Worst-case Analysis of Strategy Iteration and the Simplex Method

Thomas Dueholm Hansen

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Worst-case Analysis of Strategy Iteration and the Simplex Method

A Dissertation
Presented to the Faculty of Science
at Aarhus University
in Partial Fulfillment of the Requirements for the
PhD Degree

by
Thomas Dueholm Hansen
July 2012

Abstract

In this dissertation we study *strategy iteration* (also known as *policy iteration*) algorithms for solving *Markov decision processes* (MDPs) and *two-player turn-based stochastic games* (2TBSGs). MDPs provide a mathematical model for sequential decision making under uncertainty. They are widely used to model stochastic optimization problems in various areas ranging from operations research, machine learning, artificial intelligence, economics and game theory. The class of two-player turn-based stochastic games is a natural generalization of Markov decision processes that is obtained by introducing an adversary. 2TBSGs form an intriguing class of games whose status in many ways resembles that of *linear programming* 40 years ago. They can be solved efficiently with strategy iteration algorithms, resembling the *simplex method* for linear programming, but no polynomial time algorithm is known. Linear programming is an exceedingly important problem with numerous applications. The simplex method was introduced by Dantzig in 1947, and has since then been studied extensively. It can be shown that MDPs can be formulated as linear programs, thus, giving rise to the connection.

Strategy iteration and simplex type algorithms are local search algorithms that repeatedly improve a current candidate solution. We say that the strategy iteration algorithm repeatedly performs *improving switches*, and that the simplex method repeatedly performs *improving pivots*. The strategy iteration algorithm and the simplex method are, in fact, paradigms for solving 2TBSGs, MDPs, and linear programs. To obtain concrete algorithms we must specify an *improvement rule* (*pivoting rule*). In this dissertation we mainly focus on three improvement rules:

- Howard's improvement rule: Update the current solution by simultaneously performing all improving switches.
- RandomEdge: Update the current solution by performing a single uniformly random improving switch.

- RandomFacet: A recursively defined, randomized improvement rule that was introduced independently by Kalai and by Matoušek, Sharir and Welzl. The improvement rule works by recursively solving certain sub-problems that have been chosen randomly.

One of the main results of the dissertation is to show that there exist families of MDPs for which using the RandomEdge and RandomFacet improvement rules lead to the expected number of improvement steps being almost exponential in the size of the problem (the bounds have subexponential form). Utilizing a tight connection between MDPs and linear programming, it is shown that the same bounds apply to the corresponding pivoting rules for the simplex method for solving linear programs. Prior to this result no super-polynomial lower bounds were known for randomized pivoting rules for the simplex method. On the other hand, essentially every known deterministic pivoting rule had been shown to be exponential. The results for RandomEdge and RandomFacet are, therefore, the first of their kind for randomized pivoting rules, which solves a more than forty year old open problem.

Another important result of the dissertation is to show that Howard's improvement rule for the strategy iteration algorithm (Howard's algorithm in short) solves 2TBSGs with a fixed *discount* in a number of iterations that is polynomial in the (combinatorial) size of the corresponding games. This proves, for the first time, that 2TBSGs with a fixed discount can be solved in *strongly* polynomial time. Howard's algorithm for MDPs dates back to 1960 and is the algorithm of choice in practise for solving MDPs and 2TBSGs. It is therefore appealing that the result is obtained for this particular classical algorithm.

It is known that Howard's algorithm in general, when the discount is not fixed, may be exponential. This result relies heavily on the element of uncertainty, however. When the MDPs are *deterministic*, the bound does not apply. It is shown in the dissertation that for deterministic MDPs the number of iterations performed by Howard's algorithm may be quadratic in the size of the problem. Prior to this result the best lower bounds were linear in the size of the problem.

Resumé

I denne afhandling studerer vi *strategi iterations algoritmen* til at løse *Markov beslutningsprocesser* og *to-spiller tur-baserede stokastiske spil*. Markov beslutningsprocesser er matematiske modeller for sekventiel beslutningstagning i situationer præget af usikkerhed. De bliver brugt til at modellere stokastiske optimerings problemer inden for områder som operations analyse, automatisk læring, kunstig intelligens, økonomi og spilteori. To-spiller tur-baserede stokastiske spil er en naturlig generalisering af Markov beslutningsprocesser hvor en modspiller bliver introduceret. To-spiller tur-baserede stokastiske spil er en interessant klasse af spil hvis status på mange måder ligner *lineær programmering* for 40 år siden. De kan løses effektivt med strategi iterations algoritmen, der minder om *simplex metoden*, men der kendes ikke nogen generel polynomiel tids algoritme. Lineær programmering er et vigtigt problem der har mange anvendelses muligheder. Simplex metoden blev introduceret af Dantzig i 1947, og den er siden da blevet studeret ekstensivt. Det er muligt at vise at Markov beslutningsprocesser kan formuleres som lineære programmer, hvilket er årsagen til forbindelsen.

Strategi iterations algoritmen og simplex metoden er lokal søgnings algoritmer der gentagne gange forbedrer en nuværende kandidat løsning. Vi siger at strategi iterations algoritmen og simplex metoden gentagne gange laver *forbedrende ændringer*. Strategi iterations algoritmen og simplex metoden er mere præcist paradigmer til at løse to-spiller tur-baserede stokastiske spil, Markov beslutningsprocesser og lineær programmering. For at opnå en konkret algoritme skal der specifiseres en *forbedrings regel*. I denne afhandling fokuserer vi primært på tre forbedrings regler:

- Howard's forbedrings regel: Opdater den nuværende løsning ved at foretage alle forbedrende ændringer parallelt.

- RandomEdge: Opdater den nuværende løsning ved at foretage en tilfældig forbedrende ændring.

- RandomFacet: En rekursivt defineret randomiseret forbedrings regel der blev introduceret uafhængigt af Kalai og af Matoušek, Sharir and Welzl. Forbedrings reglen fungerer ved rekursivt at løse passende tilfældigt valgte underproblemer.

Et af hovedresultaterne i afhandlingen er at vise at der eksisterer Markov beslutningsprocesser for hvilke RandomEdge og RandomFacet beslutnings reglerne forventes at foretage næsten eksponentielt mange forbedrende ændringer i størrelsen af problemet (grænserne har *subeksponentiel* form). Ved at udnytte en tæt forbindelse mellem Markov beslutningsprocesser og lineær programmering viser vi at den samme grænse gælder for de tilsvarende forbedrings regler for simplex metoden ved løsning af lineære programmer. Før dette resultat var der ingen kendte nedre grænser for randomiserede forbedrings regler for simplex metoden der var mere end polynomielle i størrelsen af problemet. På den anden side var det kendt at stort set alle deterministiske forbedrings regler kan have eksponentiel opførsel. Resultaterne for RandomEdge og RandomFacet er derfor de første af deres art, hvilket løser et 40 år gammelt problem.

Et andet vigtigt resultat i afhandlingen er at vise at Howard's forbedrings regel for strategi iterations algoritmen (herefter henvist til som Howard's algoritme) løser to-spiller tur-baserede stokastiske spil med en fast *inflationsrate* i et polynomielt antal iterationer i den kombinatoriske størrelse af de tilsvarende spil. Dette viser for første gang at to-spiller tur-baserede stokastiske spil med fast inflationsrate kan løses i *stærkt* polynomiel tid. Howard's algoritme for Markov beslutningsprocesser og to-spiller tur-baserede stokastiske spil er den algoritme der bruges i praksis til at løse disse problemer. Det er derfor appellerende at resultatet opnås for denne klassiske algoritme.

Det er kendt at Howard's algoritme for generelle Markov beslutningsprocesser uden en fast inflationsrate har eksponentiel kompleksitet. Dette resultat afhænger dog af elementet af usikkerhed. For deterministiske Markov beslutningsprocesser gælder grænsen ikke. Vi viser i afhandlingen at Howard's algoritme kan bruge et kvadratisk antal iterationer, i størrelsen af problemet, for deterministiske Markov beslutningsprocesser. Før dette resultat var den bedste nedre grænse lineær i størrelsen af problemet.

Preface and acknowledgements

Let me start by thanking all the people I have worked with over the course of the last four years. I particularly want to thank Daniel Andersson, Oliver Friedmann, Vladimir Gurvich, Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, Peter Bro Miltersen, Troels Bjerre Sørensen, Orestis A. Telelis, and Uri Zwick. It has been a tremendous help to work with talented and experienced researchers, and to learn from their knowledge and practises. A special thanks goes out to Peter Bro Miltersen, my advisor, for helping me get off the ground as a researcher. You have always been eager to take your time to talk with me and your other students. I can truly say that any shortcomings have been on my part. I also want to give a special thanks to Uri Zwick. Thank you for taking good care of me during my time in Israel, and thank you for being patient with teaching me to be more patient and thorough. I look forward to working with you during the next year in Tel Aviv.

For my dissertation I have selected the main results obtained during the last four years. I have prioritized getting a coherent presentation. Most of the results presented in the dissertation have been published in the following papers:

[41] O. Friedmann, T. D. Hansen, and U. Zwick. A subexponential lower bound for the random facet algorithm for parity games. In *Proc. of 22nd SODA*, pages 202–216, 2011.

[42] O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *Proc. of 43rd STOC*, pages 283–292, 2011.

[59] T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is

strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. In *Proc. of 2nd ICS*, pages 253–263, 2011.

[60] T. D. Hansen and U. Zwick. Lower bounds for Howard's algorithm for finding minimum mean-cost cycles. In *Proc. of 21st ISAAC*, pages 415–426, 2010.

I start out by giving introductions to linear programming, Markov decision processes, and turn-based stochastic games in chapters 1, 2, and 3, respectively. The results of [59] are presented in chapters 2 and 3, and mainly in sections 2.2.4 and 3.4.2. The presentation in chapters 1, 2, and 3 is largely inspired by [59], however. The results of [42], [41], and [60] are presented in chapters 4, 5, and 6, respectively. The introductions and preliminaries of these papers have, to some extend, been placed in chapters 1, 2, and 3.

Contents

Notation

We use bold uppercase letters to denote matrices. For instance, $\mathbf{A}$, $\mathbf{I}$, $\mathbf{J}$, $\mathbf{M}$, and $\mathbf{P}$ are matrices. $\mathbf{I}$ is the identity matrix.

We use bold lowercase letters to denote vectors. For instance, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{g}$, $\mathbf{h}$, $\mathbf{q}$, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are vectors.

We use $\mathbf{0} = (0, 0, \dots, 0)^T$ to denote an all zero vector of suitable length.

We use $\mathbf{e} = (1, 1, \dots, 1)^T$ to denote an all one vector of suitable length.

We use $\mathbf{e}_i$ to denote the i'th unit vector of suitable length. I.e., the i'th coordinate is 1 and all other coordinates are 0.

We let $[n] = \{1, 2, \dots, n\}$.

We let $B(i)$ be the i-th element in $B \subseteq [m]$. For example $B(1)$ is the smallest element in B.

We let $\mathbf{A}_i \in \mathbb{R}^{1\times n}$ be the i-th row of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. Similarly, $\mathbf{c}_i$ is the i-th element of a vector $\mathbf{c} \in \mathbb{R}^m$.

For every $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $B \subseteq [m]$, we let $\mathbf{A}_B$ be the matrix obtained from $\mathbf{A}$ by picking the rows with indices in B. I.e., $(\mathbf{A}_B)_i = \mathbf{A}_{B(i)}$. Similarly, $\mathbf{c}_B$ is the vector consisting of the components of $\mathbf{c} \in \mathbb{R}^m$ with indices in B.

$\mathbb{N} = \{1, 2, 3, \dots\}$ is the set of natural numbers, and $\mathbb{R}$ is the set of real numbers.

$(\mathbf{u}^k)_{k=0}^N = (\mathbf{u}^0, \mathbf{u}^1, \dots, \mathbf{u}^N)$ is a finite sequence of vectors.

$(\mathbf{u}^k)_{k=0}^\infty = (\mathbf{u}^0, \mathbf{u}^1, \dots)$ is an infinite sequence of vectors.

Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ be two vectors. We write $\mathbf{u} < \mathbf{v}$ when $\mathbf{u} \leq \mathbf{v}$ and $\mathbf{u} \neq \mathbf{v}$. Similarly, we write $\mathbf{u} > \mathbf{v}$ when $\mathbf{u} \geq \mathbf{v}$ and $\mathbf{u} \neq \mathbf{v}$.

Chapter 1

Linear programming

The need to solve optimization problems involving linear constraints and linear objectives, leading to the term "linear programming", arose during World War II in connection with planning of military operations. After the war such techniques, among others, were applied for industrial purposes, giving birth to the field of operations research. The *simplex method*, published by Dantzig in 1947 (see [24]), was the first practical algorithm for solving linear programming problems. The simplex method is a general paradigm for solving linear programs, and in order to get a concrete algorithm a specific *pivoting rule* must be employed. The simplex method was named as one of the top 10 most influential algorithms of the 20th century in a special issue of the journal *Computing in Science & Engineering* (see [20]).

In the 1970's much effort was put into characterizing efficient computation theoretically. Informally, a problem was said to be efficiently computable if the time required to solve the problem was proportional to the time required to describe the problem. Formally, an algorithm is said to run in (weakly) polynomial time if the number of steps of a corresponding Turing machine is bounded by a polynomial in the number of bits of the input. Alternatively, in the arithmetic model of computation, an algorithm is *strongly* polynomial if the number of arithmetic operations performed is polynomial in the number of numbers in the input. I.e., polynomial time algorithms may depend on the bit-complexity, whereas strongly polynomial time algorithms may not.

In 1972 Klee and Minty [78] showed that Dantzig's original pivoting rule can lead to exponential behavior for carefully constructed examples. Following this work almost all known deterministic pivoting rules have been shown to be exponential (see Section 1.2.1). The complexity of randomized

pivoting rules remained open for many years. Only recently did Friedmann, Hansen, and Zwick [41, 42] manage to prove super-polynomial (subexponential) lower bounds for two of the most natural, and most studied, randomized pivoting rules suggested to date.

In 1979 Khachiyan [76] showed that the *ellipsoid method* solves linear programs in polynomial time. In 1984 Karmarkar [73] introduced the *interior point method*, an algorithm with polynomial complexity which is also efficient in practise. Today commercial software for solving linear programs, such as CPLEX, is based on the simplex and interior point methods. The ellipsoid and interior point methods are not strongly polynomial, however. The question of whether linear programming can be solved in strongly polynomial time remains the, arguable, most prominent open theoretical problem in the area.

In this chapter we give a relatively brief introduction to linear programming and the simplex method. The presentation is complete in the sense that everything that is relevant for later chapters is presented in some detail. We will, however, assume that the reader has some familiarity with linear programming. For textbooks on linear programming we refer to, e.g., Bertsimas and Tsitsiklis [10], Matoušek and Gärtner [87], Chvátal [19], and Schrijver [108]. The presentation of linear programming given in this chapter has been inspired by Bertsimas and Tsitsiklis [10] and Matoušek and Gärtner [87]. The main focus of the chapter is on the simplex method and pivoting rules.

The chapter is organized as follows. In Section 1.1 we first give the basic definitions needed to study linear programming and the simplex method. In Section 1.2 we present the simplex method, with a presentation of pivoting rules given in sections 1.2.1 and 1.2.2. In Section 1.3 we give an introduction to duality of linear programming, and finally in Section 1.4 we present some related results that did not fit in the other sections.

1.1 Definitions

In this section we list a number of standard definitions and lemmas related to linear programming. The purpose of the section is to introduce notation and terminology for later use. The section is meant to be brief, and we refer to textbooks on linear programming (see, e.g., [10, 87, 19, 108]) for a more extensive introduction.

We start by stating a few definitions related to the geometry of linear programming.

Definition 1.1.1 (Hyperplane and halfspace) *A* hyperplane *is a set* $\{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{a}^T\mathbf{x} = b\}$*, where* $\mathbf{a} \in \mathbb{R}^m$ *is a nonzero vector and* $b \in \mathbb{R}$ *is a constant. Similarly, a (closed)* halfspace *is a set* $\{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{a}^T\mathbf{x} \leq b\}$*, where* $\mathbf{a} \in \mathbb{R}^m$*,* $\mathbf{a} \neq \mathbf{0}$*, and* $b \in \mathbb{R}$*.*

Definition 1.1.2 (Convex set) *A set* $S \subseteq \mathbb{R}^m$ *is called* convex *if for all* $\mathbf{x}, \mathbf{y} \in S$ *and* $\lambda \in [0, 1]$*, we have* $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in S$*.*

Note that hyperplanes and halfspaces are convex.

Definition 1.1.3 (Polyhedron) *A* polyhedron *(or* polytope*)* $P \subseteq \mathbb{R}^m$ *is the intersection of finitely many halfspaces.*

Lemma 1.1.4 *Let* $S \subseteq \mathbb{R}^m$ *be the intersection of convex sets, then* S *is convex. In particular, every polyhedron* $P \subseteq \mathbb{R}^m$ *is convex.*

Note that a hyperplane can be viewed as the intersection of two halfspaces.

Definition 1.1.5 (Bounded polyhedron) *Let* $P \subseteq \mathbb{R}^m$ *be a polyhedron. If there exists a constant* $M \in \mathbb{R}$ *such that for all* $\mathbf{x} \in P$ *and* $i \in [m]$*,* $|\mathbf{x}_i| \leq M$*, then we say that* P *is* bounded*. Otherwise we say that* P *is* unbounded*.*

Definition 1.1.6 (Vertex) *Let* $P \subseteq \mathbb{R}^m$ *be a polyhedron. We say that* $\mathbf{x} \in \mathbb{R}^m$ *is a* vertex *of* P *if and only if there exists a vector* $\mathbf{c} \in \mathbb{R}^m$ *such that* $\mathbf{c}^T\mathbf{x} > \mathbf{c}^T\mathbf{x}'$ *for all* $\mathbf{x}' \in P \setminus \{\mathbf{x}\}$*.*

Definition 1.1.7 (Tight constraints) *We say that an inequality constraint* $\mathbf{a}^T\mathbf{x} \leq b$*, where* $\mathbf{a} \in \mathbb{R}^m$ *and* $b \in \mathbb{R}$*, is* tight *(or* active *or* binding*) for* $\mathbf{x} \in \mathbb{R}^m$ *if* $\mathbf{a}^T\mathbf{x} = b$*.*

The *dimension* of a polyhedron P is the smallest integer d such that P is contained in an affine subspace of dimension d. Suppose $H = \{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{a}^T\mathbf{x} \leq b\}$ is a halfspace used for the definition of a d-dimensional polyhedron P. Let $P' = P \cap \{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{a}^T\mathbf{x} = b\}$ be the polyhedron obtained by requiring the inequality constraint $\mathbf{a}^T\mathbf{x} \leq b$ to be tight. If P' has dimension $d-1$ we say that P' is a *facet* of P. A 1-dimensional polyhedron obtained by requiring inequality constraints to be tight is call an *edge*. A 0-dimensional polyhedron obtained in this way is a vertex (see Lemma 1.1.16 below).

We next define linear programs. Our main focus is going to be on the simplex method. For succinctness, we only present linear programs in standard form (and later the corresponding dual linear programs). It is well known that every linear program has an equivalent standard form. For a more general presentation of linear programming we again refer to textbooks on the topic (see, e.g., [10, 87, 19, 108]).

For the remainder of this section we let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{c} \in \mathbb{R}^m$ be given.

Definition 1.1.8 (Linear program, standard form) *A* linear program *in* standard form *(or* equational form*) is the following optimization problem over* $\mathbf{x} \in \mathbb{R}^m$*:*

$$\begin{array}{lllll} & \max & \mathbf{c}^T\mathbf{x} & & \\ (P) & s.t. & \mathbf{A}^T\mathbf{x} & = & \mathbf{b} \\ & & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

We refer to $\mathbf{c}^T\mathbf{x}$ *as the* objective function*, to* $\mathbf{A}^T\mathbf{x} = \mathbf{b}$ *as the* equality constraints*, and to* $\mathbf{x} \geq \mathbf{0}$ *as the* non-negativity constraints*. In particular, we refer to the inequality* $\mathbf{x}_j \geq 0$ *as the* j*-th non-negativity constraint.*

Note that we let $\mathbf{A}$ be transposed, and that the number of equations and variables are n and m, respectively. We do this in order to be consistent with the notation used in Chapter 2 for Markov decision processes. It should be pointed out, however, that this is not the traditional way of presenting linear programs in standard form.

Definition 1.1.9 (Feasible solution) *Let* $P = \{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{A}^T\mathbf{x} = \mathbf{b} \wedge \mathbf{x} \geq \mathbf{0}\}$*. We say that* $\mathbf{x} \in \mathbb{R}^m$ *is a* feasible solution *for the linear program* (P) *if and only* $\mathbf{x} \in P$*. We refer to* P *as the set of feasible solutions. We say that* $\mathbf{c}^T\mathbf{x}$ *is the* value *of* $\mathbf{x}$*.*

Note that the set of feasible solutions P for the linear program (P) is a polyhedron. I.e., P is the intersection of m halfspaces and n hyperplanes.

Definition 1.1.10 (Optimal solution) *A feasible solution* $\mathbf{x} \in P \subseteq \mathbb{R}^m$ *is said to be* optimal *if and only if* $\mathbf{c}^T\mathbf{x} \geq \mathbf{c}^T\mathbf{x}'$ *for all* $\mathbf{x}' \in P$*. If* $\mathbf{x}$ *is an optimal solution then we say that the* value *of* (P) *is* $\mathbf{c}^T\mathbf{x}$*.*

Definition 1.1.11 (Infeasible, unbounded) *The linear program* (P) *is said to be* infeasible *if and only if there are no feasible solutions;* $P = \emptyset$*.* (P) *is said to be* unbounded *if and only if for every* $M \in \mathbb{R}$ *there exists a feasible solution* $\mathbf{x} \in P$ *with* $\mathbf{c}^T\mathbf{x} \geq M$*.*

Solving a linear program means finding an optimal solution, or correctly report that the linear program is infeasible, or correctly report that the linear program is unbounded. The following lemma shows that we will always be in one of these three situations. The lemma follows from the fact that a polyhedron is a closed set.

Lemma 1.1.12 *Either (P) has an optimal solution, or (P) is infeasible, or (P) is unbounded.*

For every $j \in [m]$, we let $\mathbf{A}_j \in \mathbb{R}^{1\times n}$ be the j-th row of $\mathbf{A}$. Let $B \subseteq [m]$. We will use the convention that $B(i)$ is the i-th element in B. For example $B(1)$ is the smallest element in B. We let $\mathbf{A}_B$ be the matrix obtained from $\mathbf{A} \in \mathbb{R}^{m\times n}$ by picking the rows with indices in B. I.e., $(\mathbf{A}_B)_i = \mathbf{A}_{B(i)}$. We use the same notation for vectors. Note that we are able to use this notation because we let $\mathbf{A}$ be transposed in the definition of (P). I.e., $(\mathbf{A}_B)^T$ picks the columns corresponding to B for $\mathbf{A}^T$.

We assume that the columns of $\mathbf{A}$ are linearly independent. If this is not the case then we can identify a maximal set of linearly independent columns of $\mathbf{A}$ and drop the remaining columns (equality constraints) to get a new equivalent linear program. This can, for instance, be done using Gaussian elimination.

Definition 1.1.13 (Basis) *We say that $B \subseteq [m]$, with $|B| = n$, is a* basis *if and only if the rows of $\mathbf{A}_B$ are linearly independent. I.e., $\mathbf{A}_B$ is nonsingular.*

One can, equivalently, view a basis as containing variables of the linear program, non-negativity constraints, or column vectors corresponding to columns of $\mathbf{A}^T$ (rows of $\mathbf{A}$).

Definition 1.1.14 (Basic solution) *We say that $\mathbf{x} \in \mathbb{R}^m$ is a* basic solution *if and only if there exists a basis $B \subseteq [m]$ such that all the non-negativity constraints of B are tight for $\mathbf{x}$. Furthermore, we say that $\mathbf{x}$ is a* basic feasible solution *if and only if $\mathbf{x}$ is a basic solution and $\mathbf{x} \in P$.*

For every basis $B \subseteq [m]$ define $\mathbf{x}^B = (\mathbf{A}_B^{-1})^T\mathbf{b} \in \mathbb{R}^n$. Furthermore, define $\bar{\mathbf{x}}^B \in \mathbb{R}^m$ to be the extension of $\mathbf{x}^B$ to m coordinates by letting additional coordinates be zero. I.e., for all $j \in [m]$:

$$(\bar{\mathbf{x}}^B)_j = \begin{cases} (\mathbf{x}^B)_i & \text{if } j = B(i) \text{ for some } i \in [n] \\ 0 & \text{otherwise} \end{cases}$$

Then Definition 1.1.14 can be equivalently stated as saying that $\mathbf{x} \in \mathbb{R}^m$ is a basic solution if and only if there exists a basis $B \subseteq [m]$ such that $\mathbf{x} = \bar{\mathbf{x}}^B$.

Let B be a basis. The variables $\mathbf{x}_j$ with index $j \in B$ are called *basic variables* for the basic solution $\bar{\mathbf{x}}^B$, and the variables $\mathbf{x}_j$ with index $j \notin B$ are called *non-basic variables* for $\bar{\mathbf{x}}^B$. We will sometimes say that a basis contains basic variables. We say that B is a *feasible basis* if and only if $\bar{\mathbf{x}}^B \geq \mathbf{0}$ is a basic feasible solution. If $\bar{\mathbf{x}}^B$ is an optimal solution we say that B is an *optimal basis*.

Note that more than one basis may correspond to the same basic solution. This leads us to the following definition.

Definition 1.1.15 (Degeneracy) *A basic solution $\mathbf{x} \in \mathbb{R}^m$ is said to be* degenerate *if and only if more than n non-negativity constraints are tight for x. If no basic feasible solution is degenerate, and the polyhedron P is bounded, then P is said to be* simple.

Note that a basic solution is degenerate exactly when it is defined by more than one basis.

Lemma 1.1.16 *Let $\mathbf{x} \in \mathbb{R}^m$. Then $\mathbf{x}$ is a vertex of the polyhedron P if and only if $\mathbf{x}$ is a basic feasible solution for the linear program (P).*

Let us note that the same polyhedron P can be represented in many ways, and whether a vertex $\mathbf{x}$ is degenerate depends on the representation of P.

1.2 The simplex method

The *simplex method* (or *simplex algorithm*), developed by Dantzig in 1947 (see [24]), and its many variants, are still among the most widely used algorithms for solving linear programs. The simplex method is a local search algorithm that works by repeatedly improving a current solution. Geometrically, the solutions considered by the simplex method are the vertices of the polyhedron P for a linear program (P). For any non-optimal vertex $\mathbf{x}$ the simplex method moves to a new vertex $\mathbf{x}'$, connected to $\mathbf{x}$ by an edge in P, such that $\mathbf{c}^T\mathbf{x}' > \mathbf{c}^T\mathbf{x}$ (assuming that no problems arise due to degeneracy).

In this section we formally define the simplex method. We start by presenting the definition of reduced costs.

Definition 1.2.1 (Reduced costs) *The vector of* reduced costs *for a basis $B \subseteq [m]$ is defined as $\bar{\mathbf{c}}^B = \mathbf{c} - \mathbf{A}\mathbf{A}_B^{-1}\mathbf{c}_B \subseteq \mathbb{R}^m$.*

We refer to $(\bar{\mathbf{c}}^B)_j$ as the reduced cost of the variable $\mathbf{x}_j$ for B.

Lemma 1.2.2 *Let $B \subseteq [m]$ be any basis. Then for every $\mathbf{x} \in \mathbb{R}^m$ satisfying $\mathbf{A}^T\mathbf{x} = \mathbf{b}$ we have:*

$$\mathbf{c}^T\mathbf{x} \;=\; \mathbf{c}_B^T\mathbf{x}^B + (\bar{\mathbf{c}}^B)^T\mathbf{x}.$$

Proof: From the definition of $\mathbf{x}^B$, the assumption that $\mathbf{A}^T\mathbf{x} = \mathbf{b}$, and from the definition of $\bar{\mathbf{c}}^B$ we get:

$$\mathbf{c}^T\mathbf{x} - \mathbf{c}_B^T\mathbf{x}^B = \mathbf{c}^T\mathbf{x} - \mathbf{c}_B^T(\mathbf{A}_B^{-1})^T\mathbf{b} = \mathbf{c}^T\mathbf{x} - \mathbf{c}_B^T(\mathbf{A}_B^{-1})^T\mathbf{A}^T\mathbf{x} = (\bar{\mathbf{c}}^B)^T\mathbf{x}.$$

□

Lemma 1.2.3 *For any basis B we have $(\bar{\mathbf{c}}^B)_B = \mathbf{0} \in \mathbb{R}^n$. I.e., the reduced cost of every variable $\mathbf{x}_j$ with $j \in B$ is zero.*

Proof: From Definition 1.2.1 we get:

$$(\bar{\mathbf{c}}^B)_B \;=\; \mathbf{c}_B - (\mathbf{A}\mathbf{A}_B^{-1}\mathbf{c}_B)_B \;=\; \mathbf{c}_B - \mathbf{A}_B\mathbf{A}_B^{-1}\mathbf{c}_B \;=\; \mathbf{0}\,.$$

□

We will now derive the simplex method. The presentation given below resembles that of Matoušek and Gärtner [87]. Let $B \subseteq [m]$ be any basis, and let $\bar{B} = [m] \setminus B$. Using Lemma 1.2.3, observe that:

$$\begin{aligned}(\bar{\mathbf{c}}^B)^T\mathbf{x} \;&=\; ((\bar{\mathbf{c}}^B)_{\bar{B}})^T\mathbf{x}_{\bar{B}} \qquad\qquad \text{and}\\ (\mathbf{A}_B^{-1})^T\mathbf{A}^T\mathbf{x} \;&=\; \mathbf{x}_B + (\mathbf{A}_B^{-1})^T(\mathbf{A}_{\bar{B}})^T\mathbf{x}_{\bar{B}}\,.\end{aligned}$$

It then follows from Lemma 1.2.2 that the linear program (P) can be equivalently stated as:

$$\begin{array}{lll} & \max & \mathbf{c}_B^T\mathbf{x}^B + ((\bar{\mathbf{c}}^B)_{\bar{B}})^T\mathbf{x}_{\bar{B}} \\ (P^B) & s.t. & \mathbf{x}_B + (\mathbf{A}_B^{-1})^T(\mathbf{A}_{\bar{B}})^T\mathbf{x}_{\bar{B}} \;=\; (\mathbf{A}_B^{-1})^T\mathbf{b} \\ & & \mathbf{x}_B, \mathbf{x}_{\bar{B}} \;\geq\; \mathbf{0} \end{array}$$

Note that we can immediately see that $\bar{\mathbf{x}}^B$ is a basic solution for (P^B). I.e., by setting $\mathbf{x}_{\bar{B}} = \mathbf{0}$ we get $\mathbf{x}_B = (\mathbf{A}_B^{-1})^T\mathbf{b} = \mathbf{x}^B$. Also, $\bar{\mathbf{x}}^B$ is a basic feasible solution if and only if $(\mathbf{A}_B^{-1})^T\mathbf{b} \geq \mathbf{0}$. Note, furthermore, that $\mathbf{c}_B^T\mathbf{x}^B$ is a constant that does not affect the objective function. Writing the constant in the objective function ensures that, by Lemma 1.2.2, any solution $\mathbf{x}$ has the same value for (P) and (P^B).

Let $\bar{\mathbf{x}}^B$ be a basic feasible solution such that $(\mathbf{A}_B^{-1})^T\mathbf{b} \geq \mathbf{0}$. Observe that if $(\bar{\mathbf{c}}^B)_j > 0$ then any feasible solution of (P^B) with $\mathbf{x}_j > 0$ has larger value than $\mathbf{c}_B^T\mathbf{x}^B$. In particular, $(\bar{\mathbf{c}}^B)_j$ is the increase in the value if $\mathbf{x}_j$ is increased by one. Also, we may increase $\mathbf{x}_j$ as long as no other variable becomes negative. If we require that $\mathbf{x}_{\bar{B}\setminus\{j\}} = \mathbf{0}$ then $\mathbf{x}_B$ is expressed as a function of $\mathbf{x}_j$ as follows. Define $\mathbf{v}^B := (\mathbf{A}_B^{-1})^T\mathbf{b} \geq \mathbf{0}$ and $Q^B := ((\mathbf{A}_B^{-1})^T(\mathbf{A}_{\bar{B}})^T)$. For every $i \in B$ we then have:

$$\mathbf{x}_i \;=\; \mathbf{v}_i^B - Q_{i,j}^B\mathbf{x}_j$$

Note that if $Q_{i,j}^B \leq 0$ for all $i \in [n]$ then we can keep increasing $\mathbf{x}_j$ and the linear program (P) is unbounded. Otherwise we let:

$$\mathbf{x}_j \;=\; \min_{i\in[n]:\; Q_{i,j}^B > 0} \frac{\mathbf{v}_i^B}{Q_{i,j}^B}$$

In this case at least one other variable $\mathbf{x}_i$ becomes zero, and we get a new basis $B' = B \cup \{j\} \setminus \{i\}$ by exchanging i and j. If $\mathbf{x}_j > 0$ we have $\mathbf{c}^T\bar{\mathbf{x}}^{B'} > \mathbf{c}^T\bar{\mathbf{x}}^B$. We can only have $\mathbf{c}^T\bar{\mathbf{x}}^{B'} = \mathbf{c}^T\bar{\mathbf{x}}^B$ if $\bar{\mathbf{x}}^{B'} = \bar{\mathbf{x}}^B$. I.e., the basic feasible solution is degenerate.

The simplex method repeatedly performs the process described above. It is also described in Figure 1.1. It takes as input a description of the linear program and basis B for which the corresponding basic solution $\bar{\mathbf{x}}^B$ is feasible. If the value of the solution increases with every step, then we must reach a basis B^* with $\bar{\mathbf{c}}^{B^*} \leq \mathbf{0}$. I.e., there are at most $\binom{m}{n}$ bases, and we do not return to a basis once it has been used. Since all variables are non-negative, it follows from Lemma 1.2.2 that $\bar{\mathbf{x}}^{B^*}$ is an optimal solution. Q^B, $\mathbf{v}^B$, $(\bar{\mathbf{c}}^B)_{\bar{B}}$, and $\mathbf{c}_B^T\mathbf{x}^B$ can be maintained in a *tableau* for B, such that the new tableau for B' can be computed in time $O(mn)$. Using such an implementation of the simplex method is referred to as using the *tableau method*.

When we exchange one index in the basis with another we say that we perform a *pivot*. Note that there may be several choices for which index should enter the basis and which index should leave the basis. To get a concrete algorithm we must specify these choices. Such a specification is called a *pivoting rule*. Hence, a pivoting rule specifies:

(*i*) Which $j \in \{j \in [m] \mid (\bar{\mathbf{c}}^B)_j > 0\}$ is picked for entering the basis.

(*ii*) Which $i \in \operatorname{argmin}_{i\in[n]:\; Q_{i,j}^B > 0} \frac{\mathbf{v}_i^B}{Q_{i,j}^B}$ is picked picked for leaving the basis.

Function `SimplexMethod`$(\mathbf{A}, \mathbf{b}, \mathbf{c}, B)$

while $\exists j \in [m] : \ (\bar{\mathbf{c}}^B)_j > 0$ **do**

pick $j \in \{j \in [m] \mid (\bar{\mathbf{c}}^B)_j > 0\}$;

if $\forall i \in [n] : \ Q^B_{i,j} \leq 0$ **then**

return UNBOUNDED;

else

pick $i \in \underset{i \in [n]:\ Q^B_{i,j} > 0}{\operatorname{argmin}} \frac{\mathbf{v}^B_i}{Q^B_{i,j}}$;

$B \leftarrow B \cup \{j\} \setminus \{i\}$;

return $\bar{\mathbf{x}}^B$;

Figure 1.1: The simplex method.

We present some of the most well known pivoting rules in sections 1.2.1 and 1.2.2 below. When the index i that leaves the basis is uniquely defined, or when we are not concerned with which index leaves the basis, we use the following notation. For a feasible basis B and an index j, we let $B[j] = B \cup \{j\} \setminus \{i\}$ be the feasible basis obtained from B by performing a pivot with j. Note that i is uniquely defined when $\mathbf{x}^{B[j]}$ is not degenerate.

Definition 1.2.4 (Improving pivot) *Let $B \subseteq [m]$ be a feasible basis. We say that $j \in [m]$ is an* improving pivot *with respect to B if and only if $(\bar{\mathbf{c}}^B)_j > 0$.*

Using the definition of improving pivots, the simplex method can be stated as follows. Starting with a feasible basis B we repeatedly perform improving pivots until reaching an optimal basis B^* for which there are no improving pivots.

Let us note that there are still two unresolved issues for the definition of the simplex method. The first is that in case of degeneracy it is possible for the simplex method to cycle. This issue can be overcome, however, by using an appropriate pivoting rule. One such pivoting rule is the LEXICOGRAPHIC RULE by Dantzig, Orden, and Wolfe [25]. The LEXICOGRAPHIC RULE specifies which index should leave the basis in case of ties. It can then be combined with other pivoting rules which specify which index should enter the basis. The LEXICOGRAPHIC RULE can be viewed as implementing an infinitesimal perturbation of the linear program that elim-

inates degeneracy. For a description of the LEXICOGRAPHIC RULE and a proof that it eliminates cycling see, e.g., [19].

The second issue is that the simplex method needs an initial basic feasible solution. Note that if the linear program is infeasible, then no basic feasible solution exists. By solving the auxiliary linear program (P_0) shown below, we can either obtain an initial basic feasible solution for (P) or report that (P) is infeasible. Let $\mathbf{x}' = (\mathbf{x}_{m+1}, \ldots, \mathbf{x}_{m+n})^T$ be a vector of n additional variables, and let $\mathbf{e} = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$ be an all one vector. Furthermore, let $\mathcal{I}$ be a diagonal matrix such that for all $i \in [n]$, $\mathcal{I}_{i,i} = 1$ if $\mathbf{b}_i \geq 0$ and $\mathcal{I}_{i,i} = -1$ otherwise. Then (P_0) is defined as follows:

$$
\begin{array}{lll}
 & \max & -\mathbf{e}^T\mathbf{x}' \\
(P_0) & s.t. & \mathbf{A}^T\mathbf{x} + \mathcal{I}\mathbf{x}' = \mathbf{b} \\
 & & \mathbf{x}, \mathbf{x}' \geq \mathbf{0}
\end{array}
$$

Note that $B_0 = \{m+1, \ldots, m+n\}$ is a basis for (P_0) with basic feasible solution $(\mathbf{x}, \mathbf{x}') \geq \mathbf{0}$ where $\mathbf{x} = \mathbf{0}$ and $\mathbf{x}' = \mathcal{I}\mathbf{b} \geq \mathbf{0}$. Furthermore, the optimal value of (P_0) is at most 0, and it follows from Lemma 1.1.12 that (P_0) has an optimal solution. If (P) has a feasible solution $\mathbf{x} \in \mathbb{R}^m$ then $(\mathbf{x}, \mathbf{x}')$ where $\mathbf{x}' = \mathbf{0}$ is a feasible solution for (P_0). Hence, if (P_0) has value strictly smaller than 0, then (P) is infeasible. Let $(\mathbf{x}, \mathbf{x}')$ be an optimal (basic feasible) solution for (P_0), and assume that $\mathbf{x}' = \mathbf{0}$. We then obtain a basic feasible solution for (P) with basis B as follows. Let $B' = \{j \in [m] \mid \mathbf{x}_i > 0\}$. We have $|B'| \leq n$, and if $(\mathbf{x}, \mathbf{x}')$ is not degenerate then $|B'| = n$. Furthermore, the rows of $\mathbf{A}_{B'}$ are linearly independent. Let $B'' \subseteq [m]$ with $|B''| = n - |B'|$ be a set of indices such that the rows of $\mathbf{A}_{B' \cup B''}$ are linearly independent. Then we let $B = B' \cup B''$. If no such set B'' exists then there are less than n linearly independent rows for $\mathbf{A}$, and (P) is feasible but has no basic feasible solutions. Such linear programs are easy to solve.

The *two-phase simplex method* is the algorithm that first solves the auxiliary problem (P_0) using the simplex method, and then solves the original problem (P) using the simplex method.

1.2.1 Deterministic pivoting rules

As described in Section 1.2, one of the most important characteristics of a simplex algorithm is the pivoting rule it employs. In this section we present a number of important deterministic pivoting rules, and in the next section we present a number of important randomized pivoting rules. For all the pivoting rules presented in this section, except for the LEASTENTERED pivoting rule, it has been known for a long time that there exists linear

programs for which they require exponential time. Amenta and Ziegler [3] presented a unified view of these lower bounds. Prior to the work of Friedmann, Hansen, and Zwick [41, 42] no super-polynomial lower bounds were known for randomized pivoting rules. No polynomial pivoting rule is known for the simplex method.

Note that many of the pivoting rules below are not fully specified; they don't specify which index leaves the basis in case of ties. Such pivoting rules can be complemented with, for instance, the LEXICOGRAPHIC RULE.

LARGESTCOEFFICIENT

The LARGESTCOEFFICIENT pivoting rule is the original pivoting rule suggested by Dantzig [24]. In every iteration the LARGESTCOEFFICIENT pivoting rule picks the non-basic variable (index) with the largest positive reduced cost for entering the basis; i.e., the variable with the largest coefficient in the modified objective function.

Klee and Minty [78] gave an exponential lower bound for the number of improving pivots performed by the LARGESTCOEFFICIENT pivoting rule in the worst case. More precisely they presented n-dimensional hypercubes for which all 2^n vertices are visited. The lower bound of Klee and Minty was the first of its kind for (deterministic) pivoting rules for the simplex method.

LARGESTINCREASE

The LARGESTINCREASE pivoting rule performs the improving pivot that results in the largest increase for the value of the corresponding basic feasible solution. Note that for this pivoting rule it is not enough to consider the reduced costs to decide which pivot to perform. It was shown by Jeroslow [64] that the LARGESTINCREASE pivoting rule may perform exponentially many pivots.

STEEPESTEDGE

The STEEPESTEDGE pivoting rule performs the improving pivot for which the direction of the edge going from the current vertex to the next vertex is closest to the vector $\mathbf{c}$. I.e., it is the steepest edge. It was shown by Goldfarb and Sit [53] that the number of pivots required for the STEEPESTEDGE pivoting rule to reach an optimal solution may be exponential. The STEEPESTEDGE pivoting rule is known for being efficient in practise. See, e.g., Matoušek and Gärtner [87] for a description of the STEEPESTEDGE pivoting rule, and for a description of a pivoting rule that efficiently approximates STEEPESTEDGE.

Bland's rule

Bland's rule (also known as the LeastIndex or SmallestSubscribt pivoting rule) is a pivoting rule for the simplex method that was introduced by Bland [14]. In every iteration it picks the improving pivot with the smallest index for entering the basis, and it picks the basic variable with the smallest index among the candidates for leaving the basis. The ordering of the indices can be viewed as being given as input to the algorithm. Bland [14] showed that the Bland's rule simplex algorithm does not cycle. Avis and Chvátal [5] proved an exponential lower bound for the number of improving pivots performed by Bland's rule in the worst case.

ShadowVertex

The ShadowVertex pivoting rule works as follows. Starting from a basic feasible solution $\mathbf{x} \in \mathbb{R}^m$ it picks a vector $\mathbf{c}^0 \in \mathbb{R}^m$ for which $\mathbf{x}$ would be optimal in the linear program where $\mathbf{c}$ is replaced by $\mathbf{c}^0$ in the objective function. It follows from Definition 1.1.6 and Lemma 1.1.16 that such a vector exists. It then follows the path consisting of vertices that are optimal for some vector $\mathbf{c}' \in \{(1-\lambda)\mathbf{c}^0 + \lambda\mathbf{c} \mid \lambda \in [0,1]\}$. I.e., starting from zero, λ is gradually increased until the reduced cost of a variable becomes positive, in which case this improving pivot is performed. After the pivot is performed the procedure is repeated. When λ reaches 1 the optimal solution has been found. The vertices that are visited appear in the two-dimensional projection (shadow) of the polyhedron on the plane spanned by $\mathbf{c}^0$ and $\mathbf{c}$.

It was shown by Murty [95] that the ShadowVertex pivoting rule may require exponentially many improving pivots to solve a linear program. See also Amenta and Ziegler [3]. It was shown by Spielman and Teng [113] that the ShadowVertex pivoting rule has *polynomial smoothed complexity*. I.e., the running time becomes polynomial when the linear program is subjected to perturbations.

