the weights of the constraints it satisfies. Let ξ^* be the column vector of size m with the highest value and let $s = \sum_{i<j} w_{i,j}$ be the total sum of the weights. Clearly, we need to satisfy every constraint at least once so the sum of the values of the columns must be at least s. Therefore, we need at least $\lceil \frac{s}{x_{\xi^*}} \rceil$ columns to achieve the Order-Regularity of the whole, which yields a lower bound on the number of columns as we aimed for.

In the above approach, there are no constraints put on the weights themselves. Indeed, they are part of the variables of the problem: the better the weights, the better the lower bound one can expect. In fact, to obtain good lower bounds, one should intuitively give more weight to the constraints that are "difficult" to satisfy.[6] In the ideal scenario, it should be possible to find weights for which even the best possible column can only satisfy a small fraction (a fraction $\frac{x_{\xi^*}}{s}$ no larger than $1/\log_\varphi m$ with φ being the golden ratio would be great). This is what we could call a *balanced* weight function: it would monitor the hard constraints by giving them more weight.

When looking at the constructions of Section 4.3, we observe that constraints in the bottom-left corner of the constraint space seem to be satisfi-

[6] We think of the difficult constraints as the ones that are incompatible with many other constraints, especially if these other constraints are difficult themselves.

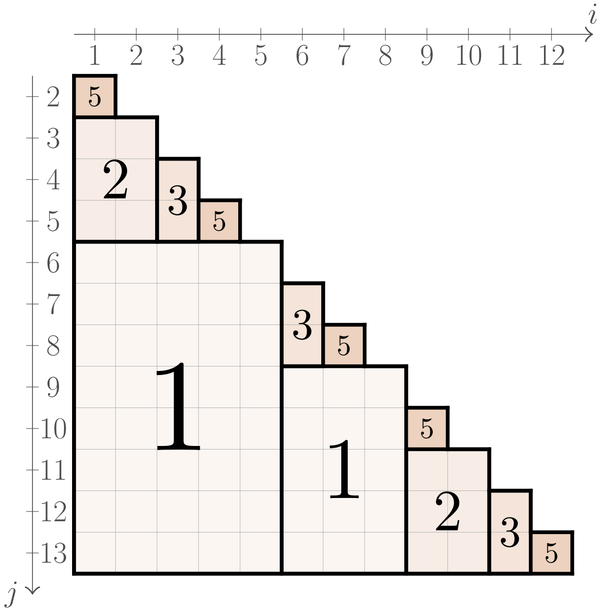


Figure 4.12: If we weighted each constraint of the constraint space according to the figure, what would be the amount of weight that is covered by each column of an OR matrix? In particular, would it be possible to force a balance between the contribution of each column? This is the question we tried to answer with the weighted constraint space approach, hoping for the existence of a suitable weight function, possibly the one represented above. Unfortunately, it seems that it is always possible to cover around $\frac{1}{8}$th of the total weight with a single column—thus a quite unbalanced situation.

able with few columns, so they should somehow receive less weight. Additionally, having in mind the Fibonacci conjecture from Hansen and Zwick (Section 4.2), it may be tempting to propose a weight function in the spirit of the one suggested in Figure 4.12, in the hope of finding a bound of the kind $n \geq \lceil \frac{s}{x_{\xi^*}} \rceil \geq \log_\varphi m$. However, even with variations on how we design the weights, we did not manage to obtain any interesting bound. In fact, we even tried to optimize the weight function in general for growing values of m and what we observed was disappointing: the lower bound on the number of columns of the OR matrix seemed to be converging to... 8?! While we were not able to confirm this fact in general, we conjecture that it is always possible to cover at least $\frac{1}{8}$th of the total weight with a single column. In particular, this would mean that it is not possible to design a truly balanced weight function.

Conjecture 4.18. *For any possible weight function w, it is possible to build a column ξ such that $x_\xi \geq \frac{s}{8}$.*

Nevertheless, we still believe that a more sophisticated study of the above idea, with another definition of the value of a column for instance, could lead to interesting results.

4.5.2 A graph coloring problem for the constraint space

Consider the set of constraints (i, j), $1 \leq i < j \leq m$, that need to be satisfied. Let us create a graph G whose nodes correspond to the pairs (i, j). Then, let us add an edge between two nodes whenever the corresponding constraints are mutually exclusive, so for instance, a column will not satisfy both $(1, 4)$ and $(4, 6)$ because the first constraint implies entries 4 and 5 to be identical and the second constraint requires them to be different. Following this idea, we create a link between (i_1, j_1) and (i_2, j_2) whenever at least one of the following conditions is satisfied:

1. $i_2 = j_1$,
2. $i_2 = i_1 + 1$ and $j_2 = j_1 - 1$,
3. $i_2 = i_1 + 1$ and $j_2 = j_1$,
4. $i_2 = i_1 + 1$ and $j_2 = j_1 + 1$.

An example of the corresponding graph for $m = 5$ is given in Figure 4.13. A column of an OR matrix cannot possibly satisfy two constraints that are neighbors in G. Suppose we assigned a color to each column of an OR matrix and reported these colors on the nodes of the graph according to the constraints they solve (if a constraint is satisfied by two columns, then it does not matter which one we choose). Doing so, we obtain a proper coloring of the graph (that is, two adjacent nodes never have the same color). For a given number of rows m, a necessary condition for all the OR constraints to be satisfiable with n columns is that there exists a proper coloring of the corresponding graph with n colors. Therefore, the minimum number of colors needed to color a graph properly, that is, the well-known *chromatic number* $\chi(G)$, is a lower bound on the number of columns of an OR matrix with m rows.

The above description leads to a relaxation of the OR problem where the goal is to find the best possible lower bound on the chromatic number of G. It is a relaxation because the definition of the graph only includes pairwise mutual exclusion constraints, while there could also be triples of constraints (or more) that are mutually exclusive even though they are compatible two

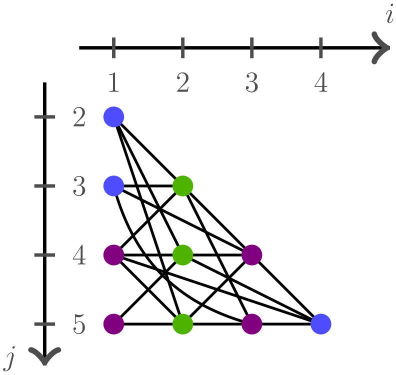


Figure 4.13: A proper coloring of the graph G for $m = 5$ with 3 colors. Notice how the coloring corresponds to the extremal OR matrix for $n = 3$, as shown in Figure 4.1

by two. These constraints would not be edges anymore but rather hyper-edges of the graph. However, we can neglect the information contained in these hyper-edges and hope that the simplification does not affect the chromatic number too heavily.

Ideally, we would like to find a lower bound on $\chi(G)$ that is explicit in m and as tight as possible. Unfortunately, this is not an easy task: lower bounds on the chromatic number of a graph are still an active topic of research today. Here are a few possibilities that we explored.

- The chromatic number satisfies $\chi(G) \geq \omega(G)$ where $\omega(G)$ is the size of the largest clique of G (that is, the largest subgraph of nodes that are all connected together). Unfortunately, we can easily see that $\omega(G) \leq 3$ in our case which only yields a trivial bound.

- Another well-known bound is given by $\chi(G) \geq \frac{N}{\alpha(G)}$ where N is the number of nodes of the graph and $\alpha(G)$ is the independence number of G, that is, the maximum number of mutually non-adjacent vertices of G. Again, one can check that in our case, $\alpha(G)$ is of the order of $N/8$ which again yields a trivial bound, possibly to be related to Conjecture 4.18 in the first approach of this section.

- Other bounds exist as well, typically involving Lovász's θ-function (1979). Moreover, a number of possible bounds are described by Matoušek and Ziegler (2004) and could be applied to the above graph. Computing these lower bounds may be a challenge in themselves. We see this approach as a valid perspective to the upper bound problem of PI.

Chapter 5

Back from Order-Regular matrices to Acyclic Unique Sink Orientations

In the previous chapters, we introduced a number of tools to study the complexity of Policy Iteration (PI). At the basis stand Acyclic Unique Sink Orientations (AUSOs), a structure that nicely characterizes the partial order of the policies of a Markov Decision Process (MDP). In Chapter 3, we extracted a number of useful properties from this structure and used them to refine the existing upper bound on the number of steps of PI. In an attempt to improve this bound once more in the case of MDPs with two actions per state, we formulated, in Chapter 4, the Order-Regularity condition (OR). Here was the idea: when we apply PI on an AUSO, we explore a sequence of policies (or vertices) that we can write as binary row-vectors and store into an $m \times n$ binary matrix, where m is the number of explored policies (that is, the number of PI steps) and n is the number of states of the MDP, or the dimension of the AUSO. The OR condition can be seen as the translation of the AUSO structure to this binary matrix, where it specifically regulates the progress of PI. (See Section 4.1 for more details.)

OR matrices and AUSOs are closely linked together since the former is a relaxation of the latter. But a natural question arises: is the relaxation tight? Or in other words, is it always possible to find an AUSO that "*realizes*" a given OR matrix? In this chapter, we show that the answer to this question is positive if the OR matrix is Odd-and-Even-Free (OEF), a special case that we introduce in Section 5.2, but negative for general OR matrices. Moreover, we show that matrices with n columns that are both

OEF and OR (or OEF-OR) allow $\sqrt{2}^{n+2} - 1$ both as an upper and lower bound on their maximum number of rows. Incidentally, since it is always possible to design an AUSO that realizes a given OEF-OR matrix, the lower bound also extends to bound the number of steps of PI in AUSOs where it improves by a factor 2 over a bound by Schurr and Szabó (2005).

This chapter is divided into four sections. First in Section 5.1, we give a formal definition of AUSOs. In Section 5.2, we introduce OEF-OR matrices and provide tight bounds on their maximum number of rows. Then, in Section 5.3, we show that there always exists an AUSO that realizes a given OEF-OR matrix. Therefrom, we deduce a new lower bound for the number of steps of PI in AUSOs. Finally, in Section 5.4, we propose an algorithm to find a realizing AUSO to any OR matrix, whenever possible. We thereby show, among other things, that such an AUSO does not always exist.

5.1 Acyclic Unique Sink Orientations of cubes

Definition 5.1 (Cube)**.** Let $N = \{1, \ldots, n\}$. We call *cube* of dimension n or *n-cube* the tuple $\mathcal{C} = (V, E)$ where $V = \{\alpha : \alpha \subseteq N\}$ is the set of *vertices* and $E = \{(\alpha, \alpha \oplus \{k\}) : \alpha \in V, k \in N\}$ is the set of *edges*. Here the "$\oplus$" operator can be seen as a symmetric difference or a XOR. Given $\alpha, \beta \in V$, we define $[\alpha, \beta]$ as (the vertex set of) the *subcube* of (V, E) in which α and β are antipodal, that is:

$$[\alpha, \beta] = \{\gamma \in V : (\alpha \cap \beta) \subseteq \gamma \subseteq (\alpha \cup \beta)\}.$$

Given a subcube $C = [\alpha, \beta]$, we refer to $\alpha \oplus \beta = (\alpha \cup \beta) \setminus (\alpha \cap \beta)$ as the *free dimensions* of C, also denoted $\mathrm{free}(C)$, that is, the set of dimensions along which we are allowed to move in C. We also define the *distance* between any two vertices α and β as $d(\alpha, \beta) \triangleq |\alpha \oplus \beta|$.

Definition 5.1 is illustrated on Figure 5.1. We observe that $[\alpha, \beta] = [\alpha \cap \beta, \alpha \cup \beta]$, where the latter representation can be thought of as the canonical form of the subcube because it is independent of the chosen antipodal vertices that describe it. Furthermore, given $\gamma \in C$, the definition of $\mathrm{free}(C)$ indicates that modifying γ along any dimensions $s \subseteq \mathrm{free}(C)$ should result in a vertex of C. Equivalently, the symmetric difference between two vertices of C should belong to $\mathrm{free}(C)$. We state this simple fact without a proof.