LeastEntered

The LeastEntered pivoting rule was introduced by Zadeh [123]. It is a pivoting rule that keeps track of how many times every variable has entered the basis. In each iteration it then performs the improving pivot with the variable that has entered the basis the least number of times before. Friedmann [40] recently proved a lower bound of subexponential form for the number of improving pivots performed by the LeastEntered pivoting

rule. More precisely, he showed that there exists linear programs in standard form with n equations and $m = O(n)$ variables, for which the number of pivots performed by the LeastEntered pivoting rule is $2^{\Omega(\sqrt{n})}$.

1.2.2 Randomized pivoting rules

RandomEdge

Perhaps the most natural randomized pivoting rule, suggested already by Dantzig himself, is RandomEdge, which among all improving pivots (edges) for the current basis (vertex) chooses one uniformly at random. The upper bounds currently known for RandomEdge are still exponential (see Gärtner and Kaibel [48]). (For additional results regarding RandomEdge, see [17, 47, 50, 6].) RandomEdge is also applicable in a much wider abstract setting. Matoušek and Szabó [89] showed that it can be subexponential on acyclic unique sink orientations (see Section 3.3.2 for a description of unique sink orientations). In Chapter 4 we present a result of Friedmann, Hansen and Zwick [42], showing that there exist standard form linear programs with n equations and $2n$ variables for which the expected number of improving pivots performed by RandomEdge is $2^{\Omega(n^{1/4})}$. This is the first super-polynomial lower bound for RandomEdge for linear programming.

RandomFacet

The RandomFacet pivoting rule is a recursively defined pivoting rule that works as follows. For a given basis B, we pick a uniformly random index $j \notin B$, and recursively solve the sub-problem in which we require that j never enters the basis. I.e., we have $\mathbf{x}_j = 0$. After obtaining an optimal basis B' for this sub-problem we perform an improving pivot with j, if possible, and restart the process from $B'[j]$. If j is not an improving pivot then B' is optimal. Geometrically the algorithm can be viewed as follows. Starting from some vertex $\mathbf{x}$ we pick a uniformly random facet that contains $\mathbf{x}$ and find an optimal vertex within that facet. If possible, we then make an improving pivot leaving that facet and repeat.

Kalai [70, 71] and Sharir and Welzl [111] independently introduced the RandomFacet pivoting rule. It was shown independently by Kalai [70, 71] and by Matoušek, Sharir, and Welzl [88] that the expected number of improving pivots performed by the RandomFacet pivoting rule is subexponential in the size of the linear program. More precisely, for a linear program in standard form with n equations and m variables the number of pivots is at most $2^{O(\sqrt{n \log m})}$. The pivoting rules of Kalai [70, 71] and of Matoušek *et al.*

[88] are in a sense dual to each other (see Goldwasser [54]). If a d-dimensional linear program is represented by (n–d) inequalities and d non-negativity constraints (a total of n facets), then the previously mentioned bound becomes $2^{O(\sqrt{(n-d)\log n})}$. In particular, if $n = \Omega(d^2)$ the bound is still exponential. For the dual linear program (see Section 1.3 below) the corresponding bound is, however, $2^{O(\sqrt{d \log n})}$ which is always subexponential.

The RandomFacet algorithm can, in fact, be used to solve a more general class of abstract problems known as *LP-type* problems. We describe LP-type problems in Section 3.3.1. (See also Gärtner [44] for an application of the algorithm to even more general abstract optimization problems.) Matoušek [86] gave a lower bound of subexponential form for the abstract setting of acyclic unique sink orientations (see Section 3.3.2), which can be viewed as a special case of LP-type problems. The lower bound essentially matches a corresponding upper bound for this setting by Gärtner [45].

Friedmann, Hansen, and Zwick [41, 42] gave a lower bound of subexponential form for RandomFacet for linear programs. We present this result in Chapter 5. For linear programs in standard form with n equations and $m = n \log n$ variables, the lower bound is of the form $2^{\Omega(n^{1/3}/\log^{2/3} n)}$.

Randomized Bland's rule

The Randomized Bland's rule is another natural randomized pivoting rule for the simplex method. It works by ordering the indices of the variables uniformly at random, and then it uses Bland's rule [14] in the usual way with these indices. I.e., it repeatedly picks the improving pivot with the smallest index for entering the basis, and the basic variable with the smallest index for leaving the basis. The Randomized Bland's rule is referred to as a "one-permutation" variant of the RandomFacet pivoting rule pivoting rule by Matoušek [86]. In Section 5.8 we show that the expected number of improving pivots performed by the Randomized Bland's rule may be as large as $2^{\Omega(\sqrt{n}/\log n)}$ for linear programs in standard form with n equations and $m = O(n \log n)$ variables.

RaisingTheBar

The RaisingTheBar pivoting rule is a generalization of RandomEdge that was introduced by Kalai [69]. It works as follows. It takes as a parameter an integer M. Starting from some vertex $\mathbf{x}^0 \in \mathbb{R}^m$ it performs a random walk of length M in the graph defined by the vertices of the polyhedron with value at least $v = \mathbf{c}^T\mathbf{x}^0$. I.e., we can view the hyperplane defined by

$\mathbf{c}^T\mathbf{x} = v$ as a "bar" that we stay above during the random walk. After M steps the process is repeated, meaning that "the bar is raised". Note that when $M = 1$ the pivoting rule is the same as RandomEdge (assuming that no neighbor of $\mathbf{x}^0$ has the same value as $\mathbf{x}^0$). Note also that not all pivots performed by the pivoting rule are improving.

1.3 Duality

Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{c} \in \mathbb{R}^m$ be given, and consider the corresponding linear program (P) from Definition 1.1.8. The value $\mathbf{c}^T\mathbf{x}$ for every feasible solution $\mathbf{x}$ of (P) is a lower bound for the optimal value of (P). It is natural to ask whether there is a simple way of providing upper bounds for the optimal value of (P). Observe that for every $\mathbf{y} \in \mathbb{R}^n$ and every feasible solution $\mathbf{x} \in \mathbb{R}^m$ for (P) we have:

$$\mathbf{y}^T\mathbf{A}^T \ \geq \ \mathbf{c}^T \quad \Rightarrow \quad \mathbf{y}^T\mathbf{b} \ = \ \mathbf{y}^T\mathbf{A}^T\mathbf{x} \ \geq \ \mathbf{c}^T\mathbf{x}$$

Hence, if $\mathbf{Ay} \ \geq \ \mathbf{c}$ then $\mathbf{b}^T\mathbf{y}$ is an upper bound for the optimal value of (P). Then it is natural to ask how good an upper bound for the value of (P) can be achieved in this way. Interestingly, the answer to this question can be formulated as a linear program (D) as shown below.

$$(P) \quad \begin{array}{lrcl} \max & \mathbf{c}^T\mathbf{x} & & \\ s.t. & \mathbf{A}^T\mathbf{x} & = & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array} \qquad (D) \quad \begin{array}{lrcl} \min & \mathbf{b}^T\mathbf{y} & & \\ s.t. & \mathbf{Ay} & \geq & \mathbf{c} \end{array}$$

We say that (D) is the *dual* linear program, and that (P) is the *primal* linear program. We use similar terminology for describing (D) as we did for (P). We say that $\mathbf{y} \in \mathbb{R}^n$ is a *feasible solution* for (D) if and only if $\mathbf{Ay} \geq \mathbf{c}$. Furthermore, we say that a feasible solution $\mathbf{y} \in \mathbb{R}^n$ is *optimal* for (D) if and only if $\mathbf{b}^T\mathbf{y} \leq \mathbf{b}^T\mathbf{y}'$ for all $\mathbf{y}' \in \{\mathbf{y} \in \mathbb{R}^n \mid \mathbf{Ay} \geq \mathbf{c}\}$. We say that (D) is *infeasible* if there are no feasible solutions for (D), and we say that (D) is *unbounded* if for every $M \in \mathbb{R}$ there is a feasible solution $\mathbf{y} \in \mathbb{R}^n$ for (D) with $\mathbf{b}^T\mathbf{y} \leq M$.

The following lemma follows from the discussion above.

Theorem 1.3.1 (Weak duality) *Let $\mathbf{x} \in \mathbb{R}^m$ be a primal feasible solution and $\mathbf{y} \in \mathbb{R}^n$ be a dual feasible solution. Then $\mathbf{c}^T\mathbf{x} \ \leq \ \mathbf{b}^T\mathbf{y}$.*

Corollary 1.3.2 *We have:*

(*i*) *If* $\mathbf{x} \in \mathbb{R}^m$ *and* $\mathbf{y} \in \mathbb{R}^n$ *are feasible solutions to* (P) *and* (D)*, respectively, and* $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$*, then* $\mathbf{x}$ *and* $\mathbf{y}$ *are optimal.*

(*ii*) *If* (P) *is unbounded then* (D) *is infeasible.*

(*iii*) *If* (D) *is unbounded then* (P) *is infeasible.*

(*iv*) *If* (P) *and* (D) *are both feasible, then both* (P) *and* (D) *have optimal solutions.*

The following theorem shows that the best upper bound obtained from (D) matches the best lower bound obtained from (P).

Theorem 1.3.3 (Strong duality) *The primal linear program* (P) *has an optimal solution* $\mathbf{x}^* \in \mathbb{R}^m$ *if and only if the dual linear program* (D) *has an optimal solution* $\mathbf{y}^* \in \mathbb{R}^n$*. Furthermore, if* $\mathbf{x}^*$ *and* $\mathbf{y}^*$ *are optimal for* (P) *and* (D)*, respectively, then* $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$*.*

Proof: We will show that the existence of $\mathbf{x}^*$ implies the existence of $\mathbf{y}^*$. The other direction follows from transforming (D) to standard form and applying the same argument.

We present a proof that is based on the correctness of the simplex method for non-cycling pivoting. We assume for simplicity that (P) has at least one basic feasible solution. The theorem can also be proved in other ways that do not rely on this assumption. See, e.g., [10, 87].

Since (P) has a basic feasible solution it follows from the existence of non-cycling pivoting rules for the simplex method that there exists an optimal basis B. Recall that the reduced costs for such an optimal basis B are non-positive; $\bar{\mathbf{c}}^B = \mathbf{c} - \mathbf{A}\mathbf{A}_B^{-1}\mathbf{c}_B \leq \mathbf{0}$. Hence, $\mathbf{y}^B := \mathbf{A}_B^{-1}\mathbf{c}_B$ is dual feasible. Furthermore, we get that:

$$\mathbf{b}^T\mathbf{y}^B \;=\; \mathbf{b}^T\mathbf{A}_B^{-1}\mathbf{c}_B \;=\; (\mathbf{x}^B)^T\mathbf{c}_B \;=\; \mathbf{c}^T\bar{\mathbf{x}}^B \;.$$

Hence, it follows from Corollary 1.3.2 (*i*) that $\mathbf{y}^B$ must be optimal for (D). □

Theorem 1.3.4 (Complementary slackness) *Two vectors* $\mathbf{x} \in \mathbb{R}^m$ *and* $\mathbf{y} \in \mathbb{R}^n$ *are optimal for* (P) *and* (D)*, respectively, if and only if:*

$$\mathbf{A}^T\mathbf{x} \;=\; \mathbf{b} \tag{1.1}$$

$$\mathbf{x} \;\geq\; \mathbf{0} \tag{1.2}$$

$$\mathbf{A}\mathbf{y} \;\geq\; \mathbf{c} \tag{1.3}$$

$$(\mathbf{A}\mathbf{y} - \mathbf{c})^T\mathbf{x} \;=\; 0 \tag{1.4}$$

Proof: Assume that $\mathbf{x}$ and $\mathbf{y}$ satisfy (1.1), (1.2), and (1.3), and observe that:

$$\mathbf{y}^T\mathbf{b} - \mathbf{c}^T\mathbf{x} \;=\; \mathbf{y}^T\mathbf{A}^T\mathbf{x} - \mathbf{c}^T\mathbf{x} \;=\; (\mathbf{A}\mathbf{y} - \mathbf{c})^T\mathbf{x}. \tag{1.5}$$

Hence, if $\mathbf{x}$ and $\mathbf{y}$ satisfy (1.4) we get $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$, and it follows from Corollary 1.3.2 that $\mathbf{x}$ and $\mathbf{y}$ must be optimal.

On the other hand, if $\mathbf{x}$ and $\mathbf{y}$ are optimal for (P) and (D), respectively, then $\mathbf{x}$ and $\mathbf{y}$ are feasible solutions to (P) and (D), and (1.1), (1.2), and (1.3) must be satisfied. Furthermore, it follows from the strong duality theorem that $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$. We then get from (1.5) that (1.4) is satisfied. □

For additional details about duality we refer to, e.g., [10, 87, 19, 108].

1.4 Related results

Although no polynomial versions of the simplex method are known, linear programs can be solved in polynomial time using either the *ellipsoid method* of Khachiyan [76], or the *interior point method* of Karmarkar [73]. The best known bounds for interior point methods show that a linear program in standard form with m variables and $n \leq m$ equations can be solved using $O(m^3L)$ arithmetic operations, where L is the number of bits needed to describe the linear program. The first bound of this form was by Renegar [103]. For more on the ellipsoid method and its combinatorial consequences, see Grötschel *et al.* [56]. For more on interior point methods, see Nesterov and Nemirovskii [96] and Ye [119].

The ellipsoid and interior point methods are polynomial, but are not *strongly polynomial*, i.e., their running times depend on the numerical values of the coefficients appearing in the program, even in the *unit-cost* model in which each arithmetical operation is assumed to take constant time. A strongly polynomial bound can only depend on n, the number of equations, and m, the number of variables, and not on L, the bit complexity. Furthermore, the ellipsoid and the interior point methods have a strong numerical flavor, as opposed to the more combinatorial flavor of simplex methods. It is a major open problem whether there exists a strongly polynomial time algorithm for solving linear programs. A polynomial pivoting rule for the simplex method would provide a positive answer to this open problem, assuming that every iteration is performed in strongly polynomial time.

The best known (expected) time bound for solving linear programs, when the bound is independent of L (the bit complexity), comes from the simplex method with the RandomFacet pivoting rule [70, 71, 88]. More precisely,

the bound is obtained in conjunction with Clarkson's linear programming algorithm [21]. Matoušek, Sharir, and Welzl [88] show that the resulting algorithm solves d-dimensional linear programs with n facets in expected time $O(d^2 n + 2^{\sqrt{d \log d}})$.

It is not known whether there exists a pivoting rule that requires a polynomial number of pivoting steps on *any* linear program. This is again one of the most important open problems in the field of linear programming. The existence of such a polynomial pivoting rule would imply, of course, that the *diameter* of the edge-vertex graph of any polyhedron is polynomial in the number of facets defining it. The Hirsch conjecture (see, e.g., [24], pp. 160,168), which states that the diameter of the graph defined by an n-facet d-dimensional polyhedron is at most $n - d$ has recently been refuted by Santos [105]. The best upper bound known on the diameter is a quasi-polynomial bound of $n^{\log d + 1}$ obtained by Kalai and Kleitman [72] (see also [70]).

Chapter 2

Markov decision processes

In 1957 Bellman [7] introduced *Markov decision processes* (MDPs); a one-player special case of Shapley's stochastic games from 1953 [110]. MDPs model long-term sequential decision making under stochastic uncertainty. Many variants of MDPs have been studied in the literature. MDPs are widely used to model stochastic optimization problems in various areas ranging from operations research, machine learning, artificial intelligence, economics and game theory. For a thorough treatment of MDPs and their numerous practical applications, see the books of Howard [63], Derman [30], Puterman [101] and Bertsekas [9].

One of the most practical algorithms for solving MDPs is the *policy iteration* algorithm, introduced by Howard in 1960 [63]. MDPs can be solved using linear programming, and Howard's policy iteration algorithm can be viewed as a generalization of the simplex method for MDPs. Until a few years ago the worst-case complexity of Howard's algorithm remained a mystery. The best upper bound, due to Mansour and Singh [85], was only slightly better than the trivial exponential upper bound, but it was widely believed that the algorithm was (strongly) polynomial. Adapting a result of Friedmann [38], Fearnley [34], however, recently managed to prove an exponential lower bound for the number of iterations performed by Howard's algorithm, when the bound must be independent of the bit-complexity. On the positive side, Ye [121] recently showed that for discounted MDPs with a fixed discount the number of iterations performed by Howard's algorithm is polynomially bounded, proving that the algorithm is, in fact, strongly polynomial in this case. The bound of Ye was later improved by Hansen, Miltersen, and Zwick [59].

In this chapter we give an introduction to infinite-horizon MDPs with

finite sets of *states* and *actions*. We consider three criteria for optimality: *discounted reward*, *average reward*, and *total reward*. The presentation of MDPs in this chapter is mainly based on Puterman [101] and Hansen, Miltersen, and Zwick [59].

A Markov decision process is composed of a finite set of *states*, and for each state a finite, non-empty set of *actions*. In each time unit (or epoch), the MDP is in exactly one of the states. A *controller* (or *decision maker* or *player*) must choose one of the actions associated with the current state. Using an action incurs an immediate *reward* (or *payoff*), and results in a probabilistic transition to a new state according to a probability distribution that depends on the action. The process goes on indefinitely. The goal of the controller is to *maximize* the incurred rewards according to some criterion:

- For the discounted reward criterion the controller tries to maximize the expected total *discounted* reward of the infinite sequence of actions taken, with respect to a fixed *discount factor*.
- For the average reward criterion the controller tries to maximize the expected limiting average of the rewards of the infinite sequence of actions taken.
- For the total reward criterion the controller tries to maximize the expected total reward of the infinite sequence of actions taken.

In some cases it is more natural to talk about *costs* than rewards, in which case the controller tries to *minimize* the incurred costs.

As we will see, MDPs can be solved using linear programming, and the problem can therefore be solved in polynomial time. There is, however, no known *strongly polynomial* time algorithm for solving the problem. This is arguably the most prominent unresolved theoretical problem related to MDPs.

The chapter is organized as follows. Section 2.1 contains definitions for MDPs that are shared by all three criteria described above. In Section 2.2 we study the discounted reward criterion. We introduce the ValueIteration and PolicyIteration algorithms, and show how the solution of a discounted MDP can be formulated as a linear program. We also show that the PolicyIteration algorithm can be used to solve discounted MDPs, with a fixed discount, in strongly polynomial time. In sections 2.3 and 2.4 we study the average reward criterion and total reward criterion, respectively. We introduce the PolicyIteration algorithm for these criteria, and show a linear programming formulation for the solution of such MDPs.

2.1 Definitions

We next formally define Markov decision processes.

Definition 2.1.1 (Markov decision processes) *A* Markov decision process *(MDP) is a tuple $M = (S, A, s, c, p)$, where*

- *S is a set of states,*
- *A is a set of actions,*
- *$s : A \to S$ assigns each action to the state from which it can be used,*
- *$c : A \to \mathbb{R}$ associates each action with a reward,*
- *and $p : A \to \Delta(S)$ associates each action with a probability distribution over states according to which the next state is chosen when a is used.*

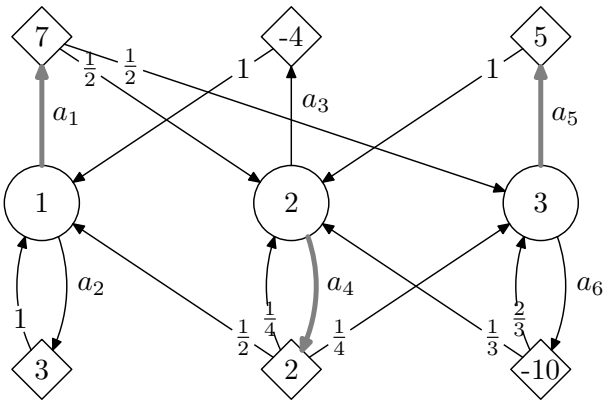
For every $i \in S$, we let $A_i = \{a \in A \mid s(a) = i\}$ be the set of actions that can be used from i. We assume that $A_i \neq \emptyset$, for every $i \in S$.

We use $n = |S|$ and $m = |A|$ to denote the number of states and actions, respectively, in an MDP.

Figure 2.1 shows an example of an MDP. The states are presented as circles numbered from 1 to 3. I.e., the set of states is $S = \{1, 2, 3\}$. The actions are represented as arrows leaving the states. The set of actions is $A = \{a_1, \ldots, a_6\}$. We, for instance, have $s(a_4) = 2$. The rewards of the actions are shown inside the diamond-shaped vertices. For instance, $c(a_4) = 2$. Finally, the probability distribution associated with an action is shown as numbers labelling the edges leaving the corresponding diamond-shaped vertex. For instance, the probability of moving to state 1 when using action a_4 is $\frac{1}{2}$.

An MDP can be represented in different ways. It can be compactly represented by matrices, as shown in Definition 2.1.2 below, and it can be viewed as a graph, as shown in Definition 2.1.3 below. The matrix representation greatly simplifies manipulation of MDPs, and we will often use it when reasoning about MDPs in general. The graphical representation is a good tool for understanding concrete MDPs, and we will generally use this representation when proving lower bounds for algorithms.

Definition 2.1.2 (Matrix representation) *Let $M = (S, A, s, c, p)$ be an MDP. We may assume, without loss of generality, that $S = [n] = \{1, \ldots, n\}$ and $A = [m] = \{1, \ldots, m\}$. We let $\mathbf{P} \in \mathbb{R}^{m \times n}$, where $\mathbf{P}_{a,i} = p(a)_i$ is the*



$$\mathbf{J} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 1 & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 7 \\ 3 \\ -4 \\ 2 \\ 5 \\ -10 \end{bmatrix}$$

$$\mathbf{P}_\pi = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{c}_\pi = \begin{bmatrix} 7 \\ 2 \\ 5 \end{bmatrix}$$

k	$\mathbf{e}_1^T \mathbf{P}_\pi^k$		
0	1	0	0
1	0	$\frac{1}{2}$	$\frac{1}{2}$
2	$\frac{2}{8}$	$\frac{5}{8}$	$\frac{1}{8}$
3	$\frac{10}{32}$	$\frac{13}{32}$	$\frac{9}{32}$
$\vdots$		$\vdots$	

Figure 2.1: Example of a simple MDP and a policy π.

probability of ending up in state i after taking action a, for every $a \in A = [m]$ and $i \in S = [n]$, be the probability matrix *of the MDP, and $\mathbf{c} \in \mathbb{R}^m$, where $\mathbf{c}_a = c(a)$ is the reward of action $a \in A = [m]$, be its* reward vector. *We also let $\mathbf{J} \in \mathbb{R}^{m \times n}$ be a matrix such that $\mathbf{J}_{a,i} = 1$ if $a \in A_i$, and 0 otherwise.*

Definition 2.1.3 (Graphical representation) *Let $M = (S, A, s, c, p)$ be an MDP. We let $G_M = (V_0, V_R, E_0, E_R, c, p)$ be a directed bipartite graph associated with M, where $V_0 = S$, $V_R = A$, $E_0 = \{(i, a) \mid a \in A_i\} \subseteq V_0 \times V_R$ and $E_R = \{(a, i) \mid p(a)_i > 0\} \subseteq V_R \times V_0$. Each edge $e = (i, a) \in E_0$ has a reward $c(a)$ associated with it, and each edge $e = (a, i) \in E_R$ has a probability $p(a)_i$ associated with it. We refer to G_M as the* graphical representation *of the MDP. We refer to V_0 as the set of* decision *vertices, and to V_R as the set of* randomization *vertices.*

Figure 2.1 shows both the graphical representation and the matrix representation of a simple MDP. The diamond-shaped vertices are randomization vertices.

We say that an action $a \in A$ is *deterministic* if there is a state $i \in S$ such that $p(a)_i = 1$. (Note that for every $j \neq i$ we then have $p(a)_j = 0$.) We say that an MDP is deterministic if all its actions are deterministic. For succinctness, if a is a deterministic action we usually omit the corresponding vertex from the graphical representation of the MDP and simply represent a by an edge (i, j), where $i = s(a)$ and $p(a)_j = 1$. Note that the graph G_M is then no longer bipartite. In particular, the graphical representation of a deterministic MDP is simply a weighted directed graph, and its incidence matrix is $(\mathbf{P} - \mathbf{J})$.

A *policy* for the controller of an MDP is a rule that specifies which action should be taken in each situation. The decision may in general depend on the current state of the process and possibly on the *history*. A policy is *positional* if it is deterministic and history independent. A policy is *optimal* for some optimality criterion if it maximizes the incurred rewards according to this criterion. It can be shown that every MDP has an optimal positional policy for the three optimality criteria we consider. We are therefore going to restrict our attention to the use of positional policies. In particular, whenever we use the word policy we will mean a positional policy. We refer to, e.g., Puterman [101] for a proof of the fact that it is not necessary for the controller to use randomized history dependent policies. It is worth pointing out that we will actually come close to obtaining this result in this chapter.

Definition 2.1.4 (Policies) *A (positional)* policy *(or* strategy*) π is a mapping $\pi : S \to A$ such that $\pi(i) \in A_i$, for every $i \in S$. We say that the*

controller uses policy π if whenever the MDP is in state i, the action $\pi(i)$ is used. We let $\Pi = \Pi(M)$ be the set of all policies.

When convenient, we identify a policy π with the set $\pi(S) \subseteq A$. We let $\mathbf{P}_\pi \in \mathbb{R}^{n \times n}$ be the matrix obtained by selecting the *rows* of $\mathbf{P}$ whose indices belong to π. Note that $(\mathbf{P}_\pi)_i = \mathbf{P}_{\pi(i)}$. We will assume that the actions are ordered according to states, such that for every policy π, by the above convention we have $\mathbf{J}_\pi = \mathbf{I}$. Note that $\mathbf{P}_\pi$ is a (row) *stochastic matrix*; its elements are non-negative and the elements in each row sum to 1. Thus, $\mathbf{P}_\pi$ defines a Markov chain. Let the initial state for a run of the Markov chain defined by $\mathbf{P}_\pi$ be picked according to some probability distribution $\mathbf{b} \in \mathbb{R}^n$. Then the probabilities of being in the different states after exactly k steps are given by the vector $\mathbf{b}^T \mathbf{P}_\pi^k$. In particular, if an MDP is started in state i and the controller uses policy π, then the probabilities of being in the different states after k steps are given by the vector $\mathbf{e}_i^T \mathbf{P}_\pi^k$, where $\mathbf{e}_i$ is the i'th unit vector. We let $X_\pi^k(i)$ be the random variable corresponding to the state the MDP is in after k transitions when the controller uses policy π starting from state i. I.e.,

$$\Pr\left[X_\pi^k(i) = j\right] \;=\; (\mathbf{e}_i^T \mathbf{P}_\pi^k)_j \;.$$

Similar to the definition of $\mathbf{P}_\pi$, we let $\mathbf{c}_\pi \in \mathbb{R}^n$ be the vector containing the rewards of the actions that belong to π. The pair $\mathbf{P}_\pi, \mathbf{c}_\pi$ is thus a Markov chain with rewards assigned to its states. A run of an MDP generates an infinite sequence of rewards. Let $Y_\pi^k(i)$ be the random variable corresponding to the reward observed after k transitions when the MDP starts in state i and the controller uses the policy π. The expected reward observed at time k when starting from state i and using policy π is then:

$$\mathrm{E}\left[Y_\pi^k(i)\right] = \mathbf{e}_i^T \mathbf{P}_\pi^k \mathbf{c}_\pi \;.$$

Figure 2.1 shows a policy π for a simple MDP. The policy is represented by bold gray arrows. The corresponding matrix $\mathbf{P}_\pi$ and vector $\mathbf{c}_\pi$ are also shown. Furthermore, the table at the lower right corner shows the probabilities of being in different states after k steps when starting in state 1.

We present three different criteria for optimality of an MDP. Each is based on a separate way of evaluating an infinite sequence of rewards, as shown by the following three definitions.

Definition 2.1.5 (Values, discounted reward) *For every policy π, we define the* value *of state $i \in S$, for the discounted reward criterion, to be the*

expected total discounted reward, for some discount factor $0 < \gamma < 1$*, when the MDP is started in state* i *and the controller plays according to* π*:*

$$\mathrm{val}_\pi^D(i) \;=\; \sum_{k=0}^{\infty} \gamma^k E\left[Y_\pi^k(i)\right] \;=\; \sum_{k=0}^{\infty} \mathbf{e}_i^T (\gamma \mathbf{P}_\pi)^k \mathbf{c}_\pi$$

Definition 2.1.6 (Values, average reward) *For every policy* π*, we define the value of state* $i \in S$*, for the average reward criterion, to be the limiting average of the rewards when the MDP is started in state* i *and the controller plays according to* π*:*

$$\mathrm{val}_\pi^A(i) \;=\; \liminf_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} E\left[Y_\pi^k(i)\right] \;=\; \liminf_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{e}_i^T \mathbf{P}_\pi^k \mathbf{c}_\pi$$

Definition 2.1.7 (Values, total reward) *For every policy* π*, we define the value of state* $i \in S$*, for the total reward criterion, to be the expected total sum of rewards when the MDP is started in state* i *and the controller plays according to* π*:*

$$\mathrm{val}_\pi^T(i) \;=\; \sum_{k=0}^{\infty} E\left[Y_\pi^k(i)\right] \;=\; \sum_{k=0}^{\infty} \mathbf{e}_i^T \mathbf{P}_\pi^k \mathbf{c}_\pi$$

Note that values for the total reward criterion are not always well-defined. We will introduce a *stopping condition* that ensures the existence of values in this case.

If it is clear from the context whether we are using the discounted reward criterion, the total reward criterion, or the average reward criterion, or if we want to describe all three criteria jointly, we simply use $\mathrm{val}_\pi(i)$ instead of $\mathrm{val}_\pi^D(i)$, $\mathrm{val}_\pi^A(i)$, or $\mathrm{val}_\pi^T(i)$.

For all three criteria we make the following definition.

Definition 2.1.8 (Optimal policies) *Let* $M = (S, A, s, r, p)$ *be an MDP and let* $\pi \in \Pi$ *be a policy. We say that* π *is* optimal *for* M *if and only if for all* $\pi' \in \Pi$ *and all* $i \in S$ *we have* $\mathrm{val}_\pi(i) \geq \mathrm{val}_{\pi'}(i)$*.*

Solving an MDP means finding an optimal policy. Since an optimal policy must obtain the maximum value for all states simultaneously, it is non-trivial to prove that an optimal policy always exist. It is well known, however, that optimal policies do always exist for the optimality criteria presented here. The first such result was established by Shapley [110].

Theorem 2.1.9 *Every MDP has an optimal policy.*

We will later show proofs of Theorem 2.1.9 for discounted rewards, total rewards, and average rewards. For an alternative, more extensive, presentation of these proofs see, e.g., Puterman [101].

Note that Definition 2.1.8 implies that if two policies π and π' are both optimal then $\mathrm{val}_\pi(i) = \mathrm{val}_{\pi'}(i)$ for all states i. We say that the *optimal value* of a state is its value for an optimal policy.

2.2 The discounted reward criterion

When operating with the discounted reward criterion, the MDP is associated with an additional parameter $0 < \gamma < 1$. Recall that the value of a state i for some policy π is then defined as $\mathrm{val}^D_\pi(i) = \sum_{k=0}^{\infty} \mathbf{e}_i^T (\gamma \mathbf{P}_\pi)^k \mathbf{c}_\pi$.

There are two ways of interpreting the discount factor γ. The discount factor may be viewed as the inverse of the rate of inflation. Thus, a reward of $c(a)$ of action a at time k corresponds to a reward of $\gamma^k c(a)$ at time 0. Alternatively, $1-\gamma$ may be viewed as the probability that the MDP stops after each step. The expected reward of an action a at time k is then again $\gamma^k c(a)$, as γ^k is the probability that the MDP reaches time k.

Lemma 2.2.1 *For any policy π, the matrix $(\mathbf{I} - \gamma \mathbf{P}_\pi)$ is nonsingular and*

$$(\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \;=\; \sum_{k=0}^{\infty} (\gamma \mathbf{P}_\pi)^k \;\geq\; \mathbf{I}\,.$$

Proof: Assume for the sake of contradiction that $(\mathbf{I} - \gamma \mathbf{P}_\pi)$ is singular. Then there exists $\mathbf{x} \in \mathbb{R}^n$ such that $(\mathbf{I} - \gamma \mathbf{P}_\pi)\mathbf{x} = \mathbf{0}$ and $\mathbf{x} \neq \mathbf{0}$. Let $i = \mathrm{argmax}_{\,i \in [n]} |\mathbf{x}_i|$. Then $0 = |\mathbf{x}_i - \gamma \mathbf{P}_{\pi(i)}\mathbf{x}| \geq (1-\gamma)|\mathbf{x}_i| > 0$; a contradiction.

Since $(\gamma \mathbf{P}_\pi)^t \to 0$ for $t \to \infty$, we observe that

$$\mathbf{I} = \lim_{t\to\infty} \mathbf{I} - (\gamma \mathbf{P}_\pi)^t = \lim_{t\to\infty} (\mathbf{I} - \gamma \mathbf{P}_\pi) \sum_{k=0}^{t-1} (\gamma \mathbf{P}_\pi)^k = (\mathbf{I} - \gamma \mathbf{P}_\pi) \sum_{k=0}^{\infty} (\gamma \mathbf{P}_\pi)^k$$

and the identity from the lemma follows.

Finally, since $\mathbf{P}^0_\pi = \mathbf{I}$ and all entries of $\mathbf{P}^k_\pi$, for all k, are non-negative, we get that $(\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \geq \mathbf{I}$. □

As a consequence of Lemma 2.2.1, the value of every policy π and state i always exists.

Definition 2.2.2 (Value vectors) *For every policy $\pi \in \Pi$, we define the* value vector $\mathbf{v}^\pi \in \mathbb{R}^n$ *by:*

$$\mathbf{v}^\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{c}_\pi \ .$$

I.e., $(\mathbf{v}^\pi)_i = \mathrm{val}^D_\pi(i)$ *for all* $i \in S = [n]$.

Given two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, we say that $\mathbf{u} \leq \mathbf{v}$ if and only if $\mathbf{u}_i \leq \mathbf{v}_i$, for every $1 \leq i \leq n$. We say that $\mathbf{u} < \mathbf{v}$ if and only if $\mathbf{u} \leq \mathbf{v}$ and $\mathbf{u} \neq \mathbf{v}$. We interpret $\mathbf{u} \geq \mathbf{v}$ and $\mathbf{u} > \mathbf{v}$ similarly. Hence, a policy π is optimal if and only if $\mathbf{v}^\pi \geq \mathbf{v}^{\pi'}$ for all $\pi' \in \Pi$.

Definition 2.2.3 (One-step operators) *Define the operators* $\mathcal{T} : \mathbb{R}^n \to \mathbb{R}^n$ *and* $\mathcal{T}_\pi : \mathbb{R}^n \to \mathbb{R}^n$*, for every policy π, as follows:*

$$\begin{aligned} \mathcal{T}_\pi \mathbf{v} &= \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{v} \ , \\ \mathcal{T} \mathbf{v} &= \max_{\pi \in \Pi} \mathcal{T}_\pi \mathbf{v} = \max_{\pi \in \Pi} \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{v} \ . \end{aligned}$$

Note that $\mathcal{T}_\pi \mathbf{v}$ are the values obtained by taking one step according to π and then receiving the corresponding value of $\mathbf{v}$. Also note that the individual components of $\mathcal{T}_\pi \mathbf{v}$ and $\mathcal{T} \mathbf{v}$ only depend on the action used from the corresponding state. In particular, $(\mathcal{T} \mathbf{v})_i = \max_{a \in A_i} \mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{v}$, and the operator $\mathcal{T}$ is well-defined although the maximum is taken over vectors.

The operators $\mathcal{T}$ and $\mathcal{T}_\pi$, for $\pi \in \Pi$, are contractions with Lipschitz constant γ.

Lemma 2.2.4 *For every* $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ *and* $\pi \in \Pi$ *we have:*

$$\begin{aligned} \|\mathcal{T} \mathbf{u} - \mathcal{T} \mathbf{v}\|_\infty &\leq \gamma \|\mathbf{u} - \mathbf{v}\|_\infty \quad \textit{and} \\ \|\mathcal{T}_\pi \mathbf{u} - \mathcal{T}_\pi \mathbf{v}\|_\infty &\leq \gamma \|\mathbf{u} - \mathbf{v}\|_\infty \ . \end{aligned}$$

Proof: Let $i \in S$, and assume that $(\mathcal{T} \mathbf{u})_i \geq (\mathcal{T} \mathbf{v})_i$. Let

$$\begin{aligned} a &= \operatorname*{argmax}_{a \in A_i} \mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{u} \quad \text{and} \\ b &= \operatorname*{argmax}_{b \in A_i} \mathbf{c}_b + \gamma \mathbf{P}_b \mathbf{v} \ . \end{aligned}$$

Then,

$$\begin{aligned} (\mathcal{T} \mathbf{u} - \mathcal{T} \mathbf{v})_i &= (\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{u}) - (\mathbf{c}_b + \gamma \mathbf{P}_b \mathbf{v}) \\ &\leq (\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{u}) - (\mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{v}) \\ &= \gamma \mathbf{P}_a (\mathbf{u} - \mathbf{v}) \\ &\leq \gamma \|\mathbf{u} - \mathbf{v}\|_\infty \ . \end{aligned}$$

The last inequality follows from the fact that the elements in $\mathbf{P}_a$ are non-negative and sum up to 1. The case when $(\mathcal{T}\mathbf{u})_i \leq (\mathcal{T}\mathbf{v})_i$ is analogous. Also, the proof for $\mathcal{T}_\pi$ is the same, only simpler since $a = b$. □

The Banach fixed point theorem now implies that the operators $\mathcal{T}$ and $\mathcal{T}_\pi$, for $\pi \in \Pi$, have unique fixed points. Also, from Definition 2.2.2 we get that $\mathbf{v}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}^\pi = \mathcal{T}_\pi \mathbf{v}^\pi$.

Corollary 2.2.5 *There is a unique vector $\mathbf{v}^* \in \mathbb{R}^n$ such that $\mathcal{T}\mathbf{v}^* = \mathbf{v}^*$. Also, $\mathbf{v}^\pi$ is the unique vector satisfying $\mathcal{T}_\pi \mathbf{v}^\pi = \mathbf{v}^\pi$.*

Corollary 2.2.5 also serves as an alternative proof for $(\mathbf{I} - \gamma \mathbf{P}_\pi)$ being nonsingular. As we will see later a policy is optimal if and only if it has value vector $\mathbf{v}^*$. We therefore refer to $\mathbf{v}^*$ as the *optimal value vector*.

Lemma 2.2.6 *Let $\pi \in \Pi$ and $\mathbf{v} \in \mathbb{R}^n$. If $\mathcal{T}_\pi \mathbf{v} > \mathbf{v}$ then $\mathcal{T}_\pi^2 \mathbf{v} = \mathcal{T}_\pi \mathcal{T}_\pi \mathbf{v} > \mathcal{T}_\pi \mathbf{v}$. The same implication is true for $\geq$, $<$ and $\leq$.*

Proof: Suppose $\mathcal{T}_\pi \mathbf{v} > \mathbf{v}$. Since all entries of $\mathbf{P}_\pi$ are non-negative we get:

$$\mathcal{T}_\pi^2 \mathbf{v} = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi (\mathcal{T}_\pi \mathbf{v}) > \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{v} = \mathcal{T}_\pi \mathbf{v} \ .$$

The three other cases are proved analogously. □

The following corollary follows from repeated use of Lemma 2.2.6 and the fact that $\mathcal{T}_\pi^N \mathbf{v} \to \mathbf{v}^\pi$ for $N \to \infty$.

Corollary 2.2.7 *Let π and π' be two policies. If $\mathcal{T}_{\pi'} \mathbf{v}^\pi > \mathbf{v}^\pi$, then $\mathbf{v}^{\pi'} > \mathbf{v}^\pi$. The same implication is true for $\geq$, $<$ and $\leq$.*

Note that $\mathcal{T}_{\pi'} \mathbf{v}^\pi > \mathbf{v}^\pi$ can be equivalently stated as $\mathbf{c}_{\pi'} - (\mathbf{I} - \gamma \mathbf{P}_{\pi'}) \mathbf{v}^\pi > \mathbf{0}$. I.e., if an action is better for one step with respect to the values of the current policy, then it is also better to keep using this action. This motivates the following definition.

Definition 2.2.8 (Reduced costs, improving switches) *The* reduced cost *vector $\bar{\mathbf{c}}^\pi \in \mathbb{R}^m$ corresponding to a policy π is defined to be*

$$\bar{\mathbf{c}}^\pi \ = \ \mathbf{c} - (\mathbf{J} - \gamma \mathbf{P}) \mathbf{v}^\pi \ .$$

We say that an action $a \in A$ is an improving switch *with respect to a policy π if and only if $\bar{\mathbf{c}}_a^\pi > 0$.*

Note that this definition is very similar to the definition of reduced costs (Definition 1.2.1) and improving pivots (Definition 1.2.4) for linear programming. We will later see that the problem of solving an MDP can be formulated as a linear program, and in this case the definitions are the same.

It is important to stress the difference between $\mathbf{c}_\pi \in \mathbb{R}^n$, the vector obtained by selecting the entries of $\mathbf{c} \in \mathbb{R}^m$ corresponding to policy π, and the reduced cost vector $\bar{\mathbf{c}}^\pi = \mathbf{c} - (\mathbf{J} - \gamma\mathbf{P})\mathbf{v}^\pi \in \mathbb{R}^m$ of Definition 2.2.8.

The following lemma corresponds to Lemma 1.2.3 for linear programming.

Lemma 2.2.9 *For any policy π we have $(\bar{\mathbf{c}}^\pi)_\pi = \mathbf{0} \in \mathbb{R}^n$. I.e., the reduced cost of every action used in π is zero.*

Proof: From Definition 2.2.8 and Definition 2.2.2 we get:

$$(\bar{\mathbf{c}}^\pi)_\pi = \mathbf{c}_\pi - (\mathbf{I} - \gamma\mathbf{P}_\pi)\mathbf{v}^\pi = \mathbf{c}_\pi - (\mathbf{I} - \gamma\mathbf{P}_\pi)(\mathbf{I} - \gamma\mathbf{P}_\pi)^{-1}\mathbf{c}^\pi = \mathbf{0} \ .$$

□

We also get the following lemma which we will later see corresponds to Lemma 1.2.2.

Lemma 2.2.10 *For every two policies π', π we have*

$$\mathbf{v}^{\pi'} - \mathbf{v}^\pi \ = \ (I - \gamma P_{\pi'})^{-1}(\bar{\mathbf{c}}^\pi)_{\pi'} \ .$$

Proof: By Definition 2.2.2 and Definition 2.2.8 we have

$$\begin{aligned} \mathbf{v}^{\pi'} - \mathbf{v}^\pi &= (I - \gamma P_{\pi'})^{-1}\mathbf{c}_{\pi'} - \mathbf{v}^\pi \\ &= (I - \gamma P_{\pi'})^{-1}(\mathbf{c}_{\pi'} - (I - \gamma P_{\pi'})\mathbf{v}^\pi) \\ &= (I - \gamma P_{\pi'})^{-1}(\bar{\mathbf{c}}^\pi)_{\pi'} \ . \end{aligned}$$

□

Let $B \subseteq A$ be a set of actions such that $|B \cap A_i| \leq 1$ for all $i \in S$. I.e., B contains at most one action that can be used from each state. Also, if $B \cap A_i \neq \emptyset$ then let $a(B, i)$ be the unique element in $B \cap A_i$. For every policy π we then define $\pi[B]$ to be the policy satisfying, for all $i \in S$:

$$\pi[B](i) \ = \ \begin{cases} a(B,i) & \text{if } B \cap A_i \neq \emptyset \\ \pi(i) & \text{otherwise} \end{cases}$$

We say that $\pi[B]$ is obtained from π by performing the switches in B.

Definition 2.2.11 (Improving set) *A set of actions $B \subseteq A$ is said to be* improving *with respect a policy π if and only if it satisfies the following conditions:*

- *There exists an action $a \in B$ that is an improving switch with respect to π.*
- *$|B \cap A_i| \leq 1$ for all $i \in S$.*
- *For every action $a \in B$, we have $\bar{\mathbf{c}}^\pi \geq 0$.*

Note that not all the actions in an improving set B must be improving switches. None of the actions can have negative reduced cost, however.

Theorem 2.2.12 *Let π be a policy.*

(i) Let $B \subseteq A$ be improving with respect to π. Then $\mathbf{v}^{\pi[B]} > \mathbf{v}^\pi$.

(ii) If no action is an improving switch with respect to π, then π is optimal.

Proof: We first prove (i). Since all actions of π as well as B have non-negative reduced cost, and since at least one action in B is an improving switch with respect to π, we have $(\bar{\mathbf{c}}^\pi)_{\pi[B]} > \mathbf{0}$. Recall that $(\bar{\mathbf{c}}^\pi)_{\pi[B]} \in \mathbb{R}^n$ is the vector obtained from $\bar{\mathbf{c}}^\pi$ by only including components with indices in $\pi[B]$. We then observe that:

$$(\bar{\mathbf{c}}^\pi)_{\pi[B]} \;=\; \mathbf{c}_{\pi[B]} - (\mathbf{I} - \gamma \mathbf{P}_{\pi[B]})\mathbf{v}^\pi \;=\; \mathcal{T}_{\pi[B]}\mathbf{v}^\pi - \mathbf{v}^\pi \;>\; \mathbf{0}\,,$$

and (i) follows from Corollary 2.2.7.

To prove (ii) we assume that there exists another policy π' such that $\mathbf{v}^\pi \not\geq \mathbf{v}^{\pi'}$. From Lemma 2.2.10 it follows that:

$$\mathbf{v}^{\pi'} - \mathbf{v}^\pi \;=\; (I - \gamma P_{\pi'})^{-1}(\bar{\mathbf{c}}^\pi)_{\pi'}\,.$$

Since there are no improving switches with respect to π we have, by definition, that $\bar{\mathbf{c}}^\pi \leq \mathbf{0}$. From Lemma 2.2.1 it follows that $(I - \gamma P_{\pi'})^{-1} \geq \mathbf{I}$. I.e., all entries of $(I - \gamma P_{\pi'})^{-1}$ are non-negative, and we, thus, get $(I - \gamma P_{\pi'})^{-1}(\bar{\mathbf{c}}^\pi)_{\pi'} \leq \mathbf{0}$; a contradiction.

□

In order to prove the existence of an optimal policy, we next define the *policy extraction* operator.

Definition 2.2.13 (Policy extraction operator) *The* policy extraction *operator* $\mathcal{P} : \mathbb{R}^n \to \Pi$ *is defined as follows:*

$$\mathcal{P}\mathbf{v} \;=\; \operatorname*{argmax}_{\pi\in\Pi} \; \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{v} \; .$$

Note that the policy extraction operator can also be defined componentwise as $(\mathcal{P}\mathbf{v})_i = \operatorname{argmax}_{a\in A_i} \mathbf{c}_a + \gamma \mathbf{P}_a \pi$, where $i \in S$. In particular, $\mathcal{P}$ is well-defined although argmax is taken over vectors. If multiple actions obtain the maximum value we use an arbitrary one of them.

The following relation between the one-step operator and the policy extraction operator is immediate.

Lemma 2.2.14 *For every* $\mathbf{v} \in \mathbb{R}^n$ *we have* $\mathcal{T}\mathbf{v} = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}$, *where* $\pi = \mathcal{P}\mathbf{v}$.

The following simple lemma provides an interesting relation between the policy extraction operator and reduced cost vectors.

Lemma 2.2.15 *For every policy* π *we have*

$$(\mathcal{P}\mathbf{v}^\pi)(i) \;=\; \operatorname*{argmax}_{a\in A_i} \; \bar{\mathbf{c}}^\pi_a \quad , \quad i \in S \; .$$

Proof: Observe that for $\bar{\mathbf{c}}^\pi_a = \mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{v}^\pi - \mathbf{v}^\pi_i$, the last term $\mathbf{v}^\pi_i$ only depends on i, not a. Hence,

$$\operatorname*{argmax}_{a\in A_i} \; \bar{\mathbf{c}}^\pi_a + \gamma \mathbf{P}_a \mathbf{v}^\pi - \mathbf{v}^\pi_i = \operatorname*{argmax}_{a\in A_i} \; \bar{\mathbf{c}}^\pi_a + \gamma \mathbf{P}_a \mathbf{v}^\pi = (\mathcal{P}\mathbf{v}^\pi)(i) \; .$$

□

The following lemma supplies a simple proof of Theorem 2.1.9 for the discounted reward criterion. (This is, in fact, the original proof given by Shapley [110].)

Lemma 2.2.16 *Let* $\mathbf{v}^* \in \mathbb{R}^n$ *be the unique fixed point of* $\mathcal{T}$, *and let* $\pi^* = \mathcal{P}\mathbf{v}^*$. *Then,* π^* *is an optimal policy.*

Proof: From Lemma 2.2.9 we know that $(\bar{\mathbf{c}}^{\pi^*})_{\pi^*} = \mathbf{0}$, and from Lemma 2.2.15 we then get that $\max_{a\in A_i} \bar{\mathbf{c}}^{\pi^*}_a = (\bar{\mathbf{c}}^{\pi^*})_{\pi^*(i)} = 0$ for all $i \in S$. It follows that $\bar{\mathbf{c}}^{\pi^*} \leq \mathbf{0}$, and then from Theorem 2.2.12 (ii) that π^* is optimal. □

Function `ValueIteration`$(\mathbf{u}^0, \epsilon)$

$k \leftarrow 0$;
repeat
  $\mathbf{u}^{k+1} \leftarrow \mathcal{T}\mathbf{u}^k$;
  $k \leftarrow k+1$;
until $\|\mathbf{u}^k - \mathbf{u}^{k-1}\|_\infty < \frac{\epsilon(1-\gamma)}{2\gamma}$;
return $\mathbf{u}^k$;

Figure 2.2: The ValueIteration algorithm.

2.2.1 The ValueIteration algorithm

The ValueIteration algorithm, given in Figure 2.2, repeatedly applies the one-step operator $\mathcal{T}$ to an initial vector $\mathbf{u}^0 \in \mathbb{R}^n$, generating a sequence of vectors $(\mathbf{u}^k)_{k=0}^N$, where $\mathbf{u}^{k+1} = \mathcal{T}\mathbf{u}^k$, until the difference between two successive vectors is sufficiently small, i.e., $\|\mathbf{u}^N - \mathbf{u}^{N-1}\|_\infty < \frac{\epsilon(1-\gamma)}{2\gamma}$ for some $\epsilon \geq 0$.

The ValueIteration algorithm was introduced for MDPs by Bellman [7]. It can be viewed as a special case of an algorithm by Shapley [110] for solving the more general class of stochastic games. When the ValueIteration algorithm is initialized with the zero vector, $\mathbf{u}^0 = \mathbf{0}$, the vector $\mathbf{u}^k$ can be viewed as the optimal values for the MDP that terminates after k steps. The ValueIteration algorithm can, thus, be viewed as a dynamic programming algorithm. See also Littman, Dean, and Kaelbling [81] for a presentation of the ValueIteration algorithm.

We say that a policy π is ϵ-optimal if $\|\mathbf{v}^* - \mathbf{v}^\pi\|_\infty \leq \epsilon$, for some $\epsilon \geq 0$.

Theorem 2.2.17 *Let $(\mathbf{u}^k)_{k=0}^N$ be the sequence of value vectors generated by a call* ValueIteration$(\mathbf{u}^0, \epsilon)$*, for some $\epsilon > 0$. Let $\mathbf{v}^*$ be the optimal value vector. Then:*

(i) For every $0 \leq k \leq N$ we have

$$\|\mathbf{v}^* - \mathbf{u}^k\|_\infty \;\leq\; \gamma^k \, \|\mathbf{v}^* - \mathbf{u}^0\|_\infty \; .$$

(ii) $\|\mathbf{v}^* - \mathbf{u}^N\|_\infty \leq \frac{\epsilon}{2}$.

(iii) The policy $\pi = \mathcal{P}\mathbf{u}^N$ is ϵ-optimal.

(iv) $N \leq \frac{1}{1-\gamma} \log \frac{2\|\mathbf{v}^* - \mathbf{u}^0\|_\infty}{\epsilon}$.

Proof: We first prove (i). By Lemma 2.2.4 and the fact that $\mathcal{T}\mathbf{v}^* = \mathbf{v}^*$, we have

$$\|\mathbf{v}^* - \mathbf{u}^k\|_\infty \;=\; \|\mathcal{T}\mathbf{v}^* - \mathcal{T}\mathbf{u}^{k-1}\|_\infty \;\leq\; \gamma\,\|\mathbf{v}^* - \mathbf{u}^{k-1}\|_\infty\,.$$

The claim follows easily by induction.

Next, we prove (ii). Since, $\mathcal{T}^k\mathbf{u}^N \to \mathbf{v}^*$ for $k \to \infty$, it follows from the triangle inequality, repeated use of Lemma 2.2.4, and finally the termination criterion that:

$$\begin{aligned}
\|\mathbf{v}^* - \mathbf{u}^N\|_\infty \;&\leq\; \gamma\|\mathbf{v}^* - \mathbf{u}^{N-1}\|_\infty \\
&\leq\; \gamma\sum_{k=0}^{\infty}\|\mathbf{u}^{N+k} - \mathbf{u}^{N+k-1}\|_\infty \\
&=\; \gamma\sum_{k=0}^{\infty}\|\mathcal{T}^k\mathbf{u}^N - \mathcal{T}^k\mathbf{u}^{N-1}\|_\infty \\
&\leq\; \gamma\sum_{k=0}^{\infty}\gamma^k\|\mathbf{u}^N - \mathbf{u}^{N-1}\|_\infty \\
&=\; \frac{\gamma}{1-\gamma}\|\mathbf{u}^N - \mathbf{u}^{N-1}\|_\infty \;\leq\; \frac{\epsilon}{2}\,.
\end{aligned}$$

Let $\pi = \mathcal{P}\mathbf{u}^N$. To prove (iii) we first observe that, by the triangle inequality:

$$\|\mathbf{v}^* - \mathbf{v}^\pi\|_\infty \leq \|\mathbf{v}^* - \mathbf{v}^N\|_\infty + \|\mathbf{v}^N - \mathbf{v}^\pi\|_\infty\,.$$

Using (ii) it, thus, suffices to show that $\|\mathbf{v}^N - \mathbf{v}^\pi\|_\infty \leq \epsilon/2$. Next, observe that:

$$\begin{aligned}
\|\mathbf{v}^N - \mathbf{v}^\pi\|_\infty &\leq \|\mathbf{v}^N - \mathcal{T}_\pi\mathbf{v}^N\|_\infty + \|\mathcal{T}_\pi\mathbf{v}^N - \mathbf{v}^\pi\|_\infty \\
&= \|\mathcal{T}\mathbf{v}^{N-1} - \mathcal{T}\mathbf{v}^N\|_\infty + \|\mathcal{T}_\pi\mathbf{v}^N - \mathcal{T}_\pi\mathbf{v}^\pi\|_\infty \\
&\leq \gamma\|\mathbf{v}^{N-1} - \mathbf{v}^N\|_\infty + \gamma\|\mathbf{v}^N - \mathbf{v}^\pi\|_\infty\,.
\end{aligned}$$

The first step follows from the triangle inequality. The second step follows from the fact that $\mathbf{v}^N = \mathcal{T}\mathbf{v}^{N-1}$, $\mathcal{T}_\pi\mathbf{v}^N = \mathcal{T}\mathbf{v}^N$, and $\mathbf{v}^\pi = \mathcal{T}_\pi\mathbf{v}^\pi$. The third step follows from Lemma 2.2.4. By rearranging the expression and using the termination criterion we get:

$$\|\mathbf{v}^N - \mathbf{v}^\pi\|_\infty \;\leq\; \frac{\gamma}{1-\gamma}\|\mathbf{v}^{N-1} - \mathbf{v}^N\|_\infty \;\leq\; \frac{\epsilon}{2}\,.$$

Finally, we prove (iv). Let $K = \frac{2\|\mathbf{v}^* - \mathbf{u}^0\|_\infty}{\epsilon(1-\gamma)}$, and assume for the sake of contradiction that $K < N$. Using (i) and the termination criterion we get:

$$\frac{\epsilon}{2} \;<\; \|\mathbf{v}^* - \mathbf{u}^K\|_\infty \;\leq\; \gamma^K \|\mathbf{v}^* - \mathbf{u}^0\|_\infty \,.$$

By rewriting this inequality, and using that $\log_{1/\gamma} x = \frac{\log x}{\log 1/\gamma} \leq \frac{\log x}{1-\gamma}$, we get a contradiction:

$$K \;<\; \log_{1/\gamma} \frac{2\|\mathbf{v}^* - \mathbf{u}^0\|_\infty}{\epsilon} \;\leq\; \frac{1}{1-\gamma} \log \frac{2\|\mathbf{v}^* - \mathbf{u}^0\|_\infty}{\epsilon} \,.$$

□

For every MDP M, let $L(M) = L(\mathbf{P}, \mathbf{c}, \mathbf{J}, \gamma)$ be the number of bits needed to describe M. More precisely, for every number $a = p/q$, where $p \in \mathbb{Z}$ and $q \in \mathbb{N}$ are relatively prime, we use $1 + \lceil \log_2(|p|+1) \rceil + \lceil \log_2(q+1) \rceil$ bits. Using Cramer's rule and simple bounds on the size of determinants, one can prove that the number of bits needed to describe a component of the value vector $\mathbf{v}^\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{c}_\pi$, for some policy π, is at most $4L(M)$. For details see, e.g., Schrijver [108]. We get:

Lemma 2.2.18 *Let π and π' be two policies such that $\mathbf{v}^\pi \neq \mathbf{v}^{\pi'}$. Then $\|\mathbf{v}^\pi - \mathbf{v}^{\pi'}\|_\infty \leq 2^{-4L(M)}$.*

Corollary 2.2.19 *If π is ϵ-optimal for $0 \leq \epsilon < 2^{-4L(M)-1}$, then π is optimal.*

From Definition 2.1.5 it easily follows that, for every policy π and state i, $|(\mathbf{v}^\pi)_i| \leq \mathbf{c}_{\max}/(1-\gamma) \leq (2^{L(M)})/(1-\gamma)$, where $\mathbf{c}_{\max} = \max_{a \in A} |\mathbf{c}_a|$. Hence, $\|\mathbf{v}^* - \mathbf{0}\|_\infty \leq (2^{L(M)})/(1-\gamma)$. Combining Theorem 2.2.17 (iv) and Corollary 2.2.19 we then get:

Theorem 2.2.20 *Let $\mathbf{u}^N = \text{VALUEITERATION}(\mathbf{0}, 2^{-4L(M)-2})$, and $\pi = \mathcal{P}\mathbf{u}^N$. Then π is optimal, and the number of iterations is at most*

$$N = O\left(\frac{L(M) \log \frac{1}{1-\gamma}}{1-\gamma} \right) \,.$$

Function `PolicyIteration`(π^0)

$k \leftarrow 0$;
while $\exists B \subseteq A : B$ *is improving w.r.t.* π^k **do**
  $\pi^{k+1} \leftarrow \pi^k[B]$;
  $k \leftarrow k + 1$;
return π^k;

Figure 2.3: The PolicyIteration algorithm.

2.2.2 The PolicyIteration algorithm

The PolicyIteration algorithm is the most widely used algorithm for solving MDPs in practice. It was introduced by Howard [63] in 1960. The PolicyIteration algorithm is given in Figure 2.3. It starts with some initial policy π^0 and generates an improving sequence $\pi^0, \pi^1, \ldots, \pi^N$ of policies, ending with an optimal policy π^N. In each iteration the algorithm *evaluates* the current policy π^k and computes the value vector $\mathbf{v}^k = \mathbf{v}^{\pi^k}$ by solving a system of linear equations. The next policy π^{k+1} is obtained from π^k by performing a non-empty set of improving switches $B \subseteq A$ with respect to π^k, such that $\pi^{k+1} = \pi^k[B]$.

It follows from Theorem 2.2.12 (i) that the value vectors strictly improve with each iteration, $\mathbf{v}^{k+1} > \mathbf{v}^k$. Hence, the number of iterations is bounded by the number of policies. Moreover, since there are no improving switches with respect to the final policy π^N, we know from Theorem 2.2.12 (ii) that π^N is optimal. We get:

Theorem 2.2.21 *For every initial policy* π^0, PolicyIteration(π^0) *terminates after a finite number of iterations. If* $(\mathbf{v}^k)_{k=0}^N$ *is the sequence of value vectors generated by the call, then* $\mathbf{v}^{k-1} < \mathbf{v}^k \leq \mathbf{v}^*$, *for every* $1 \leq k < N$. *Furthermore,* $\mathbf{v}^N = \mathbf{v}^*$ *and* π^N *is an optimal policy.*

The set of improving switches that is performed is decided by an *improvement rule*. There is, in fact, a whole family of PolicyIteration algorithms using different improvement rules. Note that during every iteration the current values are computed. This is done by solving a system of linear equations, which can, for instance, be done using Gaussian elimination with $O(n^3)$ arithmetic operations. The most natural variant is, perhaps, the one in which the algorithm selects the best improving switch from each vertex

and performs all these switches simultaneously. This was the original improvement rule suggested by Howard [63], and we will refer to the algorithm obtained by using this improvement rule as Howard's PolicyIteration algorithm. To be precise, the improvement rule updates the policy by choosing the action from each state with the largest reduced cost. I.e., $\pi^{k+1} = \mathcal{P}\mathbf{v}^k$.

For the remainder of this section we focus on Howard's PolicyIteration algorithm. Let π^0 be some initial policy. We next relate the sequences of value vectors obtained by running PolicyIteration(π^0), with Howard's improvement rule, and ValueIteration($\mathbf{v}^{\pi^0}, \epsilon$). The following lemmas appear, e.g., in Meister and Holzbaur [92].

Lemma 2.2.22 *Let $\pi^0 \in \Pi$ and $\pi^1 = \mathcal{P}\mathbf{v}^{\pi^0}$. Then $\mathcal{T}\mathbf{v}^{\pi^0} \leq \mathbf{v}^{\pi^1}$.*

Proof: Recall that $\mathcal{T}_{\pi^0}\mathbf{v}^{\pi^0} = \mathbf{v}^{\pi^0}$, implying that $\mathcal{T}\mathbf{v}^{\pi^0} \geq \mathbf{v}^{\pi^0}$. From Lemma 2.2.14 we get that $\mathcal{T}\mathbf{v}^{\pi^0} = \mathcal{T}_{\pi^1}\mathbf{v}^{\pi^0}$. It follows that $\mathcal{T}_{\pi^1}\mathbf{v}^{\pi^0} \geq \mathbf{v}^{\pi^0}$, and we get from Corollary 2.2.7 that $\mathcal{T}\mathbf{v}^{\pi^0} \leq \mathbf{v}^{\pi^1}$. □

In the following we will use $\mathbf{v}^k$ as short-hand notation for $\mathbf{v}^{\pi^k}$. Using Lemma 2.2.22, we immediately get:

Lemma 2.2.23 *Let π^0 be any policy. Let $(\mathbf{v}^k)_{k=0}^N$ and $(\mathbf{u}^k)_{k=0}^\infty$ be the value vectors generated by* PolicyIteration(σ^0)*, using Howard's improvement rule, and* ValueIteration($\mathbf{v}^{\pi^0}, 0$)*, respectively. Then, $\mathbf{v}^k \geq \mathbf{u}^k$, for every $0 \leq k \leq N$.*

Proof: We prove the lemma by induction. We have $\mathbf{v}^0 = \mathbf{u}^0$. Suppose now that $\mathbf{v}^k \geq \mathbf{u}^k$. Then, by Lemma 2.2.22 and the monotonicity of the value iteration operator, we have:

$$\mathbf{v}^{k+1} \;\geq\; \mathcal{T}\mathbf{v}^k \;\geq\; \mathcal{T}\mathbf{u}^k \;=\; \mathbf{u}^{k+1}.$$

□

Combining Theorem 2.2.17 and Lemma 2.2.23, we get:

Lemma 2.2.24 *Let $(\mathbf{v}^k)_{k=0}^N$ be the sequence of value vectors generated by the call* PolicyIteration(π^0)*, for some $\pi^0 \in \Pi$. Let $\mathbf{v}^*$ be the optimal value vector. Then, for every $0 \leq k \leq N$ we have*

$$\|\mathbf{v}^* - \mathbf{v}^k\|_\infty \;\leq\; \gamma^k \,\|\mathbf{v}^* - \mathbf{v}^0\|_\infty.$$

Combining Lemma 2.2.24 with Theorem 2.2.20 we get the following corollary. In Theorem 2.2.20 the ValueIteration algorithm is initialized with the zero vector. Using a similar argument to the one given before Theorem 2.2.20 it is not difficult to see, however, that the value vector of any policy $\mathbf{v}^{\pi^0}$ works as well. I.e., $\|\mathbf{v}^* - \mathbf{v}^{\pi^0}\|_\infty \leq (2^{L(M)+1})/(1-\gamma)$, for any policy π^0.

Corollary 2.2.25 *Starting with any policy π^0, the number of iterations performed by* PolicyIteration(π^0)*, with Howard's improvement rule, is at most:*

$$N = O\left(\frac{L(M)\log\frac{1}{1-\gamma}}{1-\gamma}\right) .$$

2.2.3 Linear programming formulation

Linear programming formulations for MDPs date back to the work of Manne [84], De Ghellinck [28], and d'Epenoux [29]. In this section we show how a discounted MDP can be formulated as a linear program. The linear program we present is due to d'Epenoux [29]. Although MDPs can be solved in polynomial time using linear programming, the preferred way of solving MDPs in practice is with the PolicyIteration algorithm.

Before introducing a linear programming formulation of discounted MDPs it is helpful to first make another definition. We let $\mathbf{e} = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$ be an all one vector.

Definition 2.2.26 (Flux vectors) *For every policy π, define $\mathbf{x}^\pi \in \mathbb{R}^n$ as follows*

$$\mathbf{x}^\pi \;=\; (\mathbf{e}^T(\mathbf{I} - \gamma\mathbf{P}_\pi)^{-1})^T .$$

It will sometimes be convenient to extend $\mathbf{x}^\pi$ to m coordinates by letting additional coordinates be zero. We therefore define $\bar{\mathbf{x}}^\pi \in \mathbb{R}^m$ by:

$$(\bar{\mathbf{x}}^\pi)_a = \begin{cases} (\mathbf{x}^\pi)_i & \text{if } a = \pi(i) \text{ for some } i \in S \\ 0 & \text{otherwise} \end{cases}$$

Using Lemma 2.2.1, it is not difficult to see that for every $i \in S$, we have

$$(\mathbf{x}^\pi)_i \;=\; \sum_{k=0}^{\infty} \mathbf{e}^T(\gamma P_\pi)^k \mathbf{e}_i \;=\; \sum_{j\in S}\sum_{k=0}^{\infty} \mathbf{e}_j^T(\gamma P_\pi)^k \mathbf{e}_i .$$

Using the interpretation of $1-\gamma$ as a stopping probability, we thus get that $(\mathbf{x}^\pi)_i$ is the expected total number of times of passing through state i, when

the controller uses the policy π, and where each state is used, in turn, as the starting state. Recall that $X_\pi^k(j)$ is the random variable corresponding to the state after k steps when the MDP starts in state j and the controller uses policy π. We then have:

$$(\mathbf{x}^\pi)_i \;=\; \sum_{j\in S}\sum_{k=0}^{\infty}\gamma^k \Pr\left[X_\pi^k(j)=i\right] \;.$$

Note that since an optimal policy maximizes the value of every state, it also maximizes the sum of the values of the states. Flux vectors provide an alternative way of summing the values of all the states:

Lemma 2.2.27 *For every policy π, we have*

$$\mathbf{e}^T\mathbf{v}^\pi = (\mathbf{c}_\pi)^T\mathbf{x}^\pi \;.$$

Proof: By Definition 2.2.2 and then Definition 2.2.26, we get

$$\mathbf{e}^T\mathbf{v}^\pi \;=\; \mathbf{e}^T(\mathbf{I}-\gamma\mathbf{P}_\pi)^{-1}\mathbf{c}_\pi \;=\; (\mathbf{x}^\pi)^T\mathbf{c}_\pi \;=\; (\mathbf{c}_\pi)^T\mathbf{x}^\pi \;.$$

□

It is also useful to make the following simple observations.

Lemma 2.2.28 *For every policy π, we have $\mathbf{x}^\pi \geq \mathbf{e}$.*

Proof: By Definition 2.2.26, Lemma 2.2.1, and the fact that $\mathbf{e}^T(\mathbf{P}_\pi)^k \geq \mathbf{0}^T$, for every $k \geq 0$, we get:

$$(\mathbf{x}^\pi)^T \;=\; \mathbf{e}^T(\mathbf{I}-\gamma\mathbf{P}_\pi)^{-1} \;=\; \sum_{k=0}^{\infty}\mathbf{e}^T(\gamma\mathbf{P}_\pi)^k \geq \mathbf{e}^T \;.$$

□

Note that Lemma 2.2.28 implies that $\bar{\mathbf{x}}^\pi \neq \bar{\mathbf{x}}^{\pi'}$ for $\pi \neq \pi'$. Consider the following pair of primal (P) and dual (D) linear programs. Let us note that, in the literature, sometimes (D) is referred to as the primal and (P) as the dual. We let (P) be the primal because this provides the most natural interpretation of the simplex method for the results presented in chapters 4 and 5.

$$\begin{array}{llrcl} & \max & \mathbf{c}^T\mathbf{x} & & \\ (P) & s.t. & (\mathbf{J}-\gamma\mathbf{P})^T\mathbf{x} & = & \mathbf{e} \\ & & \mathbf{x} & \geq & \mathbf{0} \end{array} \qquad \begin{array}{llrcl} & \min & \mathbf{e}^T\mathbf{y} & & \\ (D) & s.t. & (\mathbf{J}-\gamma\mathbf{P})\mathbf{y} & \geq & \mathbf{c} \end{array}$$

Lemma 2.2.29 *Let $\mathbf{x} \in \mathbb{R}^m$ be any feasible solution for (P), then for all $i \in S$ there exists $a \in A_i$ such that $\mathbf{x}_a > 0$.*

Proof: The equality constraints of (P) can also be stated as:

$$\forall i \in S: \quad \sum_{a \in A_i} \mathbf{x}_a = 1 + \sum_{b \in A} \gamma \mathbf{P}_{b,i} \mathbf{x}_b$$

Since $\mathbf{x}_a \geq 0$ for all $a \in A$, the right-hand-side is positive. Then there must exist for all states $i \in S$ an action $a \in A_i$ such that $\mathbf{x}_a > 0$. □

Lemma 2.2.30 *For every policy π, the vector $\bar{\mathbf{x}}^\pi \in \mathbb{R}^m$ is a basic feasible solution of (P) with basis π. There are no other basic feasible solutions of (P). I.e., there is a one-to-one correspondence between policies and basic feasible solutions of (P).*

Proof: Observe first that, due to Lemma 2.2.1, π is, indeed, a basis for (P). From Definition 2.2.26 we see that:

$$(\mathbf{I} - \gamma \mathbf{P}_\pi)^T \mathbf{x}^\pi \;=\; \mathbf{e} \quad \Rightarrow \quad (\mathbf{J} - \gamma \mathbf{P})^T \bar{\mathbf{x}}^\pi \;=\; \mathbf{e}\,,$$

and from Lemma 2.2.28 we know that $\bar{\mathbf{x}}^\pi \geq \mathbf{0}$. Hence, $\bar{\mathbf{x}}^\pi$ is a basic feasible solution for (P).

Next, we show that every basic feasible solution $\mathbf{x}$ for (P) is equal to $\bar{\mathbf{x}}^\pi$ for some policy π. From Lemma 2.2.29 we know that for all $i \in S$ there exists $a \in A_i$ such that $\mathbf{x}_a > 0$. Furthermore, since $\mathbf{x}$ is a basic feasible solution there are at most n positive variables. It follows that the basis $B = \{a \in A \mid \mathbf{x}_a > 0\}$ for $\mathbf{x}$ can be interpreted as a policy π, and from the definition of $\bar{\mathbf{x}}^\pi$ that $\mathbf{x} = \bar{\mathbf{x}}^B = \bar{\mathbf{x}}^\pi$. □

Note that the definition of reduced costs for MDPs, Definition 2.2.8, and the definition of reduced costs for linear programs, Definition 1.2.1, are exactly the same for the primal linear program (P). I.e., for $B = \pi$ and $\mathbf{A} = (\mathbf{J}\text{–}\gamma\mathbf{P})$ we have $\mathbf{A}_B^{-1}\mathbf{c}_B = (\mathbf{I}\text{–}\gamma\mathbf{P}_\pi)^{-1}\mathbf{c}_\pi = \mathbf{v}^\pi$. It follows that every pivoting rule for the simplex method corresponds directly to an improvement rule for the PolicyIteration algorithm. Similarly, every improvement rule for the PolicyIteration algorithm that only performs a single improving switch in every iteration corresponds directly to a pivoting rule for the simplex method. Hence, the PolicyIteration algorithm is a generalization of the simplex method for MDPs. In particular, Howard's PolicyIteration algorithm may be viewed as a parallel version of the simplex algorithm in which several pivoting steps are performed simultaneously.

Theorem 2.2.31 *For every MDP we have:*

(*i*) *The linear programs* (P) *and* (D) *are both feasible.*

(*ii*) *Let* $\mathbf{x}^*$ *be any optimal solution of* (P)*. Then every policy* $\pi \subseteq \{a \in A \mid \mathbf{x}^*_a > 0\}$ *is optimal. Furthermore, there is always at least one policy with this property.*

(*iii*) *Every optimal solution* $\mathbf{y}^*$ *of* (D) *is equal to the optimal value vector* $\mathbf{v}^*$*.*

Proof: We first observe that (P) and (D) are both feasible. The feasibility of (P) follows from Lemma 2.2.30. Let π be an optimal policy. Then we know from Theorem 2.2.12 (ii) that there are no improving switches with respect to π, such that $\bar{\mathbf{c}}^\pi = \mathbf{c} - (\mathbf{J} - \gamma\mathbf{P})\mathbf{v}^\pi \leq \mathbf{0}$. It follows that $\mathbf{v}^\pi$ is a feasible solution for (D).

From the complementary slackness theorem, Theorem 1.3.4, we know that two vectors $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^n$ are optimal for (P) and (D), respectively, if and only if:

$$(\mathbf{J} - \gamma\mathbf{P})^T\mathbf{x} = \mathbf{e} \tag{2.1}$$

$$\mathbf{x} \geq \mathbf{0} \tag{2.2}$$

$$(\mathbf{J} - \gamma\mathbf{P})\mathbf{y} \geq \mathbf{c} \tag{2.3}$$

$$((\mathbf{J} - \gamma\mathbf{P})\mathbf{y} - \mathbf{c})^T\mathbf{x} = 0 \tag{2.4}$$

Equations (2.1) and (2.2) imply that $\mathbf{x}$ is a feasible solution to (P). Lemma 2.2.29 then says that for all $i \in S$ there exists $a \in A_i$ such that $\mathbf{x}_a > 0$. Let $\pi \subseteq \{a \in A \mid \mathbf{x}_a > 0\}$ be any policy obtained by picking one such action from each state. Note that such a policy always exists. It now follows from equation 2.4 that:

$$((\mathbf{J}_\pi - \gamma\mathbf{P}_\pi)\mathbf{y} - \mathbf{c}_\pi)^T\mathbf{x}_\pi = 0 \quad \Rightarrow \quad \mathbf{y} = (\mathbf{I} - \gamma\mathbf{P}_\pi)^{-1}\mathbf{c}_\pi \ .$$

Hence, $\mathbf{y} = \mathbf{v}^\pi$. From equation (2.3) we then get that:

$$\mathbf{0} \geq \mathbf{c} - (\mathbf{J} - \gamma\mathbf{P})\mathbf{v}^\pi = \bar{\mathbf{c}}^\pi \ .$$

I.e., there are no improving switches with respect to π, and it follows from Theorem 2.2.12 (ii) that π is optimal, and then that $\mathbf{y} = \mathbf{v}^\pi = \mathbf{v}^*$. □

2.2.4 Strongly polynomial bound for fixed discount

As mentioned in Section 2.2.2, Meister and Holzbaur [92] established, decades ago, that the number of iterations performed by Howard's PolicyIteration algorithm, when the discount factor is fixed, is polynomially bounded in the *bit size* of the input. Their bound, however, is not *strongly polynomial*, as it does not depend solely on the number of states and actions of the MDP. The first strongly polynomial time algorithm for solving MDPs with a fixed discount factor was an interior point algorithm of Ye [120], which solves the problem using $O(n^4 \log \frac{n}{1-\gamma})$ arithmetic operations.

Very recently, Ye [121] presented a surprisingly simple proof that Howard's PolicyIteration algorithm terminates after at most $O\big(\frac{mn}{1-\gamma} \log \frac{n}{1-\gamma}\big)$ iterations, where n is the number of states, m is the total number of actions, and $0 < \gamma < 1$ is the discount factor. In particular, when the discount factor is constant, the number of iterations is $O(mn \log n)$. Since each iteration only involves solving a system of linear equations, Ye's result established for the first time that Howard's PolicyIteration algorithm is a *strongly polynomial* time algorithm, when the discount factor is constant. Ye's proof is based on a careful analysis of an LP formulation of the MDP problem, with LP duality and complementary slackness playing crucial roles.

In Hansen, Miltersen, and Zwick [59] we significantly improve and extend Ye's [121] analysis. We show that Howard's PolicyIteration algorithm actually terminates after at most $O\big(\frac{m}{1-\gamma} \log \frac{n}{1-\gamma}\big)$ iterations, improving Ye's bound by a factor of n. Interestingly, the only added ingredient needed to obtain this significant improvement is a well-known relationship between Howard's PolicyIteration algorithm and Bellman's [7] ValueIteration algorithm. The connection between the two algorithms was presented in Section 2.2.2, specifically lemmas 2.2.23 and 2.2.24. We also extended the result to 2-player turn-based stochastic games, a generalization of MDPs, and showed that the same bound holds for the StrategyIteration algorithm, a generalization of PolicyIteration, in this setting. This extension of the result is presented in Section 3.4.2.

In this section present the results of Ye [121] and Hansen, Miltersen, and Zwick [59]. The presentation will follow Hansen, Miltersen, and Zwick [59]. We start by deriving a few technical lemmas.

Lemma 2.2.32 *For every policy π, we have*

$$\mathbf{e}^T \mathbf{x}^\pi = \frac{n}{1-\gamma} .$$

Proof: By Definition 2.2.26, Lemma 2.2.1, and the fact that $\mathbf{e}^T(\mathbf{P}_\pi)^k\mathbf{e} = n$, for every $k \geq 0$, we have:

$$(\mathbf{x}^\pi)^T\mathbf{e} \;=\; \mathbf{e}^T(\mathbf{I}-\gamma\mathbf{P}_\pi)^{-1}\mathbf{e} \;=\; \sum_{k=0}^{\infty}\mathbf{e}^T(\gamma\mathbf{P}_\pi)^k\mathbf{e} \;=\; n\sum_{k=0}^{\infty}\gamma^k \;=\; \frac{n}{1-\gamma}\;.$$

□

Lemma 2.2.33 *For every two policies π, π', we have*

$$\mathbf{e}^T(\mathbf{v}^{\pi'}-\mathbf{v}^\pi) \;=\; (\bar{\mathbf{c}}^\pi)^T_{\pi'}\mathbf{x}^{\pi'}\;.$$

Proof: By Lemma 2.2.10 and then Definition 2.2.32, we have:

$$\mathbf{e}^T(\mathbf{v}^{\pi'}-\mathbf{v}^\pi) \;=\; \mathbf{e}^T(\mathbf{I}-\gamma\mathbf{P}_{\pi'})^{-1}(\bar{\mathbf{c}}^\pi)_{\pi'} \;=\; (\mathbf{x}^{\pi'})^T(\bar{\mathbf{c}}^\pi)_{\pi'} \;=\; (\bar{\mathbf{c}}^\pi)^T_{\pi'}\mathbf{x}^{\pi'}\;.$$

□

Note that Lemma 2.2.33 corresponds exactly to Lemma 1.2.2 for linear programming. I.e., from Lemma 2.2.27 we know that $\mathbf{e}^T\mathbf{v}^\pi = (\mathbf{c}_\pi)^T\mathbf{x}^\pi$ and $\mathbf{e}^T\mathbf{v}^{\pi'} = (\mathbf{c}_{\pi'})^T\mathbf{x}^{\pi'}$, and when this change is made the two lemmas become the same.

Recall that by Theorem 2.2.12, we have $\bar{\mathbf{c}}^{\pi^*} \leq \mathbf{0}$ for every optimal policy π^*. Hence, it is natural to operate with negative reduced costs.

Lemma 2.2.34 *Let π', π be two policies such that $\mathbf{v}^{\pi'} \leq \mathbf{v}^\pi$ and let $a = \pi'(i)$ where $i \in S$. Then,*

$$(\mathbf{v}^\pi - \mathbf{v}^{\pi'})_i \;\geq\; -(\bar{\mathbf{c}}^\pi)_a\;.$$

Proof: By Definition 2.2.2 and Definition 2.2.8 we have:

$$(\mathbf{v}^\pi)_i - (\mathbf{v}^{\pi'})_i \;=\; (\mathbf{v}^\pi)_i - (\mathbf{c}_a + \gamma\mathbf{P}_a\mathbf{v}^{\pi'}) \;\geq$$

$$-\mathbf{c}_a - \gamma\mathbf{P}_a\mathbf{v}^\pi + (\mathbf{v}^\pi)_i \;=\; -(\bar{\mathbf{c}}^\pi)_a\;.$$

□

Lemma 2.2.35 *Let π'', π be two policies such that $\mathbf{v}^{\pi''} \leq \mathbf{v}^\pi$ and let $a =$ argmin $_{a\in\pi''}(\bar{\mathbf{c}}^\pi)_a$. Then,*

$$\|\mathbf{v}^\pi - \mathbf{v}^{\pi''}\|_\infty \;\leq\; -\frac{n}{1-\gamma}(\bar{\mathbf{c}}^\pi)_a\;.$$

Proof: As $\mathbf{v}^{\pi''} \leq \mathbf{v}^{\pi}$, we get using Lemma 2.2.33 and then Lemma 2.2.32 that

$$\begin{aligned} \|\mathbf{v}^{\pi} - \mathbf{v}^{\pi''}\|_{\infty} &\leq \mathbf{e}^T(\mathbf{v}^{\pi} - \mathbf{v}^{\pi''}) = -(\bar{\mathbf{c}}^{\pi})^T_{\pi''}\mathbf{x}^{\pi''} \\ &\leq -(\bar{\mathbf{c}}^{\pi})_a \, \mathbf{e}^T\mathbf{x}^{\pi''} = -\frac{n}{1-\gamma}(\bar{\mathbf{c}}^{\pi})_a \, . \end{aligned}$$

□

Lemma 2.2.36 *Let π'', π', π be three policies such that $\mathbf{v}^{\pi''} \leq \mathbf{v}^{\pi'} \leq \mathbf{v}^{\pi}$. Let $a = \text{argmin}_{a \in \pi''}(\bar{\mathbf{c}}^{\pi})_a$ and suppose that $a \in \pi'$. Then,*

$$\|\mathbf{v}^{\pi} - \mathbf{v}^{\pi'}\|_{\infty} \geq \frac{1-\gamma}{n} \|\mathbf{v}^{\pi} - \mathbf{v}^{\pi''}\|_{\infty} \, .$$

Proof: Let $i \in S$ be the state for which $\pi''(i) = \pi'(i) = a$. By Lemma 2.2.34 and Lemma 2.2.35 we get

$$\begin{aligned} \|\mathbf{v}^{\pi} - \mathbf{v}^{\pi'}\|_{\infty} &\geq (\mathbf{v}^{\pi} - \mathbf{v}^{\pi'})_i \geq -(\bar{\mathbf{c}}^{\pi})_a \\ &\geq \frac{1-\gamma}{n} \|\mathbf{v}^{\pi} - \mathbf{v}^{\pi''}\|_{\infty} \, . \end{aligned}$$

□

Let us note that Ye [121] proved a variant of Lemma 2.2.36 for which the third policy π is an optimal policy. This weaker formulation of the lemma suffices for the proof that we are about to present. For the extension to 2-player turn-based stochastic games presented in Section 3.4.2, we need the stronger lemma, however.

We next present the main lemma needed to obtain a strongly polynomial bound for a fixed discount factor. The lemma shows that an action is implicitly eliminated by the PolicyIteration algorithm after a certain number of iterations.

Lemma 2.2.37 *Let $(\pi^k)_{k=0}^N$ be the sequence of policies generated by the* PolicyIteration *algorithm, starting from a policy π^0. Let $L = \log_{1/\gamma} \frac{n}{1-\gamma}$. Then, every policy π^k contains an action that does not appear in any policy π^ℓ, where $k + L < \ell \leq N$.*

Proof: By the correctness of the PolicyIteration algorithm, $\pi^* = \pi^N$ is an optimal policy. Let $a = \text{argmin}_{a \in \pi^k}(\bar{\mathbf{c}}^{\pi^*})_a$.

Suppose, for the sake of contradiction, that $a \in \pi^\ell$, for some $k + L < \ell \leq N$. From Theorem 2.2.21 we know that $\mathbf{v}^{\pi^k} \leq \mathbf{v}^{\pi^\ell} \leq \mathbf{v}^{\pi^*}$. Using Lemma 2.2.36, with $\pi'' = \pi^k$, $\pi' = \pi^\ell$ and $\pi = \pi^*$, we get that

$$\|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^\ell}\|_\infty \;\geq\; \frac{1-\gamma}{n} \|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^k}\|_\infty \,.$$

On the other hand, using Lemma 2.2.24, we get that

$$\|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^\ell}\|_\infty \;\leq\; \gamma^{\ell-k} \|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^k}\|_\infty \,.$$

Combining the two inequalities we get

$$\gamma^L \;>\; \gamma^{\ell-k} \;\geq\; \frac{1-\gamma}{n} \,,$$

a contradiction. □

Theorem 2.2.38 *The* PolicyIteration *algorithm, starting from any initial policy, terminates with an optimal policy after at most* $(m+1)(1 + \log_{1/\gamma} \frac{n}{1-\gamma}) = O(\frac{m}{1-\gamma} \log \frac{n}{1-\gamma})$ *iterations.*

Proof: Let $\bar{L} = \lfloor 1 + \log_{1/\gamma} \frac{n}{1-\gamma} \rfloor$. Consider policies $\pi^0, \pi^{\bar{L}}, \pi^{2\bar{L}}, \ldots$. By Lemma 2.2.37, every policy in this subsequence contains a new action that would never be used again. As there are only m actions, the total number of policies in the sequence is at most $(m+1)\bar{L} = (m+1)(1 + \log_{1/\gamma} \frac{n}{1-\gamma})$. Finally, note that $\log_{1/\gamma} x = \frac{\log x}{\log 1/\gamma} \leq \frac{\log x}{1-\gamma}$. □

Ye [121] used essentially the following lemma for his bound instead of Lemma 2.2.24.

Lemma 2.2.39 *Let* $\mathbf{v}^*$ *be the optimal value vector, let* π *be a policy, let* $a = \text{argmax}_{a \in A} (\bar{\mathbf{c}}^\pi)_a$*, and let* B *be an improving set with respect to* π *such that* $a \in B$*. Then:*

$$\mathbf{e}^T \mathbf{v}^* - \mathbf{e}^T \mathbf{v}^{\pi[B]} \;\leq\; \left(1 - \frac{1-\gamma}{n}\right) (\mathbf{e}^T \mathbf{v}^* - \mathbf{e}^T \mathbf{v}^\pi) \,.$$

Proof: The following two inequalities are derived in a similar way to lemmas 2.2.34 and 2.2.35, respectively, but with the roles of the policies switched:

$$(\mathbf{v}^{\pi[B]} - \mathbf{v}^\pi)_i \;\geq\; (\bar{\mathbf{c}}^\pi)_a \,,$$
$$\mathbf{e}^T(\mathbf{v}^* - \mathbf{v}^\pi) \;\leq\; \frac{n}{1-\gamma} (\bar{\mathbf{c}}^\pi)_a \,.$$

Using $\mathbf{e}^T(\mathbf{v}^{\pi[B]} - \mathbf{v}^\pi) \geq (\mathbf{v}^{\pi[B]} - \mathbf{v}^\pi)_i$ and combining the two inequalities we get:

$$\mathbf{e}^T\mathbf{v}^* - \mathbf{e}^T\mathbf{v}^\pi \;\leq\; \frac{n}{1-\gamma}(\mathbf{e}^T\mathbf{v}^{\pi[B]} - \mathbf{e}^T\mathbf{v}^\pi) \quad \Rightarrow$$

$$\mathbf{e}^T\mathbf{v}^* - \mathbf{e}^T\mathbf{v}^{\pi[B]} \;\leq\; \left(1 - \frac{1-\gamma}{n}\right)(\mathbf{e}^T\mathbf{v}^* - \mathbf{e}^T\mathbf{v}^\pi)\,.$$

□

Repeated use of Lemma 2.2.39 also gives a bound for the convergence of the value vectors generated by Howard's PolicyIteration algorithm. When this bound is used in the proof of Lemma 2.2.37, we get that the number of iterations is at most $O\big(\frac{mn}{1-\gamma}\log\frac{n}{1-\gamma}\big)$. Note, however, that there is much larger freedom in the choice of the set B in Lemma 2.2.39. In particular, Ye [121] shows that the same bound holds for Dantzig's [24] LargestCoefficient pivoting rule for the simplex method when run on the linear program of the MDP. When viewed as a PolicyIteration algorithm this improvement rule repeatedly performs the single improving switch with largest reduced cost. Since in Lemma 2.2.39 we only measure the improvement made in each iteration, the bound also holds for the LargestIncrease pivoting rule.