Lemma 5.1. *Let C be a subcube and γ be any vertex of C. Then $\gamma \oplus s \in C$ iff $s \subseteq \mathrm{free}(C)$.*

In Definition 5.1, we also observe that $[\alpha, \beta]$ describes the smallest subcube that contains α and β, who are thus antipodal to each other. It is

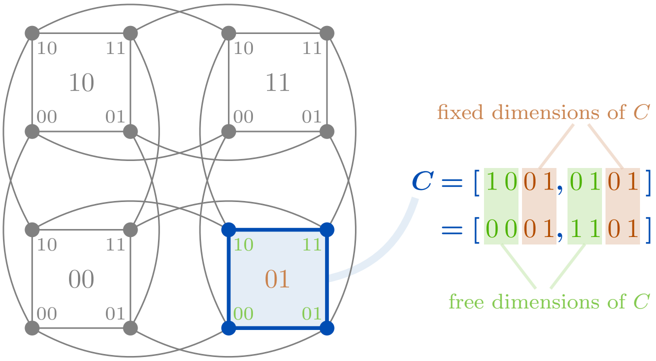


Figure 5.1: Example of a 4-cube with a subcube C of dimension 2. Note that C can be described through any of its pairs of antipodal vertices.

therefore clear that any other subcube containing α and β should also contain $[\alpha, \beta]$, as the following lemma states.

Lemma 5.2. *Let α and β be vertices of a subcube C. Then $[\alpha, \beta] \subseteq C$.*

Proof. Let $C = [\alpha', \beta']$ and let γ be any vertex of $[\alpha, \beta]$. We can see that the following inclusion relations hold:

$$(\alpha' \cap \beta') \subseteq (\alpha \cap \beta) \subseteq \gamma \subseteq (\alpha \cup \beta) \subseteq (\alpha' \cup \beta').$$

where the first and last inclusions come from the fact that $(\alpha' \cap \beta') \subseteq \alpha, \beta \subseteq (\alpha' \cup \beta')$ and the second and third inclusions come from the definition of γ being in $[\alpha, \beta]$. Therefore $[\alpha, \beta] = [\alpha \cap \beta, \alpha \cup \beta] \subseteq [\alpha', \beta'] = C$. □

The following lemma also formulates some—more advanced—cube inclusion property that will be useful in our future developments. Its statement is also represented in a schematic way in Figure 5.2.

Lemma 5.3. *Let α, β and ω be vertices of a subcube such that $d(\alpha, \omega) \geq d(\gamma, \omega)$ for all $\gamma \in [\alpha, \beta]$. Then $[\alpha, \beta] \subseteq [\alpha, \omega]$.*

Proof. Let $\gamma \in [\alpha, \beta]$ and assume that $\gamma \notin [\alpha, \omega]$. Let $s = (\gamma \oplus \alpha) \subseteq (\alpha \oplus \beta)$, where the inclusion comes from Lemma 5.1, and $s \neq \emptyset$ since γ must be different from α. Then, still using Lemma 5.1, $s \not\subseteq (\alpha \oplus \omega)$ and there exists $s' \subseteq s$ such that $s' \neq \emptyset$ and $s' \cap (\alpha \oplus \omega) = \emptyset$. But then we have $\gamma' \triangleq \alpha \oplus s' \subseteq [\alpha, \beta]$, yet $d(\gamma', \omega) = d(\alpha, \omega) + |s'| > d(\alpha, \omega)$ which contradicts the assumptions. □

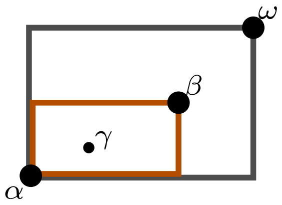


Figure 5.2: The idea of Lemma 5.3 represented schematically. The rectangles can be thought of as a 2D projection of the cubes. If all the vertices in $[\alpha, \beta]$ are “closer” from ω than α, then $[\alpha, \beta] \subseteq [\alpha, \omega]$.

Remark 5.1. Alternatively to Definition 5.1, a subcube C could also be written as a word in $\{0, 1, *\}^n$, where $*$ can be seen as a “wildcard” that can be both 0 and 1 and that is used to represent the free dimensions of C. For instance in Figure 5.1, we could represent the subcube C with the string $**01$. Although this representation allows a more intuitive interpretation than the one of Definition 5.1, it often leads to longer proof mechanisms for the results of this chapter, which is why we prefer the latter.

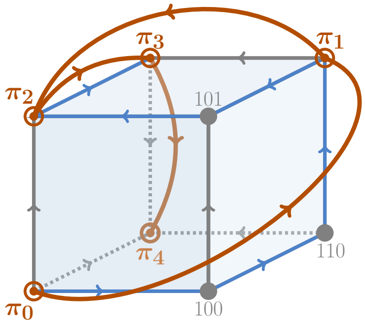
Before we formally define AUSOs, we need to define an orientation on the edges of the cube.

Definition 5.2 (Orientation)**.** We call O an *orientation* (of the edges) of the n-cube iff for all $\gamma \in V$ and all $k \in N$, O contains either $(\gamma, \gamma \oplus \{k\})$ or $(\gamma \oplus \{k\}, \gamma)$ but not both. Such an orientation extends to any subcube of the n-cube and, for all $\alpha, \beta \in V$, we then talk about the orientation of $[\alpha, \beta]$ as induced from O. Furthermore, given an orientation O, we refer to a *sink* of a subcube $[\alpha, \beta]$ as any vertex $\sigma \in [\alpha, \beta]$ that has only incoming edges in $[\alpha, \beta]$ (that is, a vertex for which there is no $k \in \alpha \oplus \beta$ such that $(\sigma, \sigma \oplus \{k\}) \in O$).

Definition 5.3 (Acyclic Unique Sink Orientation)**.** An orientation O of the n-cube $\mathcal{C}$ is called a *Unique Sink Orientation* (USO) iff every subcube of $\mathcal{C}$ has a unique sink. An orientation of $\mathcal{C}$ that is both USO and acyclic is called an *Acyclic Unique Sink Orientation* (AUSO) of $\mathcal{C}$.

Definition 5.4 (Outmap)**.** Given an orientation O of the n-cube $\mathcal{C}$, we define the outmap T^γ of a vertex $\gamma \in V$ as the set $\{k \in N : (\gamma, \gamma \oplus \{k\}) \in O\}$, that is, T^γ is the set of dimensions spanned by the outgoing edges in γ. In that respect, the outmap of a sink is the empty set.

Note that given $\gamma \in [\alpha, \beta]$, $T^\gamma \cap (\alpha \oplus \beta)$ is the restriction of the outmap of γ to the subcube $[\alpha, \beta]$. Interestingly, Szabó and Welzl (2001) show that an orientation O of the n-cube is USO iff the corresponding outmaps satisfy



$$\begin{array}{r}
\pi_0 : \\
\pi_1 : \\
\pi_2 : \\
\pi_3 : \\
\pi_4 :
\end{array}
\begin{bmatrix}
0 & 0 & 0 \\
1 & 1 & 1 \\
0 & 0 & 1 \\
0 & 1 & 1 \\
0 & 1 & 0
\end{bmatrix}$$

Figure 5.3: The extremal example for Policy Iteration in an AUSO of dimension 3. The PI jumps are represented in red. Translating Proposition 3.5 into the AUSO framework, we can guarantee the existence of a path in the cube (here represented in blue) going through each policy of the PI-sequence.

the following condition:

$$(T^\alpha \oplus T^\beta) \cap (\alpha \oplus \beta) \neq \emptyset \quad \text{for all } \alpha, \beta \in V.$$

This condition simply requires that no two vertices share the same outmap, even when restricted to any subcube $[\alpha, \beta]$. It can be seen as an alternative definition of USOs in terms of their outmap.

Definition 5.5 (Policy Iteration)**.** The Policy Iteration update rule can be extended to AUSOs as follows. Given an AUSO, we start from an initial vertex π_0 at step $i = 0$, and then at any step i, we jump from π_i to $\pi_{i+1} = \pi_i \oplus T^{\pi_i}$. The algorithm stops whenever $T^{\pi_i} = \emptyset$, that is, when the (unique) sink has been reached. As for MDPs, this algorithm always converges. As in the previous chapters, we refer to the sequence of vertices $\pi_0, \dots, \pi_{m-1}$ as the PI-sequence. In matrix form, this PI-sequence always corresponds to an OR matrix which we say was *realized* from the AUSO.

The policies of an MDP (with n states and two actions per state) now correspond to vertices of a matching AUSO and the improvement set of a policy corresponds to the outmap of the corresponding vertex in the AUSO. Note that PI is also known as *Bottom-Antipodal* in the AUSO framework. Figure 5.3 illustrates how it works on an example. Schurr and Szabó (2005) showed that it may require $\sqrt{2}^n$ steps to converge on an AUSO. In what follows, we improve this bound by a factor 2.

Remark 5.2. A vertex or an outmap can be represented by either a subset of N or a binary vector (the $k-th$ entry of the binary vector equals 1 iff

$k \in N$). Indeed, both representations are equivalent and any operation on sets has its counterpart in binary vectors and vice versa. Nevertheless, it is often natural to think of a vertex (or a policy) as a binary vector (that represents its "coordinates") and of an outmap as a subset of N (because it represents a *set* of (improving) "directions").

The topic of (A)USOs has already been well studied. See, e.g., the works of Gärtner, Schurr, Szabó and Welzl for more on the topic. In particular, see Gärtner (2003) for comprehensive lecture notes introducing randomized algorithms using the framework of AUSOs.

5.2 Odd-and-Even-Free matrices: a special class of Order-Regular matrices

In this section, we consider a restricted class of OR matrices, that we call Odd-and-Even-Free, and that allow $\sqrt{2}^{n+2} - 1$ both as an upper and lower bound on their maximum number of rows. Though it does not improve the bound from Theorem 4.13, the lower bound also holds for OR matrices. In the next section, we will show that these matrices are a special case for which a corresponding AUSO always exists. This will allow us to extend the lower bound to AUSOs as well, this time resulting in an (admittedly modest) improvement of a factor 2 over the state of the art bound.

We organize this section in three parts. First, we define the class of matrices of interest and formulate some simple properties. Then, we state the key contraction property that leads, in the last part, to the announced bounds. In this section, we assume that the reader is confortable with the notions developed in Chapter 4, Section 4.1, even though we give a short reminder.

5.2.1 Odd-and-Even-Free matrices

As we saw in the previous chapter, Section 4.1, the OR problem consists in bounding the maximum number of rows of an OR matrix with n columns, for any given value of n. Let us recall that we defined any matrix $A \in \{0,1\}^{m \times n}$ to be Order-Regular (OR) whenever for every pair of rows $1 \leq i < j \leq m$, there exists a column k such that

$$A_{i,k} \neq A_{i+1,k} = A_{j,k} = A_{j+1,k}. \tag{5.1}$$

We added the convention that $A_{m+1,k} = A_{m,k}$ to cover the cases where $j + 1 = m + 1$ in the above condition. In other words, an OR matrix needs to satisfy the combinatorial condition (5.1) for every pair of rows (i, j), $1 \leq i < j \leq m$, which we call the constraints of the problem. The number of constraints naturally grows with the number of rows m and we seek for the smallest value of m for which it is no longer possible to satisfy all the constraints (minus 1).

In the dual version of the OR problem, we rather fix the number of rows m and try to satisfy every constraint with as few columns as possible. With that view, the problem at hand becomes a constraint satisfaction problem in which each column contributes in satisfying a subset of the constraints. The contribution of each column can be conveniently visualized using the constraint space introduced in the previous chapter (see Definition 4.2). It can be observed that it is often efficient to satisfy the constraints (i, j) where the index i is odd using different columns than to satisfy the constraints where the index i is even. In the constraint space, this means to color the columns of the space with an odd index i separately from (that is, with different colors than) the columns of the space with an even index i, as illustrated in Figure 5.4. We now formalize this type of construction by introducing Odd-and-Even-Free matrices.

Definition 5.6 (Odd-and-Even-Free matrices)**.** We say that a matrix $A^o \in \{0,1\}^{m\times n}$ is *Odd-Free* (OF) iff for any row i and column k of A^o, it holds that

$$A^o_{i,k} = 0 \;\Rightarrow\; i \text{ is an odd number.}$$

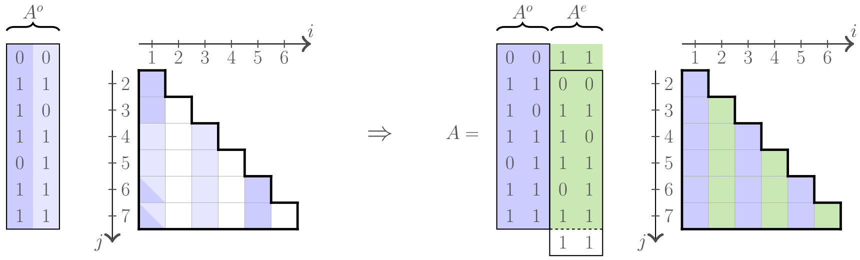
Alternatively, A^o is OF iff $A^o_{i,k} = 1$ for all even row indices i and all columns k. (The rows of A^o with an even index are fixed to only 1's so only the rows with an odd index are "*free*" to be composed of both 0's or 1's, hence the name). We define an *Even-Free* (EF) matrix $A^e \in \{0,1\}^{m\times n}$ analogously with the zeros appearing only on rows with an even index. We call *Odd-and-Even-Free* (OEF) any matrix A that can be written as $[A^o|A^e]$ (after a possible permutation of its columns) where A^o is OF and A^e is EF.