Kitahara and Mizuno [77] extended the analysis of Ye [121] to general (non-degenerate, standard form) linear programs in the following way. Suppose the minimum and the maximum value of a basic variable for any basic feasible solution is δ and β, respectively. Then they showed that the LargestCoefficient and LargestIncrease pivoting rules perform at most $O(mn\frac{\beta}{\delta}\log(n\frac{\beta}{\delta}))$ pivoting steps, where n is the number of constraints and m is the number of variables. Note that for MDPs we have $\delta \geq 1$ and $\beta \leq \frac{n}{1-\gamma}$.

2.3 The average reward criterion

We next give an introduction to the average reward criterion. The section is largely based on Puterman [101]. Several of the proofs have been omitted, and we refer to Puterman [101] for additional details.

Recall that for the average reward criterion the value of a state i for a policy π is defined as:

$$\mathrm{val}^A_\pi(i) = \liminf_{N\to\infty} \frac{1}{N}\sum_{k=0}^{N-1} \mathbf{e}_i^T\mathbf{P}_\pi^k\mathbf{c}_\pi$$

The proof of the following lemma can be found in, e.g., Chung [18].

Lemma 2.3.1 *For every policy π, there exists a* limiting matrix $\mathbf{P}^*_\pi$ *such that:*

$$\mathbf{P}^*_\pi = \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} \mathbf{P}^k_\pi$$

Note that $\mathrm{val}^A_\pi(i) = \mathbf{e}_i^T \mathbf{P}^*_\pi \mathbf{c}_\pi$. In particular, when operating with policies there is no need to use $\liminf$.

Definition 2.3.2 (Values, potentials, valuations) *Let π be any policy, and let $\mathbf{g}^\pi \in \mathbb{R}^n$ and $\mathbf{h}^\pi \in \mathbb{R}^n$ be defined as:*

$$\begin{aligned} \mathbf{g}^\pi &= \mathbf{P}^*_\pi \mathbf{c}_\pi \\ \mathbf{h}^\pi &= \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N-1} \sum_{k=0}^{t} \mathbf{P}^k_\pi (\mathbf{c}_\pi - \mathbf{g}^\pi) \end{aligned}$$

We say that $\mathbf{g}^\pi$ and $\mathbf{h}^\pi$ are the vectors of values *and* potentials*, respectively. For every state $i \in S$, we let the pair $\nu^\pi_i = (\mathbf{g}^\pi_i, \mathbf{h}^\pi_i)$ be the* valuation *of state i, and $\nu^\pi = (\mathbf{g}^\pi, \mathbf{h}^\pi)$ be the corresponding vector of valuations.*

It can be shown that $(\mathbf{I} - \mathbf{P}_\pi + \mathbf{P}^*_\pi)$ is nonsingular, and that

$$\mathbf{h}^\pi = (\mathbf{I} - \mathbf{P}_\pi + \mathbf{P}^*_\pi)^{-1}(\mathbf{I} - \mathbf{P}^*_\pi)\mathbf{c}_\pi = (\mathbf{I} - \mathbf{P}_\pi)^\# \mathbf{c}_\pi \,, \tag{2.5}$$

where $(\mathbf{I} - \mathbf{P}_\pi)^\#$ is the Drazin inverse of $(\mathbf{I} - \mathbf{P}_\pi)$. I.e., the limit defining $\mathbf{h}^\pi$ always exists, and $\mathbf{h}^\pi$ is well-defined. For details see Puterman [101]. We state the following theorem without proof. For a proof we refer to Puterman [101]. It shows how $\mathbf{g}^\pi$ and $\mathbf{h}^\pi$ can be computed efficiently.

Theorem 2.3.3 *Let π be any policy. Suppose $\mathbf{g} \in \mathbb{R}^n$ and $\mathbf{h} \in \mathbb{R}^n$ satisfy the following equations:*

$$\mathbf{g} = \mathbf{P}_\pi \mathbf{g} \tag{2.6}$$

$$\mathbf{h} = \mathbf{c}_\pi - \mathbf{g} + \mathbf{P}_\pi \mathbf{h} \tag{2.7}$$

Then $\mathbf{g} = \mathbf{g}^\pi$ and $\mathbf{h} = \mathbf{h}^\pi + \mathbf{u}$ where $(\mathbf{I} - \mathbf{P})\mathbf{u} = \mathbf{0}$. Furthermore, let $\mathcal{R}^\pi \subseteq 2^S$ be the set of recurrent classes of $\mathbf{P}_\pi$. If

$$\forall R \in \mathcal{R}^\pi : \quad \sum_{i\in R} \mathbf{h}_i = 0 \,, \tag{2.8}$$

then $\mathbf{h} = \mathbf{h}^\pi$. Also, $\mathbf{g}^\pi$ and $\mathbf{h}^\pi$ satisfy (2.6), (2.7), and (2.8).

When $\mathbf{P}_\pi$ is aperiodic we get $\mathbf{P}^*_\pi = \lim_{k\to\infty} \mathbf{P}^k_\pi$, and $\mathbf{h}^\pi = \sum_{k=0}^{\infty} \mathbf{P}^k_\pi(\mathbf{c}_\pi - \mathbf{g}^\pi)$. Note that $\mathbf{g}^\pi_i = \mathbf{e}_i^T \mathbf{P}^*_\pi \mathbf{c}_\pi$. If $\mathbf{P}_\pi$ is aperiodic then $(\mathbf{e}_i^T \mathbf{P}^*_\pi)$ is the limiting, stationary distribution reached when starting in state i. Hence, in this case $\mathbf{h}^\pi_i$ can be interpreted as the expected total difference in reward between starting in state i and starting according to $(\mathbf{e}_i^T \mathbf{P}^*_\pi)$.

We will compare valuations lexicographically such that, for two policies π and π', we write $\nu^\pi_i > \nu^{\pi'}_i$ if and only if $\mathbf{g}^\pi_i > \mathbf{g}^{\pi'}_i$, or $\mathbf{g}^\pi_i = \mathbf{g}^{\pi'}_i$ and $\mathbf{h}^\pi_i > \mathbf{h}^{\pi'}_i$. We use $\geq$, $<$, and $\leq$ in the same way. We use the same notation for comparing vectors of valuations as we do for value vectors for the discounted case.

We next establish a connection between values and potentials, and values of the corresponding discounted MDP. We let $\mathbf{v}^{\pi,\gamma}$ be the value vector for policy π for the discounted reward criterion when using discount factor γ. Using a Laurent series expansion for $\mathbf{v}^{\pi,\gamma}$ it is possible to prove the following lemma.

Lemma 2.3.4 *Let π be any policy, then*

$$\mathbf{v}^{\pi,\gamma} \;=\; \frac{1}{1-\gamma}\mathbf{g}^\pi + \mathbf{h}^\pi + f^\pi(\gamma)$$

where $f^\pi(\gamma) \in \mathbb{R}^n$ is a vector, and $\lim_{\gamma\uparrow 1} f^\pi(\gamma) = \mathbf{0}$.

It should be noted that the Laurent series is defined in terms of $\rho = \frac{1-\gamma}{\gamma}$, and not γ. In particular, $\rho \downarrow 0$ for $\gamma \uparrow 1$. The function $f^\pi(\gamma)$ represents the higher order terms of the Laurent series as well as $\frac{1-\gamma}{\gamma}\mathbf{h}^\pi$. For details see Puterman [101].

By multiplying both sides of the equation in Lemma 2.3.4 by $(1-\gamma)$, and taking the limit for γ going to 1 we get the following important Corollary:

Corollary 2.3.5 *Let π be any policy. Then:*

$$\mathbf{g}^\pi \;=\; \lim_{\gamma\uparrow 1}\; (1-\gamma)\mathbf{v}^{\pi,\gamma}\;.$$

Let $(\gamma_k)_{k=0}^{\infty}$ be a sequence of discount factors converging to 1. Since for every discount factor there is an optimal policy, and since there are only finitely many policies, there must be a policy that is optimal for discount factors arbitrarily close to 1. In fact, the following lemma shows something stronger. For a proof of the lemma see, e.g., Derman [30].

Lemma 2.3.6 *There exists a discount factor $\gamma(M) < 1$ and a policy π^*, such that for all $\gamma \in [\gamma(M), 1)$, π^* is optimal for the discounted MDP with discount factor γ.*

Andersson and Miltersen [4] gave an explicit bound for $\gamma(M)$, showing that $\gamma(M) = 1 - ((n!)^2 2^{2n+3} C^{2n^2})^{-1}$ has the desired property, where C is an upper bound for every numerator and denominator of (rational) numbers used to describe $\mathbf{P}$ and $\mathbf{c}$. In particular, the number of bits needed to describe $\gamma(M)$ is polynomial in the number of bits needed to describe M.

By combining Lemma 2.3.6 and Corollary 2.3.5 it follows that π^* is optimal under the average reward criterion, which proves the correctness of Theorem 2.1.8. It should be noted that not every policy that is optimal under the average reward criterion is optimal under the discounted reward criterion for all discount factors sufficiently close to 1.

2.3.1 The PolicyIteration algorithm

We saw in Section 2.2.2 that the PolicyIteration algorithm solves a discounted MDP by generating a sequence of policies with increasing values. For the average reward criterion we will instead define the policy iteration algorithm such that it generates a sequence of policies with increasing valuations. To define the algorithm we first introduce the notions of reduced costs and improving switches for the average reward criterion.

Let $\bar{\mathbf{c}}^{\pi,\gamma}$ be the reduced cost vector of π for the MDP with discount factor $\gamma < 1$. From Definition 2.2.8 and Lemma 2.3.4 we get:

$$\begin{aligned}
\bar{\mathbf{c}}^{\pi,\gamma} &= \mathbf{c} - (\mathbf{J} - \gamma\mathbf{P})\mathbf{v}^{\pi,\gamma} \\
&= \mathbf{c} - (\mathbf{J} - \gamma\mathbf{P}) \left(\frac{1}{1-\gamma}\mathbf{g}^\pi + \mathbf{h}^\pi + f^\pi(\gamma) \right) \\
&= \frac{\gamma}{1-\gamma}\big[(\mathbf{P} - \mathbf{J})\mathbf{g}^\pi\big] + \big[\mathbf{c} - \mathbf{J}\mathbf{g}^\pi - (\mathbf{J} - \mathbf{P})\mathbf{h}^\pi\big] + \hat{f}^\pi(\gamma) \qquad (2.9)
\end{aligned}$$

where $\hat{f}^\pi(\gamma) = (\gamma - 1)\mathbf{P}\mathbf{h}^\pi - (\mathbf{J} - \gamma\mathbf{P})f^\pi(\gamma)$. Note that $\hat{f}^\pi(\gamma) \to \mathbf{0}$ for $\gamma \uparrow 1$. When γ goes to 1, the first term becomes more important than the second term. This leads us to the following definition of reduced costs for the average reward criterion, in which the reduced cost of an action is a pair corresponding to the first and second term in (2.9).

Definition 2.3.7 (Reduced costs) *For every policy π, define the reduced costs, $\bar{\mathbf{c}}^\pi \in \mathbb{R}^m \times \mathbb{R}^m$, of π as:*

$$\bar{\mathbf{c}}^\pi = \big((\mathbf{P} - \mathbf{J})\mathbf{g}^\pi \ , \ \mathbf{c} - \mathbf{J}\mathbf{g}^\pi - (\mathbf{J} - \mathbf{P})\mathbf{h}^\pi\big) \ .$$

As in the case of valuations we make comparisons with reduced cost vectors lexicographically. When convenient we will write $(\mathbf{0}, \mathbf{0}) = \mathbf{0}$.

Note the similarity between equation (2.9) and equations (2.6) and (2.7) of Theorem 2.3.3. Equations (2.6) and (2.7) can, to some extent, be viewed as the equation $(\bar{\mathbf{c}}^{\pi,\gamma})_\pi = \mathbf{0}$ in the limit when $\gamma \to 1$.

Definition 2.3.8 (Improving switch) *We say that an action $a \in A$ is an* improving switch *with respect to a policy π if and only if $\bar{\mathbf{c}}^\pi_a > 0$.*

We define *improving sets* analogously to Definition 2.2.11. I.e., an improving set is a set of improving switches that can be incorporated in a policy. To be more precise an improving set may also contain actions with reduced cost zero. The following theorem corresponds directly to Theorem 2.2.12 for the discounted reward criterion.

Theorem 2.3.9 *Let π be a policy.*

(i) Let $B \subseteq A$ be improving with respect to π. Then $\nu^{\pi[B]} > \nu^\pi$.

(ii) If no action is an improving switch with respect to π, then π is optimal.

Proof: The proof is very similar to the proof of Theorem 2.2.12. Using Lemma 2.3.4 and equation (2.9) we move to the discounted setting and make use of the lemmas of Section 2.2. When we let the discount factor go to 1 we get back to the average reward criterion.

We first prove (i). Recall that, by Lemma 2.2.9, $(\bar{\mathbf{c}}^{\pi,\gamma})_\pi = \mathbf{0}$ for all $\gamma < 1$. Since B is an improving set with respect to π we get that $(\bar{\mathbf{c}}^\pi)_{\pi[B]} > \mathbf{0}$. This implies that for γ sufficiently close to 1 we have, from equation (2.9), $(\bar{\mathbf{c}}^{\pi,\gamma})_{\pi[B]} > \mathbf{0}$. From Lemma 2.2.1 we know that $(\mathbf{I} - \gamma \mathbf{P}_{\pi[B]})^{-1} \geq \mathbf{I}$ for every $\gamma < 1$. We then get from Lemma 2.2.10 that when γ is sufficiently close to 1:

$$\mathbf{v}^{\pi[B],\gamma} - \mathbf{v}^{\pi,\gamma} \;=\; (\mathbf{I} - \gamma \mathbf{P}_{\pi[B]})^{-1} (\bar{\mathbf{c}}^{\pi,\gamma})_{\pi[B]} \;\geq\; (\bar{\mathbf{c}}^{\pi,\gamma})_{\pi[B]} \,.$$

Furthermore, from Lemma 2.3.4 we have:

$$\mathbf{v}^{\pi[B],\gamma} - \mathbf{v}^{\pi,\gamma} \;=\; \frac{1}{1-\gamma}\big[\mathbf{g}^{\pi[B]} - \mathbf{g}^\pi\big] + \big[\mathbf{h}^{\pi[B]} - \mathbf{h}^\pi\big] + f(\gamma) \,,$$

where $f(\gamma) \to \mathbf{0}$ for $\gamma \uparrow 1$.

Using equation (2.9) we, thus, have, for all γ sufficiently close to 1:

$$\frac{1}{1-\gamma}\big[\mathbf{g}^{\pi[B]} - \mathbf{g}^\pi\big] + \big[\mathbf{h}^{\pi[B]} - \mathbf{h}^\pi\big] + f(\gamma) \;\geq$$
$$\frac{\gamma}{1-\gamma}\big[(\mathbf{P}_{\pi[B]} - \mathbf{I})\mathbf{g}^\pi\big] + \big[\mathbf{c}_{\pi[B]} - \mathbf{g}^\pi - (\mathbf{I} - \mathbf{P}_{\pi[B]})\mathbf{h}^\pi\big] + (\hat{f}^\pi(\gamma))_{\pi[B]} \,.$$

Hence, we must have that if $(\mathbf{P}_{\pi[B]} - \mathbf{I})\mathbf{g}^{\pi} > \mathbf{0}$ then $\mathbf{g}^{\pi[B]} - \mathbf{g}^{\pi} > \mathbf{0}$. Similarly, if $(\mathbf{P}_{\pi[B]} - \mathbf{I})\mathbf{g}^{\pi} = \mathbf{0}$ and $\mathbf{c}_{\pi[B]} - \mathbf{g}^{\pi} - (\mathbf{I} - \mathbf{P}_{\pi[B]})\mathbf{h}^{\pi} > \mathbf{0}$, then $\mathbf{g}^{\pi[B]} - \mathbf{g}^{\pi} > \mathbf{0}$, or $\mathbf{g}^{\pi[B]} - \mathbf{g}^{\pi} = \mathbf{0}$ and $\mathbf{h}^{\pi[B]} - \mathbf{h}^{\pi} > \mathbf{0}$. It follows that $\nu^{\pi[B]} > \nu^{\pi}$.

We next prove (ii).

Assume for the sake of contradiction that there exists another policy π' such that $\mathbf{g}^{\pi} \not\geq \mathbf{g}^{\pi'}$. Since there are no improving switches with respect to π we know that $\bar{\mathbf{c}}^{\pi} \leq \mathbf{0}$. Using equation (2.9) we get that $\bar{\mathbf{c}}^{\pi,\gamma} \leq \hat{f}^{\pi}(\gamma)$. From Lemma 2.2.10 we get that for all $\gamma < 1$:

$$\mathbf{v}^{\pi',\gamma} - \mathbf{v}^{\pi,\gamma} \;=\; (\mathbf{I} - \gamma\mathbf{P}_{\pi'})^{-1}(\bar{\mathbf{c}}^{\pi,\gamma})_{\pi'} \;=\; \sum_{k=0}^{\infty}(\gamma\mathbf{P}_{\pi})^{k}(\bar{\mathbf{c}}^{\pi,\gamma})_{\pi'} \;.$$

Let $\hat{F}_{\gamma} = \max_{a\in A} |(\hat{f}^{\pi}(\gamma))_a|$. Then we have $(\bar{\mathbf{c}}^{\pi,\gamma})_{\pi'} \leq \hat{F}_{\gamma}\mathbf{e}$, and we get:

$$\mathbf{v}^{\pi',\gamma} - \mathbf{v}^{\pi,\gamma} \;\leq\; \sum_{k=0}^{\infty}(\gamma\mathbf{P}_{\pi})^{k}\hat{F}_{\gamma}\mathbf{e} \;=\; \sum_{k=0}^{\infty}\gamma^{k}\hat{F}_{\gamma}\mathbf{e} \;=\; \frac{1}{1-\gamma}\hat{F}_{\gamma}\mathbf{e} \;.$$

Using Lemma 2.3.4 we then get:

$$\frac{1}{1-\gamma}\left[\mathbf{g}^{\pi'} - \mathbf{g}^{\pi}\right] + \left[\mathbf{h}^{\pi'} - \mathbf{h}^{\pi}\right] + f(\gamma) \leq \frac{1}{1-\gamma}\hat{F}_{\gamma}\mathbf{e} \;,$$

where $f(\gamma) \to \mathbf{0}$ for $\gamma \uparrow 1$. In particular, we must have $\mathbf{g}^{\pi'} - \mathbf{g}^{\pi} \leq \hat{F}_{\gamma}\mathbf{e}$ for all γ sufficiently close to 1. This is impossible, however, since $\hat{F}_{\gamma} \to 0$ for $\gamma \uparrow 1$ and $\mathbf{g}^{\pi} \not\geq \mathbf{g}^{\pi'}$. We, thus, get a contradiction, and the proof is complete.

□

It follows from Theorem 2.3.9 that when we define the PolicyIteration algorithm in the same way as for the discounted reward criterion, as shown in Figure 2.3, we get the following theorem. Note that the policy π^N returned by the PolicyIteration algorithm gives the maximum value vector $\mathbf{g}^{\pi^N}$, but not necessarily the maximum vector of valuations ν^{π^N}.

Theorem 2.3.10 *For every initial policy π^0, PolicyIteration(π^0) terminates after a finite number of iterations. If $(\nu^k)_{k=0}^{N}$ is the sequence of valuation vectors generated by the call, then $\nu^{k-1} < \nu^k$ for every $1 \leq k < N$. Furthermore, π^N is an optimal policy.*

Note that Theorem 2.3.10 also proves that there exist optimal policies as claimed by Theorem 2.1.9.

During each iteration of the PolicyIteration algorithm the value vector $\mathbf{g}^{\pi^k}$ and the vector of potentials $\mathbf{h}^{\pi^k}$ are computed for the current policy π^k. This can be done using Theorem 2.3.3. I.e., $\mathbf{g}^{\pi^k}$ and $\mathbf{h}^{\pi^k}$ can be found by solving a system of linear equations, which can, for instance, be done using Gaussian elimination with $O(n^3)$ arithmetic operations. The set of performed improving switches is again decided by an improvement rule. Just as for the discounted reward criterion, the original improvement rule suggested by Howard [63] constructs the next policy π^{k+1} such that for all states i:

$$\pi^{k+1}(i) = \operatorname*{argmax}_{a \in A_i} \bar{\mathbf{c}}_a^{\pi^k} .$$

We again refer to the algorithm obtained by using this improvement rule as Howard's PolicyIteration algorithm.

Determining the worst-case complexity of Howard's PolicyIteration algorithm was stated explicitly as an open problem at least 25 years ago. (It is mentioned, among other places, in Schmitz [107], Littman *et al.* [81] and Mansour and Singh [85].) It was shown by Fearnley [34] in 2010 that Howard's PolicyIteration algorithm may require an exponential number of iterations in n, the number of states. Fearnley's proof is for the total reward criterion, and as we will see in Section 2.4 this can be viewed as a special case of the average reward criterion. Prior to this result it was widely believed that the number of iterations was always polynomial in n and m. The work of Fearnley [34] followed a similar result by Friedmann [38] for *parity games*.

Together with the results of Ye [121] (and the improved bound of Hansen, Miltersen, and Zwick [59]), this provides a *complete characterization* of the complexity of Howard's PolicyIteration algorithm. The algorithm is *strongly polynomial* for discounted MDPs with a fixed discount factor, but *exponential* for the total and average reward criteria. Furthermore, for every MDP there exists a discount factor close enough to 1 such that every policy has the same set of improving switches as for the average reward criterion, see, e.g., Andersson and Miltersen [4]. By picking the discount factor sufficiently close to 1 it can then be shown that Fearnley's lower bound also holds for the discounted setting. See Hollanders, Delvenne, and Jungers [62] for details. Hence, when the discount factor is part of the input the number of iterations may also be exponential in the number of states and actions.

The best known upper bound for the number of iterations performed by Howard's PolicyIteration algorithm for MDPs with the average reward criterion is an $O(\frac{2^n}{n})$ upper bound by Mansour and Singh [85] for MDPs with n states and two actions per state.

2.3.2 Linear programming formulation

The solution of an MDP for the average reward criterion can be formulated as the following pair of primal (P) and dual (D) linear programs. We refer to Puterman [101] for a more detailed presentation of these linear programs.

$$(P)\quad \begin{array}{lrcl} \max & \mathbf{c}^T\mathbf{x} & & \\ s.t. & (\mathbf{J}-\mathbf{P})^T\mathbf{x} & = & \mathbf{0} \\ & (\mathbf{J}-\mathbf{P})^T\mathbf{y}+\mathbf{J}^T\mathbf{x} & = & \mathbf{e} \\ & \mathbf{x},\mathbf{y} & \geq & \mathbf{0} \end{array} \qquad (D)\quad \begin{array}{lrcl} \min & \mathbf{e}^T\mathbf{g} & & \\ s.t. & (\mathbf{J}-\mathbf{P})\mathbf{g} & \geq & \mathbf{0} \\ & (\mathbf{J}-\mathbf{P})\mathbf{h}+\mathbf{J}\mathbf{g} & \geq & \mathbf{c} \end{array}$$

For any feasible solution $(\mathbf{x},\mathbf{y})$ to (P), the vectors $\mathbf{x},\mathbf{y} \in \mathbb{R}^m$ can be interpreted as representing a flow in the following sense. Initially we assign a flow of one unit to every state. This flow is eventually transferred to the vector $\mathbf{x}$, which is required to satisfy flow conservation. I.e., the flow entering a state i equals the flow that leaves the state i. When flow is passed along an action it is distributed according to the probability distribution of the action. Before transferring the initial flow to $\mathbf{x}$ it can be passed from state to state temporarily. The vector $\mathbf{y}$ counts how many units of flow is passed through each action in this way. We can also think of $\mathbf{x}$ as a stationary distribution over the actions that has been scaled up by a factor of n. It is worth pointing out that there is no direct connection between policies and basic feasible solutions of (P). In particular, a lower bound for a PolicyIteration algorithm for the average reward criterion does not necessarily say anything about the behavior of the corresponding pivoting rule for (P). (There does, however, exist a discount factor sufficiently close to 1 for which we would get the desired connection for the linear program for the corresponding discounted MDP.)

If $(\mathbf{x}^*,\mathbf{y}^*)$ and $(\mathbf{g}^*,\mathbf{h}^*)$ are optimal solutions to (P) and (D), respectively, then $\mathbf{g}^*$ is the optimal value vector, and we can get an optimal policy π^* as follows. For each state $i \in S$, π^* uses an action $a \in A_i$ with $\mathbf{x}_a > 0$ if possible. If this is not possible then π^* uses an action $a \in A_i$ with $\mathbf{y}_a > 0$.

2.4 The total reward criterion

For the total reward criterion the value of a state i for a policy π is defined as:

$$\mathrm{val}_\pi^T(i) \;=\; \sum_{k=0}^{\infty} \mathbf{e}_i^T \mathbf{P}_\pi^k \mathbf{c}_\pi$$

This series may not converge, and we therefore need to make assumptions about the MDP to ensure convergence. For this purpose we introduce the notion of a terminal state:

Definition 2.4.1 (Terminal state) *We say that a state* $\text{T} \in S$ *is a terminal state if for all actions* $a \in A_i$, $\mathbf{P}_a = \mathbf{e}_{\text{T}}$ *and* $\mathbf{c}_a = 0$. *I.e.,* T *is an absorbing state for every policy* π, *and once* T *is reached no additional reward is accumulated.*

For the remainder of this section we assume that the MDP under consideration has a terminal state T. Note that reaching the terminal state corresponds to terminating the MDP. Also, since the decision at T is fixed we will let T and its actions be implicit in the description of the MDP M. We therefore do not consider T to be an element of S, and we let $\mathbf{P}$, $\mathbf{J}$, and $\mathbf{c}$ be the matrices and vectors obtained by removing the column corresponding to T and the rows corresponding to A_{T}. I.e., rows of $\mathbf{P}$ may sum to less than one, and $1 - \mathbf{P}_a\mathbf{e}$ is the probability of moving to T when using action a. Note that this provides a natural interpretation of the discounted reward criterion. The rows of the matrix $\gamma\mathbf{P}$ sum to $\gamma < 1$, and for each action we move to the terminal state with probability $(1 - \gamma)$.

Definition 2.4.2 (Stopping condition) *A policy* π *is said to satisfy the* stopping condition *if from each state* i *there is positive probability of eventually reaching the terminal state. If every policy satisfies the stopping condition we say that the MDP satisfies the stopping condition.*

Let us note that if from some state i we never reach the terminal state, then we also never reach the terminal state from the states reachable from i. Since every state reachable from i can be reached with at most n transitions, the stopping condition can also be stated as:

$$\forall i \in S: \quad (\mathbf{P}_\pi^n)_i\mathbf{e} < 1\,. \tag{2.10}$$

In the remainder of this section we assume that the MDP under consideration satisfies the stopping condition.

The following lemma is analogous to Lemma 2.2.1 for the discounted case.

Lemma 2.4.3 *For any policy* π *satisfying the stopping condition, the matrix* $(\mathbf{I} - \mathbf{P}_\pi)$ *is nonsingular and*

$$(\mathbf{I} - \mathbf{P}_\pi)^{-1} \;=\; \sum_{k=0}^{\infty} \mathbf{P}_\pi^k \;\geq\; \mathbf{I}\,.$$

Proof: The proof is similar to the proof of Lemma 2.2.1.

Assume for the sake of contradiction that $(\mathbf{I} - \mathbf{P}_\pi)$ is singular. Then there exists $\mathbf{x} \in \mathbb{R}^n$ such that $(\mathbf{I} - \mathbf{P}_\pi)\mathbf{x} = \mathbf{0}$ and $\mathbf{x} \neq \mathbf{0}$. By multiplying repeatedly by $\mathbf{P}_\pi$ we get that for all $k > 0$, $\mathbf{P}_\pi^{k-1}\mathbf{x} = \mathbf{P}_\pi^k\mathbf{x}$. It follows by induction that $(\mathbf{I} - \mathbf{P}_\pi^n)\mathbf{x} = \mathbf{0}$. Let $i = \operatorname{argmax}_{i\in[n]} |\mathbf{x}_i|$. Then $0 = |\mathbf{x}_i - \mathbf{P}^n_{\pi(i)}\mathbf{x}| \geq |\mathbf{x}_i| - \mathbf{P}^n_{\pi(i)}\mathbf{e}|\mathbf{x}_i| > 0$; a contradiction. The last inequality follows from (2.10).

We get from (2.10) that $(\mathbf{P}_\pi^n)^t \to 0$ for $t \to \infty$, implying that $(\mathbf{P}_\pi)^t \to 0$ for $t \to \infty$. The identity from the lemma then follows from:

$$\mathbf{I} \;=\; \lim_{t\to\infty} \mathbf{I} - \mathbf{P}_\pi^t \;=\; \lim_{t\to\infty} (\mathbf{I} - \mathbf{P}_\pi) \sum_{k=0}^{t-1} \mathbf{P}_\pi^k \;=\; (\mathbf{I} - \mathbf{P}_\pi) \sum_{k=0}^{\infty} \mathbf{P}_\pi^k \,.$$

Finally, since $\mathbf{P}_\pi^0 = \mathbf{I}$ and all entries of $\mathbf{P}_\pi^k$, for all k, are non-negative, we get that $(\mathbf{I} - \mathbf{P}_\pi)^{-1} \geq \mathbf{I}$. □

From Lemma 2.4.3 we get that values exist for policies satisfying the stopping condition. The following definition is therefore well-defined.

Definition 2.4.4 (Value vectors) *For every policy $\pi \in \Pi$, we define the* value vector $\mathbf{v}^\pi \in \mathbb{R}^n$ *by:*

$$\mathbf{v}^\pi = (\mathbf{I} - \mathbf{P}_\pi)^{-1}\mathbf{c}_\pi \,.$$

I.e., $(\mathbf{v}^\pi)_i = \mathrm{val}^T_\pi(i)$ for all $i \in S = [n]$.

Note that if the MDP is deterministic and the objective is to find a policy with minimum cost, then the problem becomes a single-source shortest path problem. The value of state i satisfies

$$\mathrm{val}^T_\pi(i) = \mathbf{c}_{\pi(i)} + \sum_{j\in S} \mathbf{P}_{\pi(i),j} \mathrm{val}^T_\pi(j) \,,$$

and in the case of the single-source shortest path problem, this recursively defines the distance to the terminal state T. Solving MDPs satisfying the stopping condition for the total reward criterion may in general be viewed as a stochastic shortest path problem.

We next consider the relationship between the total reward criterion and the average reward criterion. For this purpose we will momentarily assume that the terminal state T is represented explicitly. Note that, assuming that the MDP satisfies the stopping condition, the terminal state is reached from every state for every policy. Hence, the limiting average of the observed

rewards is always zero, such that $\mathbf{g}^\pi = \mathbf{0}$ for every policy π. Furthermore, since the limit $\lim_{t\to\infty} \sum_{k=0}^{t} \mathbf{P}_\pi^k \mathbf{c}_\pi$ always exists, we have:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N-1} \sum_{k=0}^{t} \mathbf{P}_\pi^k \mathbf{c}_\pi \;=\; \lim_{t\to\infty} \sum_{k=0}^{t} \mathbf{P}_\pi^k \mathbf{c}_\pi \;.$$

The limit on the left is also known as a Cesaro limit. Since $\mathbf{g}^\pi = \mathbf{0}$ it follows that $\mathbf{h}^\pi = \mathbf{v}^\pi$. I.e., the values for the total reward criterion may be viewed as the potentials for the average reward criterion.

For the remainder of the section we will again assume that T is represented implicitly.

We now introduce the definition of reduced costs and improving switches for the total reward criterion. Note that this definitions is analogous to Definition 2.2.8 for the discounted case, as well as Definition 2.3.7 for the average reward criterion.

Definition 2.4.5 (Reduced costs, improving switches) *For every policy π, define the reduced costs, $\bar{\mathbf{c}}^\pi \in \mathbb{R}^m$, of π as:*

$$\bar{\mathbf{c}}^\pi \;=\; \mathbf{c} - (\mathbf{J} - \mathbf{P})\mathbf{v}^\pi \;.$$

We say that an action $a \in A$ is an improving switch *with respect to π if and only if $\bar{\mathbf{c}}_a^\pi > 0$.*

We again let an improving set be a set of improving switches (possibly including actions with reduced cost zero) that can be incorporated in a policy, as in Definition 2.2.11. The following theorem corresponds directly to Theorem 2.2.12 for the discounted reward criterion, and to Theorem 2.3.9 for the average reward criterion.

Theorem 2.4.6 *Let π be a policy.*

(*i*) *Let $B \subseteq A$ be improving with respect to π. Then $\mathbf{v}^{\pi[B]} > \mathbf{v}^\pi$.*

(*ii*) *If no action is an improving switch with respect to π, then π is optimal.*

Proof: Following the discussion above, (*i*) can be seen as a direct consequence of Theorem 2.3.9 (*i*).

(*ii*) can be proved in the same way that Theorem 2.2.12 (*ii*) was proved. In particular, we also have an analog of Lemma 2.2.10 for the total reward criterion. □

From Theorem 2.4.6 we get that the PolicyIteration algorithm, defined in the same way as for the discounted reward criterion and the average reward criterion as shown in Figure 2.3, correctly computes an optimal policy in a finite number of iterations. This proves the existence of an optimal policy, Theorem 2.1.9.

During each iteration of the PolicyIteration algorithm the current value vector is computed, and an improvement rule decides which improving switches to perform. The original improvement rule suggested by Howard picks for each state the action with the largest reduced cost. As mentioned in Section 2.3.1, Fearnley [34] showed that the number of iterations required for Howard's PolicyIteration algorithm to solve an MDP for the average reward criterion may, in the worst case, be exponential in the number of states n. This result was, in fact, proved for the total reward criterion. It then holds for the average reward criterion as well because of the close connection between the two settings. I.e., because values for the total reward criterion may be viewed as potentials for the average reward criterion.

2.4.1 Linear programming formulation

The linear program for solving MDPs for the total reward criterion can be understood in essentially the same way as for the discounted reward criterion. We therefore refer to Section 2.2.3 for proofs of the lemmas and theorems of this section.

The primal (P) and dual (D) linear programs for the total reward criterion are:

$$(P)\quad \begin{array}{ll} \max & \mathbf{c}^T\mathbf{x} \\ s.t. & (\mathbf{J}-\mathbf{P})^T\mathbf{x} = \mathbf{e} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \qquad (D)\quad \begin{array}{ll} \min & \mathbf{e}^T\mathbf{y} \\ s.t. & (\mathbf{J}-\mathbf{P})\mathbf{y} \geq \mathbf{c} \end{array}$$

A variable $\mathbf{x}_a$, for $a \in A$, of the primal linear program (P) counts the number of times action a is used before reaching the terminal state t, when we sum over all starting states. It is again helpful to introduce flux vectors, which are well-defined due to Lemma 2.4.3 for policies that satisfy the stopping condition.

Definition 2.4.7 (Flux vectors) *For every policy π satisfying the stopping condition, define $\mathbf{x}^\pi \in \mathbb{R}^n$ as follows*

$$\mathbf{x}^\pi = (\mathbf{e}^T(\mathbf{I}-\mathbf{P}_\pi)^{-1})^T .$$

Furthermore, define the extension of $\mathbf{x}^\pi$ *to* m *coordinates to be* $\bar{\mathbf{x}}^\pi \in \mathbb{R}^m$, *where for all* $a \in A$:

$$(\bar{\mathbf{x}}^\pi)_a = \begin{cases} (\mathbf{x}^\pi)_i & \text{if } a = \pi(i) \text{ for some } i \in S \\ 0 & \text{otherwise} \end{cases}$$

As for the discounted reward criterion we have:

$$(\mathbf{x}^\pi)_i \;=\; \sum_{k=0}^{\infty} \mathbf{e}^T (P_\pi)^k \mathbf{e}_i \;=\; \sum_{j \in S} \sum_{k=0}^{\infty} \mathbf{e}_j^T (P_\pi)^k \mathbf{e}_i \;.$$

We also get the following two simple lemmas which correspond to lemmas 2.2.28 and 2.2.29, respectively.

Lemma 2.4.8 *For every policy* π *satisfying the stopping condition, we have* $\mathbf{x}^\pi \geq \mathbf{e}$.

Lemma 2.4.9 *Let* $\mathbf{x} \in \mathbb{R}^m$ *be any feasible solution for* (P), *then for all* $i \in S$ *there exists* $a \in A_i$ *such that* $\mathbf{x}_a > 0$.

Note that if a policy π does not satisfy the stopping condition, then there is a least one state i from which we never reach the terminal state T. It follows that there is an action that is used an infinite number of times for π, and, hence, π can not be represented as a feasible solution for (P). If the MDP does not satisfy the stopping condition (not every policy satisfies the stopping condition), then the polyhedron of the linear program (P) is unbounded. Also, if no policy satisfies the stopping condition then (P) is infeasible. The following lemma says that we still get the same connection between basic feasible solutions and policies that satisfy the stopping condition as we did in the discounted case.

Lemma 2.4.10 *For every policy* π *satisfying the stopping condition, the vector* $\bar{\mathbf{x}}^\pi \in \mathbb{R}^m$ *is a basic feasible solution of* (P) *with basis* π. *There are no other basic feasible solutions of* (P). *I.e., there is a one-to-one correspondence between policies that satisfy the stopping condition and basic feasible solutions of* (P).

Proof: The proof is essentially identical to the proof of Lemma 2.2.30. Observe first that, due to Lemma 2.4.3, every policy π satisfying the stopping condition is a basis for (P). From Definition 2.2.26 we see that:

$$(\mathbf{I} - \mathbf{P}_\pi)^T \mathbf{x}^\pi \;=\; \mathbf{e} \quad \Rightarrow \quad (\mathbf{J} - \mathbf{P})^T \bar{\mathbf{x}}^\pi \;=\; \mathbf{e}\,,$$

and from Lemma 2.4.8 we know that $\bar{\mathbf{x}}^\pi \geq \mathbf{0}$. Hence, $\bar{\mathbf{x}}^\pi$ is a basic feasible solution for (P).

Next, we show that every basic feasible solution $\mathbf{x}$ for (P) is equal to $\bar{\mathbf{x}}^\pi$ for some policy π. From Lemma 2.4.9 we know that for all $i \in S$ there exists $a \in A_i$ such that $\mathbf{x}_a > 0$. Furthermore, since $\mathbf{x}$ is a basic feasible solution there are at most n positive variables. It follows that the basis $B = \{a \in A \mid \mathbf{x}_a > 0\}$ for $\mathbf{x}$ can be interpreted as a policy π, and from the definition of $\bar{\mathbf{x}}^\pi$ that $\mathbf{x} = \bar{\mathbf{x}}^B = \bar{\mathbf{x}}^\pi$. □

Note that, just like it was the case for discounted MDPs, the definition of reduced costs for MDPs with the total reward criterion, Definition 2.4.5, and the definition of reduced costs for linear programs, Definition 1.2.1, are exactly the same for the primal linear program (P). Let π be a policy that satisfies the stopping condition, and assume that $\pi' = \pi[a]$, for $a \in A \setminus \pi$, does not satisfy the stopping condition. It follows that if $(\bar{\mathbf{c}}^\pi)_a > 0$ then the linear program (P) is unbounded. Also, from Theorem 2.3.9 we know that π' must have a state with a better value for the average reward criterion than π. It again follows that every pivoting rule for the simplex method corresponds directly to an improvement rule for the PolicyIteration algorithm. Similarly, every improvement rule for the PolicyIteration algorithm that only performs a single improving switch in every iteration corresponds directly to a pivoting rule for the simplex method.

The following lemma follows from Theorem 2.3.9.

Lemma 2.4.11 *Assume that* $\mathrm{val}_\pi^A(i) \leq 0$ *for every policy* π *and state* $i \in S$. *If* π' *is a policy that satisfies the stopping condition, then all policies reachable from* π' *by repeatedly performing improving switches satisfy the stopping condition.*

The proof of the following theorem is analogous to the proof of Theorem 2.2.31.

Theorem 2.4.12 *For every MDP we have:*

(*i*) *There exist optimal solutions to the linear programs* (P) *and* (D) *if and only if there exists an optimal policy* π^* *satisfying the stopping condition*

(*ii*) *Let* $\mathbf{x}^*$ *be any optimal solution of* (P). *Then every policy* $\pi \subseteq \{a \in A \mid \mathbf{x}_a^* > 0\}$ *is optimal.*

(*iii*) *Every optimal solution* $\mathbf{y}^*$ *of* (D) *is equal to the optimal value vector* $\mathbf{v}^*$.

Chapter 3

Turn-based stochastic games

In 1953 Shapley [110] introduced his model of *stochastic games*. In stochastic games, the players perform *simultaneous*, or *concurrent*, actions. In this chapter we consider the subclass of *turn-based* stochastic games (or stochastic games with *perfect information*). We refer to such games as *2-player turn-based stochastic games* (2TBSG). 2TBSGs form an intriguing class of games whose status in many ways resembles that of linear programming 40 years ago. They can be solved efficiently with strategy iteration (or policy iteration) algorithms, resembling the simplex method for linear programming, but no polynomial time algorithm is known. In fact, two-player turn-based stochastic games generalize Markov decision processes (MDPs) in a natural way, and MDPs can be viewed as a special case of linear programming, thus, giving rise to the connection. More precisely, 2TBSGs model long-term sequential decision making in uncertain, i.e., stochastic, and adversarial environments. A 2TBSG is an MDP in which certain decisions are made by an adversary. Conversely, MDPs may be viewed as degenerate 2TBSGs in which one of the players has no influence on the game.

Like an MDP, a 2TBSG is composed of a finite set of states, and for each state a finite, non-empty set of actions. The only difference between an MDP and a 2TBSG is that for the 2TBSG the set of states is partitioned into two sets controlled by player 1, *the minimizer*, and player 2, *the maximizer*, respectively. A 2TBSG is played in the same way as an MDP. In each time unit, the 2TBSG is in exactly one of the states. Whenever the game is in some state, the player controlling that state decides which action should be used next. Using an action incurs an immediate reward (or payoff, or cost), which is paid by player 1 to player 2. (The game is therefore a *zero-sum* game.) Using an action also results in a probabilistic transition to a new

state according to a probability distribution that depends on the action. The process goes on indefinitely. The goal of player 1 is to minimize the incurred rewards according to some criterion while the goal of player 2 is to maximize the incurred rewards according to the same criterion. As in Chapter 2 we consider infinite-horizon 2TBSGs where the sets of states and actions are finite, and we consider the discounted reward criterion, average reward criterion, and total reward criterion. All these criteria are defined as in Chapter 2.

For books on 2TBSGs see Filar and Vrieze [35] and Neyman and Sorin [97]. For a general book on game theory see, e.g., Osborne and Rubinstein [98]. This Chapter is mainly based on Hansen, Miltersen, and Zwick [59]. The chapter is organized as follows. In Section 3.1 we introduce 2TBSGs for all three of the above mentioned criteria. In Section 3.2 we introduce the StrategyIteration algorithm, and use it to prove the existence of optimal strategies. In Section 3.3 we briefly survey a number of generalizations and specializations of 2TBSGs, and go through the best known time bounds for solving these problems. In Section 3.4 we take a closer look at 2TBSGs with the discounted reward criterion, and show that the StrategyIteration algorithm (with Howard's improvement rule) is strongly polynomial when the discount factor is fixed.

3.1 Definitions

Formally, a 2-player turn-based stochastic game is defined as follows.

Definition 3.1.1 (2-player turn-based stochastic games) *A* 2-player turn-based stochastic game *(2TBSG) is a tuple* $G = (S_1, S_2, A, s, c, p)$*, where*

- S_1 *is a set of states controlled by player* 1*,*
- S_2 *is a set of states controlled by player* 2*,*
- A *is a set of actions,*
- $s : A \to S_1 \cup S_2$ *assigns each action to the state from which it can be used,*
- $c : A \to \mathbb{R}$ *associates each action with a reward,*
- *and* $p : A \to \Delta(S_1 \cup S_2)$ *associates each action with a probability distribution over states according to which the next state is chosen when* a *is used.*

We let $S = S_1 \cup S_2$, and for every $i \in S$, we let $A_i = \{a \in A \mid s(a) = i\}$ be the set of actions that can be used from i. We assume that $A_i \neq \emptyset$, for every $i \in S$.

We let $A^1 = \bigcup_{i \in S_1} A_i$ and $A^2 = \bigcup_{i \in S_1} A_2$ be the sets of actions that belong to player 1 and 2, respectively. Finally, we use $n = |S|$ and $m = |A|$ to denote the number of states and actions, respectively, in a 2TBSG.

Just like MDPs, 2TBSGs can be represented in different ways. In this chapter we will use the matrix representation as shown in Definition 3.1.2. Note that this definition is completely analogous to Definition 2.1.2. The graphical representation can also be defined analogously to Definition 2.1.3.

Definition 3.1.2 (Matrix representation) *Let $G = (S_1, S_2, A, s, c, p)$ be a 2TBSG. We may assume, without loss of generality, that $S = [n] = \{1, \ldots, n\}$ and $A = [m] = \{1, \ldots, m\}$. We let $\mathbf{P} \in \mathbb{R}^{m \times n}$, where $\mathbf{P}_{a,i} = p(a)_i$ is the probability of ending up in state i after taking action a, for every $a \in A = [m]$ and $i \in S = [n]$, be the* probability matrix *of the 2TBSG, and $\mathbf{c} \in \mathbb{R}^m$, where $\mathbf{c}_a = c(a)$ is the reward of action $a \in A = [m]$, be its* reward vector*. We also let $\mathbf{J} \in \mathbb{R}^{m \times n}$ be a matrix such that $\mathbf{J}_{a,i} = 1$ if $a \in A_i$, and 0 otherwise.*

Note that a 2TBSG is then described by $(S_1, S_2, \mathbf{P}, \mathbf{c}, \mathbf{J})$. I.e., we need to explicitly describe which states belong to which player. We again say that an action $a \in A$ is *deterministic* if there is a state $i \in S$ such that $p(a)_i = 1$, and we say that a 2TBSG is deterministic if all its actions are deterministic.

A *strategy* (or *policy*) for a player is a possibly probabilistic rule that specifies the action to be taken in each situation, given the full history of play so far. One of the fundamental results in the theory of MDPs and 2TBSGs, is that both players have *positional* optimal strategies (see Definition 3.1.6 below). A positional strategy is a strategy that is both *deterministic* and *memoryless*. A *memoryless* strategy is a strategy that depends only on the current state, and not on the full history. MDPs and 2TBSGs are *solved* by finding optimal positional strategies for the players. Since there exist optimal positional strategies we may, without loss of generality, restrict our attention to such strategies. In particular, whenever we use the word strategy we will mean a positional strategy. The fact that it is not necessary for the players to use randomized history dependent strategies follows from the corresponding result for MDPs, and from the connection between optimal strategies and Nash equilibria (see Theorem 3.1.9 below). We will not include a proof of this.

Definition 3.1.3 (Strategies, strategy profiles) *A (positional)* strategy π_j *for player* j*, where* $j \in \{1,2\}$*, is a mapping* $\pi_j : S_j \to A^j$ *such that* $\pi_j(i) \in A_i$*, for every* $i \in S_j$*. We say that player* j *uses strategy* π_j *if whenever the game is in state* i*, player* j *chooses action* $\pi_j(i)$*. A* strategy profile $\pi = (\pi_1, \pi_2)$ *is simply a pair of strategies for the two players. We let* $\Pi_j = \Pi_j(G)$*, for* $j \in \{1,2\}$*, be the set of all strategies of player* j*, and let* $\Pi = \Pi(G) = \Pi_1 \times \Pi_2$ *be the set of all strategy profiles in* G*.*

We note that a strategy profile $\pi = (\pi_1, \pi_2)$ may be viewed as a mapping $\pi : S \to A$, i.e., as a strategy in a 1-player version of the game. In particular, π is a policy for the MDP corresponding to the game. We can, thus, immediately make use of a large number of definitions, lemmas, and theorems from Chapter 2.

All strategies considered in this chapter are positional. When convenient, we identify a strategy π_j with the set $\pi_j(S) \subseteq A^j$, and a strategy profile π with the set $\pi(S) \subseteq A$. A strategy profile $\pi = (\pi_1, \pi_2)$, when viewed as a subset of A, is simply the *union* $\pi_1 \cup \pi_2$. We again let $\mathbf{P}_\pi \in \mathbb{R}^{n \times n}$ be the matrix obtained by selecting the *rows* of $\mathbf{P}$ whose indices belong to π. Note that $\mathbf{P}_\pi$ is a (row) *stochastic matrix*; its elements are non-negative and the elements in each row sum to 1. Thus $\mathbf{P}_\pi$ defines a Markov chain. Similarly, $\mathbf{c}_\pi \in \mathbb{R}^n$ is the vector containing the rewards of the actions that belong to π. The pair $\mathbf{P}_\pi, \mathbf{c}_\pi$ is thus a Markov chain with costs assigned to its states. We refer to Section 2.1 for a more extensive discussion of $\mathbf{P}_\pi$ and $\mathbf{c}_\pi$.

Since strategy profiles correspond to policies we will reuse the definitions of values from Section 2.1. For every strategy profile $\pi = (\pi_1, \pi_2) \in \Pi$, we let $\mathrm{val}^D_\pi(i)$ be the value of state $i \in S$ for the discounted reward criterion, as described in Definition 2.1.5. Similarly, $\mathrm{val}^A_\pi(i)$ is the value of state $i \in S$ for the average reward criterion, as described in Definition 2.1.6. Finally, $\mathrm{val}^T_\pi(i)$ is the value of state $i \in S$ for the total reward criterion, as described in Definition 2.1.7. We again use $\mathrm{val}_\pi(i)$ when describing all three criteria jointly, or when it is clear from the context which criterion is used.

Since there are two players with opposite objectives in 2TBSGs we need a different definition of optimal strategies compared to the definition of optimal policies for MDPs. Note, however, that if the strategy of one of the players is fixed then the game can be viewed as an MDP. Indeed, if we remove the unused actions of the player whose strategy is fixed, we may transfer control of his states to the other player. In this case the *optimal counter-strategy* (or *best reply* or *best response*) of the player whose strategy is not fixed should be the optimal policy in the corresponding MDP. We get the following definition.

Definition 3.1.4 (Optimal counter-strategies) *Let G be a 2TBSG and let $\pi_1 \in \Pi_1(G)$ and $\pi_2 \in \Pi_2(G)$ be strategies for player 1 and 2, respectively. Define for all states $i \in S$:*

$$\begin{aligned} \overline{\mathrm{val}}_{\pi_1}(i) &= \max_{\pi_2' \in \Pi_2(G)} \mathrm{val}_{\pi_1,\pi_2'}(i) \\ \underline{\mathrm{val}}_{\pi_2}(i) &= \min_{\pi_1' \in \Pi_1(G)} \mathrm{val}_{\pi_1',\pi_2}(i) \end{aligned}$$

We say that a strategy π_1' for player 1 is an optimal counter-strategy *against π_2, if and only if $\mathrm{val}_{\pi_1',\pi_2}(i) = \overline{\mathrm{val}}_{\pi_2}(i)$ for all states $i \in S$. Similarly, π_2' for player 2 is an optimal counter-strategy against π_1, if and only if $\mathrm{val}_{\pi_1,\pi_2'}(i) = \underline{\mathrm{val}}_{\pi_1}(i)$ for all states $i \in S$.*

It follows from Theorem 2.1.9 that optimal counter-strategy always exist, since we get an MDP when the strategy of one player is fixed and MDPs always have optimal policies. Note that for a given strategy the optimal counter-strategy need not be unique.

Let π_1 be some strategy. If π_2' is an optimal counter-strategy against π_1, then $\mathrm{val}_{\pi_1,\pi_2'}(i) = \overline{\mathrm{val}}_{\pi_1}(i)$ is the best possible value that player 2 can obtain for state i when player 1 uses the strategy π_1. We may, thus, view $\overline{\mathrm{val}}_{\pi_1}(i)$ as an upper bound for the value player 1 can guarantee for state i. It is natural to ask what the best guarantee a player can get from a single strategy is. This leads to the following definition.

Definition 3.1.5 (Upper and lower values) *For every state $i \in S$, we define the* upper value, $\overline{\mathrm{val}}(i)$, *and* lower value, $\underline{\mathrm{val}}(i)$, *as:*

$$\begin{aligned} \overline{\mathrm{val}}(i) &= \min_{\pi_1 \in \Pi_1} \overline{\mathrm{val}}_{\pi_1}(i) = \min_{\pi_1 \in \Pi_1} \max_{\pi_2 \in \Pi_2} \mathrm{val}_{\pi_1,\pi_2}(i) \\ \underline{\mathrm{val}}(i) &= \max_{\pi_2 \in \Pi_2} \underline{\mathrm{val}}_{\pi_2}(i) = \max_{\pi_2 \in \Pi_2} \min_{\pi_1 \in \Pi_1} \mathrm{val}_{\pi_1,\pi_2}(i) \end{aligned}$$

Note that we must have $\underline{\mathrm{val}}(i) \leq \overline{\mathrm{val}}(i)$ for all states i. Intuitively, if both players play reasonably we would expect the resulting value to be between $\underline{\mathrm{val}}(i)$ and $\overline{\mathrm{val}}(i)$. I.e., the players can always choose to use the strategies that give these guarantees. We will say that a strategy is optimal exactly when it achieves the best possible guarantee against the other player.

Definition 3.1.6 (Optimal strategies) *We say that a strategy π_1 for player 1 is* optimal *if and only if $\overline{\mathrm{val}}_{\pi_1}(i) = \overline{\mathrm{val}}(i)$ for all states $i \in S$. Similarly, a strategy π_2 for player 2 is optimal if and only if $\underline{\mathrm{val}}_{\pi_2}(i) = \underline{\mathrm{val}}(i)$ for all states $i \in S$. We say that a strategy profile (π_1, π_2) is optimal if and only if π_1 and π_2 are optimal.*

The following theorem shows the existence of optimal strategies. We will present a proof of the theorem in Section 3.2. The theorem was first established by Shapley [110].

Theorem 3.1.7 *Let G be any 2TBSG. Then there exists an optimal strategy profile. Furthermore, for all states $i \in S$, the upper and lower values are the same, such that we may define:*

$$\mathrm{val}(i) \;=\; \overline{\mathrm{val}}(i) \;=\; \underline{\mathrm{val}}(i)\ .$$

In particular, if (π_1, π_2) is an optimal strategy profile then $\mathrm{val}_{\pi_1,\pi_2}(i) = \mathrm{val}(i)$.

By *solving* a 2TBSG we mean computing an optimal strategy profile. The following definition and theorem gives an alternative interpretation of optimal strategies. The theorem is standard for finite two-player zero-sum games. It was also established by Shapley [110]. We include the proof for completeness.

Definition 3.1.8 (Nash equilibrium) *A strategy profile $(\pi_1, \pi_2) \in \Pi$ is a* Nash equilibrium *if and only if π_1 is an optimal counter-strategy against π_2, and π_2 is an optimal counter-strategy against π_1.*

Theorem 3.1.9 *Let $(\pi_1, \pi_2) \in \Pi$ be a strategy profile.*

(i) If (π_1, π_2) is a Nash equilibrium then

$$\forall i \in S: \quad val_{\pi_1,\pi_2}(i) = \overline{\mathrm{val}}(i) = \underline{\mathrm{val}}(i)\ ,$$

and (π_1, π_2) is optimal.

(ii) If (π_1, π_2) is optimal, then (π_1, π_2) is a Nash equilibrium.

Proof: We first prove (i). Let $i \in S$ be any state. Since π_2 is an optimal counter-strategy against π_1 we have:

$$\mathrm{val}_{\pi_1,\pi_2}(i) \;=\; \max_{\pi_2' \in \Pi_2(G)} \mathrm{val}_{\pi_1,\pi_2'}(i) \;\geq\; \min_{\pi_1' \in \Pi_1(G)} \max_{\pi_2' \in \Pi_2(G)} \mathrm{val}_{\pi_1',\pi_2'}(i)$$

Also, since π_1 is an optimal counter-strategy against π_2 we have:

$$\forall \pi_1' \in \Pi_1: \quad val_{\pi_1,\pi_2}(i) \;\leq\; \mathrm{val}_{\pi_1',\pi_2}(i) \;\leq \max_{\pi_2' \in \Pi_2(G)} \mathrm{val}_{\pi_1',\pi_2'}(i)$$

$$\Rightarrow \quad val_{\pi_1,\pi_2}(i) \leq \min_{\pi_1' \in \Pi_1(G)} \max_{\pi_2' \in \Pi_2(G)} \mathrm{val}_{\pi_1',\pi_2'}(i)$$

We have proved that $val_{\pi_1,\pi_2}(i) = \overline{\mathrm{val}}(i)$. Also, since $val_{\pi_1,\pi_2}(i) = \overline{\mathrm{val}}_{\pi_1}(i)$, it follows that π_1 is an optimal strategy. The proof that $val_{\pi_1,\pi_2}(i) = \underline{\mathrm{val}}(i)$ and that π_2 is optimal is analogous.

We next prove (ii). Let $i \in S$ be any state. Since (π_1, π_2) is an optimal strategy profile we get from Theorem 3.1.7 that:

$$\mathrm{val}_{\pi_1,\pi_2}(i) \;=\; \mathrm{val}(i) \;=\; \underline{\mathrm{val}}(i) \;=\; \underline{\mathrm{val}}_{\pi_2}(i)\,.$$

It follows that π_1 is an optimal counter-strategy against π_2. The proof that π_2 is an optimal counter-strategy against π_1 is analogous.

□

Let us note that if we can prove the existence of a Nash equilibrium then Theorem 3.1.7 follows from Theorem 3.1.9 (i). In fact, this is how we will prove the existence of optimal strategies in Section 3.2.

3.2 The StrategyIteration algorithm

We will next see how the PolicyIteration algorithm of Howard [63] for solving MDPs can be extended to 2TBSGs in a natural way. We refer to the resulting algorithm as the StrategyIteration algorithm. The StrategyIteration algorithm was described for 2-player stochastic games by Rao *et al.* [102]. (Their algorithm actually works on more general imperfect information games for which it is a non-terminating approximation algorithm.) Hoffman and Karp [61] earlier described a related algorithm for a somewhat different class of stochastic games.

For every strategy profile $\pi = (\pi_1, \pi_2)$, we define a vector of reduced costs $\bar{\mathbf{c}}^\pi$ as we did for MDPs. For the discounted reward criterion $\bar{\mathbf{c}}^\pi$ is defined as in Definition 2.2.8. For the average reward criterion $\bar{\mathbf{c}}^\pi$ is defined as in Definition 2.3.7. For the total reward criterion $\bar{\mathbf{c}}^\pi$ is defined as in Definition 2.4.5. In this section we consider all three criteria jointly, and we therefore generally do not specify which definition of $\bar{\mathbf{c}}^\pi$ is used. We again use the reduced costs to define improving switches.

Definition 3.2.1 (Improving switches) *We say that an action $a \in A^1$ is an* improving switch *for player 1 with respect to a strategy profile π if and only if $\bar{\mathbf{c}}^\pi_a < 0$. Similarly, an action $a \in A^2$ is an improving switch for player 2 with respect to π if and only if $\bar{\mathbf{c}}^\pi_a > 0$.*

The following lemma is a direct consequence of theorems 2.2.12, 2.3.9, and 2.4.6. I.e., since a policy is optimal for an MDP if and only if there are no improving switches, we get a similar statement about optimal counter-strategies.

Lemma 3.2.2 *A strategy $\pi_1 \in \Pi_1$ is an optimal counter-strategy against $\pi_2 \in \Pi_2$ if and only if no action $a \in A^1$ is an improving switch for player 1 with respect to (π_1, π_2). Similarly, $\pi_2 \in \Pi_2$ is an optimal counter-strategy against $\pi_1 \in \Pi_1$ if and only if player 2 has no improving switches.*

From the definition of Nash equilibria and Theorem 3.1.9 we get the following important corollary, which can be viewed as an analog of theorems 2.2.12 (ii), 2.3.9 (ii), and 2.4.6 (ii).

Corollary 3.2.3 *A strategy profile $\pi = (\pi_1, \pi_2)$ is a Nash equilibrium and optimal if and only if neither player has an improving switch with respect to π.*

We define improving sets for 2TBSGs analogously to Definition 2.2.11 for MDPs.

Definition 3.2.4 (Improving set) *A set of actions $B \subseteq A^j$ is said to be* improving for player j *with respect a strategy profile π if and only if it satisfies the following conditions:*

- *There exists an action $a \in B$ that is an improving switch for player j with respect to π.*
- *$|B \cap A_i| \leq 1$ for all $i \in S$.*
- *Every action $a \in B$, we have $\overline{\mathbf{c}}^\pi \leq 0$ if $j = 1$ and $\overline{\mathbf{c}}^\pi \geq 0$ if $j = 2$.*

We again let $\pi_j[B]$ be the strategy obtained from π_j by incorporating the set of actions $B \subseteq A^j$. We then get the following theorem which can be viewed as an analog of theorems 2.2.12 (i), 2.3.9 (i), and 2.4.6 (i). A similar theorem can be proved with the roles of the two players reversed.

Theorem 3.2.5 *Let $\pi = (\pi_1, \pi_2) \in \Pi$ be a strategy profile such that π_1 is an optimal counter-strategy against π_2. Let $B \subseteq A^2$ be an improving set for player 2 with respect to π, and let π_1' be an optimal counter-strategy against $\pi_2[B]$. Then*

$$\forall i \in S: \quad \mathrm{val}_{\pi_1', \pi_2[B]}(i) \;\geq\; \mathrm{val}_{\pi_1, \pi_2}(i)\,,$$

with strict inequality for at least one state. For the average reward criterion the inequalities are expressed in terms of valuations.

Function `StrategyIteration`(π_1^0, π_2^0)

$k \leftarrow 0$;
repeat
  while $\exists B_1 \subseteq A^1 : B_1$ *is improving w.r.t.* (π_1^k, π_2^k) *for player 1* **do**
    $\pi_1^k \leftarrow \pi_1^k[B_1]$;
  if $\exists B_2 \subseteq A^2 : B_2$ *is improving w.r.t.* (π_1^k, π_2^k) *for player 2* **then**
    $(\pi_1^{k+1}, \pi_2^{k+1}) \leftarrow (\pi_1^k, \pi_2^k[B])$;
    $k \leftarrow k + 1$;
  else
    return (π_1^k, π_2^k);

Figure 3.1: The StrategyIteration algorithm.

Proof: Since π_1 is an optimal counter-strategy against π_2, we get from Lemma 3.2.2 that player 1 has no improving switches with respect to π. I.e., $(\overline{\mathbf{c}}^{\pi})_{A^1} \geq \mathbf{0}$ and in particular $(\overline{\mathbf{c}}^{\pi})_{\pi_1'} \geq \mathbf{0}$.

We now view π as a policy for the MDP corresponding to the 2TBSG. With this interpretation $B \cup \pi_1'$ is an improving set with respect to π, and from theorems 2.2.12 (i), 2.3.9 (i), and 2.4.6 (i) we get that the values (valuations) for $\pi[B \cup \pi_1'] = (\pi_1', \pi_2[B])$ do not decrease compared to the values (valuations) for π, and that at least one state has larger value (valuation). □

The StrategyIteration algorithm is given in Figure 3.1. It starts with some initial strategy profile $\pi^0 = (\pi_1^0, \pi_2^0)$ and generates a sequence $(\pi^k)_{k=0}^N$, where $\pi^k = (\pi_1^k, \pi_2^k)$, of strategy profiles, ending with an optimal strategy profile π^N. The algorithm repeatedly updates the strategy π_1^k for player 1 by performing improving switches for player 1. Note that this can be viewed as running the PolicyIteration algorithm on the MDP obtained by fixing π_2^k, and therefore this inner loop always terminates. When π_1^k is an optimal counter-strategy against π_2^k, the algorithm updates π_2^k once by performing improving switches for player 2, and the process is restarted. If π_2^k can not be updated then neither of the players has an improving switch, and we know from Corollary 3.2.3 that $\pi^N = (\pi_1^N, \pi_2^N)$ is optimal. Furthermore, since π_1^k is an optimal counter-strategy against π_2^k, and π_2^{k+1} is obtained from π_2^k by performing switches from a set of actions that is improving with

respect to (π_1^k, π_2^k), we get from Theorem 3.2.5 that for all $0 \leq k < N$:

$$\forall i \in S : \quad \mathrm{val}_{\pi_1^{k+1}, \pi_2^{k+1}}(i) \ \geq \ \mathrm{val}_{\pi_1^k, \pi_2^k}(i) \ ,$$

with strict inequality for at least one state. For the average reward criterion the inequalities are expressed in terms of valuations. It follows that the same strategy profile does not appear twice, and since there are only a finite number of strategy profiles the algorithm terminates. We get the following theorem. By an iteration we mean an iteration of the outer loop.

Theorem 3.2.6 StrategyIteration(π_1^0, π_2^0) *terminates after a finite number of iterations, for every initial strategy profile* (π_1^0, π_2^0)*. After each iteration the value of at least one state has been increased, and no state has smaller value. The strategy profile* (π_1^N, π_2^N) *returned by the algorithm is optimal.*

Since the algorithm always successfully computes an optimal strategy profile, it follows that there always exists an optimal strategy profile, and we get Theorem 3.1.7. Note that the uniqueness of optimal values follows from Theorem 3.1.9, and the fact that the StrategyIteration algorithm returns a Nash equilibrium.

It is helpful to think of the StrategyIteration algorithm as only operating with the strategy for player 2, and view the inner loop as a subroutine. Indeed, optimal counter-strategies for player 1 can also be found by other means, for instance by using linear programming. During each iteration of the StrategyIteration algorithm the current values and reduced costs are computed, and an improvement rule decides which improving switches to perform. This is completely analogous to the PolicyIteration algorithm for MDPs. We will again say that Howard's improvement rule [63] picks the action from each state with largest reduced cost. I.e., π_2^{k+1} satisfies:

$$\forall i \in S_2 : \quad \pi_2^{k+1}(i) = \operatorname*{argmax}_{a \in A_i} \ \bar{\mathbf{c}}_a^{\pi^k} \ .$$

We refer to the resulting algorithm as Howard's StrategyIteration algorithm, although Howard only introduced the algorithm for MDPs. Friedmann [38] showed in 2009 that Howard's StrategyIteration algorithm can require exponentially many iterations in the number of states for parity games. Parity games are a special case of 2TBSGs, with the average reward criterion, that we will introduce in Section 3.3.6. The lower bound of Friedmann was adapted to MDPs by Fearnley [34].

3.3 Generalizations and specializations

3.3.1 LP-type problems

LP-type problems were defined by Sharir and Welzl [111] as an abstract generalization of linear programs. LP-type problems can be solved by the RandomFacet algorithm. In fact, the RandomFacet algorithm was introduced by Sharir and Welzl [111], and only later did Matoušek, Sharir and Welzl [88] show that it solves LP-type problems (and linear programs) in subexponential time.

Ludwig [82] was the first to note that the RandomFacet algorithm can be used in the setting of 2TBSGs. More precisely, he showed that the RandomFacet algorithm can be used to solve *simple stochastic games*; a specialization of 2TBSGs with two actions per state that will be introduced below. Petersson and Vorobyov [99] and Björklund *et al.* [11],[12],[13] adapted these algorithms to work on *parity games* and *mean payoff games*. Parity games and mean payoff games will also be presented below. Halman [58] later gave a formal proof that the optimization problems involved in solving all these type of games are of LP-type.

The best time bound for solving 2TBSGs expressed solely in terms of the number states and actions is derived from these results. I.e., it follows from the analysis of Matoušek, Sharir and Welzl [88] that the RandomFacet algorithm can be used to solve 2TBSGs in expected time $2^{O(\sqrt{n \log m})}$.

An LP-type problem is defined as follows. Let $A = \{1, \ldots, m\}$ be a set of abstract constraints, and let $\text{val} : 2^A \to \mathbb{R} \cup \{-\infty\}$ assign a value to every subset of abstract constraints. We say that $B \subseteq A$ is a *basis* if and only if $-\infty < \text{val}(B)$, and $\text{val}(B') < \text{val}(B)$ for all proper subsets $B' \subsetneq B$. (A, val) is an *LP-type problem* if the following two axioms are satisfied:

(*i*) Monotonicity: For any $F \subseteq G \subseteq A$, we have $\text{val}(F) \leq \text{val}(G)$.

(*ii*) Locality: For any $F \subseteq G \subseteq A$ with $-\infty < \text{val}(F) = \text{val}(G)$ and any $a \in A$, $\text{val}(G) < \text{val}(G \cup \{a\})$ implies that $\text{val}(F) < \text{val}(F \cup \{a\})$.

If the following additional axiom is satisfied we say that (A, val) is *basis-regular*.

(*iii*) If $B \subseteq A$ is a basis and $a \in A$ is an abstract constraint, then every basis of $B \cup \{a\}$ has $|B|$ elements.

Solving an LP-type problem (A, val) means finding a basis $B^* \subseteq A$ such that $\text{val}(B^*) = \text{val}(A)$.

A 2TBSG can be viewed as an LP-type problem as follows. A is the set of actions of player 2. $\mathrm{val}(F)$, for $F \subseteq A$, is the sum of the optimal values of all the states for the subgame defined by actions of F, and $\mathrm{val}(F) = -\infty$ if there is a state from which no action is available (F does not define a subgame). Note that removing actions restricts player 2 (the maximizer) which only lowers the value. This proves monotonicity. Due to the existence of optimal strategies, $B \subseteq A$ is a basis if and only if B is a strategy for player 2, which provides the third axiom. Finally, if $\mathrm{val}(F) = \mathrm{val}(G)$ for $F \subseteq G$, then there exists a strategy $\pi_2 \subseteq F$ that is optimal for both F and G. If $a \in A$ is an improving switch for player 2 with respect to π_2, then $\mathrm{val}(\pi_2) < \mathrm{val}(\pi_2[a])$, and since $\pi_2[a] \subseteq F \cup \{a\} \subseteq G \cup \{a\}$ this proves locality.

3.3.2 Unique sink orientations

An n-dimensional hypercube is an undirected graph $G = (V, E)$ with vertex set $V = \{0,1\}^n$ and edge set

$$E = \{(\mathbf{u}, \mathbf{v}) \subseteq V \times V \mid \exists i \in [n] : \mathbf{u}_i \neq \mathbf{v}_i \wedge (\forall j \neq i : \mathbf{u}_j = \mathbf{v}_j)\} .$$

I.e., there is an edge between $\mathbf{u} \in \{0,1\}^n$ and $\mathbf{v} \in \{0,1\}^n$ if and only if the Hamming distance between $\mathbf{u}$ and $\mathbf{v}$ is one. Let $\mathbf{x} \in \{0,1,*\}^n$. The $*$ symbol may be thought of as a wildcard character. Then $\mathbf{x}$ defines a subcube (a subgraph) $G_{\mathbf{x}} = (V_{\mathbf{x}}, E_{\mathbf{x}})$ as follows:

$$\begin{aligned} V_{\mathbf{x}} &= \{\mathbf{v} \in V \mid \forall i \in [n] : \ \mathbf{x}_i = * \ \vee \ \mathbf{v}_i = \mathbf{x}_i\} \\ E_{\mathbf{x}} &= E \cap (V_{\mathbf{x}} \times V_{\mathbf{x}}) \end{aligned}$$

I.e., $V_{\mathbf{x}}$ contains all the vertices that agree with $\mathbf{x}$. The dimension of $G_{\mathbf{x}}$ is the number of $*$ symbols in $\mathbf{x}$. Let $\phi : V \to \{0,1\}^n$ be a function that assigns a zero-one vector to every vertex $\mathbf{u} \in V$, such that for every edge $(\mathbf{u}, \mathbf{v}) \in E$, we have $(\phi(\mathbf{u}))_i \neq (\phi(\mathbf{v}))_i$ when $\mathbf{u}_i \neq \mathbf{v}_i$. I.e., ϕ defines an *orientation* of the edges of E, where we think of an edge $(\mathbf{u}, \mathbf{v}) \in E$ with $\mathbf{u}_i \neq \mathbf{v}_i$ as being directed towards $\mathbf{v}$ if $(\phi(\mathbf{u}))_i = 1$ and $(\phi(\mathbf{v}))_i = 0$. We say that a vertex $\mathbf{v} \in V_{\mathbf{x}}$ is a *sink* w.r.t. ϕ in $G_{\mathbf{x}}$ if and only if $(\phi(\mathbf{v}))_i = 0$ for all $i \in [n]$ with $\mathbf{x}_i = *$. We say that ϕ is a *unique sink orientation* (USO) of G if and only if there is a unique sink w.r.t. ϕ for every subcube $G_{\mathbf{x}}$, where $\mathbf{x} \in \{0,1,*\}^n$. Solving a unique sink orientation problem ϕ means finding the vertex $\mathbf{v} \in V$ with $\phi(\mathbf{v}) = \mathbf{0}$. An algorithm can *evaluate* ϕ for vertices in V, and the complexity of an algorithm is the number of vertex evaluations needed for the algorithm to find the unique sink of G w.r.t. ϕ in the worst case.

Unique sink orientations of hypercubes were defined by Stickney and Watson [114] as an abstract generalization of P-matrix linear complementarity problems (for a definition of P-matrix linear complementarity problems see Section 3.3.3 below).

When the directed graph obtained by orienting the edges of G according to ϕ is *acyclic*, we say that ϕ is an *acyclic unique sink orientation* (AUSO) of G. Note that for an AUSO ϕ we can assign a unique value $\text{val}(\mathbf{u})$ to every vertex $\mathbf{u} \in V$, such that an edge $(\mathbf{u}, \mathbf{v}) \in E$ is directed towards $\mathbf{v}$ for ϕ if and only if $\text{val}(\mathbf{v}) > \text{val}(\mathbf{u})$. It is not difficult to see that an AUSO is an LP-type problem. I.e., for an n-dimensional AUSO we have $2n$ abstract constraints: two constraints, corresponding to 0 and 1, for every $i \in [n]$. The value of a subset of abstract constraints is the value of the sink of the corresponding subcube, and $-\infty$ if there is an index for which neither 0 nor 1 is available. Gärtner [45] showed that the RandomFacet algorithm solves n-dimensional AUSOs in expected time $2^{O(\sqrt{n})}$. The analysis of the RandomFacet algorithm is particularly nice for this case. Matoušek [86] also constructed a family of AUSOs such that the RandomFacet algorithm when run on a *random* instance from the family, has an expected running time $2^{\Omega(\sqrt{n})}$. I.e., the bound for RandomFacet is tight up to a constant factor in the exponent for AUSOs.

Note that if every state controlled by player 2 for a 2TBSG has exactly two actions, then a strategy π_2 for player 2 can be interpreted as a zero-one vector. It is then not difficult to show that the 2TBSG defines an AUSO. I.e., strategies $\pi_2 \in \Pi_2$ of player 2 correspond to vertices of a hypercube. We then get an AUSO ϕ by saying that $(\phi(\pi_2))_i = 1$ if and only if changing the decision at state i is an improving switch w.r.t. $\pi = (\pi_1, \pi_2)$, where π_1 is an optimal counter-strategy against π_2. (By making infinitesimal perturbations, we may assume without loss of generality that no two strategy profiles have the same value vectors.)

Szabo and Welzl [116] introduced the FibonacciSeesaw algorithm for solving USOs, and showed that it uses at most F_{n+2} (the $(n{+}2)$-th Fibonacci number) vertex evaluations. (In fact, they also introduced a slightly faster but more complicated variant of the algorithm.) The FibonacciSeesaw algorithm is the fastest known deterministic algorithm for solving USOs. The best known time bound for deterministic algorithms for solving 2TBSGs, where the bound is expressed only in terms of the number of states n and actions m, is also obtained from this result.

As hinted by the connection to 2TBSGs, if ϕ is an AUSO and $\mathbf{u} \in \{0,1\}^n$ is a vertex, then the set of indices $I(\mathbf{u}) = \{i \in [n] \mid (\phi(\mathbf{u}))_i = 1\}$ can be viewed as a set of improving switches. For every $\mathbf{u} \in \{0,1\}^n$, and $B \subseteq [n]$

let $\mathbf{u}[B] \in \{0,1\}^n$ be defined as:

$$(\mathbf{u}[B])_i \;=\; \begin{cases} 1-\mathbf{u}_i & \text{if } i \in B \\ \mathbf{u}_i & \text{otherwise} \end{cases}$$

I.e., $\mathbf{u}[B]$ is obtained from $\mathbf{u}$ by performing the switches in B. Furthermore, for $\mathbf{u} \in \{0,1\}^n$ define:

$$\mathit{cube}(\mathbf{u}, B) \;=\; \{\mathbf{v} \in \{0,1\}^n \mid \forall i \in [n] :\; i \in B \vee \mathbf{v}_i = \mathbf{u}_i\} \; .$$

It is not difficult to see that if ϕ is a USO and ϕ' is obtained from ϕ by reversing the orientation of every edge, then ϕ' is also a USO (see, e.g., [109]). Let $\mathrm{val} : V \to \mathbb{R}^n$ be an assignment of values that is consistent with an AUSO ϕ. It follows that for every vertex $\mathbf{u}$ and every non-empty set $B \subseteq I(\mathbf{u})$ we have $\mathrm{val}(\mathbf{u}[B]) > \mathrm{val}(\mathbf{u})$. To see this, let $V' = \mathit{cube}(\mathbf{u}, I(\mathbf{u}))$. If the orientations of all the edges are reversed, then $\mathbf{u}$ becomes the unique sink of the subcube consisting of vertices in V'. Since the orientation is acyclic it is easy to see that every vertex has a directed path to the sink. The claim follows.

It follows from the above discussion that StrategyIteration algorithms can be defined for AUSOs in the usual way. We refer to the algorithm corresponding to Howard's improvement rule as the SwitchAll algorithm, since it repeatedly updates the current vertex by performing all the improving switches. It follows from Mansour and Singh [85] that the SwitchAll algorithm performs at most $O(\frac{2^n}{n})$ iterations. The same upper bound holds for 2TBSGs with two choices per state. Schurr and Szabó [109] gave a $\Omega(2^{n/2})$ lower bound for the SwitchAll algorithm.

The following lemma was proved together with Uri Zwick. It can be viewed as a strengthening of a condition by Madani [83]. Romain Hollanders came up with a similar condition independently.

Lemma 3.3.1 *Let $A \subseteq \{0,1\}^{N \times n}$ be the matrix where the i-th row corresponds to the i-th vertex generated by a run of the* SwitchAll *algorithm. Then A satisfies the following condition:*

$$\forall 1 \leq i < j \leq N,\; \exists k \in [n] : \quad A_{i,k} \neq A_{i+1,k} = A_{j,k} = A_{j+1,k} \; , \tag{3.1}$$

where $A_{N+1,k} := A_{N,k}$ for all $k \in [n]$.

Proof: Let $\mathbf{u}^i = (A_i)^T$ be the i-th vertex generated by the SwitchAll algorithm. Since the SwitchAll algorithm performs all improving switches

we know that $I(\mathbf{u}^i) = \{k \in [n] \mid A_{i,k} \neq A_{i,k+1}\}$. Following the discussion above we know that $\text{val}(\mathbf{v}) \geq \text{val}(\mathbf{u}^i)$ for all $\mathbf{v} \in \{\mathbf{u}^i[B] \mid B \subseteq I(\mathbf{u}^i)\} = \mathit{cube}(\mathbf{u}^i, I(\mathbf{u}^i))$. Conversely, it can be shown in the same way that $\text{val}(\mathbf{v}) \leq \text{val}(\mathbf{u}^i)$ for all $\mathbf{v} \in \{\mathbf{u}^i[B] \mid B \subseteq [n] \setminus I(\mathbf{u}^i)\} = \mathit{cube}(\mathbf{u}^i, [n] \setminus I(\mathbf{u}^i))$. Note that the definition $A_{N+1,k} := A_{N,k}$ for all $k \in [n]$ means that $\mathbf{u}^N$ is the sink.

Since $\text{val}(\mathbf{u}^i) < \text{val}(\mathbf{u}^j)$ for $i < j$ we must have:

$$\forall 1 \leq i < j \leq N: \quad \mathit{cube}(\mathbf{u}^i, [n] \setminus I(\mathbf{u}^i)) \cap \mathit{cube}(\mathbf{u}^j, I(\mathbf{u}^j)) \;=\; \emptyset\,.$$

We have $\mathit{cube}(\mathbf{u}^i, [n] \setminus I(\mathbf{u}^i)) \cap \mathit{cube}(\mathbf{u}^j, I(\mathbf{u}^j)) \neq \emptyset$ if and only if for every $k \in [n]$ we have one of the following:

(i) $(\mathbf{u}^i)_k = (\mathbf{u}^j)_k$, or alternatively $A_{i,k} = A_{j,k}$.

(ii) $k \in [n] \setminus I(\mathbf{u}^i)$, or alternatively $A_{i,k} = A_{i,k+1}$.

(iii) $k \in I(\mathbf{u}^j)$, or alternatively $A_{j,k} \neq A_{j,k+1}$.

Hence, it must be the case that $A_{i,k} \neq A_{j,k}$, $A_{i,k} \neq A_{i,k+1}$, and $A_{j,k} = A_{j,k+1}$. It follows that A satisfies (3.1). □

Let $f(n)$ be the maximum integer N such that there exists a matrix $A \in \{0,1\}^{N \times n}$ satisfying (3.1). Note that $2^{\lfloor n/2 \rfloor} \leq f(n) \leq 2^n$. It follows from Lemma 3.3.1 that the SwitchAll algorithm performs at most $f(n)$ iterations on n-dimensional AUSOs. By doing exhaustive search with a computer program we have shown that $f(n) = F_{n+2}$ for $n \leq 6$, and that $f(7) \geq 34 = F_{7+2}$. We conjecture:

Conjecture 3.3.2 (Hansen and Zwick) *$f(n) = F_{n+2}$. I.e., the number of iterations performed by the* SwitchAll *algorithm for n-dimensional AUSOs is at most F_{n+2}.*

Note that the bound suggested by the above conjecture matches the bound for the FibonacciSeesaw algorithm by Szabo and Welzl [116]. Any improvement over the $O(\frac{2^n}{n})$ bound by Mansour and Singh [85] would be interesting.

3.3.3 P-matrix linear complementarity problems

There is no known linear programming formulation for 2TBSGs. (Natural attempts fail. See Condon [23].) The problem of solving 2TBSGs can, however, be formulated as a P-matrix (generalized) linear complementarity problem. This was shown by Gärtner and Rüst [49] for (non-binary) simple

stochastic games (see a description of simple stochastic games below), and by Jurdziński and Savani [68] for discounted 2TBSGs with two actions per state.

Let $\mathbf{M} \in \mathbb{R}^{n \times n}$ be a matrix and $\mathbf{q} \in \mathbb{R}^n$ be a vector. A linear complementarity problem (LCP) is the problem of finding a feasible solution $\mathbf{w}, \mathbf{z} \in \mathbb{R}^n$ to the following set of constraints:

$$\begin{aligned} \mathbf{w} &= \mathbf{M}\mathbf{z} + \mathbf{q} \\ \mathbf{w}, \mathbf{z} &\geq \mathbf{0} \\ \mathbf{w}^T \mathbf{z} &= 0 \end{aligned}$$

We say that the LCP is a P-matrix LCP if $\mathbf{M}$ is a P-matrix. A square matrix is a P-matrix if all of its principal minors are positive.

The P-matrix LCPs of Jurdziński and Savani [68] for discounted 2TBSGs with two actions per state are derived as follows. Let $G = (S_1, S_2, A, s, c, p)$ be a 2TBSG. Let $\mathcal{I} \in \mathbb{R}^{n \times n}$ be a diagonal matrix with 1 and -1 on the diagonal, such that $\mathcal{I}_{i,i} = -1$ for $i \in S_1$, and $\mathcal{I}_{i,i} = 1$ for $i \in S_2$. Since there are two actions per state we can partition the set of actions into two disjoint strategy profiles π^a and π^b. Consider then the following set of constraints:

$$(\mathbf{I} - \gamma \mathbf{P}_{\pi^a})\mathbf{y} - \mathcal{I}\mathbf{z} = \mathbf{c}_{\pi^a} \tag{3.2}$$

$$(\mathbf{I} - \gamma \mathbf{P}_{\pi^b})\mathbf{y} - \mathcal{I}\mathbf{w} = \mathbf{c}_{\pi^b} \tag{3.3}$$

$$\mathbf{w}, \mathbf{z} \geq \mathbf{0} \tag{3.4}$$

$$\mathbf{w}^T \mathbf{z} = 0 \tag{3.5}$$

It can be proved in a similar way to the proof of Theorem 2.2.31 that if $\mathbf{y}, \mathbf{w}, \mathbf{z} \in \mathbb{R}^n$ is a feasible solution for (3.2-3.5), then $\mathbf{y}$ is the optimal value vector. I.e., (3.4) and (3.5) ensure that for every state $i \in S$ either $\mathbf{w}_i = 0$ or $\mathbf{z}_i = 0$, which means that $\mathbf{y}$ is the value vector of the corresponding strategy profile π. It then follows from (3.2), (3.3), (3.4), and the definition of $\mathcal{I}$ that neither player has an improving switch with respect to π, which means that π is optimal. By eliminating $\mathbf{y}$ we get an equivalent LCP where:

$$\begin{aligned} \mathbf{M} &= \mathcal{I}(\mathbf{I} - \gamma \mathbf{P}_{\pi^b})(\mathbf{I} - \gamma \mathbf{P}_{\pi^a})^{-1}\mathcal{I} \\ \mathbf{q} &= \mathcal{I}(\mathbf{I} - \gamma \mathbf{P}_{\pi^b})(\mathbf{I} - \gamma \mathbf{P}_{\pi^a})^{-1}\mathbf{c}_{\pi^a} - \mathcal{I}\mathbf{c}_{\pi^b} \end{aligned}$$

Jurdziński and Savani [68] then show that $\mathbf{M}$ is a P-matrix.

It is interesting to note that a P-matrix LCP can be expressed as an USO, as shown by Stickney and Watson [114]. The USOs arising from P-matrix LCPs need not be acyclic. It is also worth pointing out that the

problem of solving P-matrix LCPs is in the complexity class $\mathbf{PPAD} \cap \mathbf{PLS}$ (see, e.g., Daskalakis and Papadimitriou [27]).

P-matrix LCPs can be solved using interior point methods (see, e.g., [119]). It was shown, however, by Rüst [104] that the P-matrix LCPs arising from discounted 2TBSGs may be ill-conditioned, such that the bounds for the interior point methods depend linearly on $\frac{1}{1-\gamma}$.

3.3.4 Simple stochastic games

Simple stochastic games are a special case of 2TBSGs that can be defined as follows. Let $G = (V_1, V_2, V_R, E, \textsc{t})$ be a directed graph with vertex set $V = V_1 \cup V_2 \cup V_R \cup \{\textsc{t}\}$, where V_1, V_2, and V_R are disjoint, and edge set $E \subseteq (V_1 \cup V_2 \cup V_R) \times V$ where every vertex $v \in V_1 \cup V_2 \cup V_R$ has two outgoing edges. We interpret G as a graphical presentation of a 2TBSG as follows. V_j, for $j \in \{1, 2\}$, is the set of states controlled by player j, and edges leaving $v \in V_1 \cup V_2$ are actions. V_R is a set of randomization vertices, and when a vertex $v \in V_R$ is reached the next vertex is obtained by moving along either of the two edges leaving v with probability $\frac{1}{2}$. The transition probabilities of actions are defined correspondingly. Every action has reward 0, except a special action with reward 1 that forms a self-loop at $\textsc{t}$. We use the average reward criterion, and, hence, the value of a vertex for some strategy profile is equal to the probability of reaching $\textsc{t}$ for that strategy profile.

Simple stochastic games (SSGs) were introduced by Condon [22], who showed that the problem of solving SSGs is in $\mathbf{NP} \cap \mathit{co}\mathbf{NP}$. It remains a long-standing open problem to find a polynomial time algorithm for solving SSGs, and SSGs is one of the few problems in $\mathbf{NP} \cap \mathit{co}\mathbf{NP}$ with this status. The problem of solving SSGs is also known to be in $\mathbf{PPAD} \cap \mathbf{PLS}$ (see, e.g., Daskalakis and Papadimitriou [27]). Furthermore, Andersson and Miltersen [4] showed that the task of solving SSGs is polynomial time (Turing) equivalent to the task of solving 2TBSGs with the average reward criterion, and to the task of solving 2TBSGs with the discounted reward criterion where the discount factor is part of the input. (A similar result for the total reward criterion then follows easily.)