By definition, an OF (resp. EF) matrix cannot be OR but it can possibly satisfy all the constraints (i, j) for which i is odd (resp. even). Following this idea, we say that a matrix A is Odd-Order-Regular (OOR) (resp. Even-Order-Regular (EOR)) whenever for any rows $1 \le i < j < m$ with i odd (resp. even), there exists a column k such that the Condition (5.1) is satisfied, with the usual convention that $A_{j,k} = A_{j+1,k}$ when $j = m$. A is OR iff it is both OOR and EOR.

In terms of the constraint space, an OF matrix A^o covers the columns corresponding to odd values of i whereas an EF matrix A^e covers the columns for even values of i.

Remarks 5.3. The following observations can be made.

1. When searching for large OF matrices that are OOR, we may assume that the matrices have an odd number of rows. Indeed, if an OF



$$\overbrace{\begin{array}{cc} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{array}}^{A^o} \quad \Rightarrow \quad A = \begin{array}{cc|cc} \multicolumn{2}{c}{A^o} & \multicolumn{2}{c}{A^e} \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ & & 1 & 1 \end{array}$$

Figure 5.4: From an OOR matrix A^o, we can build an OR matrix A with twice as many columns and the same number of rows.

matrix with an even number of rows is OOR, then we may add a row of ones at the end of it without altering the OOR nor the OF property. An analogous observation can be made for EF matrices that are EOR.

2. If there exists an OOR (resp. EOR) matrix with n columns and m rows, then it is not hard to see that there must also exist such a matrix that is OF (resp. EF). Therefore, when searching for the largest OOR or EOR matrices with a fixed number of columns, it is enough to explore only the OF and EF matrices.

The following lemma is also a straightforward property of OOR and EOR matrices.

Lemma 5.4. *For any odd k, removing the first k rows of an EOR (resp. OOR) matrix yields an OOR (resp. EOR) matrix.*

Interestingly, if one finds an OOR matrix, then it is easy to find an EOR matrix with one more row, as the following construction shows.

Construction 5.1. From an OOR matrix $A^o \in \{0,1\}^{m \times n}$, one can build an EOR matrix $A^e \in \{0,1\}^{(m+1) \times n}$ as follows:

$$A^e = \begin{bmatrix} r \\ A^o \end{bmatrix}$$

where r is a row of ones[1]. It is easy to check that A^e is EOR. Using this construction, one can also use A^o to build an OR matrix $A \in \{0,1\}^{m \times 2n}$ as Figure 5.4 illustrates.

Of course, a converse construction exists to obtain an OOR matrix from an EOR matrix. Construction 5.1 suggests the following lemma.

[1] Choosing r to be a row of ones ensures that if A^o is OF on top of being OOR, then the constructed matrix A^e is EF.

Lemma 5.5. *There exists an OOR matrix with m rows and n columns iff there exists an EOR matrix with $m+1$ rows and n columns.*

Proof. From an $m \times n$ OOR matrix, Construction 5.1 shows how to build an $(m+1) \times n$ EOR matrix. Moreover, take an $(m+1) \times n$ EOR matrix A^e of the form $\left[\begin{smallmatrix} r \\ A^o \end{smallmatrix}\right]$ where r is the first row of A^e. Then from Lemma 5.4, A^o is OOR. □

5.2.2 The contraction matrix

When studying OF matrices, it is striking that OR constraints are always satisfied by pairs so that if some pair (i,j) is satisfied with j even, then the pair $(i, j+1)$ is also satisfied. Furthermore, we should not care about constraints (i,j) for which i is even because OF matrices are unable to satisfy them anyway. These observations motivate the following contraction of OF matrices.

Definition 5.7 (Contraction matrix)**.** Given an OF matrix $A^o \in \{0,1\}^{m\times n}$ (with m odd according to Remark 5.3), we build its related *contraction matrix* $C(A^o) \in \{0,1\}^{\frac{m+1}{2}\times n}$ as follows: for all i, $1 \le i \le \frac{m+1}{2}$ and $k, 1 \le k \le n$:

$$C_{i,k}(A^o) = \begin{cases} 0 & \text{if } A^o_{2i-1,k} = 0 \\ 1 & \text{if } A^o_{2i-1,k} = 1. \end{cases}$$

In the definition, the even row indices play no role since they correspond to entries that are always equal to 1.

Note that the matrix A^o can be uniquely recovered from its contraction matrix: simply apply the inverse transformation. Furthermore, the OOR property of an OF matrix A^o can be checked looking only at its contraction matrix, as the following lemma states.

Lemma 5.6. *An OF matrix $A^o \in \{0,1\}^{m\times n}$ with m odd satisfies a pair of constraints $(2i-1, 2j-2)$ and $(2i-1, 2j-1)$ for $1 \le i < j \le \frac{m+1}{2}$ iff there exists a column k of A^o such that:*

$$\overline{C}_{i,k}(A^o) \cdot C_{j,k}(A^o) = 1, \tag{5.2}$$

where $\overline{C}(A^o) = \mathbb{1} - C(A^o)$ and $\mathbb{1}$ is a matrix of 1's.

Proof. A^o satisfies the constraints $(2i-1, 2j-2)$ and $(2i-1, 2j-1)$ iff there exists a column k such that $A^o_{2i-1,k} = 0$ and $A^o_{2i,k} = A^o_{2j-2,k} = A^o_{2j-1,k} = A^o_{2j,k} = 1$. Since $A^o_{2i,k}, A^o_{2j-2,k}$ and $A^o_{2j,k}$ are forced to be equal to 1, this is equivalent to requiring the existence of k where $A^o_{2i-1,k} = 0$ and $A^o_{2j-1,k} = 1$. But by definition of the contraction matrix, this is the case iff $C_{i,k}(A^o) = 0$ and $C_{j,k}(A^o) = 1$ which yields the result. □

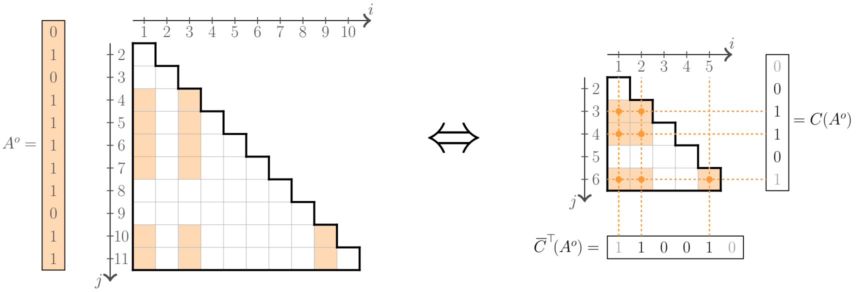


Figure 5.5: The contribution in the constraint space of an OF column-matrix A^o can be represented by its contraction matrix $C(A^o)$ that is half as long. Indeed, constraints of the constraint space are always satisfied by pairs. The simplified constraint space on the right shows how a pair of constraints $(2i-1, 2j-2)$ and $(2i-1, 2j-1)$ is satisfied iff the corresponding product $\overline{C}_i(A^o) \cdot C_j(A^o) = 1$. Note that the first and last rows of the contraction matrix have been drawn in gray because a column of an OF matrix that aims to contribute as much as possible to the constraint space should always start with a "0" and end with a "1".

Example 5.1. Figure 5.5 shows how an OF column-matrix A^o contributes to the constraint space and how the contribution of A^o can be represented through its contraction matrix $C(A^o)$ in a simplified constraint space that is four times as small and that is devoted only to the odd columns of the constraint space. Note that A^o can be uniquely recovered from $C(A^o)$.

The following corollary adapts the OOR condition of A^o to its contraction matrix.

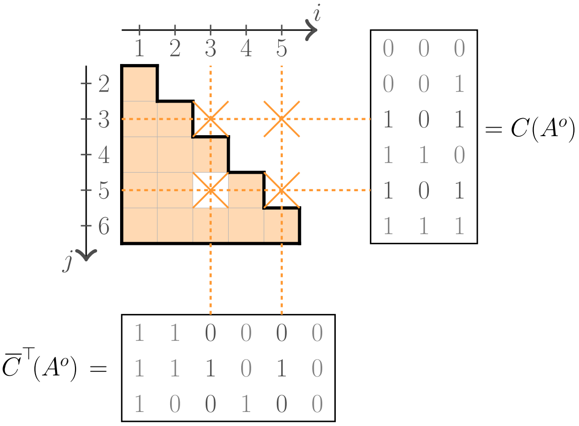
Corollary 5.7. *An OF matrix $A^o \in \{0,1\}^{m \times n}$ with m odd is OOR iff for all i, j with $1 \leq i < j \leq \frac{m+1}{2}$, there exists a column k that verifies (5.2).*

In Corollary 5.7, the simplified constraint (i, j) can be mapped on the pair of constraints $(2i-1, 2j-2)$ and $(2i-1, 2j-1)$ in the original constraint space. We will sometimes abuse of terminology by saying that a contraction matrix $C(A^o)$ is OOR if it verifies every simplified constraint (i, j), $1 \leq i < j \leq \frac{m+1}{2}$, thereby meaning that the corresponding OF matrix A^o is OOR.

5.2.3 Tight bounds

Before we state the main result of this section, we formulate an upper bound for the number of rows of OF matrices.

Proposition 5.8. *Let $A^o \in \{0,1\}^{m \times n}$ be an OF matrix that is OOR. Then $m \leq 2^{n+1} - 1$.*



$$\begin{array}{|ccc|}0&0&0\\0&0&1\\1&0&1\\1&1&0\\1&0&1\\1&1&1\end{array} = C(A^o)$$

$$\bar{C}^\top(A^o) = \begin{array}{|cccccc|}1&1&0&0&0&0\\1&1&1&0&1&0\\1&0&0&1&0&0\end{array}$$

Figure 5.6: An example on which the contraction matrix $C(A^o)$ has two identical rows. Here we see that a hole necessarily appears in the simplified constraint space and hence A^o cannot be OOR.

Proof. First, using Remark 5.3, we may assume that m is odd. Let $C(A^o)$ be the contraction matrix of A^o. We shall prove that $C(A^o)$ cannot contain twice the same row. The idea is illustrated by Figure 5.6. Suppose that there exist i and j, $1 \leq i < j \leq \frac{m+1}{2}$, such that $r^{(i)}$ and $r^{(j)}$, respectively the $i-th$ and $j-th$ rows of $C(A^o)$, are identical. From Lemma 5.6, if we want the constraints $(2i-1, 2j-2)$ and $(2i-1, 2j-1)$ to be satisfied by A^o, we must have $\bar{r}_k^{(i)} \cdot r_k^{(j)} = 1$ for some k. But since we assumed that $r^{(i)} = r^{(j)} \triangleq r$, we have instead $\bar{r}_k^{(i)} \cdot r_k^{(j)} = (1 - r_k) \cdot r_k = 0$ for all k. Therefore, if $C(A^o)$ did contain twice the same row, it would not be OOR.

Therefore, $C(A^o)$ can have at most 2^n rows and from it, we can recover A^o using the reverse contraction transformation from Definition 5.7 and A^o can thus have at most $2^{n+1} - 1$ rows. □

We now state the main result of this section.