3.3.5 Mean payoff games

Mean payoff games are deterministic 2TBSGs with the average reward criterion. Mean payoff games were introduced by Ehrenfeucht and Mycielski[31] and studied by Gurvich, Karzanov, and Khachiyan[57]. There is no known polynomial time algorithm for solving mean payoff games. However, Zwick

and Paterson [125] gave a pseudo-polynomial time algorithm for the problem. Boros *et al.* [16] used this result to show that mean payoff games have smoothed polynomial complexity, and to derive a fully polynomial time approximation scheme (FPTAS) for mean payoff games. Furthermore, Jurdziński [65] showed that the problem of solving mean payoff games is in $\mathbf{UP} \cap co\mathbf{UP}$.

3.3.6 Parity games

Parity games form an intriguing special case of mean payoff games. A parity game can be described as a directed graph $G = (V_0, V_1, E, p)$ as follows. The set of vertices (states) $V = V_0 \cup V_1$ is partitioned into the vertices controlled by player 0, V_0, and the vertices controlled by player 1, V_1. $E \subseteq V \times V$ is the set of edges (actions), and $p : V \to \mathbb{N}$ assign an integer *priority* to every vertex. We say that the value of a vertex $v \in V$ for a strategy profile π (a choice of an outgoing edge from every vertex) is *even* if the parity of the largest priority on the cycle reached from v by following edges of π is even, and that the value is *odd* otherwise. The goal of player 0 (even) is to reach an even cycle, and the goal of player 1 (odd) is to reach an odd cycle. Hence, the game is a win-lose game, where player 0 wins from a vertex if and only if he can force the play to reach an even cycle.

A parity game can then be viewed as a mean payoff game as follows. We let the reward of every edge $(u, v) \in E$ be $c(u, v) = (-n)^{p(u)}$, where $n = |V|$. I.e., the rewards of all the edges leaving a vertex u are defined by the priority of u. Then the largest priority on a cycle is even if and only if the average reward of the cycle is positive. Hence, we may now view the game as a mean payoff game. I.e., we use the average reward criterion, and the goal of player 0 is to maximize the incurred rewards, while the goal of player 1 is to minimize the incurred rewards. For additional details about the reduction from parity games to mean payoff games, see Puri [100].

The problem of *solving* a parity game, i.e., determining which of the two players has a *winning strategy*, is known to be equivalent to the problems of complementation of ω-tree automata, and to the problem of μ-calculus model checking. (See [32],[33],[115],[55].)

It is a major open problem whether parity games (or any of the game families presented in this section) can be solved in polynomial time. There are various exponential time algorithms for solving parity games (see, e.g., [124],[66]). The theoretically fastest deterministic algorithm currently known for solving parity games runs in $2^{O(\sqrt{n}\log n)}$ time, where n is the number of vertices in the game, and is due to Jurdziński, Paterson, and Zwick [67]. No

such *deterministic* subexponential algorithms are known for mean payoff games or 2TBSGs.

Vöge and Jurdziński [118] defined StrategyIteration algorithms for parity games without relying on the reduction to mean payoff games. They did this by defining discrete valuations of strategy profiles. They then suggested an improvement rule that corresponds to Howard's improvement rule. The StrategyIteration algorithm of Vöge and Jurdziński [118] and another StrategyIteration algorithm of Schewe [106] behave extremely well in practice (see [43]), and it was hoped that such algorithms would solve parity games in polynomial time. Friedmann [38] recently showed, however, that this is not the case by exhibiting parity games on which these two StrategyIteration algorithms take exponential time.

3.4 Discounted turn-based stochastic games

We next take a closer look at 2-player turn-based stochastic games for the discounted reward criterion. Most of the definitions, lemmas, and theorems of this section have counterparts in Section 2.2. Due to the close connection to MDPs many of the proofs have been omitted. The purpose of the section is to introduce the ValueIteration algorithm for discounted 2TBSGs and relate it to the StrategyIteration algorithm.

As shown in Lemma 2.2.1, for every strategy profile π, the matrix $(\mathbf{I} - \gamma\mathbf{P}_\pi)$ is nonsingular, and we can define the vector of values as follows.

Definition 3.4.1 (Value vectors) *For every strategy profile $\pi \in \Pi$, we define the* value vector $\mathbf{v}^\pi \in \mathbb{R}^n$ *by:*

$$\mathbf{v}^\pi = (\mathbf{I} - \gamma\mathbf{P}_\pi)^{-1}\mathbf{c}_\pi \ .$$

I.e., $(\mathbf{v}^\pi)_i = \mathrm{val}^D_\pi(i)$ *for all* $i \in S = [n]$.

Definition 3.4.2 (One-step operator) *The* one-step operator $\mathcal{T} : \mathbb{R}^n \to \mathbb{R}^n$ *is defined componentwise as follows:*

$$(\mathcal{T}\mathbf{v})_i \ = \ \begin{cases} \min_{a \in A_i} \ \mathbf{c}_a + \gamma\mathbf{P}_a\mathbf{v} \ , & \text{if } i \in S_1 \\ \max_{a \in A_i} \ \mathbf{c}_a + \gamma\mathbf{P}_a\mathbf{v} \ , & \text{if } i \in S_2 \end{cases}$$

where $\mathbf{v} \in \mathbb{R}^n$ *is a vector.*

Note that $\mathcal{T}$ is a generalization of the corresponding operator for MDPs (Definition 2.2.3). More precisely, if $S = S_2$ then we get exactly the same operator as for MDPs.

The operator $\mathcal{T}$ is a contraction with Lipschitz constant γ. The proof of the following lemma is the same as the proof of Lemma 2.2.4, except that the case analysis takes into account whether a state belongs to player 1 or player 2.

Lemma 3.4.3 *For every $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and $\pi \in \Pi$ we have:*

$$\|\mathcal{T}\mathbf{u} - \mathcal{T}\mathbf{v}\|_\infty \;\leq\; \gamma \, \|\mathbf{u} - \mathbf{v}\|_\infty \; .$$

The Banach fixed point theorem implies that the operator $\mathcal{T}$ has a unique fixed point.

Corollary 3.4.4 *There is a unique vector $\mathbf{v}^* \in \mathbb{R}^n$ such that $\mathcal{T}\mathbf{v}^* = \mathbf{v}^*$.*

We next define the *strategy extraction* operators, which generalize the policy extraction operator for MDPs (Definition 2.2.13).

Definition 3.4.5 (Strategy extraction operators) *The* strategy extraction *operators $\mathcal{P}_1 : \mathbb{R}^n \to \Pi_1$ and $\mathcal{P}_2 : \mathbb{R}^n \to \Pi_2$ and $\mathcal{P} : \mathbb{R}^n \to \Pi$ are defined as follows:*

$$\begin{array}{rcl} (\mathcal{P}_1\mathbf{v})(i) & = & \operatorname{argmin}_{a \in A_i} \mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{v} \quad , \quad i \in S_1 \; , \\ (\mathcal{P}_2\mathbf{v})(i) & = & \operatorname{argmax}_{a \in A_i} \mathbf{c}_a + \gamma \mathbf{P}_a \mathbf{v} \quad , \quad i \in S_2 \; . \end{array}$$

and

$$\mathcal{P}\mathbf{v} \;=\; (\mathcal{P}_1\mathbf{v}, \mathcal{P}_2\mathbf{v}) \; .$$

Note that $\mathcal{P}_2$ corresponds to the policy extraction operator, and if $S = S_2$ then $\mathcal{P} = \mathcal{P}_2$. The following relation between the one-step and strategy extraction operator is immediate.

Lemma 3.4.6 *For every $\mathbf{v} \in \mathbb{R}^n$ we have $\mathcal{T}\mathbf{v} = (\mathbf{c} + \gamma\mathbf{P}\mathbf{v})_\pi$, where $\pi = \mathcal{P}\mathbf{v}$.*

We repeat here the definition of reduced costs for 2TBSGs. As described in Section 3.1 it is the same as for policies for MDPs (Definition 2.2.8).

Definition 3.4.7 (Reduced costs, improving switches) *The* reduced cost *vector $\bar{\mathbf{c}}^\pi \in \mathbb{R}^m$ corresponding to a strategy profile π is defined to be*

$$\bar{\mathbf{c}}^\pi \;=\; \mathbf{c} - (\mathbf{J} - \gamma\mathbf{P})\mathbf{v}^\pi \; .$$

We say that an action $a \in A$ is an improving switch *with respect to a policy π if and only if $\bar{\mathbf{c}}^\pi_a > 0$.*

The following two lemmas generalize lemmas 2.2.15 and 2.2.9. Their proofs have been omitted since they are the same as their counterparts for MDPs.

Lemma 3.4.8 *For every strategy profile π we have*

$$\begin{aligned}(\mathcal{P}_1\mathbf{v}^\pi)(i) &= \operatorname{argmin}_{a\in A_i} \bar{\mathbf{c}}^\pi_a \quad , \quad i \in S_1 \, , \\ (\mathcal{P}_2\mathbf{v}^\pi)(i) &= \operatorname{argmax}_{a\in A_i} \bar{\mathbf{c}}^\pi_a \quad , \quad i \in S_2 \, .\end{aligned}$$

Lemma 3.4.9 *For any strategy profile π we have $(\bar{\mathbf{c}}^\pi)_\pi = \mathbf{0} \in \mathbb{R}^n$. I.e., the reduced cost of every action used in π is zero.*

The following lemma supplies a proof of Theorem 3.1.7 for the discounted reward criterion that does not rely on the StrategyIteration algorithm. It is, in fact, the original proof given by Shapley [110]. It is a generalization of Lemma 2.2.16, and the proof is essentially the same.

Lemma 3.4.10 *Let $\mathbf{v}^* \in \mathbb{R}^n$ be the unique fixed point of $\mathcal{T}$, and let $\pi^* = \mathcal{P}\mathbf{v}^*$. Then, π^* is an optimal strategy profile.*

Proof: From Lemma 3.4.9 we know that $(\bar{\mathbf{c}}^{\pi^*})_{\pi^*} = \mathbf{0}$, and from Lemma 3.4.8 we then get that $(\bar{\mathbf{c}}^{\pi^*})_{\pi^*(i)} = \min_{a\in A_i} \bar{\mathbf{c}}^{\pi^*}_a = 0$ for all $i \in S_1$. It follows that $(\bar{\mathbf{c}}^{\pi^*})_{A^1} \geq \mathbf{0}$. Similarly, $(\bar{\mathbf{c}}^{\pi^*})_{\pi^*(i)} = \max_{a\in A_i} \bar{\mathbf{c}}^{\pi^*}_a = 0$ for all $i \in S_2$, such that $(\bar{\mathbf{c}}^{\pi^*})_{A^2} \leq \mathbf{0}$. Hence, neither player has an improving switch with respect to π^*, and from Corollary 3.2.3 we get that π^* is a Nash equilibrium, and then from Theorem 3.1.9 that π^* is optimal. □

3.4.1 The ValueIteration algorithm

The ValueIteration algorithm for 2TBSGs is the same as for MDPs, except that it uses the more general one-step operator. In fact, with the notation we have introduced the description of the ValueIteration algorithm for 2TBSGs is identical to the description for MDPs. The description of the ValueIteration algorithm is repeated in Figure 3.2. Starting from some initial vector $\mathbf{u}^0 \in \mathbb{R}^n$ it repeatedly applies the one-step operator $\mathcal{T}$ until the resulting change is sufficiently small.

The ValueIteration algorithm can be viewed as a special case of an algorithm by Shapley [110] for solving the more general class of stochastic games. When the ValueIteration algorithm is initialized with the zero vector, $\mathbf{u}^0 = \mathbf{0}$, the vector $\mathbf{u}^k$ can be viewed as the optimal values for the

Function `ValueIteration`$(\mathbf{u}^0, \epsilon)$

$k \leftarrow 0$;
repeat
  $\mathbf{u}^{k+1} \leftarrow \mathcal{T}\mathbf{u}^k$;
  $k \leftarrow k+1$;
until $\|\mathbf{u}^k - \mathbf{u}^{k-1}\|_\infty < \frac{\epsilon(1-\gamma)}{2\gamma}$;
return $\mathbf{u}^k$;

Figure 3.2: The ValueIteration algorithm.

2TBSG that terminates after k steps. The ValueIteration algorithm can, thus, be viewed as a dynamic programming algorithm.

The following theorem is analogous to Theorem 2.2.17, and it can be proved in the same way.

Theorem 3.4.11 *Let $(\mathbf{u}^k)_{k=0}^N$ be the sequence of value vectors generated by a call* ValueIteration$(\mathbf{u}^0, \epsilon)$*, for some $\epsilon > 0$. Let $\mathbf{v}^*$ be the optimal value vector. Then:*

(i) *For every $0 \leq k \leq N$ we have $\|\mathbf{v}^* - \mathbf{u}^k\|_\infty \leq \gamma^k \|\mathbf{v}^* - \mathbf{u}^0\|_\infty$.*

(ii) $\|\mathbf{v}^* - \mathbf{u}^N\|_\infty \leq \frac{\epsilon}{2}$.

(iii) *If $\pi = \mathcal{P}\mathbf{u}^N$, then $\|\mathbf{v}^* - \mathbf{v}^\pi\|_\infty \leq \epsilon$.*

(iv) $N \leq \frac{1}{1-\gamma} \log \frac{2\|\mathbf{v}^* - \mathbf{u}^0\|_\infty}{\epsilon}$.

In Section 2.2.1 we showed how the minimum distance between value vectors can be lower bounded by using Cramer's rule and bounds on the size of determinants. (See, e.g., Schrijver [108].) The same technique can be applied to 2TBSGs. Let $L(G) = L(\mathbf{P}, \mathbf{c}, \mathbf{J}, \gamma)$ be the number of bits needed to describe a 2TBSG G. The following theorem corresponds to Theorem 2.2.20 for MDPs. For details on how to prove the theorem see Section 2.2.1. The theorem was proved by Meister and Holzbaur [92] for MDPs and by Littman [80] for 2TBSGs.

Theorem 3.4.12 *Let $\mathbf{u}^N =$* ValueIteration$(\mathbf{0}, 2^{-4L(G)-2})$*, and $\pi = \mathcal{P}\mathbf{u}^N$. Then π is optimal, and the number of iterations is at most*

$$N = O\left(\frac{L(G) \log \frac{1}{1-\gamma}}{1-\gamma}\right) .$$

Let $\pi^0 = (\pi_1^0, \pi_2^0)$ be a strategy profile where π_1^0 is an optimal counter-strategy against π_2^0. We next relate the sequences of value vectors generated by ValueIteration($\mathbf{v}^{\pi^0}, \epsilon$), and by StrategyIteration(π_1^0, π_2^0) when using Howard's improvement rule. The following lemmas for the case of MDPs are well-known and appear, e.g., in Meister and Holzbaur [92]. The proofs for the 2-player case are essentially identical. (They may be folklore.)

Lemma 3.4.13 *Let $\pi_2 \in \Pi_2$, let $\pi_1 \in \Pi_1$ be an optimal counter-strategy against π_2, let $\pi_2' = \mathcal{P}_2\mathbf{v}^{\pi_1,\pi_2}$, and let π_1' be an optimal counter-strategy against π_2'. Then $\mathbf{v}^{\pi_1',\pi_2'} \geq \mathcal{T}\mathbf{v}^{\pi_1,\pi_2}$.*

Proof: Let $\pi = (\pi_1, \pi_2)$ and $\pi' = (\pi_1', \pi_2')$. Let us first note that if π is optimal then $\mathbf{v}^{\pi'} = \mathbf{v}^{\pi} = \mathcal{T}\mathbf{v}^{\pi}$. Otherwise we know that $\pi_2' = \pi_2[B]$ where B is improving with respect to (π_1, π_2). We then get from Theorem 3.2.5 that $\mathbf{v}^{\pi'} \geq \mathbf{v}^{\pi}$.

Let $i \in S$. From Lemma 3.4.6 we know that $(\mathcal{T}\mathbf{v}^{\pi})_i = \mathbf{c}_{\pi'(i)} + \gamma\mathbf{P}_{\pi'(i)}\mathbf{v}^{\pi}$. We then get:

$$(\mathcal{T}\mathbf{v}^{\pi})_i \;=\; \mathbf{c}_{\pi'(i)} + \gamma\mathbf{P}_{\pi'(i)}\mathbf{v}^{\pi} \;\leq\; \mathbf{c}_{\pi'(i)} + \gamma\mathbf{P}_{\pi'(i)}\mathbf{v}^{\pi'} \;=\; (\mathbf{v}^{\pi'})_i.$$

□

The following lemma corresponds to Lemma 2.2.23. We will use $\mathbf{v}^k$ as short-hand notation for $\mathbf{v}^{\pi_1^k,\pi_2^k}$.

Lemma 3.4.14 *Let $\pi^0 = (\pi_1^0, \pi_2^0)$ be a strategy profile such that π_1^0 is an optimal counter-strategy against π_2^0. Let $(\mathbf{v}^k)_{k=0}^N$ and $(\mathbf{u}^k)_{k=0}^\infty$ be the value vectors generated by* StrategyIteration(π_1^0, π_2^0)*, using Howard's improvement rule, and* ValueIteration($\mathbf{v}^{\pi^0}, 0$)*, respectively. Then, $\mathbf{v}^k \geq \mathbf{u}^k$, for every $0 \leq k \leq N$.*

Proof: We prove the lemma by induction. We have $\mathbf{v}^0 = \mathbf{u}^0$. Suppose now that $\mathbf{v}^k \geq \mathbf{u}^k$. Then, by Lemma 3.4.13 and the monotonicity of the value iteration operator, we have:

$$\mathbf{v}^{k+1} \;\geq\; \mathcal{T}\mathbf{v}^k \;\geq\; \mathcal{T}\mathbf{u}^k \;=\; \mathbf{u}^{k+1}.$$

□

Combining Theorem 3.4.11 (i) and Lemma 3.4.14, we get:

Lemma 3.4.15 *Let $(\mathbf{v}^k)_{k=0}^N$ be the sequence of value vectors generated by the call* StrategyIteration(π_1^0, π_2^0)*, where π_1^0 is an optimal counter-strategy against π_2^0. Let $\mathbf{v}^*$ be the optimal value vector. Then, for every $0 \leq k \leq N$ we have*

$$\|\mathbf{v}^* - \mathbf{v}^k\|_\infty \ \leq \ \gamma^k \, \|\mathbf{v}^* - \mathbf{v}^0\|_\infty.$$

As shown in Section 2.2.2 we get the following corollary. For 2TBSGs the corollary is obtained by using Theorem 3.4.12.

Corollary 3.4.16 *Starting with any strategy profile (π_1^0, π_2^0) where π_1^0 is an optimal counter-strategy against π_2^0, the number of iterations performed by* StrategyIteration(π_1^0, π_2^0)*, with Howard's improvement rule, is at most:*

$$N = O\left(\frac{L(G)\log\frac{1}{1-\gamma}}{1-\gamma}\right) .$$

3.4.2 Strongly polynomial bound for fixed discount

In Section 2.2.4 we presented strongly polynomial bounds for Howard's PolicyIteration algorithm for solving MDPs with a fixed discount factor. The first such bound was obtained by Ye [121]. In Hansen, Miltersen, and Zwick [59], we later improved the bound of Ye, and showed how to extend the result to the StrategyIteration algorithm with Howard's improvement rule for solving discounted 2TBSGs. We present this result next. What makes our analysis of the StrategyIteration algorithm surprising is the fact that Ye's analysis relies heavily on the LP formulation of MDPs. Our proof is based on finding natural game-theoretic quantities that correspond to the LP-based quantities used by Ye, and by reestablishing, via direct means, (improved versions) of the bounds obtained by Ye using LP duality.

We show in Hansen *et al.* [59] that the StrategyIteration algorithm with Howard's improvement rule solves discounted 2TBSGs in at most $O\big(\frac{m}{1-\gamma}\log\frac{n}{1-\gamma}\big)$ iterations. This supplies the *first* strongly polynomial algorithm for solving 2TBSGs, with a fixed discount factor. Prior to our strongly polynomial bound for the strategy iteration algorithm, the best time bound available for the problem of solving discounted 2TBSGs, with a fixed discounted factor, was a polynomial, but not strongly polynomial, bound of Littman [80]. Littman's bound was presented in Theorem 3.4.12. It is obtained by using the ValueIteration algorithm.

The following lemma corresponds exactly to Lemma 2.2.36 for MDPs, and it can be proved in the same way. I.e., we may view strategy profiles as policies for the MDP that is obtained by giving all the states to one player.

Lemma 3.4.17 *Let π'', π', π be three strategy profiles such that $\mathbf{v}^{\pi''} \leq \mathbf{v}^{\pi'} \leq \mathbf{v}^{\pi}$. Let $a = \text{argmin}_{a \in \pi''}(\overline{\mathbf{c}}^{\pi})_a$ and suppose that $a \in \pi'$. Then,*

$$\|\mathbf{v}^{\pi} - \mathbf{v}^{\pi'}\|_{\infty} \ \geq \ \frac{1-\gamma}{n} \|\mathbf{v}^{\pi} - \mathbf{v}^{\pi''}\|_{\infty} .$$

The proof of the following lemma is also essentially the same as the proof of Lemma 2.2.37, its counterpart for MDPs. The main difference is the use of Lemma 3.4.15 instead of Lemma 2.2.24. The lemma shows that an action is implicitly eliminated by the StrategyIteration algorithm after a certain number of iterations.

Lemma 3.4.18 *Let $(\pi^k)_{k=0}^N$ be the sequence of strategy profiles generated by the* StrategyIteration *algorithm, starting from any strategy profile $\pi^0 = (\pi_1^0, \pi_2^0)$ where π_1^0 is an optimal counter-strategy against π_2^0. For every $k \geq 0$ let $\pi^k = (\pi_1^k, \pi_2^k)$, and let $L = \log_{1/\gamma} \frac{n}{1-\gamma}$. Then, every strategy π_2^k contains an action that does not appear in any strategy π_2^{ℓ}, where $k + L < \ell \leq N$.*

Proof: By the correctness of the StrategyIteration algorithm, $\pi^* = \pi^N$ is an optimal strategy profile. Let $a = \text{argmin}_{a \in \pi^k}(\overline{\mathbf{c}}^{\pi^*})_a$. By Theorem 3.1.9, we know that π^* is a Nash equilibrium such that neither player has any improving switches. I.e., $(\overline{\mathbf{c}}^{\pi^*})_{A^1} \geq \mathbf{0}$ and $(\overline{\mathbf{c}}^{\pi^*})_{A^2} \leq \mathbf{0}$. In particular, if π^k is not optimal, then $(\overline{\mathbf{c}}^{\pi^*})_a < 0$ and $a \in A^2$. We may assume, therefore, that a belongs to player 2.

Suppose, for the sake of contradiction, that $a \in \pi^{\ell}$, for some $k + L < \ell \leq N$. From Theorem 3.2.6 we know that $\mathbf{v}^{\pi^k} \leq \mathbf{v}^{\pi^{\ell}} \leq \mathbf{v}^{\pi^*}$. Using Lemma 3.4.17, with $\pi'' = \pi^k$, $\pi' = \pi^{\ell}$ and $\pi = \pi^*$, we get that

$$\|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^{\ell}}\|_{\infty} \ \geq \ \frac{1-\gamma}{n} \|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^k}\|_{\infty} .$$

On the other hand, using Lemma 3.4.15, we get that

$$\|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^{\ell}}\|_{\infty} \ \leq \ \gamma^{\ell-k} \|\mathbf{v}^{\pi^*} - \mathbf{v}^{\pi^k}\|_{\infty} .$$

Combining the two inequalities we get

$$\gamma^L \ > \ \gamma^{\ell-k} \ \geq \ \frac{1-\gamma}{n} ,$$

a contradiction. □

Using Lemma 3.4.18, the following theorem is the derived in the same way as Theorem 2.2.38. Note that if the StrategyIteration algorithm is

given the strategy profile $\pi^0 = (\pi_1^0, \pi_2^0)$ as input, and the strategy π_1^0 is not an optimal counter-strategy against π_2^0, then such an optimal counter-strategy is found in the first iteration.

Theorem 3.4.19 *The* StrategyIteration *algorithm, starting from any initial strategy profile, terminates with an optimal strategy profile after at most* $(m+1)(1+\log_{1/\gamma}\frac{n}{1-\gamma}) = O(\frac{m}{1-\gamma}\log\frac{n}{1-\gamma})$ *iterations.*

Ye's [121] results and our results, combined with the recent results of Friedmann [38] and Fearnley [34], supply a *complete characterization* of the complexity of the policy/strategy iteration algorithm for MDPs/2TBSGs. The policy/strategy iteration algorithms are *strongly polynomial* for a fixed discount factor, but *exponential* for the total and average reward criteria. Furthermore, for every MDP (or 2TBSG) there exists a discount factor close enough to 1 such that every policy has the same set of improving switches as for the average reward criterion, see, e.g., Andersson and Miltersen [4]. By picking the discount factor sufficiently close to 1 it can then be shown that Fearnley's lower bound also holds for the discounted setting. See Hollanders, Delvenne, and Jungers [62] for details. Hence, when the discount factor is part of the input the number of iterations may also be exponential in the number of states and actions.

It remains a major open problem to obtain a polynomial time algorithm for the problem of solving 2TBSGs in general. One appealing approach for attacking this problem could be through the study of interior point methods. In an earlier work, Ye [120] gave a polynomial time algorithm for the analogous MDP problem. His algorithm uses interior point methods and its analysis relies again on the LP formulation of the MDP problem. Given the "deLPfication" of Ye's [121] analysis of the PolicyIteration algorithm carried out here (and in Section 2.2.4), one could hope that interior point methods for MDPs have natural interpretations that can be translated to 2TBSGs. It should also be pointed out that interior point methods for 2TBSGs can be obtained through P-matrix linear complementarity problems (see Section 3.3.3). It is not known how to obtain good bounds in this way, however.

Chapter 4

The RandomEdge algorithm

4.1 Introduction

In this chapter we present the results of Friedmann, Hansen and Zwick [42]. We prove a lower bound for the expected number of iterations performed by the simplex method when using the RandomEdge pivoting rule. We refer to the resulting algorithm as the RandomEdge algorithm. Recall that the RandomEdge pivoting rule repeatedly performs a uniformly random improving pivot until finding an optimal solution. It is, arguably, the simplest randomized pivoting rule for the simplex method. We show that there exist linear programs in standard form with n equations and $2n$ variables, for which the expected number of steps is $2^{\Omega(n^{1/4})}$. I.e., the lower bound is of subexponential form. The diameter of such a linear program is n, meaning that the result has no consequences for the polynomial Hirsch conjecture.

Prior to the results of Friedmann, Hansen and Zwick [42], super-polynomial lower bounds for the RandomEdge algorithm had only been proved for abstract problems. Matoušek and Szabó [89] showed that the expected number of iterations required for the RandomEdge algorithm to solve n-dimensional acyclic unique sink orientations (AUSOs) may be as large as $2^{\Omega(n^{1/3})}$. See Section 3.3.2 for a description of AUSOs. The best known upper bound for RandomEdge is still exponential (see Gärtner and Kaibel [48]). (For additional results regarding RandomEdge, see [17, 47, 50, 6].)

The study of randomized pivoting rules should not be confused with the study of the *average case* behavior of the simplex algorithm, with deterministic pivoting rules, under various probability distributions. (For more on this subject, see, e.g., [112, 2, 117, 1, 15].) Our result is also orthogonal to the *smoothed analysis* of the simplex algorithm performed by Spielman

and Teng [113], which may be used to explain why the simplex algorithm behaves so well in practice. Kelner and Spielman [75] use the results of [113] to obtain a new polynomial time algorithm for linear programming. Their algorithm applies the simplex algorithm to a suitably transformed linear program. For our lower bound we rely on the linear program having exactly the form we describe. It is not likely that the same behavior would be observed after transforming the linear program.

The linear programs on which RandomEdge performs an expected subexponential number of iterations are obtained using the close relation between the simplex method for linear programming and PolicyIteration algorithms for Markov decision processes. The connection we exploit was described in Section 2.4.1. More precisely, we will study the behavior of the RandomEdge algorithm when applied to MDPs with the total reward criterion. In this case, the RandomEdge algorithm works by repeatedly performing a uniformly random improving switch. We show that there exist MDPs with n states and $2n$ actions per state, for which the expected number of iterations is $2^{\Omega(n^{1/4})}$. The same lower bound then holds for the simplex method with the RandomEdge pivoting rule.

Recall that the StrategyIteration algorithm for solving 2-player turn-based stochastic games is a generalization of the PolicyIteration algorithm for solving MDPs, as discussed in Chapter 3. Recall also that Howard's improvement rule performs all improving switches *simultaneously*. Friedmann [38] started the line of work pursued here by showing that the StrategyIteration algorithm with Howard's improvement rule may require an exponential number of iterations to solve certain parity games. Parity games were presented in Section 3.3.6. Fearnley [34] adapted Friedmann's construction to work for MDPs. The paper of Melekopoglou and Condon [93] may be seen as a precursor of the papers of Friedmann [38] and Fearnley [34]. It deals, however, only with deterministic PolicyIteration algorithms that perform one switch at a time.

To obtain concrete linear programs on which the simplex algorithm with the RandomEdge pivoting rule performs an expected subexponential number of pivoting steps, we followed a similar path. We started by constructing parity games on which the RandomEdge algorithm performs an expected subexponential number of iterations. We converted these parity games into MDPs, and then to concrete linear programs. As the translation between our parity games and MDPs is a relatively simple step, we directly present the MDP version of our construction. As a consequence, our bound can be described and understood without knowing anything about parity games. In this chapter we will only present the lower bound for MDPs. We refer to

Friedmann, Hansen, and Zwick [42][1] for a description of the corresponding construction for parity games.

In high level terms, our MDPs, and the linear programs corresponding to them, are constructions of 'fault tolerant' binary counters. The challenge in designing such counters is making sure that they count 'correctly' under most sequences of random choices made by the RandomEdge algorithm. Our constructions are very different from the constructions used to obtain lower bounds for deterministic pivoting rules.

The chapter is organized as follows. In Section 4.2 we review MDPs with the total reward criterion. We introduce the notation used in this chapter, and, in particular, describe how to interpret certain weighted directed graphs as MDPs. In sections 4.3 and 4.4 we informally present a simplified version of our lower bound construction as a warm-up before presenting the full construction. In sections 4.5 and 4.6 we take a look at the shortcomings of the simplified construction and arrive at a complete description of our lower bound construction for RandomEdge. In sections 4.7 and 4.8 we then prove that the RandomEdge algorithm requires expected subexponential time to solve our MDPs.

4.2 Markov Decision Processes

In this chapter we are going to study MDPs with the total reward criterion. For a detailed description of such MDPs we refer to Section 2.4. We are going to use the graphical representation of MDPs. To be precise, we use a slightly more general class of graphs than those described in Definition 2.1.3.

Let $G = (V_0, V_R, E, c, p)$ be a weighted directed graph. Since we use the total reward criterion we implicitly assume that G has a special *sink* vertex $\textsc{t}$ that will be interpreted as the terminal state. The set of vertices $V = V_0 \cup V_R$ is partitioned into decision vertices (states) and randomization vertices. We require that there is at least one edge leaving every vertex, but otherwise there are no restrictions on the set of edges. In particular, we do not require the graph to be bipartite. Let $E_R = E \cap (V_R \times V \cup \{\textsc{t}\})$. For every randomization vertex $v \in V_R$, $p : E_R \to [0, 1]$ defines a probability distribution for the edges leaving v. Furthermore, $c : E \to \mathbb{R}$ assigns a reward to every edge, including edges from E_R. We may now define an MDP $M = M(G)$ from G in the following way. The set of states of M is $S = V_0$, and the set of actions is $A = E \cap (V_0 \times V)$. Consider the random

[1] A full version of the paper that contains a description of the construction for parity games is available at `http://cs.au.dk/~tdh/papers/random_edge.pdf`.

process that starts at a vertex $v \in V$, while it is in a randomization vertex it moves to a new vertex at random according to the probability distribution for the outgoing edges, and it stops when reaching a decision vertex (or the sink $\textsc{t}$). For every action $a = (u, v) \in A$, the probability of moving to a state $i \in S$ is the probability that the described process reaches the corresponding vertex in G when starting from v. Furthermore, the reward of a is the expected sum of rewards encountered by this process. In order for the MDP to be well-defined we require that the described process stops with probability 1 regardless of the starting state. For the remainder of the chapter we will study graphs of the form described above.

A policy $\pi : V_0 \to (V \cup \{\textsc{t}\})$ maps every decision vertex $u \in V_0$ to another vertex $v \in V \cup \{\textsc{t}\}$ where $(u, v) \in E$. Note that a policy can be equivalently interpreted as mapping vertices to edges. The definition given here is more convenient for our purposes, however. Recall that the value $\text{val}_\pi(v)$ of a vertex v (a state) is the expected total sum of rewards encountered before reaching the sink $\textsc{t}$. We will assign values to both decision vertices and randomization vertices. When the values are well-defined they can be found as the unique solution to the following system of equations:

$$\begin{aligned}
\text{val}_\pi(\textsc{t}) &= 0 \\
\text{val}_\pi(u) &= c(u, \pi(u)) + \text{val}_\pi(\pi(u)) \quad , \quad \text{if } u \in V_0 \\
\text{val}_\pi(u) &= \sum_{v:(u,v)\in E} p(u, v)(c(u, v) + \text{val}_\pi(v)) \quad , \quad \text{if } u \in V_R
\end{aligned}$$

The values are always well-defined when π satisfies the stopping condition (Definition 2.4.2). I.e., when $\textsc{t}$ is reached from every vertex with probability 1.

An edge $(u, v) \in V_0 \times (V \cup \{\textsc{t}\})$ is an improving switch with respect to a policy π if and only if:

$$c(u, v) + \text{val}_\pi(v) > \text{val}_\pi(u) \ .$$

Note that when the right-hand-side is subtracted from the left-hand-side we get the reduced cost as defined in Definition 2.4.5. Let $I(\pi) \subseteq E$ be the set of improving switches. The RandomEdge algorithm works in iteration starting from some initial policy π^0. Let π^k be the policy at the beginning of the k-th iteration. Then π^{k+1} is obtained from π^k by performing a uniformly random improving switch. This is repeated until there are no improving switches, in which case the resulting policy is optimal. For additional details about the PolicyIteration algorithm see section 2.4 and 2.2.2.

4.3 Simplified lower bound construction

We start with a description of a simplified version of the lower bound construction. As mentioned above, the construction may be seen as an implementation of 'fault tolerant' randomized counters. Let G_n be a family of lower bound MDPs, where n denotes the number of bits of a corresponding binary counter. In this section we describe, in a simpler setting, how to interpret policies for G_n as counter configurations, and we describe a sequence of improving switches that generates policies corresponding to all 2^n counter configurations.

The first important idea is to use rewards with a certain exponential structure. More precisely, we say that an action a has priority $p \in \mathbb{N}$, if the reward of a is $c(a) = (-N)^p$, where N is some integer. For the simplified construction it suffices to use $N = 2$. As a consequence, larger priorities dominate smaller priorities when computing the values of a policy. Intuitively, the RandomEdge pivoting rule is tricked into undoing previous work in order to reach actions with large even priorities. We should point out that the use of priorities is motivated by parity games (see Section 3.3.6).

A schematic description of a simplified lower bound MDP G_n is given in Figure 4.1. Vertices of V_0 are shown as circles, and actions are shown as arrows. Actions are labelled by 0 and 1 for reference. The initial policy consists of all the actions labelled 0. The small hexagons describe the priorities of the actions. All other rewards are zero. Note that in the simplified construction there are no randomization vertices.

The MDP of Figure 4.1 is composed of n identical levels, each corresponding to a single bit of the counter. The 1-st, i-th and $(i+1)$-st levels are shown explicitly in the figure. Levels are separated by dashed lines. In addition to the n levels there is a special $(n+1)$-st level with a *sink* vertex (terminal state) t. The i-th level contains four vertices of V_0, namely, a_i, b_i, u_i and w_i. Each of these four vertices is associated with two actions, labelled 0 and 1 in Figure 4.1. A policy π can, thus, be described by a zero-one vector with each entry defining the choice of some state. We use the shorthand notation $b_i(\pi) = 0$ to say that $\pi(b_i) = (b_i, u_1)$, and similarly for the remaining states and actions. Furthermore, when π is uniquely identified from the context, we simply write $b_i = 0$ instead of $b_i(\pi) = 0$.

It is worth pointing out that G_n does not satisfy the stopping condition. All the policies that we consider do satisfy the stopping condition, however. It is also not difficult to check that every cycle in G_n has negative reward. Hence, according to Lemma 2.4.11, from a policy that satisfies the stopping condition it is not possible to reach one that does not satisfy the stopping

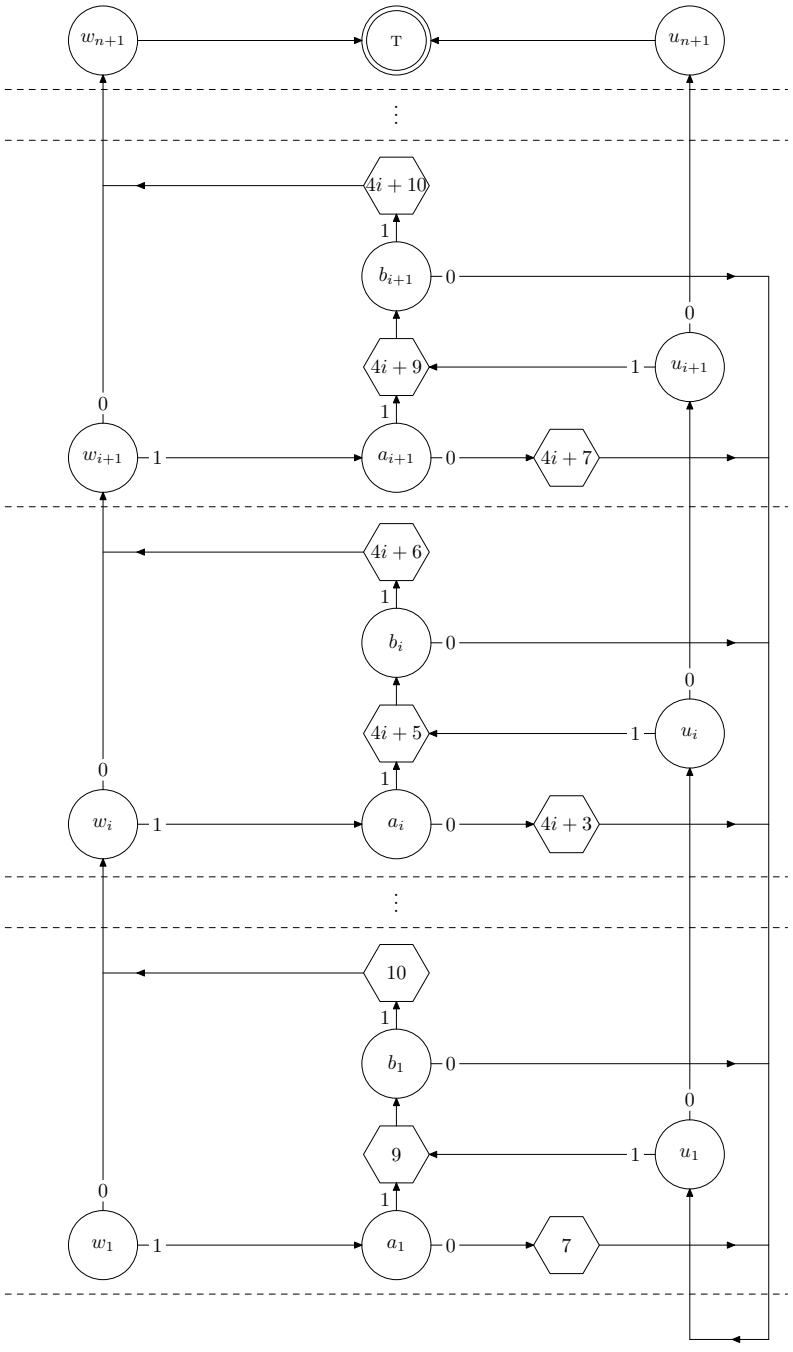


Figure 4.1: Simplified RandomEdge MDP construction.

condition by repeatedly performing improving switches.

A policy π is interpreted as a configuration $\mathfrak{b} \in \{0,1\}^n$ of the counter in the following straight-forward way. $\mathfrak{b}_i = 1$ if and only if $b_i(\pi) = 1$. Note that a bit is 1 exactly when the action with even priority in the corresponding level is used.

The incrementation of the binary counter by one is realized by transitioning through five phases. At the beginning of the first phase we require as an invariant that for all levels $i \in [n]$, b_i, a_i, u_i and w_i have the same values (i.e., matching choices). Also, let $k_\mathfrak{b} = \min\{i \in [n] \mid \mathfrak{b}_i = 0\}$ be the index of the bit that is going to be set, where $[n] = \{1, \ldots, n\}$. The five phases are then defined as follows:

1. Make $b_{k_\mathfrak{b}} = 1$.
2. Make $u_{k_\mathfrak{b}} = 1$ and $u_i = 0$, for $i < k_\mathfrak{b}$.
3. Make $b_i = 0$ and $a_i = 0$, for $i < k_\mathfrak{b}$.
4. Make $a_{k_\mathfrak{b}} = 1$.
5. Make $w_{k_\mathfrak{b}} = 1$ and $w_i = 0$, for $i < k_\mathfrak{b}$.

Note that after finishing the five phases we have $b_{k_\mathfrak{b}} = a_{k_\mathfrak{b}} = u_{k_\mathfrak{b}} = w_{k_\mathfrak{b}} = 1$ and $b_i = a_i = u_i = w_i = 0$ for $i < k_\mathfrak{b}$. Hence, the counter has been incremented by one, and the invariant is reestablished.

4.4 Improving switches

Next, let us briefly argue that all the changes made during the five phases are improving switches. In fact, in this section we list all the improving switches that could be made at different times during the five phases. In order for the RandomEdge improvement rule to generate the desired sequence of policies, we, thus, need to invent gadgets that delay the undesirable switches. We begin by introducing notation and terminology to describe policies and improving switches.

Let π be some policy. We say that a state is *stable* if it is not an improving switch to change the decision at that state, and that the state is *unstable* otherwise. For every policy π, either $b_i = 0$ or $b_i = 1$, and either b_i is stable or b_i is unstable. There are, thus, four different combinations of choices and stability. It will be convenient to define subsets of such combinations in order to succinctly describe policies and improving switches. For this

purpose we define the following symbols:

$$\begin{aligned}
* &= \{(0, stable), (0, unstable), (1, stable), (1, unstable)\} \\
0 &= \{(0, stable)\} \\
\overline{\uparrow} &= \{(0, stable), (0, unstable)\} \\
\uparrow &= \{(0, stable), (0, unstable), (1, stable)\} \\
1 &= \{(1, stable)\} \\
\downarrow &= \{(0, stable), (1, stable), (1, unstable)\} \\
\Downarrow &= \{(0, stable), (1, unstable)\}
\end{aligned}$$

We let $S = \{*, 0, \overline{\uparrow}, \uparrow, 1, \downarrow, \Downarrow\}$ be the set of symbols.

We let $\overline{\mathsf{b}}_i(\pi)$, or simply $\overline{\mathsf{b}}_i$ when π is clear from the context, be the pair representing the combination of b_i for π. Using this notation we for instance write $\overline{\mathsf{b}}_i(\pi) \in \uparrow$. Slightly abusing notation we will also write $b_i(\pi) \in \uparrow$ to mean that there exists a pair $(x, y) \in \uparrow$ such that $b_i(\pi) = x$. We refer to $b_i(\pi) \in \uparrow$ as a decision constraint, and a constraint involving the second component as a stability constraint. Note that decision constraints involving, for instance, $\uparrow$ are always satisfied. Let Π be the set of all policies. We can then, for instance, define $\Pi_{\overline{\mathsf{b}}_i(\pi) \in \uparrow} = \{\pi \in \Pi \mid \overline{\mathsf{b}}_i(\pi) \in \uparrow\}$ or $\Pi_{b_i(\pi) \in \uparrow} = \{\pi \in \Pi \mid b_i(\pi) \in \uparrow\}$. A statement such as $\Pi_{b_i(\pi) \in \uparrow} = \Pi_{\overline{\mathsf{b}}_i(\pi) \in \uparrow}$ then means that if $b_i = 1$ then b_i is stable.

Definition 4.4.1 (Chain) *We say that a set of vertices $C = \{c_1, \ldots, c_k\} \subseteq V_0$ is a* chain *(or path) if the vertices of C can be ordered such that for all $1 < i \leq k$, $(c_i, c_{i-1}) \in E$. We say that c_1 is the* head *of C, and that c_k is the* tail *of C.*

Note that vertices $u_1, \ldots, u_{n+1}$ form a chain starting from u_1 and moving to u_{n+1}. Similarly, the vertices $w_1, \ldots, w_{n+1}$ form a chain. We will refer to $u_1, \ldots, u_{n+1}$ as the *right lane*, and to $w_1, \ldots, w_{n+1}$ as the *left lane*. Note also that every prefix or suffix of a chain is itself a chain.