Theorem 5.9. *Let $A \in \{0,1\}^{m \times n}$ be an OEF matrix that is OR. Then $m \leq \sqrt{2}^{n+2} - 1$ whenever $n > 1$.*

Proof. First, we decompose A into its OF and EF parts A^o and A^e that we assume to have k and $n-k$ columns respectively. From Proposition 5.8 and with the help of Lemma 5.5, we know that the number of rows of A^o and A^e are bounded by $2^{k+1} - 1$ and 2^{n-k+1} respectively, and that m is bounded by the minimum of the two, that is:

$$m \leq \max_k \min\{2^{k+1} - 1, 2^{n-k+1}\}.$$

Working out the inequality, we find that k should be equal to $\lceil \frac{n}{2} \rceil$ and hence that:

$$m \leq \begin{cases} 2^{\frac{n}{2}+1} - 1 & \text{if } n \text{ is even,} \\ 2^{\frac{n-1}{2}+1} & \text{if } n \text{ is odd.} \end{cases}$$

The fact that $2^{\frac{n}{2}+1} - 1 \geq 2^{\frac{n-1}{2}+1}$ except when $n = 1$ gives the result. □

We will now show that the bound of Theorem 5.9 can be attained for even values of n.

Construction 5.2. Let $C^{(1)} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ and $C^{(\ell)} \in \{0,1\}^{2^\ell \times \ell}$, $\ell > 1$, be defined as follows:

$$C^{(\ell)} = \left[\begin{array}{c|c} 0 & \\ \vdots & C^{(\ell-1)} \\ 0 & \\ \hline 1 & \\ \vdots & C^{(\ell-1)} \\ 1 & \end{array}\right].$$

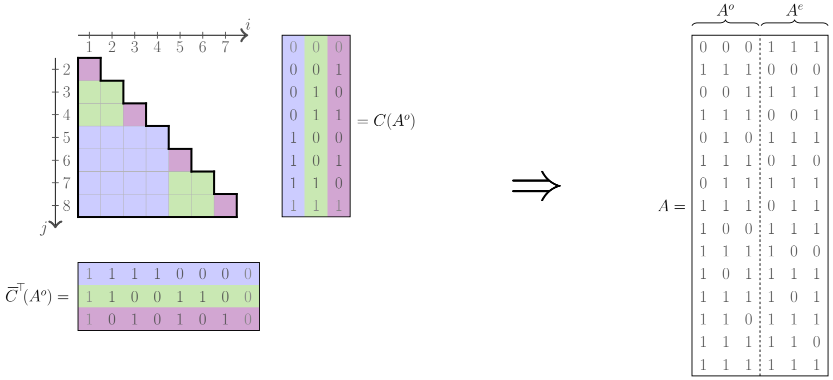
Consequently, the rows of $C^{(\ell)}$ are arranged lexicographically and form a binary counter. Then, we build an OF matrix $A^o \in \{0,1\}^{m \times n}$ from the reverse contraction transformation of Definition 5.7 with the contraction matrix $C(A^o) = C^{(n)}$. The resulting number of rows of A^o is $m = 2^{n+1} - 1$.

Lemma 5.10. *The matrices $A^o \in \{0,1\}^{m \times n}$ obtained from Construction 5.2 are OOR.*

Proof. We show that $C(A^o)$ is OOR by induction on $C^{(\ell)}$. Clearly $C^{(1)}$ is OOR. Now let us assume that $C^{(\ell-1)}$ is OOR. In that case, by definition of $C^{(\ell)}$, for every i, j with $1 \leq i < j \leq 2^{\ell-1}$ or $2^{\ell-1} < i < j \leq 2^{\ell}$, there exists a column k of $C^{(\ell)}$ such that $\overline{C}^{(\ell)}_{i,k} \cdot C^{(\ell)}_{j,k} = 1$. Furthermore, for every i, j with $1 \leq i \leq 2^{\ell-1} < j \leq 2^{\ell}$, we have $\overline{C}^{(\ell)}_{i,1} \cdot C^{(\ell)}_{j,1} = 1$ using the first column of $C^{(\ell)}$. Therefore, $C^{(\ell)}$ is OOR for all $\ell \geq 1$, and in particular for $\ell = n$, hence A^o is OOR as well. □

Theorem 5.11. *For even values of n, it is possible to build an OEF matrix $A \in \{0,1\}^{m \times n}$ with $m = \sqrt{2}^{n+2} - 1$ rows that is OR.*

Proof. Using Construction 5.2, we can build an OF matrix A^o with $\frac{n}{2}$ columns and $\sqrt{2}^{n+2} - 1$ rows. From Lemma 5.10, A^o is OOR. Then, we can use Construction 5.1 with A^o to build an OEF matrix A that is OR and that has twice as many columns as A^o and the same number of rows. □



$$\begin{array}{|ccc|}\hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline\end{array} = C(A^o)$$

$$\overline{C}^{\top}(A^o) = \begin{array}{|cccccccc|}\hline 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline\end{array}$$

$$\Rightarrow$$

$$A = \begin{array}{|ccc:ccc|} \multicolumn{3}{c}{\overbrace{\hphantom{0\ 0\ 0}}^{A^o}} & \multicolumn{3}{c}{\overbrace{\hphantom{0\ 0\ 0}}^{A^e}} \\ \hline 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ \hline\end{array}$$

Figure 5.7: The simplified constraint space on the left can be completely covered by a contraction matrix $C(A^o)$ with 3 columns and 2^3 rows. The rows of $C(A^o)$ correspond to all the possible binary vectors with 3 bits, ordered lexicographically. From the definition of $C(A^o)$, one can easily recover A^o by applying the inverse transformation. Corollary 5.7 guarantees that A^o is OOR. Then Construction 5.1 can be used to build an EOR matrix A^e with the same dimensions as A^o and both matrices can be assembled to create an OR matrix A.

Example 5.2. Figure 5.7 illustrates how to build an OEF matrix with 6 columns and $\sqrt{2}^{6+2} - 1 = 15$ rows that is OR using Constructions 5.2 and 5.1.

Remark 5.4. Using a similar analysis as above, it is also possible to derive $\sqrt{2}^{n+1}$ as a tight upper bound on the number of rows of Order-Regular OEF matrices with an odd number of columns. However, except when $n = 1$, this bound is overruled by the bound of Theorem 5.9.

5.3 From Odd-and-Even-Free matrices to AUSOs

The rows of an n-columns Odd-and-Even-Free Order-Regular (OEF-OR) matrix, described in the previous section, can be seen as the policies of a PI-sequence and represented as vertices of the n-cube. In this section, we show that to any OEF-OR matrix A corresponds an AUSO and a starting vertex such that applying PI from that vertex yields a sequence of policies corresponding to the original matrix A. Therefore, we show that the lower bound from Theorem 5.11 also holds to bound the number of steps of PI on AUSOs. In order to get there, we will first translate some properties of OEF-OR matrices into the AUSO framework. Then we will formulate an algorithm to develop any OEF-OR matrix into an orientation of a cube. We will then show that this orientation is AUSO and that applying PI on this AUSO indeed allows to recover the original OEF-OR matrix.

5.3.1 Odd-and-Even-Free Order-Regular matrices in the cube

Let A be an n-column OEF-OR matrix whose rows $\pi_0, \dots, \pi_{m-1}$ are binary vectors that correspond to vertices of the n-cube. We would like to find an AUSO that realizes A. For that to happen, we need the outmaps of the vertices to match $T^{\pi_{i-1}} = \pi_{i-1} \oplus \pi_i$ for all $0 < i < m$.

Let $C_i = [\pi_{i-1}, \pi_i]$. The fact that the π_i's originate from an OR matrix implies the following lemma.

Lemma 5.12. *If $\pi_0, \dots, \pi_{m-1}$ come from an OR matrix, then for all $j > i$, it holds that $\pi_{i-1} \notin C_j$.*

Proof. It is easy to see that $\pi_{i-1} \notin C_j = [\pi_{j-1}, \pi_j]$ if (and only if) there exists a $k \notin (\pi_{j-1} \oplus \pi_j)$ such that $\pi_{i-1}(k) \neq \pi_{j-1}(k)$ $(= \pi_j(k))$. This is ensured by the OR condition. Indeed, remember that A being OR means that for all $1 \leq i < j \leq m$, there exists a column k such that $A_{i,k} \neq A_{i+1,k} = A_{j,k} = A_{j+1,k}$, and therefore such that $\pi_{i-1}(k) \neq \pi_{j-1}(k) = \pi_j(k)$ (because π_{i-1} corresponds to the $i - th$ row of A). Since $\pi_{j-1}(k) \neq \pi_j(k)$ for all $k \in (\pi_{j-1} \oplus \pi_j)$, there must exist a $k \notin (\pi_{j-1} \oplus \pi_j)$ such that $\pi_{i-1}(k) \neq \pi_{j-1}(k)$ $(= \pi_j(k))$. □

Lemma 5.12 can be seen as the translation of Proposition 3.4 into the framework of cubes. Regarding OEF matrices now, they share the following important property.

Lemma 5.13. *Let $\omega \in V$ be the vertex corresponding to the binary vector of all ones. If $\pi_0, \dots, \pi_{m-1}$ come from an OEF matrix, then $\omega \in C_i$ for all $0 < i < m$.*

Proof. It suffices to see that $\omega = \pi_{i-1} \cup \pi_i \in C_i$ for all i. □

Lemma 5.13 expresses the fact that all cubes C_i share a common vertex ω. This property will be a crucial point towards our result[2].

[2] In fact, Lemma 5.13 is the only property we need to require about the matrix A in addition to its Order-Regularity for the below AUSO reconstruction algorithm to work. This allows to build AUSOs from a wider class of matrices than the OEF-OR matrices alone. However, we do not believe that matrices within that class allow for a better lower bound than the one from Theorem 5.11.

5.3.2 A greedy AUSO reconstruction algorithm

The goal of the following algorithm is to construct an AUSO from an OEF-OR matrix and ensure at the same time that applying PI on the reconstructed AUSO allows recovering the original OEF-OR matrix. For that purpose, we constructively give an orientation to all edges one by one, in agreement with the OEF-OR matrix, such that in the end, we obtain an AUSO. This algorithm is greedy because it never reconsiders a chosen orientation. The main contribution of this chapter is to show that with an OEF-OR matrix as input, this procedure actually leads to an AUSO.

Algorithm 6: Greedy AUSO Reconstruction

```
Input: An OEF-OR matrix $A \in \{0,1\}^{m \times n}$ with the rows $\pi_0, ..., \pi_{m-1}$,
       $i = m - 1$.
Output: An Acyclic Unique Sink Orientation of the $n$-cube that
        realizes $A$ when applying PI from $\pi_0$.
$O = \emptyset$, the already set orientations of the edges.
while $i > 0$ do
    $C_i = [\pi_{i-1}, \pi_i]$.
    $F_i = C_i \setminus \bigcup_{j>i} C_j$.
    for $\alpha \in F_i,\ k \in (\pi_{i-1} \oplus \pi_i)$ do
        $O \leftarrow O \cup \begin{cases} (\alpha, \alpha \oplus \{k\}) & \text{if } d(\alpha, \pi_{i-1}) < d(\alpha \oplus \{k\}, \pi_{i-1}) \\ & \text{or if } \alpha \oplus \{k\} \notin F_i, \\ (\alpha \oplus \{k\}, \alpha) & \text{otherwise.} \end{cases}$
    $i \leftarrow i - 1$.
return $O$.
```

Example 5.3. Algorithm 6 is best understood with an example. Figure 5.8 illustrates how it works on an OEF-OR matrix with 4 columns that achieves the lower bound from Theorem 5.11.

The idea of Algorithm 6 follows the way PI works in AUSOs, that is, jumping from π_{i-1} to π_i in subcubes $C_i = [\pi_{i-1}, \pi_i]$ such that the out-edges of π_{i-1} exactly span C_i. In Algorithm 6, we visit every C_i, starting from the last ones, and choose a “suitable” orientation for all their *unbound* edges (that is, the edges that did not yet receive an orientation, represented in green on Figure 5.8). This description yields two questions:

- *What is a suitable orientation?* The answer comes from Proposition 3.5: we know that there should exist some path from π_{i-1} to π_i

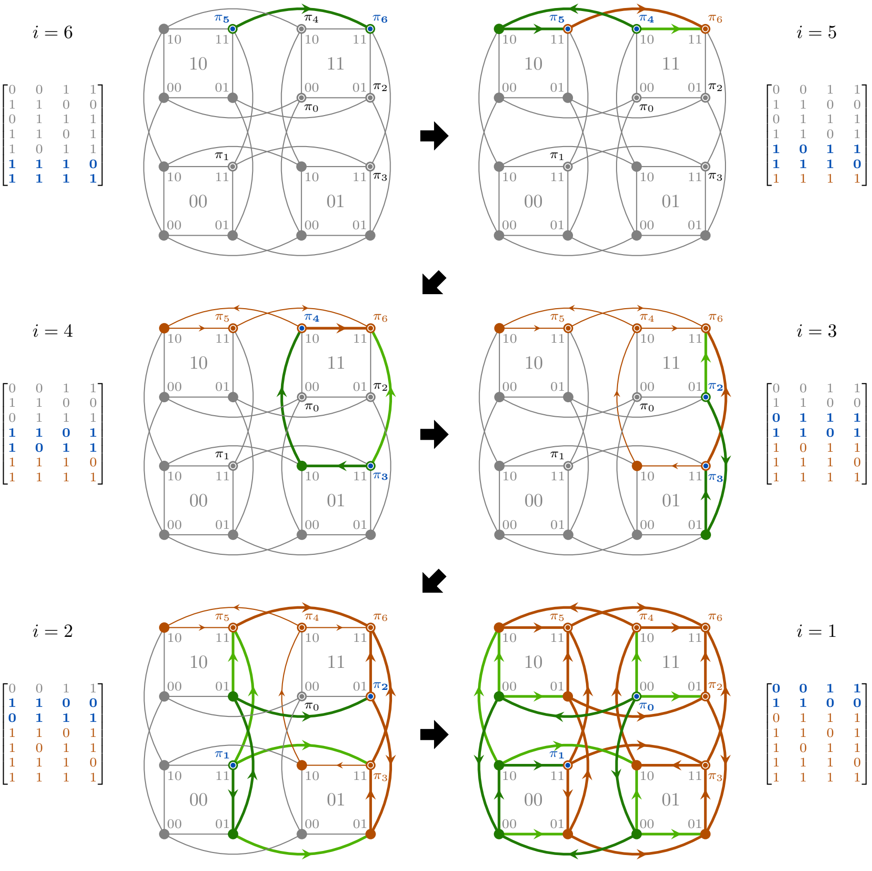


Figure 5.8: At every step i of Algorithm 6, the cube $C_i = [\pi_{i-1}, \pi_i]$ is depicted by the bold edges in the corresponding cube. The policies π_{i-1} and π_i appear in blue (to identify them with the policies of the sequence, note that the digits in the middle of a square correspond to the columns 3 and 4 while the digits in the corners correspond to the columns 1 and 2). The vertices and edges of the previous cubes $C_j, j > i$, appear in red whereas the vertices of F_i and their adjacent edges in C_i (that we refer to as "*unbound*") appear in green. At every step, all the (green) edges of C_i are given an orientation. C_i is guaranteed to be AUSO (Proposition 5.16). Moreover, after the algorithm has finished, applying PI on the obtained orientation with the starting vertex π_0 yields back the original OEF-OR matrix.

in C_i, according to O. So we choose to orient every unbound edge, first towards the previous subcubes $C_j, j > i$, if adjacent, to avoid creating cycles, or towards π_i, hoping that we at least create this path. (Note that the green edges realizing this path appear in a darker green on Figure 5.8.) Here, the unbound edges are the ones adjacent to the vertices of F_i, that is, adjacent to the vertices of C_i that were not contained in the previous subcubes $C_j, j > i$.

- *Why do we start from the last subcubes?* This is indicated by Lemma 5.12. Indeed, this lemma guarantees that $\pi_{i-1} \in F_i$, hence all its adjacent edges in C_i are still unbound. This is a property that we need if we want to both match the outmap of π_{i-1} imposed by the OEF-OR matrix, and start a path to connect π_{i-1} to π_i.

Of course in general, the above greedy approach is not enough to guarantee that the result will be AUSO. However, we will now show that it is indeed enough when an OEF-OR matrix is taken as input.

Observe that at step 6 of Algorithm 6, the same edge can never be considered more than once. Indeed, the edges considered at this step are all in C_i but not in any $C_j, j > i$, since by definition of F_i, the vertex α is not in any of these cubes. Algorithm 6 was designed to allow the two following simple consequences whose proofs are omitted.

Lemma 5.14. *For an orientation O obtained from Algorithm 6, we have $(\pi_{i-1}, \pi_{i-1} \oplus \{k\}) \in O$ iff $k \in (\pi_{i-1} \oplus \pi_i)$.*

Lemma 5.15. *Let $C \subseteq C_i$. If $C \subseteq F_i$, then the vertex of C closest to π_i is the (unique) sink of C. Otherwise, F_i does not contain a sink of C.*

Lemma 5.14 simply ensures that $T^{\pi_{i-1}} = \pi_{i-1} \oplus \pi_i$ which will ensure that it is possible to recover the PI-sequence from applying PI on the recovered AUSO. Lemma 5.15 translates the fact that the vertices of F_i are somehow unstable in the sense that their incident edges in C_i always lead outside F_i if possible. Therefore $C \subseteq C_i$ can have a sink in F_i if and only if it is entirely enclosed in F_i.

5.3.3 The greedy AUSO reconstruction algorithm is correct

Proposition 5.16. *C_i is an AUSO for every $0 < i < m$ according to the orientation O obtained from Algorithm 6 with an OEF-OR matrix given as input.*

Proof. We first show the unique sink property of C_i. Then we will show that it is acyclic.

$\boldsymbol{C_i}$ **is unique sink.** By induction, let us assume that C_j is AUSO for all $j > i$ and let us show that C_i is AUSO. To achieve this, let $C \subseteq C_i$ be any subcube of C_i and let us show that C must have a unique sink. Three cases are possible for C.

1. $\boldsymbol{C \subseteq F_i}$**.** The vertex of C closest to π_i is the unique sink, according to Lemma 5.15.

2. $\boldsymbol{C \subseteq C_i \setminus F_i}$**.** We show that $C \subseteq C_j$ for some $j > i$ which implies that C has a unique sink since C_j is USO. Let α be the vertex furthest away from ω in C. Since $C \cap F_i = \emptyset$, we know that $\alpha \in C_j$ for some $j > i$. Furthermore, we know that $\omega \in C_j$ as well (Lemma 5.13). Therefore we have $C \subseteq [\alpha, \omega] \subseteq C_j$, where the first inclusion comes from Lemma 5.3 and the second comes from Lemma 5.2.

3. $\boldsymbol{C \not\subseteq F_i}$ **and** $\boldsymbol{C \not\subseteq C_i \setminus F_i}$**.** (Hence $C \cap F_i \neq \emptyset$ and $C \cap C_i \setminus F_i \neq \emptyset$.) First let us observe that C can be naturally decomposed as follows:

$$C = \left\{ \bigcup_{\ell > i} C_{(\ell)} \right\} \cup (C \cap F_i),$$

where $C_{(\ell)} \triangleq C \cap C_\ell$ is the subcube of C from stage ℓ of Algorithm 6. We now make two claims about these subcubes. First we show that all non-empty $C_{(\ell)}$'s have a non-empty intersection. Then we show that the "oldest" non-empty subcube attracts all its neighbors and therefore creates a sink in C.

Claim 1. Let ω' be the vertex of C closest to ω, that is, such that $\omega' = \omega \oplus t$ for t satisfying $t \cap \text{free}(C) = \emptyset$. Then $\omega' \in C_{(\ell)}$ for all non-empty $C_{(\ell)}$'s.

> Let $\alpha \in C \cap C_\ell = C_{(\ell)}$. Since α and ω' are in C, we know from Lemma 5.1 that there exists $s \subseteq \text{free}(C)$ such that $\alpha = \omega' \oplus s = \omega \oplus t \oplus s$, where $s \cap t = \emptyset$ because $t \cap \text{free}(C) = \emptyset$. Similarly since α and ω are in C_ℓ, there exists $s' \subseteq \text{free}(C_\ell)$ such that $\alpha = \omega \oplus s'$. Therefore, $s' = t \oplus s$. But since, $s \cap t = \emptyset$, we also have $s \oplus t = s \cup t$, hence we have $t \subseteq s' \subseteq \text{free}(C_\ell)$. Therefore, since from Lemma 5.13, $\omega \in C_\ell$, we can again apply Lemma 5.1 and obtain that $\omega \oplus t = \omega' \in C_\ell$. Since ω' is both in C and C_ℓ, it is also in $C_{(\ell)}$.

Claim 2. Let α, β be neighbors in C and let $m' > i$ be the index of the "oldest" stage where the subcube $C_{(m')}$ is non-empty, that is, such that $C_{(m')} \neq \emptyset$ and there is no $\ell > m'$ for which $C_{(\ell)} \neq \emptyset$. If

$\alpha \in C_{(m')}$ and $\beta \notin C_{(m')}$, then $(\beta, \alpha) \in O$. (That is, all edges between $C_{(m')}$ and $C \setminus C_{(m')}$ are directed towards the vertices of $C_{(m')}$).

> Indeed, given $\alpha \in C_{(m')}$, it appears from step 6 of Algorithm 6 that any neighbor β of α that is not in $C_{(m')}$ must be part of some F_ℓ, $i \leq \ell < m'$ and therefore that the edge between the two vertices must be oriented towards α, that is, $(\beta, \alpha) \in O$.

From the induction hypothesis, $C_{(m')}$ has a unique sink σ, and hence from Claim 2, σ is also a sink of C.

We now show that the sink σ of C is unique. From Lemma 5.15, C has no sink in $C \cap F_i$. So any hypothetical other sink should belong to some subcube $C_{(\ell')}$, $\ell' < m'$, such that $\sigma \notin C_{(\ell')}$ (because $C_{(\ell')}$ must have a unique sink). We know that $C_{(\ell')} \cap C_{(m')}$ is non-empty (thanks to Claim 1) and that it has a unique sink, say τ. But from Claim 2, since $\tau \in C_{(m')}$, it must also be the (unique) sink of $C_{(\ell')}$ which means that the sink τ of $C_{(\ell')}$ must be in $C_{(m')}$. Since $C_{(m')}$ has a unique sink, τ cannot be a sink of C so the sink σ must be unique.

C_i is acyclic. Let $\mathcal{U}_i$ be the vertex set of the graph induced by $\bigcup_{j \geq i} C_j$. We show inductively that this induced graph is acyclic according to O, which implies that C_i is acyclic as well.

By definition, $\mathcal{U}_i = \mathcal{U}_{i+1} \cup F_i$ and $\mathcal{U}_{i+1} \cap F_i = \emptyset$. We now consider any vertex $\alpha \in F_i$ and any $k \in \text{free}(C_i)$ and make the following observations.

1. If $\alpha \oplus \{k\} \notin F_i$, then from step 6 of Algorithm 6, $(\alpha, \alpha \oplus \{k\}) \in O$, so it is impossible to go from vertices of $\mathcal{U}_{i+1}$ to vertices of F_i.

2. Otherwise if $\alpha \oplus \{k\} \in F_i$, then $(\alpha, \alpha \oplus \{k\}) \in O$ iff $d(\alpha, \pi_i) > d(\alpha \oplus \{k\}, \pi_i)$, so edges of F_i always go towards π_i. Therefore, we cannot have a cycle in F_i since taking an edge always decreases the distance to some fixed vertex.

Since $\mathcal{U}_{m-1} = F_{m-1}$, it is acyclic. Moreover, we can see that $\mathcal{U}_i$ is acyclic provided that $\mathcal{U}_{i+1}$ also is. Indeed, $\mathcal{U}_i$ is the disjoint union of two acyclic sets $\mathcal{U}_{i+1}$ and F_i and all edges between the two are oriented towards $\mathcal{U}_{i+1}$ only. □

In case some edge $(\alpha, \alpha \oplus \{k\})$ did not receive an orientation after Algorithm 6 finishes (typically because C_1 differs from the n-cube), then one may choose to orient it towards the global sink π_{m-1} to ensure the AUSO property of the whole. This can be seen as an additional step of Algorithm 6 with a cube $C_0 = [\pi_0 \oplus \omega, \pi_0]$, that is, the complete cube. With this last observation, we can formulate the main theorem of this section.

Theorem 5.17. *To any OEF-OR matrix corresponds an AUSO together with an initial vertex π_0 such that applying PI on the AUSO starting from π_0 allows to recover the original OEF-OR matrix.*

As we will see in the next section, OR matrices do not always come from an AUSO. In that sense, they describe a more general structure. However, Theorem 5.17 demonstrates that under some additional conditions, namely the OEF condition, such an AUSO does exist (so OEF-OR matrices are somehow a restriction of the AUSO condition). As a consequence, all lower bounds that exist on the number of rows of OEF-OR matrices also extend AUSOs. The following corollary consequently extends the lower bound from Theorem 5.11.