Definition 4.4.2 (Consecutive, closing) *Let $C = \{c_1, \ldots, c_k\} \subseteq V_0$ be a chain, and let c_0 be a vertex such that $(c_1, c_0) \in E$. We say that C is* consecutive *for a policy π (and c_0) if there exists an index $1 \leq j \leq k+1$ such that for all $1 \leq i < j$, $\pi(c_i) = c_{i-1}$, and for all $i \geq j$, $\pi(c_i) \neq c_{i-1}$. Furthermore, we say that C is* closing *for π (and c_0) if C is consecutive for π (and c_0), c_i is stable for $i \neq j$, and c_j is unstable. If C is closing and $j = k+1$ we say that C is* closed

Definition 4.4.3 (Opening) *Let $C = \{c_1, \ldots, c_k\} \subseteq V_0$ be a chain, let c_0 be a vertex such that $(c_1, c_0) \in E$, and let π be a policy. We say that C is* opening *for π (and c_0) if c_i, $1 \leq i \leq k$, is stable if and only if $\pi(c_i) \neq c_{i-1}$. If, additionally, every vertex of C is stable we say that C is* open.

The improving switches appearing throughout the five phases can briefly be described as follows. The key actions are those with priority $4i + 6$. For all bits $i \in [n]$ such that $\mathfrak{b}_i = 0$, b_i is unstable in phase 1, and in the remaining phases if $i > k_{\mathfrak{b}}$. When $b_{k_{\mathfrak{b}}}$ is changed to 1, the best way to reach $b_{k_{\mathfrak{b}}}$ from lower levels, while $a_{k_{\mathfrak{b}}} = 0$, is through the right lane. Hence, the right lane will be adjusted one level at a time in phase 2 until there is access from u_1 to $b_{k_{\mathfrak{b}}}$. Once this happens, all b_i and a_i vertices for $i < k_{\mathfrak{b}}$ become unstable, and the lower bits are reset in phase 3. From the start of phase 2 it is improving to switch $a_{k_{\mathfrak{b}}}$ to 1, and this happens in phase 4. When $a_{k_{\mathfrak{b}}} = 1$ the left lane again becomes useful, and it is adjusted one level at a time in phase 5. In fact, it was an improvement to switch $w_{k_{\mathfrak{b}}}$ to 1 from the beginning of phase 3, but such a change would be of little importance as it would not affect the improving switches at other vertices. b_i, for $i < k_{\mathfrak{b}}$, becomes unstable again when the access to $b_{k_{\mathfrak{b}}}$ through the left lane is restored.

We next formally define the set of policies $\Pi_{\mathfrak{b},p} \subseteq \Pi$ of phase $p \in \{1, \ldots, 5\}$ for a given setting $\mathfrak{b} \in \{0,1\}^n$ of the binary counter. We do this by assigning a symbol $s_v \in S = \{*, 0, \overline{\uparrow}, \uparrow, 1, \downarrow, \Downarrow\}$ to every vertex $v \in V_0 \setminus \{u_{n+1}, w_{n+1}\}$. The assignment of symbols is shown in Table 4.1. For every $i \in [n]$, we use a succinct notation tuple to provide information describing the i-th level: $\overline{\mathtt{b}}_i(\pi)\ \overline{\mathtt{a}}_i(\pi)\ \ \overline{\mathtt{u}}_i(\pi)\ \overline{\mathtt{w}}_i(\pi)$. We then have $\pi \in \Pi_{\mathfrak{b},p}$ if and only if the following three requirements are met. We use $v(\pi)$ as a generic way of writing $b_i(\pi)$, $a_i(\pi)$, $u_i(\pi)$, and $w_i(\pi)$, for $v = b_i$, $v = a_i$, $v = u_i$, and $v = w_i$, respectively.

- $v(\pi) \in s_v$ for all $v \in V_0 \setminus \{u_{n+1}, w_{n+1}\}$. I.e., the decision constraints should be satisfied for all vertices.
- The vertices $u_1, \ldots, u_{k_{\mathfrak{b}}}$ are a consecutive chain for π (and $\mathrm{B}_{k_{\mathfrak{b}}}$) in phase 2, and $w_1, \ldots, w_{k_{\mathfrak{b}}}$ are a consecutive chain for π (and $a_{k_{\mathfrak{b}}}$) in phase 5.
- There is at least one vertex v with symbol $\uparrow$ such that $v(\pi) = 0$, or at least one vertex v with symbol $\downarrow$ or $\Downarrow$ such that $v(\pi) = 1$.

In order to also describe which edges are improving switches, we make a similar definition in which constraints for stability should also be satisfied.

Phase	$i > k_{\mathfrak{b}}$		$i = k_{\mathfrak{b}}$	$i < k_{\mathfrak{b}}$
	$\mathfrak{b}_i = 1$	$\mathfrak{b}_i = 0$		
1	1 1 1 1	$\overline{\uparrow}$ 0 0 0	$\uparrow$ 0 0 0	1 1 1 1
2	1 1 1 1	$\overline{\uparrow}$ 0 0 0	1 $\overline{\uparrow}$ $\uparrow$ 0	1 1 $\downarrow$ 1
3	1 1 1 1	$\overline{\uparrow}$ 0 0 0	1 $\overline{\uparrow}$ 1 $*$	$\Downarrow$ $\Downarrow$ 0 1
4	1 1 1 1	$\overline{\uparrow}$ 0 0 0	1 $\uparrow$ 1 $*$	0 0 0 1
5	1 1 1 1	$\overline{\uparrow}$ 0 0 0	1 1 1 $\uparrow$	$\overline{\uparrow}$ 0 0 $\downarrow$

Table 4.1: Policies and improving switches of the five phases for the simplified construction.

I.e., let $\overline{\Pi}_{\mathfrak{b},p} \subseteq \Pi$ be defined like $\Pi_{\mathfrak{b},p}$, but where the constraints should be satisfied both for decisions and stability. More precisely, we use the same assignment of symbols to define $\overline{\Pi}_{\mathfrak{b},p}$ as we did for $\Pi_{\mathfrak{b},p}$. We then have $\pi \in \overline{\Pi}_{\mathfrak{b},p}$ if and only if the following three requirements are met.

- $\overline{v}(\pi) \in s_v$ for all $v \in V_0 \setminus \{u_{n+1}, w_{n+1}\}$. I.e., both decision constraints and stability constraints should be satisfied for all vertices.
- The vertices $u_1, \ldots, u_{k_{\mathfrak{b}}}$ is a *closing* (and not just consecutive) chain for π (and $\textsc{b}_{k_{\mathfrak{b}}}$) in phase 2, and $w_1, \ldots, w_{k_{\mathfrak{b}}}$ is a *closing* chain for π (and $a_{k_{\mathfrak{b}}}$) in phase 5.
- There is at least one vertex v with symbol $\uparrow$ such that $v(\pi) = 0$, or at least one vertex v with symbol $\downarrow$ or $\Downarrow$ such that $v(\pi) = 1$.

Note that the second requirement about the vertices in the left and right lane, respectively, with $\uparrow$ and $\downarrow$ symbols, corresponds exactly to one level of the lanes being adjusted at a time.

It is possible to prove the following lemma. We will not include the proof here, but we will prove a similar lemma for the full construction.

Lemma 4.4.4 *For every setting $\mathfrak{b} \in \{0,1\}^n$ of the binary counter and every phase $p \in \{1, \ldots, 5\}$, we have $\Pi_{\mathfrak{b},p} = \overline{\Pi}_{\mathfrak{b},p}$.*

The arrows of Table 4.1 indicate that a state becomes unstable at some point during a phase. Note, in particular, that the changes that are made during the five phases correspond directly to arrows other than $\overline{\uparrow}$, and that in each phase such arrows are present. The $\overline{\uparrow}$ symbols indicate improving switches that are not allowed to happen during a phase, and, hence, must be delayed in order for the RandomEdge pivoting rule to generate the correct sequence of counter configurations.

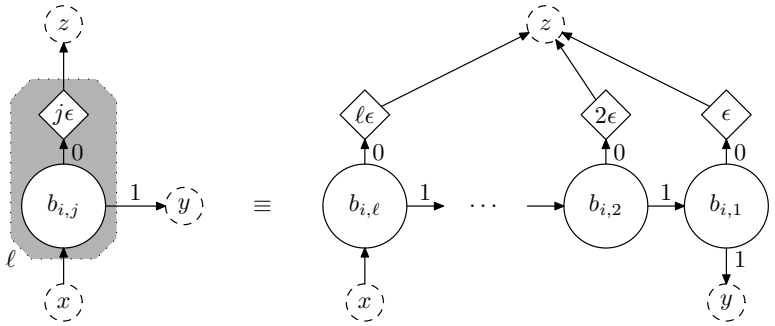


Figure 4.2: A delay gadget used for the lower bound construction.

Let us also note that two important types of delays were required in order to generate the sequence of policies corresponding to all 2^n counter configurations. First, although changing b_i to 1 for $i > k_{\mathfrak{b}}$ remained an improving switch at all times, it was delayed until the counter reached the correct configuration. Second, changing $a_{k_{\mathfrak{b}}}$ to 1 was delayed until after all the lower levels had been reset in phase 3. These delays were shown in Table 4.1 with the $\overline{\uparrow}$ symbol.

4.5 Delaying improving switches

To delay an improving switch at b_i from 0 to 1, we replace b_i by a chain of vertices, $b_{i,j}$ for $j \in [\ell]$, such that a specific sequence of improving switches must be performed to get the same effect. Such a gadget is shown in Figure 4.2. The shaded octagon labelled ℓ indicates that the vertices inside are copied ℓ times to form a chain. The diamond-shaped vertices show the rewards of actions. The rewards are necessary to get the correct improving switches. ϵ is picked sufficiently small such that these rewards are secondary compared to priorities. We let $\epsilon = N^{-(4n+8)}$, where N is used to define priorities. x, y and z are generic vertices. Actions are again labelled by 0 and 1 for reference.

We refer jointly to the $b_{i,j}$ vertices as the b_i-chain. Instead of saying that the b_i-chain is consecutive (or closing or opening) for π (and y), we simply say that the b_i-chain is consecutive (or closing or opening) for π. I.e., y is the natural continuation of the b_i-chain, and we do not point this out explicitly.

Note that it is only preferred for $b_{i,j}$ to use the action labelled 1 if it is possible to reach y, and $\mathrm{val}_\pi(y) > \mathrm{val}_\pi(z) + j\epsilon$. This proves the following simple lemma.

Lemma 4.5.1 *Let π be a policy. If $\mathrm{val}_\pi(y) \leq \mathrm{val}_\pi(z)$ then the b_i-chain is opening for π. If $\mathrm{val}_\pi(y) > \mathrm{val}_\pi(z) + \ell\epsilon$ and the b_i-chain is consecutive for π then the b_i-chain is closing for π.*

It will be convenient to describe the choices for the states of the b_i-chain and the corresponding improving switches jointly. We use the same definition of $b_{i,j}(\pi)$ and $\overline{\mathsf{b}}_{i,j}(\pi)$ as in Sections 4.3 and 4.4. Define:

$$b_i(\pi) = \begin{cases} 1 & \text{if } b_{i,j} = 1 \text{ for all } j \in [\ell] \\ 0 & \text{otherwise} \end{cases}$$

We let $\overline{\mathsf{b}}_i(\pi) \in S^\ell$ be a vector of symbols such that $(\overline{\mathsf{b}}_i(\pi))_j = \overline{\mathsf{b}}_{i,j}(\pi)$. We overload the use of symbols by writing:

$$\overline{\mathsf{b}}_i(\pi) \in \begin{cases} 0 & \text{if } \overline{\mathsf{b}}_{i,j}(\pi) \in 0 \text{ for all } j \in [\ell] \\ \uparrow & \text{if } \overline{\mathsf{b}}_{i,j}(\pi) \in \uparrow \text{ for all } j \in [\ell], \text{ and the } b_i\text{-chain is consecutive} \\ \overline{\uparrow} & \text{if } b_i = 0 \text{ and } \overline{\mathsf{b}}_i(\pi) \in \uparrow \\ 1 & \text{if } \overline{\mathsf{b}}_{i,j}(\pi) \in 1 \text{ for all } j \in [\ell] \\ \Downarrow & \text{if } \overline{\mathsf{b}}_{i,j}(\pi) \in \Downarrow \text{ for all } j \in [\ell] \end{cases}$$

$a_i(\pi)$ and $\overline{\mathsf{a}}_i(\pi)$ are defined analogously. Note that if the b_i-chain is opening for π then $\overline{\mathsf{b}}_i(\pi) \in \Downarrow$, and that if the b_i-chain is closing for π then $\overline{\mathsf{b}}_i(\pi) \in \uparrow$. Notice also that if the b_i-chain is closing for π, and π' is obtained from π by performing the single improving switch in the b_i-chain, then the b_i-chain is also closing for π'. We again slightly abuse the notation and write $b_i(\pi) \in \uparrow$ to mean that there exist pairs $(x_1, y_1), \ldots, (x_\ell, y_\ell) \in \uparrow$ such that $b_{i,j} = x_j$, for all $j \in [\ell]$.

The progress of two closing chains during some phase can be viewed as a *competition* where both chains advance with equal probability in every iteration. By using Chernoff bounds it is not difficult to analyze the probability that one chain makes a certain amount of progress before the other is closed. Specifically, we make use of the following lemma.

Lemma 4.5.2 *The probability that a closing chain acquires b new edges before a different closing chain of length a closes completely is at most $\mathbf{e}^{-\frac{1}{2}(b-a)^2/(b+a)}$.*

Proof: As the probability of each of the two competing cycles to acquire a new edge is the same, we are essentially looking at the following 'experiment'. A fair coin is repeatedly tossed until either b heads or a tails are observed,

for some $a < b$. We would like to bound the probability that b heads are observed before a heads.

The probability of getting b heads before a tails is exactly the probability of getting *less* than a tails in the first $a+b-1$ tosses, which is at most the probability of getting *at most* a heads in the first $a+b$ tosses. The above probability can be easily bounded using the Chernoff bound. Let X be the number of heads observed in the first $a+b$ tosses. Then $\mu = E[X] = \frac{a+b}{2}$. The Chernoff bound, in the case of a fair coin (see Corollary 4.10 on page 71 of [94]), states that for every $0 < \delta < 1$ we have

$$\Pr[X \leq (1-\delta)\mu] \;\leq\; \mathrm{e}^{-\mu\delta^2} \;.$$

Let $\delta = \frac{b-a}{b+a}$. Then,

$$(1-\delta)\mu \;=\; a \quad , \quad \mu\delta^2 \;=\; \frac{(b-a)^2}{2(b+a)} \;,$$

and the claim of the lemma follows. □

Another type of competition happens when a number of opening chains must open completely before some other closing chain makes a certain amount of progress. This was the case in phase 3 for the simplified construction. Competitions between closing chains and opening chains are heavily biased towards the opening chains, however, since these chains are associated with many more improving switches. To analyze such competitions we use the following lemma.

Lemma 4.5.3 *Let a be the total length of all the chains that are currently opening. Then, the probability that a closing chain acquires at least b new edges before all opening chains open completely is at most $\frac{a}{2^b}$.*

Proof: Let $p(a,b)$ be the probability that the closing cycles acquire b new edges before the opening cycles, which currently have a edges pointing into them, open completely. We can ignore switches that do not belong to the opening cycles or the closing cycle. Thus, the probability that the next relevant edge chosen belongs to the opening cycles is $\frac{a}{a+1}$, while the probability that it belongs to the closing cycle is $\frac{1}{a+1}$. We thus get the following recurrence relation:

$$\begin{aligned} p(a,0) &= 1 \\ p(0,b) &= 0 \\ p(a,b) &= \frac{a}{a+1}p(a-1,b) + \frac{1}{a+1}p(a,b-1) \end{aligned}$$

We can now easily prove by induction that $p(a,b) \leq \frac{a}{2^b}$. For $a = 0$ or $b = 0$ the inequality clearly holds. Otherwise we have:

$$\begin{aligned} p(a,b) &= \frac{a}{a+1}p(a-1,b) + \frac{1}{a+1}p(a,b-1) \\ &\leq \frac{a}{a+1}\frac{a-1}{2^b} + \frac{1}{a+1}\frac{a}{2^{b-1}} \\ &= \frac{a(a-1)+2a}{(a+1)2^b} = \frac{a}{2^b}\,. \end{aligned}$$

□

As a first step towards the full construction, b_i and a_i vertices are replaced by chains of appropriate lengths. A bit i is still interpreted as being set when $b_i = 1$, and Table 4.1 still describes policies and improving switches of the five phases, now using the extended definitions of $\overline{\mathtt{b}}_i(\pi)$ and $\overline{\mathtt{a}}_i(\pi)$. It is, in fact, possible to select the lengths of the chains in such a way that all 2^n counter configurations are generated with high probability. To do so, the lengths of the chains would have to be exponential in n, however, due to progress building up from previous increments. Next, we will see how to modify the construction such that the progress of b_i chains of higher bits is also reset during the resetting phase (phase 3 for the simplified construction).

4.6 Resetting chains of higher bits

In order to reset partially closed b_i chains of higher bits when the counter is incremented, no vertex in the i-th level can directly use the action with priority $4i + 6$. If this was possible there would be no gain in moving to the lower levels instead. To fix this problem we introduce a randomization vertex B_i, for all $i \in [n]$. B_i goes to w_{i+1} and the large even priority with a small probability $\epsilon = N^{-(4n+8)}$. A bit i will then be set when we repeatedly cycle back to B_i until eventually moving to w_{i+1} with probability 1.

In addition to the vertex B_i we also introduce a chain of $c_{i,j}$ vertices, for $j \in [g]$, identical to the b_i chain except that the actions lead to w_1 instead of u_1 and have priority 6. The full lower bound construction is shown in Figure 4.3 above. B_i goes to $c_{i,g}$ with probability $1 - \epsilon$. $c_i(\pi)$ and $\overline{\mathtt{c}}_i(\pi)$ are defined analogously to $b_i(\pi)$ and $\overline{\mathtt{b}}_i(\pi)$, and we interpret the i-th bit as being set, $\mathfrak{b}_i = 1$, if and only if $b_i(\pi) = 1$ and $c_i(\pi) = 1$. The lengths of the b_i-, c_i- and a_i-chains are parameterized by ℓ_i, g and h, respectively. For the full construction we use $N = 3n + 1$ to define priorities.

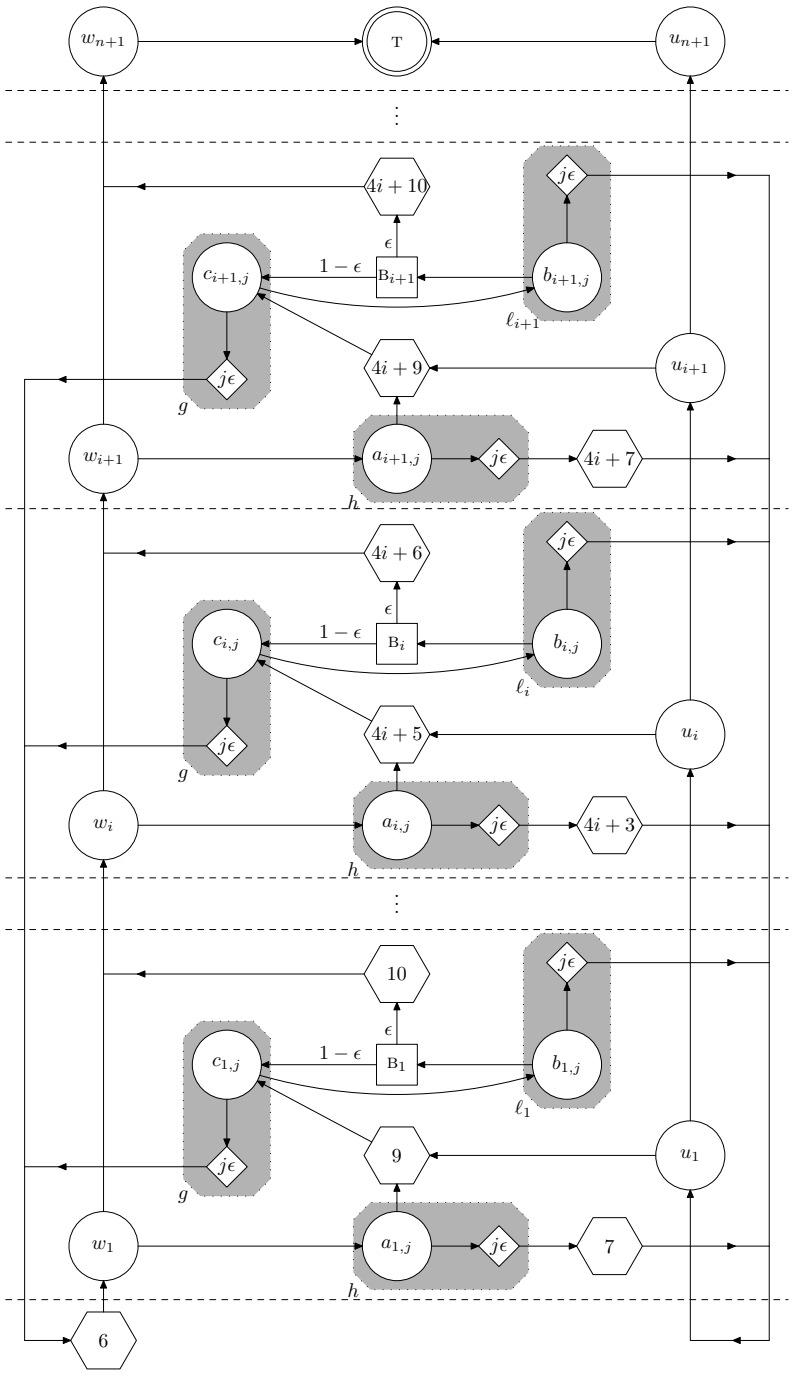


Figure 4.3: Full RandomEdge MDP construction. The interpretation of the shaded octagons is shown in Figure 4.2.

For a tuple $\zeta = (n, (\ell_i)_{1 \leq i \leq n}, g, h)$, with $n, \ell_i, g, h > 0$, we let $G_\zeta = (V_0, V_R, E, c, p)$ be the graph defined by using the parameters of ζ. Formally, G_ζ is defined as follows.

$$\begin{aligned} V_0 &:= \{a_{i,j} \mid i \in [n], j \in [h]\} \cup \{b_{i,j} \mid i \in [n], j \in [\ell_i]\} \cup \\ &\quad \{c_{i,j} \mid i \in [n], j \in [g]\} \cup \{u_i, w_i \mid i \in [n+1]\} \\ V_R &:= \{\text{B}_i \mid i \in [n]\} \end{aligned}$$

The edges and their priorities, rewards, and probabilities are defined in Table 4.2. Remember that an action a with priority p has reward $c(a) = (-N)^p$. If an edge has no priority the entry is left blank. Whenever j is used in the table we assume that $j \geq 2$. We label the edges in the same way as we did for the simplified construction. The initial policy π^0 uses the edge with label 0 from every state.

Lemma 4.6.1 *The initial policy π^0 satisfies the stopping condition, and every policy reachable from π^0 by performing improving switches satisfies the stopping condition.*

Proof: It is not difficult to see that π^0 satisfies the stopping condition. Indeed, both the left and right lane moves directly to the sink T, and from every other vertex the lanes are reached with probability 1.

The cycle $C_i = \text{B}_i, c_{i,g}, \ldots, c_{i,1}, b_{i,\ell_i}, \ldots, b_{i,1}, \text{B}_i$ has reward 0, but the probability of staying in this cycle is 0 for any policy. On the other hand, every other cycle has negative reward. Indeed, at every level i it is only possible to move to a higher level, move to the first level, or cycle temporarily in C_i with reward 0. When moving up there is a positive reward, but when moving to the first level there is a larger negative reward. Hence, every policy that does not satisfy the stopping condition has states with negative values for the average reward criterion (minus infinity for the total reward criterion). The lemma then follows from Lemma 2.4.11. I.e., since all values of π^0 are 0 for the average reward criterion, such policies can not be reached by performing improving switches. □

It follows from Lemma 4.6.1 and Lemma 2.4.10 that a lower bound for the expected number of iterations performed by the RandomEdge algorithm on G_ζ, starting from π^0, also gives a lower bound for the expected number of iterations performed by the RandomEdge algorithm for the corresponding linear program. Let us note that we could make all policies satisfy the stopping condition by always moving from u_i to u_{i+1} and from w_i to w_{i+1} with probability ϵ. This would not affect the analysis.

Vertex V_0	Successors in E	Priority	Reward	Label
$a_{i,1}$	u_1	$4i+3$	$\epsilon - N^{4i+3}$	0
	$c_{i,g}$	$4i+5$	$-N^{4i+5}$	1
$a_{i,j}$	u_1	$4i+3$	$j\epsilon - N^{4i+3}$	0
	$a_{i,j-1}$		0	1
$b_{i,1}$	u_1		ϵ	0
	$\mathrm{B}i$		0	1
$b_{i,j}$	u_1		$j\epsilon$	0
	$b_{i,j-1}$		0	1
$c_{i,1}$	w_1	6	$\epsilon + N^6$	0
	b_{i,ℓ_i}		0	1
$c_{i,j}$	w_1	6	$j\epsilon + N^6$	0
	$c_{i,j-1}$		0	1
u_{n+1}	T		0	0
u_i	u_{i+1}		0	0
	$c_{i,g}$	$4i+5$	$-N^{4i+5}$	1
w_{n+1}	T		0	0
w_i	w_{i+1}		0	0
	$a_{i,h}$		0	1
Vertex V_R	Successors in E_R	Priority	Reward	Probability
$\mathrm{B}i$	$c_{i,g}$		0	$1-\epsilon$
	w_{i+1}	$4i+6$	N^{4i+6}	ϵ

Table 4.2: Priorities, edges, and transition probabilities of G_ζ where $\zeta = (n, (\ell_i)_{1\le i\le n}, g, h)$.

The reason for the additional c_i-chain is that if $\mathfrak{b}_i = 0$ and $\mathrm{val}_\pi(w_1) = \mathrm{val}_\pi(u_1)$, then the small priority 6 obtained by moving to w_1 causes the c_i-chain to be opening while the b_i-chain is closing. On the other hand, if $\mathrm{val}_\pi(w_1)+1 \le \mathrm{val}_\pi(u_1)$, because we are in a reset phase, then the c_i-chain is closing while the b_i-chain is opening. Moreover, this is true even for higher bits than the one that was just set.

Incrementing the counter still happens through a number of phases. With the c_i-chains we get two more phases where c_i is set to 1 and 0, respectively, giving a total of seven phases. The invariant for the beginning of the first phase again requires that b_i, c_i, a_i, u_i, and w_i all have the same values. Let $\mathfrak{b} \in \{0,1\}^n$ be the counting configuration corresponding to the policy at the beginning of the first phase, and let $k_\mathfrak{b} = \min\{i \in [n] \mid \mathfrak{b}_i = 0\}$.

The seven phases are then defined as follows:

1. Make $b_{k_\mathfrak{b}} = 1$.
2. Make $c_{k_\mathfrak{b}} = 1$.
3. Make $u_{k_\mathfrak{b}} = 1$ and $u_i = 0$, for $i < k_\mathfrak{b}$.
4. Make $b_i = 0$ and $a_i = 0$, for $i < k_\mathfrak{b}$. Make $b_i = 0$, for all $i \in [n]$ such that $\mathfrak{b}_i = 0$.
5. Make $a_{k_\mathfrak{b}} = 1$.
6. Make $w_{k_\mathfrak{b}} = 1$ and $w_i = 0$, for $i < k_\mathfrak{b}$.
7. Make $c_i = 0$, for all $i \in [n]$ such that $\mathfrak{b}_i = 0$.

Phases 2 and 7 are new compared to the simplified construction, and in phase 4 (previously phase 3) higher b_i chains are also reset.

We again formally define the set of policies $\Pi_{\mathfrak{b},p} \subseteq \Pi$ that belong to phase $p \in \{1, \ldots, 7\}$ for a given setting $\mathfrak{b} \in \{0,1\}^n$ of the binary counter. As for the simplified construction, we specify a symbol s_v for every vertex $v \in \{u_1, \ldots, u_n, w_1, \ldots, w_n\}$, and a symbol s_d for every chain d of b_i, c_i, and a_i vertices. The assignment of symbols is shown in Table 4.3. For every $i \in [n]$, we use the notation tuple: $\overline{\mathtt{b}}_i(\pi)\ \overline{\mathtt{c}}_i(\pi)\ \overline{\mathtt{a}}_i(\pi)\ \ \overline{\mathtt{u}}_i(\pi)\ \overline{\mathtt{w}}_i(\pi)$. We let $\Pi_{\mathfrak{b},p}$ be the set of policies that satisfy all decision constraints. More precisely, $\pi \in \Pi_{\mathfrak{b},p}$ if and only if the following three constraints are satisfied. For a chain d we use $d(\pi)$ to denote $b_i(\pi)$, $c_i(\pi)$, or $a_i(\pi)$ correspondingly. We use similar notation for u_i and w_i vertices.

- $v(\pi) \in s_v$ for all $v \in \{u_1, \ldots, u_n, w_1, \ldots, w_n\}$, and $d(\pi) \in s_d$ for every chain d. I.e., the decision constraints should be satisfied for all vertices.
- The vertices $u_1, \ldots, u_{k_\mathfrak{b}}$ are a consecutive chain for π (and $\textsc{B}_{k_\mathfrak{b}}$) in phase 3, and $w_1, \ldots, w_{k_\mathfrak{b}}$ are a consecutive chain for π (and $a_{k_\mathfrak{b}}$) in phase 6.
- There is at least one vertex or chain v with symbol $\uparrow$ such that $v(\pi) = 0$, or at least one vertex or chain v with symbol $\downarrow$ or $\Downarrow$ such that $v(\pi) \neq 0$.

We also let $\overline{\Pi}_{\mathfrak{b},p} \subseteq \Pi$ be defined like $\Pi_{\mathfrak{b},p}$, with the same symbols, but where both decision constraints and stability constraints should be satisfied. More precisely, $\pi \in \overline{\Pi}_{\mathfrak{b},p}$ if and only if the following four requirements are met. For a chain d we use $\overline{\mathtt{d}}(\pi)$ to denote $\overline{\mathtt{b}}_i(\pi)$, $\overline{\mathtt{c}}_i(\pi)$, or $\overline{\mathtt{a}}_i(\pi)$ correspondingly. We use similar notation for u_i and w_i vertices.

- $\overline{\mathtt{v}}(\pi) \in s_v$ for all $v \in \{u_1, \ldots, u_n, w_1, \ldots, w_n\}$, and $\overline{\mathtt{d}}(\pi) \in s_d$ for every chain d. I.e., both decision constraints and stability constraints should be satisfied.
- The vertices $u_1, \ldots, u_{k_\mathfrak{b}}$ is a *closing* chain for π (and $\textsc{b}_{k_\mathfrak{b}}$) in phase 3, and $w_1, \ldots, w_{k_\mathfrak{b}}$ is a *closing* chain for π (and $a_{k_\mathfrak{b}}$) in phase 6.
- There is at least one vertex or chain v with symbol $\uparrow$ such that $v(\pi) = 0$, or at least one vertex or chain v with symbol $\downarrow$ or $\Downarrow$ such that $v(\pi) \neq 0$.
- If $\overline{\mathtt{b}}_i(\pi) \in \uparrow$ or $\overline{\mathtt{b}}_i(\pi) \in \overline{\uparrow}$ then we require the b_i-chain to be closing instead of just consecutive. The same requirements are made for $\overline{\mathtt{c}}_i(\pi)$ and $\overline{\mathtt{a}}_i(\pi)$.

We will prove the following lemma in Section 4.8.

Lemma 4.6.2 *For every setting $\mathfrak{b} \in \{0,1\}^n$ of the binary counter and every phase $p \in \{1, \ldots, 7\}$, we have $\Pi_{\mathfrak{b},p} = \overline{\Pi}_{\mathfrak{b},p}$.*

Phase	$i > k_\mathfrak{b}$		$i = k_\mathfrak{b}$	$i < k_\mathfrak{b}$
	$\mathfrak{b}_i = 1$	$\mathfrak{b}_i = 0$		
1	1 1 1 1 1	$\overline{\uparrow}$ 0 0 0 0	$\uparrow$ 0 0 0 0	1 1 1 1 1
2	1 1 1 1 1	$\overline{\uparrow}$ 0 0 0 0	1 $\uparrow$ 0 0 0	1 1 1 1 1
3	1 1 1 1 1	$\overline{\uparrow}$ 0 0 0 0	1 1 $\overline{\uparrow}$ $\uparrow$ 0	1 1 1 $\downarrow$ 1
4	1 1 1 1 1	$\Downarrow$ $\overline{\uparrow}$ 0 0 0	1 1 $\overline{\uparrow}$ 1 $*$	$\Downarrow$ 1 $\Downarrow$ 0 1
5	1 1 1 1 1	0 $\overline{\uparrow}$ 0 0 0	1 1 $\uparrow$ 1 $*$	0 1 0 0 1
6	1 1 1 1 1	0 $\overline{\uparrow}$ 0 0 0	1 1 1 1 $\uparrow$	$\overline{\uparrow}$ 1 0 0 $\downarrow$
7	1 1 1 1 1	$\overline{\uparrow}$ $\Downarrow$ 0 0 0	1 1 1 1 1	$\overline{\uparrow}$ $\Downarrow$ 0 0 0

Table 4.3: Strategies and improving switches of the seven phases for the full lower bound construction.

4.7 Transition probabilities

Next, we specify the parameters ℓ_i, g and h in order to make the RandomEdge pivoting rule generate the desired policies with high probability.

Let $\Pi_{\mathfrak{b},p}$ be the set of policies that belong to phase p, where $p \in \{1, \ldots, 7\}$, for a given setting $\mathfrak{b} \in \{0,1\}^n$ of the counter. The sets $\Pi_{\mathfrak{b},p}$ are defined by

Table 4.3, where the improving switches from each such policy are also specified. Our goal in this section is to show that if RandomEdge is run on a policy from $\Pi_{\mathfrak{b},p}$, then with a very high probability a policy from $\Pi_{\mathfrak{b},p+1}$, or from $\Pi_{\mathfrak{b}+1,1}$, if $p = 7$, is encountered. We show that the probability that this does not hold is $O(\mathbf{e}^{-n})$. The probability that one of the $7 \cdot 2^n$ phases fails is thus $O((2/\mathbf{e})^n)$, i.e., exponentially small.

As noted in Section 4.5, we can view each phase as being composed of several simultaneous competitions between various chains, some of which are trying to open while others are trying to close. In phases 4 and 7, it should be the case that chains that are opening, open completely before any other chain closes completely. In the remaining phases, several chains are closing, and we require certain chains to be the first to close completely. The competitions are described in Table 4.3 by the arrows. Chains with the symbol $\overline{\uparrow}$ are not allowed to close completely before chains with the $\Downarrow, \uparrow$ and $\downarrow$ symbols open or close. Note that the two lanes form chains of length at most n.

We let $\nu(a)$ be the value of b for which the probability $\mathbf{e}^{-\frac{1}{2}(b-a)^2/(b+a)}$ of Lemma 4.5.2 is at most $\mathbf{e}^{-n}$. It is not difficult to check that $\nu(a) = a + n + \sqrt{n^2 + 4an}$. In particular, we have $\nu(n) = (2 + \sqrt{5})n < 5n$, and $\nu(a) \leq a + 3\sqrt{an}$, for $a \geq 2n$. We also let $\rho = 2n$.

If d is a chain that is closing at a certain phase, but is not supposed to win the competition of the phase, we refer to the number of edges currently pointing into d as the *noise level* of d. To prove that competitions are won by the intended candidates, we prove that the probability that the noise level of any of the other cycles exceeds the noise bounds specified in Table 4.4, at any time during the phase, is exponentially small. Three different noise bounds ξ^b, ξ^c, ξ^a are specified for b_i-chains, c_i-chains and a_i-chains, respectively. These bounds are only relevant when the corresponding chains are closing, but not closing completely, during the phase. I.e., the bounds are only relevant for chains with the symbol $\overline{\uparrow}$ in Table 4.3. A phase ends successfully if the noise level of each chain never reaches the length of that chain.

It is not difficult to prove by induction that the probability that the noise level of a chain exceeds the noise bound given in Table 4.4 is exponentially small. Let us look, for example, at the noise levels of the b_i-chains. In phase 4, no b_i-chain is closing, so $\xi^b = 0$ is a (vacuous) upper bound on the noise level of closing b_i-chains. The same holds for phase 5. Some b_i-cycles are closing in phase 6. All these cycles, however, are completely open at the beginning of phase 6. The competition in phase 6 is with the left lane, which forms a chain of length at most n. Thus, by Lemma 4.5.2, the

Phase	Noise Bounds for $\overline{\uparrow}$-chains		
	ξ^b	ξ^c	ξ^a
1	$\nu(n) + \rho + \nu(\ell_{k_\mathfrak{b}})$	0	0
2	$\nu(n)+\rho+\nu(\ell_{k_\mathfrak{b}})+\nu(g)$	0	0
3	$2\nu(n)+\rho+\nu(\ell_{k_\mathfrak{b}})+\nu(g)$	0	$\nu(n)$
4	0	ρ	$\nu(n)+\rho$
5	0	$\rho + \nu(h)$	0
6	$\nu(n)$	$\rho+\nu(h)+\nu(n)$	0
7	$\nu(n) + \rho$	0	0

Table 4.4: Noise bounds for the chains in the seven phases.

probability that the noise level of any of the b_i-chains exceeds $\nu(n)$ is at most $\mathbf{e}^{-n}$. As mentioned $\nu(n) < 5n$. Phase 7 is a chain opening competition. By Lemma 4.5.3, the probability that the noise level of a given b_i-chain increases by more than $\rho = 2n$ is $O(n^4/2^{2n}) = o(\mathbf{e}^{-n})$. The other noise bounds can be verified in a similar manner.

We are now in a position to choose the lengths of the various chains. The length h of all the a_i-chains should satisfy $h > \nu(n) + \rho$. This is satisfied by choosing $h = 8n$. The length g of the c_i-chains should satisfy $g > \rho + \nu(8n) + \nu(n)$. As $\rho = 2n$, $\nu(8n) < 15n$ and $\nu(n) < 5n$, we can choose $g = 22n$. Finally, the length ℓ_{k+1} of the b_{k+1}-chain should satisfy $\ell_{k+1} > 2\nu(n)+\rho+\nu(\ell_k)+\nu(22n)$. As $\nu(22n) < 33n$ and $\nu(\ell_k) \leq \ell_k+3\sqrt{\ell_k n}$, it is enough to require that $\ell_{k+1} > \ell_k + 3\sqrt{\ell_k n} + 45n$. It is easy to see that this is satisfied by the choice $\ell_k = 25k^2n$. Putting everything together, we get

Theorem 4.7.1 *The expected number of improving switches performed by* RandomEdge *on the MDPs constructed in this section, which contain* $\mathcal{O}(n^4)$ *vertices and edges, is* $\Omega(2^n)$.

The primal linear programs corresponding to the MDPs constructed in this section are thus linear programs on which the simplex algorithm with the RandomEdge pivoting rule performs an expected subexponential number of iterations. We get the following corollary.

Corollary 4.7.2 *There exist linear programs in standard form with* n *equations and* $2n$ *variables, for which the expected number of steps performed by* RandomEdge *is* $2^{\Omega(n^{1/4})}$.

It is not difficult to see that the lower bound construction for RandomEdge also works for Howard's PolicyIteration algorithm [63], in which all improving switches are performed in every iteration. I.e., in every competition between closing chains the progress made is exactly the same. Also, in a competition between a closing chain and an opening chain, the opening chain opens completely in one iteration. The lower bound obtained in this way is, however, worse than the $2^{\Omega(n)}$ lower bounds of Friedmann [38] and Fearnley [34]. In a similar way, it can be shown that the lower bound works for the SwitchHalf improvement rule [85], which applies every improving switch with probability $1/2$. Performing multiple improving switches only facilitates the analysis of competing chains. In fact, this means that the lower bound extends to a family of randomized multi-switch improvement rules, in which the number of improving switches is chosen at random, and then a set of improving switches of this size is chosen at random. For more details on such improvement rules see Friedmann [39].

4.8 Proof of Lemma 4.6.2

Since $\epsilon = N^{-(4n+8)}$ is very small, terms that involve ϵ are generally only relevant when studying chains locally. We therefore use $o(1)$ to denote terms that involve ϵ when we do not need to describe values so precisely. In particular, $o(1)$ is always less than 1. Studying Figure 4.3 it is not difficult to prove the following lemma.

Lemma 4.8.1 *For every policy π and every $i \in [n]$:*

$$\mathrm{val}_\pi(c_{i,g}) = \begin{cases} N^6 + \mathrm{val}_\pi(w_1) + o(1) & \text{if } c_i(\pi) = 0 \\ \mathrm{val}_\pi(u_1) + o(1) & \text{if } c_i(\pi) = 1 \text{ and } b_i(\pi) = 0 \\ N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) & \text{if } c_i(\pi) = 1 \text{ and } b_i(\pi) = 1 \end{cases}$$

$$\mathrm{val}_\pi(a_{i,h}) = \begin{cases} -N^{4i+3} + \mathrm{val}_\pi(u_1) + o(1) & \text{if } a_i(\pi) = 0 \\ -N^{4i+5} + \mathrm{val}_\pi(c_{i,g}) & \text{if } a_i(\pi) = 1 \end{cases}$$

$$\mathrm{val}_\pi(u_i) = \begin{cases} \mathrm{val}_\pi(u_{i+1}) & \text{if } u_i(\pi) = 0 \\ -N^{4i+5} + \mathrm{val}_\pi(c_{i,g}) & \text{if } u_i(\pi) = 1 \end{cases}$$

$$\mathrm{val}_\pi(w_i) = \begin{cases} \mathrm{val}_\pi(w_{i+1}) & \text{if } w_i(\pi) = 0 \\ \mathrm{val}_\pi(a_{i,h}) & \text{if } w_i(\pi) = 1 \end{cases}$$

Before showing that we get the correct improving switches for the b_i-, c_i-, and a_i-chains, we first prove two lemmas about the values of the two lanes.

Lemma 4.8.2 *Let π be a policy and $i \in [n]$. If $u_i(\pi) = w_i(\pi) = a_i(\pi) = b_i(\pi) = c_i(\pi) = 1$, then $\mathrm{val}_\pi(u_i) = \mathrm{val}_\pi(w_i)$. In particular, if for all $i \leq k \leq n$ we either have $u_k(\pi) = w_k(\pi) = 0$ or $u_k(\pi) = w_k(\pi) = a_k(\pi) = b_k(\pi) = c_k(\pi) = 1$, then $\mathrm{val}_\pi(u_i) = \mathrm{val}_\pi(w_i)$.*

Proof: The first part of the lemma follows immediately from Lemma 4.8.1. The second part of the lemma is proved by backwards induction in i, where the basis follows from $\mathrm{val}_\pi(u_{n+1}) = \mathrm{val}_\pi(w_{n+1})$. □

Lemma 4.8.3 *Let π be a policy that satisfies the stopping condition, and let $i \in [n]$. If $b_i(\pi) = c_i(\pi) = u_i(\pi) = 1$, $a_i(\pi) = 0$, and $u_t(\pi) = 0$ for all $1 \leq t < i$, then $\mathrm{val}_\pi(w_k) \leq -N^{4k+3} + \mathrm{val}_\pi(u_1) + 1$ for all $1 \leq k \leq i$.*

Proof: The lemma is proved by induction. Observe first that the lemma is true for k if $w_k(\pi) = 1$ and $a_k(\pi) = 0$, or if $w_k(\pi) = a_k(\pi) = c_k(\pi) = 1$ and $b_k(\pi) = 0$. In particular, the lemma is true for $k = i$.

Furthermore, if $\mathrm{val}_\pi(w_{k+1}) \leq -N^{4k+7} + \mathrm{val}_\pi(u_1) + 1$, and $w_k(\pi) = 0$ or $w_k(\pi) = a_k(\pi) = c_k(\pi) = b_k(\pi) = 1$, then we also get $\mathrm{val}_\pi(w_k) \leq -N^{4k+3} + \mathrm{val}_\pi(u_1) + 1$.