Corollary 5.18. *There exists an AUSO of dimension n on which PI requires $\sqrt{2}^{n+2} - 1$ steps to converge.*

It is to be noted that this bound only improves by a factor 2 an existing bound by Schurr and Szabó (2005) that invokes much shorter arguments. However, to my eyes, the arguments that lead to the bound of Corollary 5.18 is among the most elegant contributions of this thesis.

5.4 From general OR matrices to AUSOs

To close this chapter, we answer two natural questions at this point: "*Is it always possible to reconstruct an AUSO from an OR matrix?*" and "*Has the lower bound from Corollary 5.18 any chance of being optimal?*" To answer these questions, we develop an exact AUSO reconstruction algorithm whose returned answer is either "*There is an AUSO realizing this OR matrix and here it is*" or "*There is no such AUSO.*" Using this algorithm, we are able to answer both the above questions with a negative answer.

In the next section, we provide more details about how we designed our exact AUSO reconstruction algorithm using ideas from Constraint Programming. Then, we discuss its results in Section 5.4.2.

5.4.1 An exact AUSO reconstruction algorithm inspired from Constraint Programming

In this section, we present how we construct an AUSO that realizes a given OR matrix using an approach inspired by Constraint Programming (Lecoutre, 2013; Rossi *et al.*, 2006). Despite the strong combinatorial nature of the problem at hand, the algorithm we propose is tractable for OR matrices with up to 7 columns given as input. Moreover, it is exact in the sense that if an AUSO that realizes the OR matrix exists, we will find it, and if it does not, we will find out. The outcome must be an orientation of a cube that needs to satisfy two constraints, which are central for the method.

1. The orientation must be compatible with the outmaps imposed by the OR matrix given as input. More precisely, the outmap of every vertex π_i of the cube, $0 \leq i < m$, appearing as the $(i+1)st$ row in the OR matrix verifies $T^{\pi_i} = \pi_i \oplus \pi_{i+1}$ (with $\pi_m \triangleq \pi_{m-1}$). We refer to this condition as the *outmap constraint.*

2. The orientation must be AUSO.

It turns out that an acyclic orientation of a cube is USO iff the orientation of all the 2-dimensional subcubes (which we also call *2-faces*) of the cube are USO.

Lemma 5.19 (See, e.g., Gärtner (2003) for a proof)**.** *An acyclic orientation of a cube is USO iff all its 2-faces are USO.*

Thanks to Lemma 5.19, we know that an orientation is AUSO iff the two following constraints are satisfied:

- all its 2-faces are AUSO (which we refer to as the *2D constraint*) and,
- the orientation is (globally) acyclic (which we refer to as the *acyclicity constraint*).

Note that in practice when the OR matrix given as input has many rows, we observed that the AUSOs satisfying both the outmap and the 2D constraints often satisfy the acyclicity constraint as well.

The idea of our method is to discard possible orientations of the edges whenever they are not compatible with the above constraints and the already found orientations. It returns a solution if compatible orientations have been found for all edges and it branches if no more orientations can be discarded, possibly backtracking if a branch leads to a contradiction. It stops if all branches led to contradictions.

More precisely, we think of the edges e as variables whose *domain* D_e consists of the possible orientations they can take: either directed towards ω ($\to\omega$) or away from it ($\leftarrow\omega$), with ω being the vertex corresponding to the binary vector of all ones. Originally, the domain of the edges is *full* and we will use the constraints to *prune* the domains, that is, remove incompatible orientations. We say that an edge is *bound* if its domain contains exactly one value left, *unbound* if it has more. We let $D = \bigtimes_{e\in E} D_e$ and observe that whenever all edges are bound, it is possible to recover an orientation $O(D)$ from D.

To prune the domains, we *propagate* the constraints. The way propagation happens differs depending on the constraint. Here is how we do it for each of them.

- **The outmap constraint.** This constraint only helps to launch the search. It allows to bind straightaway all the edges that are adjacent to some π_i, in agreement with their outmaps. Figure 5.9 illustrates this effect on the extremal 5×3 OR matrix. Note that given an $m \times n$ OR matrix as input, the outmap constraint allows to bind up to[3] mn edges among $2^{n-1}n$ in total. Furthermore, the outmap constraint cannot lead to trivial contradictions with the 2D nor the acyclicity constraints.

$$\begin{array}{c} \pi_0 : \\ \pi_1 : \\ \pi_2 : \\ \pi_3 : \\ \pi_4 : \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

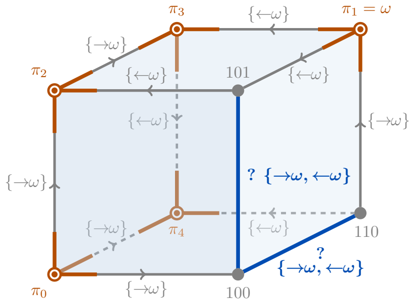


Figure 5.9: The edges that are adjacent to some π_i are marked with a red end. Their orientation can be found using the outmap constraint that is extracted from the OR matrix on the left. After propagation, the remaining edges appear in blue.

- **The 2D constraint.** Every edge is contained in exactly $n-1$ 2-faces. Let e be an unbound edge and F be a 2-face containing e. If the other edges of F are bound, then there is a chance that some orientation of D_e violates the 2D constraint (typically if it creates either a cycle or two sinks, as Figure 5.10 illustrates) and can therefore be pruned.

  To propagate the 2D constraint, we therefore need to make the above test for every edge that is susceptible to lead to some pruning (that is, typically, the unbound edges), and every 2-face containing those edges. Moreover, it may be needed for the same edge to be tested multiple times. Indeed, when two unbound edges e and e' belong to the same 2-face, it may be the case that pruning D_e is only possible after $D_{e'}$ has been pruned, which we illustrate in Figure 5.11. Therefore, in practice, we maintain a queue with the unbound edges that are still susceptible to allow some pruning, while possibly adding elements to this queue

[3] Here, "*up to*" hides the fact that edges between adjacent vertices of the OR matrix are counted twice.

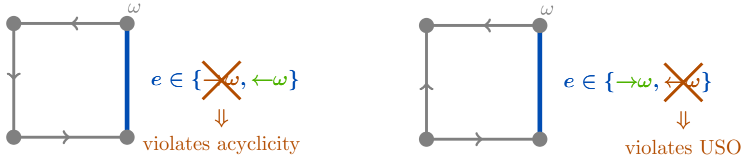
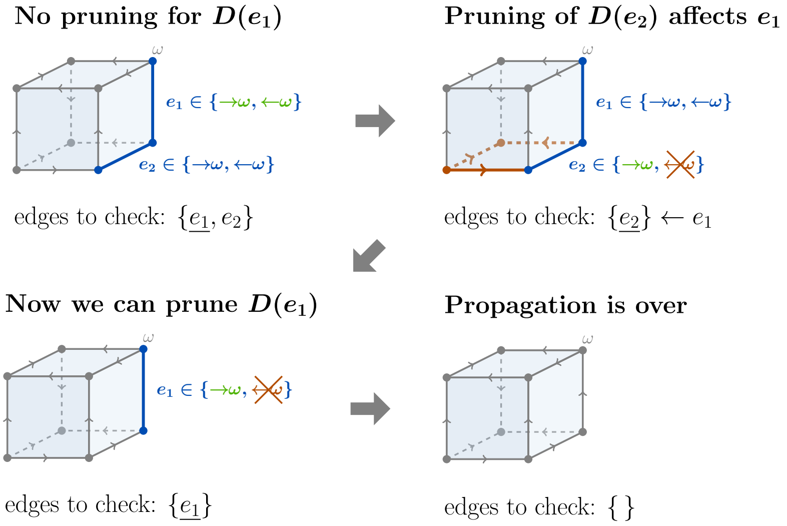

ω
$e \in \{\cancel{\rightarrow\omega}, \leftarrow\omega\}$
⇓
violates acyclicity
ω
$e \in \{\rightarrow\omega, \cancel{\leftarrow\omega}\}$
⇓
violates USO

Figure 5.10: Given an unbound edge e and a 2-face containing e as well as three other bound edges, it is possible to prune D_e iff the edges adjacent to e have opposite orientations. Applying this test to all 2-faces containing e may lead to an empty domain D_e and hence a contradiction.

every time an orientation is removed from a domain. Propagation is over whenever this queue is empty. Interestingly, among the 2-faces that contain four bound edges at the end of this process, the validity of the 2D constraint is guaranteed.


No pruning for $D(e_1)$

Pruning of $D(e_2)$ affects e_1

Now we can prune $D(e_1)$

Propagation is over

ω
$e_1 \in \{\rightarrow\omega, \leftarrow\omega\}$
$e_2 \in \{\rightarrow\omega, \leftarrow\omega\}$
edges to check: $\{\underline{e_1}, e_2\}$
ω
$e_1 \in \{\rightarrow\omega, \leftarrow\omega\}$
$e_2 \in \{\rightarrow\omega, \cancel{\leftarrow\omega}\}$
edges to check: $\{\underline{e_2}\} \leftarrow e_1$
ω
$e_1 \in \{\rightarrow\omega, \cancel{\leftarrow\omega}\}$
edges to check: $\{\underline{e_1}\}$
ω
edges to check: $\{\,\}$

Figure 5.11: The order in which we apply the 2D constraint on the edges to prune their domain influences the speed of the propagation.

- **The acyclicity constraint.** Let $e = (\alpha\backslash\{k\}, \alpha \cup \{k\})$ be an unbound edge. If there exists a directed path from $\alpha\backslash\{k\}$ to $\alpha \cup \{k\}$ in $O(D)$, then we remove $\to\omega$ from D_e, and the other way around if there exists a directed path from $\alpha \cup \{k\}$ to $\alpha \backslash \{k\}$. Being less frequently useful and more costly than the others, we only use this constraint whenever no more propagation of the 2D constraint is possible, relaunching the propagation of the latter whenever the acyclicity allows to prune a domain.

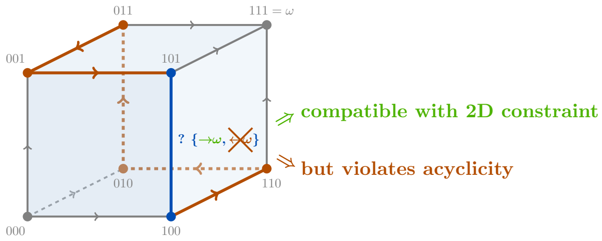

011
$111 = \omega$
001
101
? $\{\to\omega, \cancel{\leftarrow\omega}\}$
compatible with 2D constraint
but violates acyclicity
010
110
000
100

Figure 5.12: One of the rare cases where the 2D constraint finished propagation and the acyclicity still helps pruning a domain.

Using the above ingredients, we are now ready to formulate the exact AUSO reconstruction algorithm in Algorithm 7.

Note that we can always use necessary conditions about the problem to help pruning the domains even more. For instance, we could use Proposition 3.5 requiring that there exists a directed path from π_{i-1} to π_i in the cube, for all $0 < i < m$. We call it the *path constraint*. Such redundant constraint may help to achieve more pruning and less branching, which can be beneficial when the orientation of many edges are unknown.

5.4.2 What the exact AUSO reconstruction algorithm tells us

We used Algorithm 7 on several particular OR matrices with the following observations.

1. There does not always exist an AUSO that realizes a given OR matrix. We show this for instance by applying Algorithm 7 on the extremal 21×6 OR matrix or on some 13×5 OR matrices[4].

[4] Up to symmetry and equivalences, there are respectively 1, 1, 1, 4 and 1 extremal OR matrices for $n = 2$ to 6. Regarding $n = 7$, there are no matrices of 34 rows but more than a hundred of them with 33 rows.

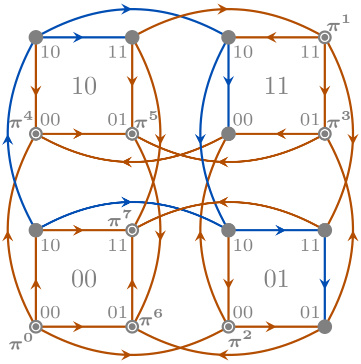
Algorithm 7: Exact AUSO Reconstruction

```
Input: An OR matrix $A \in \{0,1\}^{m \times n}$ with the rows $\pi_0, ..., \pi_{m-1}$.
Initialization: Set $D = \bigtimes_{e \in E}\{\rightarrow\omega, \leftarrow\omega\}$ to be the (originally full)
                domain of the edges and propagate the outmap
                constraint.
Output: AUSOsearch($D$): displays an AUSO that realizes $A$ when
        applying PI from $\pi_0$, if it exists.
Function AUSOsearch($D$)
    Propagate the 2D and the acyclicity constraints.
    //If some edge has an empty domain: backtrack
    if $|D_e| = 0$ for some $e \in E$ then
        return
    //If all edges are bound: display the solution
    else if $|D_e| = 1$ for all $e \in E$ then
        //It could be that $O(D)$ is not acyclic so we make a
        final check
        if $O(D)$ is acyclic then
            Display $O(D)$.
        else
            return
    //Otherwise choose an unbound edge and branch on it
    else
        Let $e$ be an unbound edge.
        AUSOsearch($D$ with $D_e = \{\rightarrow\omega\}$).
        AUSOsearch($D$ with $D_e = \{\leftarrow\omega\}$).
```

2. The $2\sqrt{2}^n - 1$ lower bound from Corollary 5.18 for the number of steps of PI in AUSOs is not optimal. Indeed, there exists a (single) AUSO that realizes the extremal 8×4 OR matrix, as shown by Figure 5.13. Note that the Greedy AUSO reconstruction algorithm from Algorithm 6 applied on this matrix fails to recover this AUSO.

3. The Fibonacci sequence fits the maximal number of steps of PI in AUSOs for $n = 1$ to 5, but already not for $n = 6$.

4. We designed a weaker version of Algorithm 7 in which we only use the outmap and the path constraints for the propagation. We then applied it on a 13×5 OR matrix for which we failed to recover a matching



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

Figure 5.13: An AUSO that realizes the extremal 8 × 4 OR matrix. We first propagated the outmap constraint to obtain the orientation of the red edges, then the 2D constraint to obtain the orientation of the blue edges, and converged without even needing to branch.

AUSO. Interestingly, we found that there is also no orientation of the 5-cube in which a path connects the 13 policies of the OR matrix. This means that the condition from Proposition 3.5 does not apply to OR matrices in general, although it was a key argument in the proof of our $2 \cdot \frac{2^n}{n} + o\left(\frac{2^n}{n}\right)$ upper bound from Theorem 3.9. This means that this latter bound does not apply as such to OR matrices, for which the $2^{n-1}+1$ upper bound from Proposition 4.5 remains the best one we know.

These observations tend to indicate that there is still room to refine the lower, upper and conjectured bounds that we know so far. We however believe that the next improvement will not come without breaking a sweat. At this stage, our analysis did not reveal any real opening towards new results, except maybe for the ideas described in Section 4.5. It thus remains a real challenge. Who will be up for it ?

Chapter 6

An unexpected journey: conclusions and perspectives

When we started looking at the complexity of Policy Iteration (PI), it was only months before Fearnley (2010) and Ye's (2011) results came out. At that time, there were still good hopes that PI would prove to be a polynomial time algorithm and of course, as many others, we were also trying to verify this statement. We even found some interesting properties on the way that we were trying to exploit.

When Fearnley's exponential complexity result appeared in the literature, we managed to adapt it to another important case. Showing that PI may run in exponential time on general discounted-reward Markov Decision Processes (MDPs) and, to observe that, identifying a lower bound on the rate at which the discount factor should tend to 1 when the problem size grows was indeed our first published result. On the way, we left open the question of improving the lower bound on this rate.

Despite Fearnley's result, the gap between upper and lower bounds was still enormous and had to be reduced. Shortly after we submitted our first result, we met with Thomas Hansen and on that occasion, we realized that the properties we had been studying were equivalent to his Order-Regularity (OR) condition on binary matrices, still unpublished at the time. Moreover, except for Hansen and Zwick's "Fibonacci" conjecture, this condition had never really been exploited before, and it was opening a wide range of new possibilities. Showing or disproving the Fibonacci conjecture was an ambitious and exciting goal that we decided to undertake.

Doing so, it quickly appeared to us that the OR structure was more intricated than anticipated and it took us some time and a number of unfruitful approaches to obtain our first results from it. In the end, the results in question are not the ones we were expecting, even less the ones we were secretly hoping for, but hopefully they are worth telling.

Eventually, we did achieve our goal of reducing the gap between the lower and upper bounds on the complexity of PI when applied to Acyclic Unique Sink Orientations (AUSOs). We indeed improved Schurr and Szabó's $\sqrt{2}^n$ lower bound (2005) and Mansour and Singh's $6\frac{2^n}{n}$ upper bound (1999) for Cube AUSOs, yet only by a factor 2 and 3 respectively. We also extended the upper bound for MDPs with up to k actions per state (and left open the case where the number of actions would be distinct in each state). Even though these improvements are admittedly modest, I believe that our proof showing the limits of the classical tools to obtain upper bounds on PI is useful in that it serves as a warning for the next researcher that would consider attacking the problem from the same angle. Moreover, the sequence of ideas that led to the improvement of the lower bound is personally my favorite of the whole thesis.

Still, the lower and the upper bounds remain far apart from each other and in between this gap stands the true worst case behavior we are looking for. In that respect, Hansen and Zwick's Fibonacci conjecture was suggesting a nice trade-off between the two bounds. When we disproved F_{n+2}—the $(n+2)nd$ Fibonacci number—to be the true maximum number of rows of an OR matrix, we provided a striking new example of a Fibonacci forgery, that is, a sequence that does look like but is not the Fibonacci sequence (Stewart, 1995). It is indeed not common to observe a "wild" (or "not made up") sequence that matches the Fibonacci sequence up to the 8th number. On the other hand, we do not rule out the Fibonacci sequence as a possible upper bound on the number of steps of PI. In fact, since the maximum size of an OR matrix with $n = 7$ columns is lower than the initially expected Fibonacci number, I even believe that the true worst case behavior should stand somewhere below $O(1.618^n)$, in the same spirit as the state of the art upper bound on the complexity of Fibonacci Seesaw from Szabó and Welzl (2001) which is also slightly lower than this bound. In that respect, our $\Omega(1.427^n)$ lower bound for OR matrices allows to restrict what can be hoped for as an upper bound, at least using the OR framework.

Talking of which, I feel that we do not loose much generality considering OR matrices instead of AUSOs. Indeed, the AUSO reconstruction algorithms that we have designed in the last chapter revealed that a realizing AUSO exists for many OR matrices, even for some of the largest ones. Moreover, OR matrices do present a novel approach to the complexity problem of PI and I therefore believe that they offer the best starting point currently known to study the question.

However, *best* does not mean *good*. Indeed, despite a simple and appealing formulation, I feel that additional ideas are necessary in order to make further progress with the OR condition. The approaches described in Section 4.5 could be a good starting point to identify such ideas, especially the graph coloring one. Alternatively, our analysis has left open a number of questions. For instance, although it is probably hard, it would be interesting to determine if our $\Omega(1.427^n)$ lower bound for extremal OR matrices can be extended to PI in AUSOs, or if our $\sqrt{2}^{n+2} - 1$ lower bound for the maximum number of steps of PI in AUSOs can be extended to MDPs. Another interesting idea that I would have liked to explore more would be to propose new multi-hop deterministic switching rules for PI and then try to fit the OR formulation to these new algorithms in the hope of a more tractable complexity analysis. Ultimately, even though many questions still expect an answer, I would not recommend studying PI's complexity as the next thesis topic because results seem too uncertain.

To conclude, I would like to mention that in practice, a good combinatorial algorithm like PI, even if known to be exponential in the worst case, is often competitive against the best weakly polynomial time algorithms such as the Interior Point method of Karmarkar (1984). And the fact that PI exhibits competitive performance against his peers, also theoretically, is interesting and even encouraging. The next step would be to assess these performances, for instance by applying *smoothed analysis* to PI in a similar fashion as what has been done for the Simplex method by Spielman and Teng (2004). To our knowledge, such an analysis has not yet been undertaken in the past. It may prove challenging because of the multi-switch nature of PI[1]. Nevertheless, I believe that smoothed analysis could finally shed some light on why PI is so efficient in practice.

[1] Likewise, this same multi-switch nature seems to prevent Post and Ye's polynomial time complexity arguments for Deterministic MDPs to be adapted to PI. Could this be related to the fact that Fearnley's example is also very close to be deterministic ? This question could be another interesting one to explore.

Bibliography

Adler, I., Papadimitriou, C., and Rubinstein, A. 2014. On simplex pivoting rules and complexity theory. In Proceedings of the 17th Conference on Integer Programming and Combinatorial Optimization (IPCO), pages 13–24.

Akian, M. and Gaubert, S. 2013. Policy iteration for perfect information stochastic mean payoff games with bounded first return times is strongly polynomial. arXiv preprint arXiv:1310.4953.

Akian, M., Cochet-Terrasson, J., Detournay, S., and Gaubert, S. 2012. Policy iteration algorithm for zero-sum multichain stochastic games with mean payoff and perfect information. arXiv preprint arXiv:1208.0446.

Altman, E. 1999. *Constrained Markov decision processes*, volume 7. CRC Press.

Altman, E. 2002. *Applications of Markov Decision Processes in communication networks*. Springer.

Amenta, N. and Ziegler, G. M. 1999. Deformed products and maximal shadows of polytopes. Contemporary Mathematics, 223:57–90.

Andersson, D. and Miltersen, P. 2009. The Complexity of Solving Stochastic Games on Graphs. Algorithms and Computation, pages 112–121.

Avis, D. and Chvátal, V. 1978. Notes on Bland's pivoting rule. Polyhedral Combinatorics, pages 24–34.

Bäuerle, N. and Rieder, U. 2011. *Markov Decision Processes with Applications to Finance*. Springer.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press.

Bertot, Y. and Castéran, P. 2013. *Interactive theorem proving and program development: Coq?Art: the calculus of inductive constructions*. Springer Science & Business Media.

Bertsekas, D. P. 1996. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts.

Bertsekas, D. P. 2007. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 3rd edition.

Bertsekas, D. P. and Tsitsiklis, J. N. 1991. An analysis of Stochastic Shortest Path problems. Mathematics of Operations Research, 16(3):580-595.

Bertsimas, D. and Vempala, S. 2004. Solving convex programs by random walks. Journal of the ACM (JACM), 51(4):540–556.

Chang, H. S. 2007. A policy improvement method for constrained average Markov decision processes. Operations Research Letters, 35(4):434–438.

Chang, H. S., Lee, H.-G., Fu, M. C., and Marcus, S. I. 2005. Evolutionary Policy Iteration for solving Markov Decision Processes. IEEE Transactions on Automatic Control, 50(11):1804–1808.

Chao, G. H. 2013. Production and availability policies through the Markov Decision Process and myopic methods for contractual and selective orders. European Journal of Operational Research, 225(3):383–392.

Chevalier, P. B. and Wein, L. M. 1993. Scheduling networks of queues: heavy traffic analysis of a multistation closed network. Operations Research, 41(4):743–758.

Cipra, B. A. 2000. The best of the 20th century: editors name top 10 algorithms. SIAM news, 33(4):1–2.

Condon, A. 1992. The complexity of stochastic games. Information and Computation, 96(2):203–224.

Csáji, B. C., Jungers, R. M., and Blondel, V. D. 2014. PageRank optimization by edge selection. Discrete Applied Mathematics, 169:73–87.

Dantzig, G. B. 1963. *Linear programming and extensions.* Princeton university press.

Daskalakis, C. and Papadimitriou, C. 2011. Continuous local search. In *In Proceedings of the 22nd Symposium on Discrete Algorithms (SODA)*, pages 790–804. SIAM.

Dekker, R., Nicolai, R. P., and Kallenberg, L. 2007. Maintenance and Markov decision models. Encyclopedia of Statistics in Quality and Reliability.

Dunagan, J. and Vempala, S. 2008. A simple polynomial-time rescaling algorithm for solving linear programs. Mathematical Programming, 114(1):101–114.

Ehrenfeucht, A. and Mycielski, J. 1979. Positional strategies for mean payoff games. International Journal of Game Theory, 8(2):109–113.

Emerson, E. A. and Jutla, C. S. 1991. Tree automata, μ-calculus and determinacy. In Proceedings of the 32nd Symposium on Foundations of Computer Science (FOCS), pages 368–377.

Fearnley, J. 2010. Exponential lower bounds for Policy Iteration. In Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP), pages 551–562.

Fearnley, J. and Savani, R. 2015a. The Complexity of All-switches Strategy Improvement. arXiv preprint arXiv:1507.04500.

Fearnley, J. and Savani, R. 2015b. The complexity of the simplex method. In Proceedings of the 47th Symposium on the Theory of Computing (STOC), pages 201–208.

Feinberg, E. A. and Huang, J. 2013. Strong polynomiality of policy iterations for average-cost MDPs modeling replacement and maintenance problems. Operations Research Letters, 41(3):249–251.

Feinberg, E. A. and Huang, J. 2014. The value iteration algorithm is not strongly polynomial for discounted dynamic programming. Operations Research Letters, 42(2):130–131.

Feinberg, E. A. and Huang, J. 2015. On the Reduction of Total-Cost and Average-Cost MDPs to Discounted MDPs. arXiv preprint arXiv:1507.00664.

Feinberg, E. A., Huang, J., and Scherrer, B. 2014. Modified policy iteration algorithms are not strongly polynomial for discounted dynamic programming. Operations Research Letters, 42(6):429–431.

Fercoq, O., Akian, M., Bouhtou, M., and Gaubert, S. 2013. Ergodic control and polyhedral approaches to PageRank optimization. IEEE Transactions on Automatic Control, 58(1):134–148.

Friedmann, O. 2009. An exponential lower bound for the Parity Game Strategy Improvement algorithm as we know it. In Proceedings of the 24th Annual IEEE Symposium on Logic In Computer Science (LICS), pages 145–156.

Friedmann, O. 2011. A Subexponential Lower Bound for Zadeh's Pivoting Rule for Solving Linear Programs and Games. Integer Programming and Combinatoral Optimization, pages 192–206.

Friedmann, O., Hansen, T., and Zwick, U. 2011. Subexponential lower bounds for randomized pivoting rules for the Simplex algorithm. In Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC), pages 283–292.

Gärtner, B. 1998. Combinatorial linear programming: Geometry can help. In Proceedings of the 2nd Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM), pages 82–96.

Gärtner, B. 2002. The Random-Facet simplex algorithm on combinatorial cubes. Random Structures & Algorithms, 20(3):353–381.

Gärtner, B. 2003. Randomized algorithms: an introduction through unique sink orientations. Lecture notes.

Gärtner, B. and Schurr, I. 2006. Linear programming and unique sink orientations. In Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithm (DA), pages 749–757.

Gärtner, B., Henk, M., and Ziegler, G. M. 1998. Randomized simplex algorithms on Klee-Minty cubes. Combinatorica, 18(3):349–372.

Gärtner, B., Morris, W. D., and Rüst, L. 2005. *Unique sink orientations of grids*. Springer.

Gärtner, B., Matoušek, J., Rüst, L., and Škovroň, P. 2008. Violator spaces: Structure and algorithms. Discrete Applied Mathematics, 156(11):2124–2141.

Gerencsér, B., Hollanders, R., Delvenne, J.-C., and Jungers, R. M. 2015. A complexity analysis of Policy Iteration through combinatorial matrices arising from Unique Sink Orientations. arXiv preprint arXiv:1407.4293.

Goldfarb, D. and Sit, W. Y. 1979. Worst case behavior of the steepest edge simplex method. Discrete Applied Mathematics, 1(4):277–285.

Grädel, E., Thomas, W., and Wilke, T. 2002. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media.

Guo, X. and Hernández-Lerma, O. 2009. *Continuous-time Markov decision processes*. Springer.

Hansen, T. 2012. *Worst-case analysis of Strategy Iteration and the Simplex method.* Ph.D. thesis, Aarhus University, Science and Technology, Department of Computer Science.

Hansen, T. D. and Zwick, U. 2015. An improved version of the Random-Facet pivoting rule for the Simplex algorithm. In Proceedings of The 47th ACM Symposium on Theory of Computing (STOC), pages 209–218.

Hansen, T. D., Miltersen, P. B., and Zwick, U. 2013. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. Journal of the ACM (JACM), 60(1):1.

Hansen, T. D., Paterson, M., and Zwick, U. 2014. Improved upper bounds for Random-Edge and Random-Jump on abstract cubes. In Proceedings of the 25th Symposium On Discrete Algorithms (SODA), pages 874–881.

Higham, N. 1996. *Accuracy and stability of numerical algorithms.* SIAM.

Hoeffding, W. 1963. Probability inequalities for sums of bounded random variables. Journal of the American statistical association, 58(301):13–30.

Hoffman, A. J. and Karp, R. M. 1966. On nonterminating stochastic games. Management Science, 12(5):359–370.

Hollanders, R., Delvenne, J.-C., and Jungers, R. M. 2012. The complexity of Policy Iteration is exponential for discounted Markov Decision Processes. In Proceedings of the 51st IEEE Conference on Decision and Control (CDC), pages 5997–6002.

Hollanders, R., Bernardes, D. F., Mitra, B., Jungers, R. M., Delvenne, J.-C., and Tarissan, F. 2014. Data-driven traffic and diffusion modeling in peer-to-peer networks: A real case study. Network Science, 2(03):341–366.

Hollanders, R., Gerenscér, B., Delvenne, J.-C., and Jungers, R. M. 2015. Improved bound on the worst case complexity of Policy Iteration. Accepted for publication in Operations Research Letters.

Holt, F. and Klee, V. 1999. A proof of the strict monotone 4-step conjecture. Contemporary Mathematics, 223:201–216.

Howard, R. 1960. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA.

Hunter, P. 2007. *Complexity and infinite games on finite graphs.* Ph.D. thesis, University of Cambridge.

Jeroslow, R. G. 1973. The simplex algorithm with the pivot rule of maximizing criterion improvement. Discrete Mathematics, 4(4):367–377.

Jurdzinski, M., Paterson, M., and Zwick, U. 2008. A deterministic subexponential algorithm for solving parity games. SIAM Journal on Computing, 38(4):1519–1532.

Kalai, G. 1992. A subexponential randomized simplex algorithm. In Proceedings of the 24th ACM Symposium on Theory of Computing (STOC), pages 475–482.

Kalai, G. 1997. Linear programming, the simplex algorithm and simple polytopes. Mathematical Programming, 79(1-3):217–233.

Kalai, G. and Kleitman, D. J. 1992. A quasi-polynomial bound for the diameter of graphs of polyhedra. Bulletin of the American Mathematical Society, 26(2):315–316.

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. Combinatorica, 4(4):373-395.

Kelner, J. A. and Spielman, D. A. 2006. A randomized polynomial-time simplex algorithm for linear programming. In Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), pages 51–60.

Khachiyan, L. G. 1980. Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics, 20(1):53–72.

Klee, V. and Minty, G. J. 1970. How good is the simplex algorithm? Technical report, DTIC Document.

Lecoutre, C. 2013. *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. John Wiley & Sons.

Lovász, L. 1979. On the Shannon capacity of a graph. IEEE Transactions on Information Theory, 25(1):1–7.

Madani, O. 2000. *Complexity results for infinite-horizon markov decision processes*. Ph.D. thesis, University of Washington.

Mahadevan, S. and Connell, J. 1992. Automatic programming of behavior-based robots using reinforcement learning. Artificial intelligence, 55(2):311–365.

Mansour, Y. and Singh, S. 1999. On the complexity of Policy Iteration. In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI), pages 401–408.

Matoušek, J. 1994. Lower bounds for a subexponential optimization algorithm. Random Structures & Algorithms, 5(4):591–607.

Matoušek, J. and Ziegler, G. M. 2004. Topological lower bounds for the chromatic number: A hierarchy. Jahresbericht der Deutschen Mathematiker-Vereinigung, 106:71–90.

Matoušek, J., Sharir, M., and Welzl, E. 1996. A subexponential bound for linear programming. Algorithmica, 16(4-5):498–516.

Meister, U. and Holzbaur, U. 1986. A Polynomial Time Bound for Howard's Policy Improvement Algorithm. OR Spectrum, 8(1):37–40.

Meyn, S. P. 2008. *Control techniques for complex networks*. Cambridge University Press.

Morris Jr, W. D. 2002a. Distinguishing cube orientations arising from linear programs. Manuscript.

Morris Jr, W. D. 2002b. Randomized pivot algorithms for P-matrix linear complementarity problems. Mathematical Programming, 92(2):285–296.

Post, I. and Ye, Y. 2015. The simplex method is strongly polynomial for deterministic markov decision processes. Mathematics of Operations Research.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Puterman, M. L. and Shin, M. C. 1978. Modified policy iteration algorithms for discounted Markov decision problems. Management Science, 24(11):1127–1137.

Rao, S. S., Chandrasekaran, R., and Nair, K. P. K. 1973. Algorithms for discounted stochastic games. Journal of Optimization Theory and Applications, 11(6):627–637.

Rossi, F., Van Beek, P., and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Santos, F. 2012. A counterexample to the Hirsch Conjecture. Annals of Mathematics, 176(1):383–412.

Scherrer, B. 2013. Improved and generalized upper bounds on the complexity of Policy Iteration. In Proceedings of the 27th Conference on Advances in Neural Information Processing Systems (NIPS), pages 386–394.

Scherrer, B. 2014. Approximate policy iteration schemes: a comparison. arXiv preprint arXiv:1405.2878.

Schurr, I. and Szabó, T. 2004. Finding the sink takes some time: An almost quadratic lower bound for finding the sink of unique sink oriented cubes. Discrete and Computational Geometry, 31(4):627–642.

Schurr, I. and Szabó, T. 2005. Jumping doesn't help in abstract cubes. In Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization (IPCO), pages 225–235.

Shapley, L. S. 1953. Stochastic games. Proceedings of the National Academy of Sciences of the United States of America, 39(10):1095.

Sharir, M. and Welzl, E. 1992. A combinatorial bound for linear programming and related problems. In Proceedings of the 9th Symposium on Theoretical Aspects of Computer Science (STACS), pages 567–579.

Smale, S. 1998. Mathematical Problems for the Next Century.

Spaan, M. T. J. 2012. Partially observable Markov decision processes. Reinforcement Learning, pages 387–414.

Spielman, D. and Teng, S. 2004. Smoothed analysis of algorithms: why the Simplex algorithm usually takes polynomial time. Journal of the ACM (JACM), 51(3):385–463.

Stewart, I. 1995. Fibonacci forgeries. Scientific American, 272:102–105.

Stickney, A. and Watson, L. 1978. Digraph models of Bard-type algorithms for the linear complementarity problem. Mathematics of Operations Research, 3(4):322–333.

Sutton, R. S. and Barto, A. G. 1998. *Reinforcement Learning*. The MIT Press.

Szabó, T. and Welzl, E. 2001. Unique Sink Orientations of Cubes. In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pages 547–555.

Todd, M. J. 2002. The many facets of linear programming. Mathematical Programming, 91(3):417–436.

Todd, M. J. 2014. An Improved Kalai–Kleitman Bound for the Diameter of a Polyhedron. SIAM Journal on Discrete Mathematics, 28(4):1944–1947.

Tseng, P. 1990. Solving H-horizon, stationary Markov Decision Problems in time proportional to log(H). Operations Research Letters, 9(4):287-297.

Vöge, J. and Jurdziński, M. 2000. A discrete strategy improvement algorithm for solving parity games. Computer Aided Verification, pages 202–215.

White, D. J. 1993. Survey of applications of Markov Decision Processes. The Journal of the Operational Research Society, 44(11):1073-1096.

Ye, Y. 2011. The Simplex and Policy-Iteration methods are strongly polynomial for the Markov Decision Problem with a fixed discount rate. Mathematics of Operations Research, 36(4):593–603.

Zwick, U. and Paterson, M. 1996. The complexity of mean payoff games on graphs. Theoretical Computer Science, 158(1):343–359.

Acronyms

MDP	Markov Decision Process
PI	Policy Iteration
2TBSG	Two-Player Turn-Based Stochastic Game
SI	Strategy Iteration
USO	Unique Sink Orientation
AUSO	Acyclic Unique Sink Orientation
OR	Order-Regular
SOR	Strongly Order-Regular *A restriction of the OR condition*
PSOR	Partially Strongly Order-Regular *A restriction of the OR condition and a relaxation of the PSOR condition*
OF	Odd-Free *relates to binary matrices whose 0-entries only appear at odd row indices*
EF	Even-Free *relates to binary matrices whose 0-entries only appear at even row indices*
OEF	Odd-and-Even-Free *relates to binary matrices whose columns are either OF or EF*
OOR	Odd-Order-Regular *relates to binary matrices that satisfy all the OR constraints* (i, j) *where* i *is an odd number*
EOR	Even-Order-Regular *relates to binary matrices that satisfy all the OR constraints* (i, j) *where* i *is an even number*
OEF-OR	Odd-and-Even-Free and Order-Regular *relates to binary matrices that are both OEF and OR*