Next, we prove that $\mathrm{val}_\pi(w_1) \leq -N^7 + \mathrm{val}_\pi(u_1) + 1$. Assume that this is not the case. Since π satisfies the stopping condition we can not have $w_1(\pi) = a_1(\pi) = 1$ and $c_1(\pi) = 0$ (this would lead to a negative cycle). Thus, we must have $w_1(\pi) = 0$ or $w_1(\pi) = a_1(\pi) = c_1(\pi) = b_1(\pi) = 1$. In both cases we move from w_1 to w_2. If $\mathrm{val}_\pi(w_2) \leq -N^{11} + \mathrm{val}_\pi(u_1) + 1$ we are done, otherwise we can repeat the argument. Since $\mathrm{val}_\pi(w_i) \leq -N^{4i+3} + \mathrm{val}_\pi(u_1) + 1$, we can not continue in this way, and the claim follows.

Finally, we observe that if $w_k(\pi) = a_k(\pi) = 1$ and $c_k(\pi) = 0$, then:

$$\begin{aligned} \mathrm{val}_\pi(w_k) &= -N^{4k+5} + o(1) + N^6 + \mathrm{val}_\pi(w_1) \\ &\leq -N^{4k+5} + N^6 - N^7 + \mathrm{val}_\pi(u_1) + 1 \\ &\leq -N^{4k+3} + \mathrm{val}_\pi(u_1) + 1\,, \end{aligned}$$

and the lemma is true for k. This completes the proof. □

Using lemmas 4.8.2 and 4.8.3, the following lemma will be helpful when proving that we get the desired improving switches.

Lemma 4.8.4 *Let π be a policy, and let $i \in [n]$. Then:*

(*i*) *If* $N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \geq 1 + N^6 + \mathrm{val}_\pi(w_1)$ *and* $\mathrm{val}_\pi(w_1) = \mathrm{val}_\pi(u_1)$, *then:*

$$\overline{\mathsf{b}}_i(\pi) \in \begin{cases} \uparrow & \text{if the } b_i\text{-chain is consecutive} \\ 1 & \text{if } b_i(\pi) = 1 \end{cases}$$

$$\overline{\mathsf{c}}_i(\pi) \in \begin{cases} \Downarrow & \text{if } b_i(\pi) = 0 \\ \uparrow & \text{if } b_i(\pi) = 1 \text{ and the } c_i\text{-chain is consecutive} \\ 1 & \text{if } b_i(\pi) = 1 \text{ and } c_i(\pi) = 1 \end{cases}$$

$$\overline{\mathsf{a}}_i(\pi) \in \begin{cases} \Downarrow & \text{if } b_i(\pi) = 0 \text{ or } c_i(\pi) = 0 \\ \uparrow & \text{if } b_i(\pi) = 1,\ c_i(\pi) = 1, \text{ and the } a_i\text{-chain is consecutive} \end{cases}$$

(*ii*) *If* $1 + N^6 + \mathrm{val}_\pi(w_1) \leq \mathrm{val}_\pi(u_1)$, $\mathrm{val}_\pi(u_1) + 1 \leq N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$, *and* $\mathrm{val}_\pi(w_{i+1}) \leq N^{4n+6}$, *then:*

$$\overline{\mathsf{b}}_i(\pi) \in \begin{cases} \Downarrow & \text{if } c_i(\pi) = 0 \\ \uparrow & \text{if } c_i(\pi) = 1 \text{ and the } b_i\text{-chain is consecutive} \\ 1 & \text{if } b_i(\pi) = 1 \text{ and } c_i(\pi) = 1 \end{cases}$$

$$\overline{\mathsf{c}}_i(\pi) \in \begin{cases} \uparrow & \text{if } b_i(\pi) = 0 \text{ and the } c_i\text{-chain is consecutive} \\ 1 & \text{if } c_i(\pi) = 1 \end{cases}$$

$$\overline{\mathsf{a}}_i(\pi) \in \begin{cases} \Downarrow & \text{if } b_i(\pi) = 0 \text{ or } c_i(\pi) = 0 \\ \uparrow & \text{if } b_i(\pi) = 1 \text{ and } c_i(\pi) = 1 \end{cases}$$

(*iii*) *If* $1 + N^6 + \mathrm{val}_\pi(w_1) \leq \mathrm{val}_\pi(u_1)$ *and* $N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \leq \mathrm{val}_\pi(u_1)$, *then:*

$$\begin{aligned} \overline{\mathsf{b}}_i(\pi) &\in \ \Downarrow \\ \overline{\mathsf{c}}_i(\pi) &\in \ \uparrow \quad \text{if } b_i(\pi) = 0 \text{ and the } c_i\text{-chain is consecutive} \\ \overline{\mathsf{a}}_i(\pi) &\in \ \Downarrow \end{aligned}$$

Proof: Recall that if the b_i-chain is opening then $\overline{\mathsf{b}}_i(\pi) \in \Downarrow$, and that if the b_i-chain is closing then $\overline{\mathsf{b}}_i(\pi) \in \uparrow$. Also, if the b_i-chain is open then $\overline{\mathsf{b}}_i(\pi) \in 0$, and that if the b_i-chain is closed then $\overline{\mathsf{b}}_i(\pi) \in 1$. The proof of the lemma relies on Lemma 4.5.1. We will only prove that in (*i*) if the b_i-chain is consecutive then $\overline{\mathsf{b}}_i(\pi) \in \uparrow$. The other statements are shown in a similar

fashion. An alternative presentation of the proof is given in Friedmann, Hansen, and Zwick [42][2].

From Lemma 4.5.1 we know that it suffices to show that $\mathrm{val}_\pi(\mathrm{B}_i) > \mathrm{val}_\pi(u_1)+\ell_i\epsilon$. Observe that $\mathrm{val}_\pi(\mathrm{B}_i) = \epsilon(N^{4i+6}+\mathrm{val}_\pi(w_{i+1}))+(1-\epsilon)\mathrm{val}_\pi(c_{i,g})$. If $c_i(\pi) = 0$ then $\mathrm{val}_\pi(c_{i,g}) \geq N^6 + \mathrm{val}_\pi(w_1) = N^6 + \mathrm{val}_\pi(u_1)$, and since $N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \geq 1 + N^6 + \mathrm{val}_\pi(w_1)$ the inequality follows. If $c_i(\pi) = 1$ and $b_i(\pi) = 0$ then $\mathrm{val}_\pi(c_{i,g}) = \ell_i\epsilon + \mathrm{val}_\pi(u_1)$, and we again get that the inequality holds since $N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \geq 1 + N^6 + \mathrm{val}_\pi(u_1)$. Finally, if $c_i(\pi) = 1$ and $b_i(\pi) = 1$, then $\mathrm{val}_\pi(\mathrm{B}_i) = N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$, and we again see that the inequality holds due to the assumption.

Note that in (ii) when proving that if $c_i(\pi) = 0$ then $\overline{\mathtt{b}}_i(\pi) \in \Downarrow$, we use that ϵ is a very small number. This is the reason that small probabilities are needed in the construction.

□

Note that $\overline{\mathtt{b}}_i(\pi) \in 0$ if $\overline{\mathtt{b}}_i(\pi) \in \Downarrow$ and $b_{i,j}(\pi) = 0$ for all $j \in [\ell_i]$. Similarly, $\overline{\mathtt{b}}_i(\pi) \in \overline{\uparrow}$ if $\overline{\mathtt{b}}_i(\pi) \in \uparrow$ and $b_i(\pi) = 0$.

We are now ready to prove Lemma 4.6.2.

Let $\mathfrak{b} \in \{0,1\}^n$. Let us first consider phase 1. I.e., let $\pi \in \Pi_{\mathfrak{b},1}$. For all $i \in [n]$ with $\mathfrak{b}_i = 1$ we have $u_i(\pi) = w_i(\pi) = a_i(\pi) = b_i(\pi) = c_i(\pi) = 1$, and for all $i \in [n]$ with $\mathfrak{b}_i = 0$ we have $u_i(\pi) = w_i(\pi) = 0$. It follows from Lemma 4.8.2 that $\mathrm{val}_\pi(u_1) = \mathrm{val}_\pi(w_1)$. Also, $\mathrm{val}_\pi(w_1) \leq \mathrm{val}_\pi(w_{i+1}) + \sum_{k=1}^{i}(N^{4k+6} - N^{4k+5}) \leq N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) - N^6 - 1$ so that we can apply Lemma 4.8.4 (i). It follows that if $\mathfrak{b}_i = 1$ then $\overline{\mathtt{b}}_i(\pi) \in 1$, $\overline{\mathtt{c}}_i(\pi) \in 1$, and $\overline{\mathtt{a}}_i(\pi) \in 1$. Note that since $\pi \in \Pi_{\mathfrak{b},1}$, if $\mathfrak{b}_i = 0$ then the b_i-chain is consecutive. It follows from Lemma 4.8.4 that $\overline{\mathtt{b}}_i(\pi) \in \uparrow$, $\overline{\mathtt{c}}_i(\pi) \in \Downarrow$, and $\overline{\mathtt{a}}_i(\pi) \in \Downarrow$.

Note that when arguing about improving switches there is no need to distinguish $\uparrow$, $\overline{\uparrow}$, and 1. The only differences are whether a chain is closed or not. Similarly, there is no reason to distinguish between $\Downarrow$ and 0. For instance, if $c_i(\pi) \in 0$ and $\overline{\mathtt{c}}_i(\pi) \in \Downarrow$, then $\overline{\mathtt{c}}_i(\pi) \in 0$.

Using Lemma 4.8.1, we see that $\mathrm{val}_\pi(u_1) = \mathrm{val}_\pi(w_1) = \sum_{i:\mathfrak{b}_i=1}(N^{4i+6} - N^{4i+5})$. It is then not difficult to see that both lanes are stable. I.e., the penalty for entering a level i with $\mathfrak{b}_i = 0$ is larger than anything that can be gained from moving back up from u_1 or w_1. We conclude that $\Pi_{\mathfrak{b},1} = \overline{\Pi}_{\mathfrak{b},1}$.

Let $k_\mathfrak{b} = \min\{i \in [n] \mid \mathfrak{b}_i = 0\}$. Phase 2, $\pi \in \Pi_{\mathfrak{b},2}$, is the same as phase 1, with the exception that $b_{k_\mathfrak{b}}(\pi) = 1$. This means that $\overline{\mathtt{b}}_i(\pi) \in 1$ and $\overline{\mathtt{c}}_i(\pi) \in \uparrow$, but otherwise the arguments given above for phase 1 also hold for phase 2, and it follows that $\Pi_{\mathfrak{b},2} = \overline{\Pi}_{\mathfrak{b},2}$.

[2] A full version of the paper that contains the proof is available at `http://cs.au.dk/~tdh/papers/random_edge.pdf`.

For phase 3, $\pi \in \Pi_{\mathfrak{b},3}$, we have $b_{k_\mathfrak{b}}(\pi) = c_{k_\mathfrak{b}}(\pi) = 1$. This means that $\overline{\mathtt{b}}_i(\pi) \in 1$, $\overline{\mathtt{c}}_i(\pi) \in 1$, and $\overline{\mathtt{a}}_i(\pi) \in \uparrow$. We also have $\mathrm{val}_\pi(c_{k_\mathfrak{b},g}) = N^{4k_\mathfrak{b}+6} + \mathrm{val}_\pi(w_{k_\mathfrak{b}+1}) = N^{4k_\mathfrak{b}+6} + \mathrm{val}_\pi(u_{k_\mathfrak{b}+1})$. Hence, it is improving for the vertices $u_1, \ldots, u_{k_\mathfrak{b}}$ to move through $c_{k_\mathfrak{b},g}$, and these vertices therefore form a closing chain. These changes do not affect other vertices because $u_1(\pi) = 1$ throughout the phase.

For phase 4, $\pi \in \Pi_{\mathfrak{b},4}$, we have $b_{k_\mathfrak{b}}(\pi) = c_{k_\mathfrak{b}}(\pi) = u_{k_\mathfrak{b}}(\pi) = 1$, $a_{k_\mathfrak{b}}(\pi) = 0$, and $u_t(\pi) = 0$ for all $1 \leq t < k_\mathfrak{b}$. We can, thus, apply Lemma 4.8.3, and we see that $\mathrm{val}_\pi(w_k) \leq -N^{4k+3} + \mathrm{val}_\pi(u_1) + o(1)$ for all $1 \leq k \leq i$. In particular, $\mathrm{val}_\pi(w_1) \leq -N^7 + \mathrm{val}_\pi(u_1) + o(1)$. It is also not difficult to see that $\mathrm{val}_\pi(w_i) \leq N^{4n+6}$ for all $i \in [n]$. We then get from Lemma 4.8.4 that:

- From (ii): $\overline{\mathtt{b}}_i(\pi) \in \Downarrow$ and $\overline{\mathtt{a}}_i(\pi) \in \Downarrow$, for all $i > k_\mathfrak{b}$ with $\mathfrak{b}_i = 0$.
- From (ii): $\overline{\mathtt{b}}_i(\pi) \in 1$, and $\overline{\mathtt{a}}_i(\pi) \in \uparrow$, for all $i > k_\mathfrak{b}$ with $\mathfrak{b}_i = 1$, and for $i = k_\mathfrak{b}$.
- From (ii) and (iii): $\overline{\mathtt{c}}_i(\pi) \in \uparrow$ for all $i \in [n]$.
- From (iii): $\overline{\mathtt{b}}_i(\pi) \in \Downarrow$ and $\overline{\mathtt{a}}_i(\pi) \in \Downarrow$, for all $i < k_\mathfrak{b}$.

The situation has not changed for the right lane, and for the left lane for w_i with $i > k_\mathfrak{b}$. For $w_{k_\mathfrak{b}}$ it is improving to move to $a_{k_\mathfrak{b},h}$. We do not specify this improving switch ($\overline{\mathtt{w}}_{k_\mathfrak{b}}(\pi) \in *$), since we do not require it to happen until phase 6. For $i < k_\mathfrak{b}$ it follows from Lemma 4.8.3 that $\overline{\mathtt{w}}_i(\pi) \in 1$.

Phase 5, $\pi \in \Pi_{\mathfrak{b},5}$, is like phase 4, except that the b_i- and a_i- chains are open for $i < k_\mathfrak{b}$. This does not affect the analysis from phase 4, however.

In phase 6, $\pi \in \Pi_{\mathfrak{b},6}$, we have $a_{k_\mathfrak{b}}(\pi) = 1$. Hence, we have $\mathrm{val}_\pi(a_{k_\mathfrak{b},h}) = N^{4k_\mathfrak{b}+6} - N^{4k_\mathfrak{b}+5} + \mathrm{val}_\pi(w_{k_\mathfrak{b}+1}) = \mathrm{val}_\pi(u_1)$. It follows that it is improving for the vertices $w_1, \ldots, w_{k_\mathfrak{b}}$ to move through $a_{k_\mathfrak{b},h}$, and these vertices therefore form a closing chain. We still have $\mathrm{val}_\pi(w_1) \leq -N^7 + \mathrm{val}_\pi(u_1) + o(1)$ throughout the phase. When the vertices move along the chain from level $i < k_\mathfrak{b}$, such that $\mathrm{val}_\pi(w_i) = \mathrm{val}_\pi(u_1) = \mathrm{val}_\pi(u_i)$, we get from Lemma 4.8.4 (ii) that $\overline{\mathtt{b}}_i(\pi) \in \uparrow$. These are the only changes compared to phase 5.

In phase 7, $\pi \in \Pi_{\mathfrak{b},7}$, we have $u_{k_\mathfrak{b}}(\pi) = w_{k_\mathfrak{b}}(\pi) = 1$ and $u_t(\pi) = w_t(\pi) = 0$ for all $1 \leq t < k_\mathfrak{b}$. Hence, we have $\mathrm{val}_\pi(u_1) = \mathrm{val}_\pi(w_1) = \mathrm{val}_\pi(w_t)$, for all $t \leq k_\mathfrak{b}$. It follows that we can apply Lemma 4.8.4 (i), and we see that $\overline{\mathtt{b}}_i(\pi) \in \uparrow$ for all $i \in [n]$, and $\overline{\mathtt{c}}_i(\pi) \in \Downarrow$ for all $i < k_\mathfrak{b}$, and for $i > k_\mathfrak{b}$ with $\mathfrak{b}_i = 0$. The situation for the remaining vertices remains the same as in phase 6.

This completes the proof.

Chapter 5

The RandomFacet algorithm

5.1 Introduction

In this chapter we present results from Friedmann, Hansen, and Zwick [41, 42]. The main result presented in this chapter is a lower bound of subexponential form for the RandomFacet pivoting rule for the simplex method. We refer to the resulting algorithm as the RandomFacet algorithm. Kalai [70, 71] and Matoušek, Sharir, and Welzl [88] independently showed that the expected number of pivots performed by the RandomFacet algorithm is subexponential. More precisely, for a linear program in standard form with n equality constraints and m variables they showed that the expected number of pivots performed by the (primal) RandomFacet algorithm is $2^{O(\sqrt{n \log m})}$. For additional details see Section 1.2.2.

Matoušek [86] constructed a family of abstract optimization problems (acyclic unique sink orientations, see Section 3.3) such that RandomFacet, when run on a *random* instance from the family, has an expected running time close to the subexponential upper bound given above. It is known, however, that the hard instances in this family do not correspond to linear programs (see Gärtner [46]).

We construct explicit linear programs such that the expected running time of the RandomFacet algorithm for a linear program in standard form with n equality constraints and $m = O(n \log n)$ variables is $2^{\Omega(n^{1/3}/\log^{2/3} n)}$. This also provides the first *explicit* construction of LP-type problems on which the RandomFacet algorithm is not polynomial.

The lower bound is obtained for a corresponding PolicyIteration algorithm for solving Markov decision processes (MDPs). Due to the close connection between the PolicyIteration algorithm for MDPs and the

simplex method for linear programming (see Section 2.4.1), this immediately translates to a lower bound for the RandomFacet algorithm for linear programming.

To obtain the lower bound for MDPs, we adapt and extend the techniques used by Friedmann [38] and Fearnley [34]. Friedmann [38] showed that the StrategyIteration algorithm with Howard's improvement rule (see Section 3.2) may require exponential time to solve parity games. Fearnley [34] adapted the result of Friedmann to show that the PolicyIteration algorithm with Howard's improvement rule for MDPs with the total reward criterion (see Section 2.4) may require an exponential number of iterations. Since Howard's StrategyIteration (PolicyIteration) algorithm is deterministic, Friedmann's [38] and Fearnley's [34] delicate constructions are designed to fool a deterministic algorithm. Fooling a randomized algorithm like the RandomFacet algorithm poses new challenges. Many difficulties, thus, need to be overcome to obtain a subexponential lower bound for the Random Facet algorithm. Instead of simulating a standard counter, runs of the RandomFacet algorithm on our MDPs essentially simulate the operation of a *randomized counter*. To ensure, with high probability, the desired behavior of the randomized counter, we *duplicate* critical gadgets used in the construction, thus achieving resilience against 'fortunate' choices of the RandomFacet algorithm.

We initially proved a lower bound for the RandomFacet algorithm for parity games and not MPDs. In fact, in Friedmann, Hansen, and Zwick [41] from SODA 2011 we only consider parity games. We later converted these parity games into MDPs in Friedmann, Hansen, and Zwick [42] from STOC 2011. The translation of our parity games to MDPs is a relatively simple step. It closely resembles the way Fearnley [34] translated Friedmann's [38] construction for parity games to MDPs. The MDP version of our construction can be described and understood without knowing anything about parity games. In this chapter we will only present the lower bound for MDPs. We refer to Friedmann, Hansen, and Zwick [41] for a description of the corresponding construction for parity games.

We also present lower bounds of subexponential form for two algorithms that are closely related to the RandomFacet algorithm. We refer to the first as the *modified* RandomFacet *algorithm* (or RandomFacet*), and to the second as the Randomized Bland's rule algorithm. As described in Section 1.2.2, the Randomized Bland's rule is Bland's rule where the indices of the variables (actions) are ordered uniformly at random. For these two algorithms no subexponential upper bounds are known. A subexponential upper bound for the Randomized Bland's rule algorithm would be

particularly interesting. For both the modified RandomFacet algorithm and the Randomized Bland's rule algorithm we prove $e^{\Omega(\sqrt{n}/\log n)}$ lower bounds for the expected number of improving pivots (switches) performed for linear programs (MDPs) with n equations (states) and $m = O(n \log n)$ variables (actions). The constructions are exactly the same as for the RandomFacet algorithm, but the analysis of each algorithm is different. In fact, in [41] from SODA 2011 we presented the analysis for the modified RandomFacet algorithm. We then *incorrectly* claimed that the expected number of steps for the RandomFacet and modified RandomFacet algorithms were the same. We here present an independent analysis (with a slightly worse bound) for the RandomFacet algorithm.

The chapter is organized as follows. In Section 5.2 we present the RandomFacet algorithm for MDPs. In Section 5.3 we present the randomized counter that we later show is simulated by the three algorithms. In sections 5.4 and 5.5 we first present a simplified version of our construction and then the full version. The simplified construction is nearly identical to the simplified construction for RandomEdge presented in Section 4.3. Finally, in sections 5.6, 5.7, and 5.8 we present the analyses of the RandomFacet, modified RandomFacet, and Randomized Bland's rule algorithms, respectively.

5.2 The RandomFacet algorithm

The RandomFacet algorithm of Kalai [70, 71] and Matoušek, Sharir and Welzl [88] is a simple and elegant randomized pivoting rule for the simplex method. As described in Section 1.2.2, it works by recursively finding an optimal solution within a uniformly random facet f that contains the current basic feasible solution. When such an optimal solution is found, it is either optimal for the entire linear program, or there exists an improving pivot that leaves the facet f. In the second case we perform the improving pivot, and repeat the procedure from the new basic feasible solution. In the following we will see how the operations of the RandomFacet algorithm, when run on the primal linear program of an MDP with the total reward criterion, can be interpreted as a PolicyIteration algorithm. For a detailed description of MDPs with the total reward criterion we refer to Section 2.4.

As in Chapter 4, we are going to use the graphical representation of MDPs. We interpret a weighted directed graph $G = (V_0, V_R, E, c, p)$ as an MDP as described in Section 4.2. I.e., V_0 is the set of decision vertices (or states), and V_R is the set of randomization vertices. We let $V = V_0 \cup V_R$

be the set of all vertices. Since we are using the total reward criterion, we implicitly assume that G has a special *sink* vertex $\textsc{t}$ (the terminal state). We also let $E_0 = E \cap (V_0 \times V \cup \{\textsc{t}\})$ be the set of edges emanating from V_0, and $E_R = E \cap (V_R \times V \cup \{\textsc{t}\})$ be the set of edges emanating from V_R. The set of actions is the set of edges E_0. $c : E \to \mathbb{R}$ assigns a reward to every edge, and $p : E_R \to [0, 1]$ defines probability distributions for the edges leaving randomization vertices.

A policy $\pi : V_0 \to (V \cup \{\textsc{t}\})$ maps every decision vertex $u \in V_0$ to another vertex $v \in V \cup \{\textsc{t}\}$ where $(u, v) \in E$. Note that a policy can be equivalently interpreted as mapping vertices to edges. We will also interpret policies as subsets of edges. I.e., $\pi \subseteq E_0$. For every policy π satisfying the stopping condition (Definition 2.4.2), the values of the vertices can be found as the unique solution to the following system of equations:

$$\begin{aligned}
\mathrm{val}_\pi(\textsc{t}) &= 0 \\
\mathrm{val}_\pi(u) &= c(u, \pi(u)) + \mathrm{val}_\pi(\pi(u)) \quad , \quad \text{if } u \in V_0 \\
\mathrm{val}_\pi(u) &= \sum_{v:(u,v)\in E} p(u, v)(c(u, v) + \mathrm{val}_\pi(v)) \quad , \quad \text{if } u \in V_R
\end{aligned}$$

An edge $(u, v) \in E_0$ is an improving switch with respect to a policy π if and only if:

$$c(u, v) + \mathrm{val}_\pi(v) > \mathrm{val}_\pi(u) \; .$$

It follows from Theorem 2.4.6 that a policy π is optimal if and only if there are no improving switches with respect to π.

The primal linear program (P) of an MDP with the total reward criterion was presented in Section 2.4.1. We repeat it here for reference:

$$\begin{array}{rlrcl}
 & \max & \mathbf{c}^T\mathbf{x} & & \\
(P) & s.t. & (\mathbf{J} - \mathbf{P})^T\mathbf{x} & = & \mathbf{e} \\
 & & \mathbf{x} & \geq & \mathbf{0}
\end{array}$$

Recall that every basic feasible solution of (P) corresponds to a policy of the MDP (Lemma 2.4.10). Recall also that the variables $\mathbf{x} \in \mathbb{R}^m$ count how many times the actions are used before reaching the terminal state $\textsc{t}$. Let $\bar{\mathbf{x}}^\pi \in \mathbb{R}^m$ be a basic feasible solution of (P) corresponding to a policy π. For some action a we have $(\bar{\mathbf{x}}^\pi)_a = 0$ if and only if this action is not used in the policy π. The facets that contain $\bar{\mathbf{x}}^\pi$ then correspond exactly to the actions that are not used by π. Furthermore, staying within a facet means not using the corresponding action. Hence, the RandomFacet pivoting rule for the simplex method can be interpreted as an improvement rule for

Algorithm 6: RandomFacet(F, π)

```
if F = π then
    return π;
else
    Choose e ∈ F \ π uniformly at random ;
    π' ← RandomFacet(F \ {e}, π) ;
    if e is an improving switch with respect to π' then
        π'' ← π'[e] ;
        return RandomFacet(F, π'');
    else
        return π';
```

Figure 5.1: The RandomFacet algorithm

the PolicyIteration algorithm as follows. Starting from a policy π, we pick a uniformly random action a that is not used by π. We remove this action from the MDP, and solve the problem recursively, giving us a policy π' that achieves the best possible values without using the action a. If a is not an improving switch with respect to π' then π' is optimal and we are done. If a is an improving switch with respect to π', then we perform the improving switch a and repeat the procedure from $\pi'' = \pi'[a]$. Note that a must now remain a part of the policy until the algorithm terminates, since every policy that does not use a has lower values than π''.

The RandomFacet algorithm, when viewed as a PolicyIteration algorithm, is shown in Figure 5.1. $F \subseteq E_0$ is the set of remaining edges. Initially $F = E_0$. The algorithm is also given the graph $G = (V_0, V_R, E, c, p)$ as input, but to simplify notation we only explicitly mention the set F. Define $G_F = (V_0, V_R, F \cup E_R, c, p)$ to be the subgraph defined by edges of $F \cup E_R$. The recursive calls can then be interpreted as being run on the subgraph $G_{F\setminus\{e\}}$.

It follows from the analysis of Kalai [70, 71] and Matoušek, Sharir, and Welzl [88] that the *expected number of improving switches* performed by the RandomFacet algorithm on any MDP is $2^{O(\sqrt{n \log m})}$, where $n = |V_0|$ is the number of decision vertices in G, and $m = |E_0|$ is the number of actions.

The operation of the RandomFacet algorithm may be described by a binary computation tree T as follows. We use the terms *nodes* and *arcs* to describe computation trees. Let Π be the set of all policies for G. Let $T = (U, D, F, \pi, e, d, u_0)$ be a tuple where

- U is the set of nodes,
- $D \subseteq U \times U$ is the set of arcs,
- $F : U \to 2^{E_0}$ assigns a subset of edges from G to every node,
- $\pi : U \to \Pi$ assigns a policy to every node,
- $e : D \to E_0$ assigns an edge from G to every arc,
- $d : D \to \{L, R\}$ assigns a symbol L (left) or R (right) to every arc,
- and $u_0 \in U$ is the root of T.

Each node u of the tree T has an edge subset $F(u) \subseteq E_0$ and a policy $\pi(u)$ assigned to it. For the root we have $F(u_0) = E_0$ and $\pi(u_0) = \pi_0$, where π_0 is the initial policy. The node u corresponds to the recursive call RandomFacet$(F(u), \pi(u))$. Each node u may have a left child u_L and a right child u_R. I.e., from every node u there are at most two arcs $(u, u_L), (u, u_R) \in D$ leaving u. Each arc (u, u') of the tree, where $u' \in \{u_L, u_R\}$, is labeled by a pair $(e(u, u'), d(u, u'))$, where $e(u, u') \in F(u) \setminus \pi(u)$ is an edge of the MDP, and:

$$d(u, u') \;=\; \begin{cases} L & \text{if } u' = u_L \\ R & \text{if } u' = u_R \end{cases}$$

The edge $e(u, u')$ is the edge picked at the first level of the recursion for RandomFacet$(F(u), \pi(u))$. If both u_L and u_R exist we must have $e(u, u_L) = e(u, u_R)$. The left child u_L exists if and only if $\pi(u) \subsetneq F(u)$, in which case $F(u_L) = F(u) \setminus \{e(u, u_L)\}$ and $\pi(u_L) = \pi(u)$.

Let u^R be the *rightmost* leaf in the subtree of u. It is obtained by following a path from u that makes a right turn whenever possible until reaching a leaf. It can also be defined recursively as follows. If u is a leaf then $u^R = u$. If u_R exists, then $u^R = (u_R)^R$. Otherwise, $u^R = (u_L)^R$. The right child u_R exists if and only if u_L exists and $e(u, u_L)$ is an improving switch with respect to $\pi((u_L)^R)$. If u_R exists, then $e(u, u_R) = e(u, u_L)$, $\pi(u_R) = \pi((u_L)^R)[e(u, u_L)]$, and $F(u_R) = F(u)$. Note that $\pi(u^R)$ is the policy returned by the recursive call of RandomFacet at u. The correctness of the algorithm thus implies that $\pi(u^R)$ is optimal for $G_{F(u)}$. In particular, $\pi((u_L)^R)$ is optimal for $G_{F(u)\setminus\{e(u,u_L)\}}$. When there exists a unique optimal policy for $G_{F(u)\setminus\{e(u,u_L)\}}$, then $\pi(u_R)$ does not depend on the subtree of u_L. By making small pertubations we may assume that there always exists

a unique optimal policy for a subgraph. We do not use this assumption, however.

Let $T = (U, D, F, \pi, e, d, u_0)$ be a computation tree. Then we define $right(T) = \{(u, v) \in D \mid d(u, v) = R\}$ to be the set of arcs going right, and $size(T) = |right(T)|$ to be the number of arcs going right. The RandomFacet algorithm is of course a randomized algorithm, so the computation tree it defines is not unique. Instead, the RandomFacet algorithm defines a probability distribution over computation trees. The random choices made by the algorithm manifest themselves in the edges of the MDP that label the arcs of the tree. Let $f(F, \pi)$ be the expected number of improving switches performed by running RandomFacet(F, π). Note that every improving switch performed by the RandomFacet algorithm corresponds to an arc going right in the computation tree. Hence, if T is a random computation tree generated by RandomFacet(F, π), then $f(F, \pi) = \mathrm{E}[size(T)]$.

Let $(u, v) \in D$ be an arc, and let $(u_0, u_1), (u_1, u_2), \ldots, (u_{k-1}, u_k)$, where $(u_{k-1}, u_k) = (u, v)$, be the sequence of arcs on the path from the root to v. Then we define $P(u, v) = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ to be the sequence of pairs such that $e_\ell = e(u_{\ell-1}, u_\ell) \in E_0$ and $d_\ell = d(u_{\ell-1}, u_\ell) \in \{L, R\}$ for every $\ell \in [k]$. Note that there is always a unique such sequence. We next consider paths that may occur in one of the possible computation trees.

Definition 5.2.1 (Potential path) *A* potential path P *is a sequence of pairs* $\langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$*, where* $e_1, e_2, \ldots, e_k \in E_0$ *are distinct edges of the MDP, and where* $d_1, d_2, \ldots, d_k \in \{L, R\}$*. We let* $paths(G)$ *be the set of all potential paths for* G *with* $d_k = R$*. (We require the paths of* $paths(G)$ *to end with an arc going right.)* .

The pairs that make up a potential path P correspond to the labels of arcs of a computation tree. If P appears in a computation tree T we write $P \in T$. We let $appear_{F,\pi}(P)$ be the event that P appears as a path in the (random) computation tree T generated by RandomFacet(F, π). We will sometimes denote $appear_{E_0,\pi_0}(P)$, where π_0 is the initial policy, by $appear(P)$ when E_0 and π_0 are clear from the context. Furthermore, we let $\mathcal{T}_{F,\pi}$ be the set of all possible computation trees generated by RandomFacet(F, π), and $p_{F,\pi}(T)$ be the probability that the computation tree T is generated. Using that every right arc of a computation tree corresponds uniquely to a path, we get the following useful lemma. This technique was first used by Gärtner [45].

Lemma 5.2.2 $f(F, \pi) = \sum_{P \in paths(G_F)} \Pr[appear_{F,\pi}(P)]$.

Proof: Let $\mathbb{1}_{P \in T}$ be an indicator variable that is 1 if $P \in T$ and 0 otherwise. Then we have:

$$\begin{aligned} f(F,\pi) &= \mathrm{E}\left[size(T)\right] = \sum_{T \in \mathcal{T}_{F,\pi}} p_{F,\pi}(T) size(T) \\ &= \sum_{T \in \mathcal{T}_{F,\pi}} p_{F,\pi}(T) |right(T)| \\ &= \sum_{T \in \mathcal{T}_{F,\pi}} p_{F,\pi}(T) |\{P \in paths(G_F) \mid P \in T\}| \\ &= \sum_{P \in paths(G_F)} \sum_{T \in \mathcal{T}_{F,\pi}} p_{F,\pi}(T) \mathbb{1}_{P \in T} \\ &= \sum_{P \in paths(G_F)} \Pr\left[appear_{F,\pi}(P)\right] . \end{aligned}$$

□

5.3 A randomized counter

Our lower bound for the RandomFacet algorithm is obtained by simulating the behavior of the *randomized counter* shown in Figure 5.2. Such a counter is composed of n bits, $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$, all initially 0. We refer to $\mathfrak{b} \in \{0,1\}^n$ as a configuration of the counter. The randomized counter works in a recursive manner, focusing each time on a subset $N \subseteq [n] := \{1, \ldots, n\}$ of the bits, such that $\mathfrak{b}_j = 0$ for all $j \in N$. Initially $N = [n]$. If $N = \emptyset$, then nothing is done. Otherwise, the counter chooses a random index $i \in N$ and recursively performs a randomized count on $N \setminus \{i\}$. When this recursive call is done, we have $\mathfrak{b}_j = 1$, for every $j \in N \setminus \{i\}$, while $\mathfrak{b}_i = 0$. Next, the i-th bit is set to 1, and all bits $j \in N \cap [i-1]$ are reset to 0. Finally, a recursive randomized count is performed on $N \cap [i-1]$.

Let $g(n)$ be the expected number of times the call RandCount($[n]$) sets a bit of the randomized counter to 1. It is not difficult to check that the behavior of RandCount(N), where $|N| = k$, is equivalent to the behavior of RandCount($[k]$). It is easy to see that $g(n)$ satisfies the following recurrence relation:

$$\begin{aligned} g(0) &= 0 \\ g(n) &= g(n-1) + 1 + \frac{1}{n} \sum_{i=0}^{n-1} g(i) \quad \text{for } n > 0 \end{aligned}$$

Algorithm 7: RANDCOUNT(N)

```
if N ≠ ∅ then
    Choose i ∈ N uniformly at random ;
    RandCount(N \ {i}) ;
    for j ∈ N ∩ [i − 1] do 𝔟_j ← 0 ;
    𝔟_i ← 1 ;
    RandCount(N ∩ [i − 1]) ;
```

Figure 5.2: A randomized counter.

Lemma 5.3.1 $g(n) = \sum_{k=1}^{n} \frac{1}{k!}\binom{n}{k}$

Proof: Observe that:

$$\frac{1}{n}\sum_{i=0}^{n-1}\sum_{k=1}^{i}\frac{1}{k!}\binom{i}{k} = \frac{1}{n}\sum_{k=1}^{n-1}\frac{1}{k!}\sum_{i=k}^{n-1}\binom{i}{k} = \frac{1}{n}\sum_{k=1}^{n-1}\frac{1}{k!}\binom{n}{k+1}$$
$$= \sum_{k=1}^{n-1}\frac{1}{(k+1)!}\binom{n-1}{k} = \sum_{k=2}^{n}\frac{1}{k!}\binom{n-1}{k-1}.$$

Then by induction:

$$\begin{aligned} g(n) &= g(n-1) + 1 + \frac{1}{n}\sum_{i=0}^{n-1} g(i) \\ &= \sum_{k=1}^{n-1}\frac{1}{k!}\binom{n-1}{k} + 1 + \frac{1}{n}\sum_{i=0}^{n-1}\sum_{k=1}^{i}\frac{1}{k!}\binom{i}{k} \\ &= \sum_{k=1}^{n-1}\frac{1}{k!}\binom{n-1}{k} + 1 + \sum_{k=2}^{n}\frac{1}{k!}\binom{n-1}{k-1} \\ &= \sum_{k=1}^{n}\frac{1}{k!}\binom{n-1}{k} + \sum_{k=1}^{n}\frac{1}{k!}\binom{n-1}{k-1} \\ &= \sum_{k=1}^{n}\frac{1}{k!}\binom{n}{k} \end{aligned}$$

□

According to Lemma 5.3.1 we can interpret $g(n)$ as the expected number of increasing subsequences in a uniformly random permutation of $[n] = \{1, \ldots, n\}$. I.e., every subset $S \subseteq [n]$ appears as an increasing subsequence with probability $1/|S|!$. The asymptotic behavior of $g(n)$ is known quite precisely:

Lemma 5.3.2 ([79],[36, p. 596–597]) $g(n) \sim \dfrac{\mathrm{e}^{2\sqrt{n}}}{2\sqrt{\pi \mathrm{e}}\, n^{1/4}}$

Note that $g(n)$ is, thus, just of the right subexponential form. The challenge, of course, is to construct MDPs on which the behavior of the RandomFacet algorithm mimics the behavior of RandCount. In order to explain the idea for doing so, we examine a simplified version of our construction first.

5.4 A simplified construction

Let G_n be a family of lower bound MDPs, where n denotes the number of bits of a corresponding binary counter. In this section we describe, in a simpler setting, how to interpret policies for G_n as counter configurations, and we describe which policy is optimal when some of the edges have been removed. A schematic description of a simplified lower bound MDP G_n is given in Figure 5.3. The simplified construction for RandomFacet is essentially the same as the simplified construction we presented for RandomEdge in Section 4.3. The only difference is that edges from b_i and a_i move to u_{i+1} instead of u_1. In fact, this difference is not needed, but it makes the construction simpler that it is only possible to move from the i-th level to the $(i+1)$-st level. In particular, it is not difficult to check that G_n satisfies the stopping condition.

As we did when proving a lower bound for RandomEdge in Chapter 4, we use rewards with a certain exponential structure. I.e., we say that an edge e has priority $p \in \mathbb{N}$, if the reward of e is $c(e) = (-N)^p$, where N is some integer. For the RandomFacet algorithm it suffices to use $N = 2$, also for the full construction. The use of priorities is again motivated by parity games. It is worth noting that we use the same priorities as we did for the RandomEdge construction for consistency. In this case it would be possible to use smaller priorities.

Vertices of V_0 are shown as circles in Figure 5.3, and actions are shown as arrows. Actions are labelled by 0 and 1 for reference. The initial policy π_0 consists of all the edges labelled 0. The small hexagons describe the priorities of the actions. All other rewards are zero. Note that in the simplified construction there are no randomization vertices.

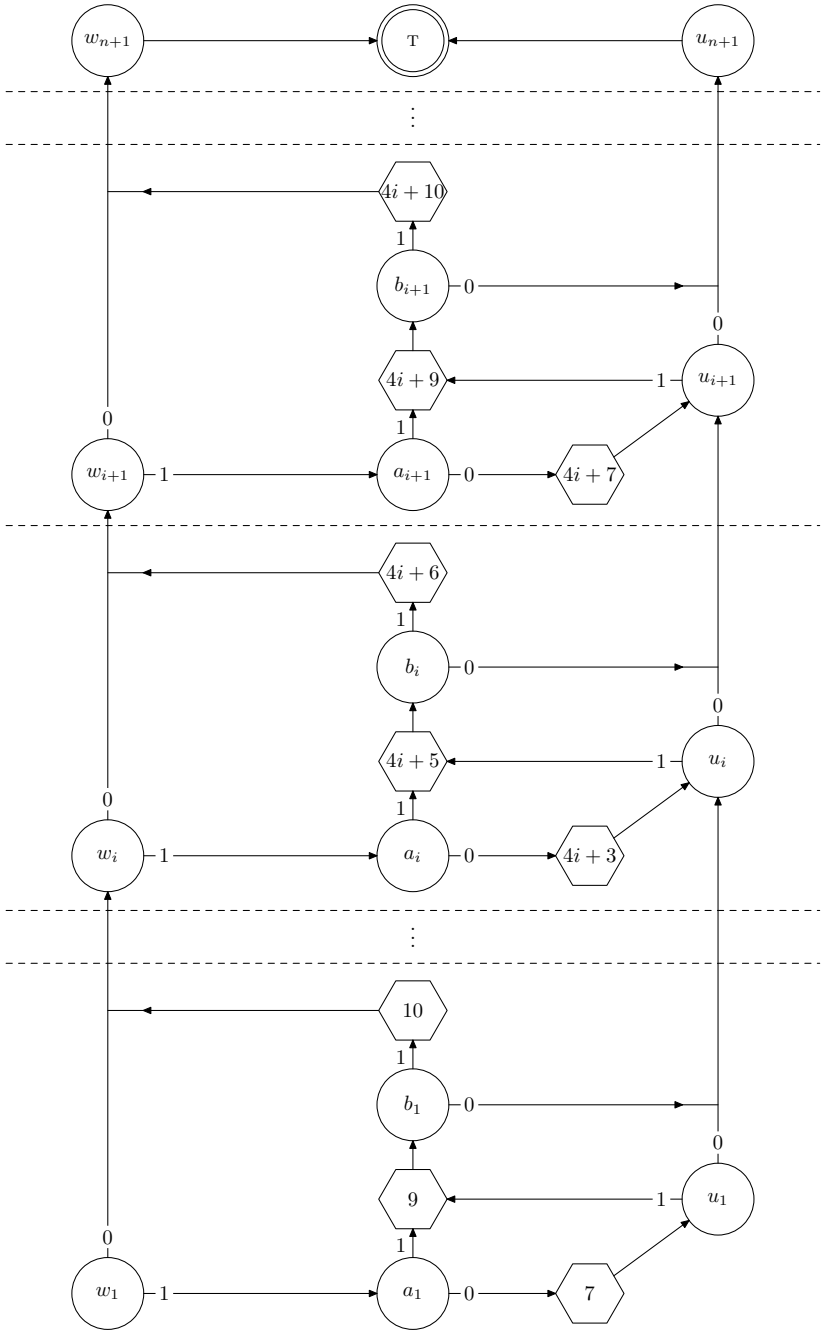


Figure 5.3: Simplified lower bound construction for the RandomFacet algorithm.

The MDP of Figure 5.3 is composed of n identical levels, each corresponding to a single bit of the counter. The 1-st, i-th and $(i+1)$-st levels are shown explicitly in the figure. Levels are separated by dashed lines. In addition to the n levels there is a special $(n+1)$-st level with a *sink* vertex (terminal state) T. The i-th level contains four vertices of V_0, namely, a_i, b_i, u_i and w_i. Each of these four vertices is associated with two actions, labelled 0 and 1 in Figure 5.3. A policy π can, thus, be described by a zero-one vector with each entry defining the choice of some state. We use the shorthand notation $b_i(\pi) = 0$ to say that $\pi(b_i) = u_{i+1}$, and similarly for the remaining states and actions. Furthermore, we use b_i^0 and b_i^1 as the names of the edges $b_i \to u_{i+1}$ and $b_i \to w_{i+1}$, respectively. We refer to b_i^0 and b_i^1 as the 0-edge and 1-edge, respectively, of b_i. Similar names are given to all the other edges. We view a policy π as a configuration of the counter $\mathfrak{b} \in \{0,1\}^n$ by saying that $\mathfrak{b}_i = 1$ if and only if $b_i(\pi) = 1$.

The RandomFacet algorithm works by removing edges and performing recursive calls with various subgraphs. Each subgraph G_F is defined by the subset F of edges that are still present. We are only interested in subgraphs in which only 1-edges leaving b_i and a_i vertices have been removed. Such subgraphs are said to be *functional*. More formally:

Definition 5.4.1 (Functional sets) *A subset $F \subseteq E_0$ is said to be* functional *if and only if $a_i^0, b_i^0, u_i^0, u_i^1, w_i^0, w_i^1 \in F$, for every $i \in [n]$.*

Recall that we interpret policies as subsets of edges. Given a subset $F \subseteq E_0$, we define the *reset level* of the subgraph G_F to be the highest level in which $b_i^1 \in F$ but $a_i^1 \notin F$. If there is no level satisfying this condition, the reset level is defined to be 0. Formally,

Definition 5.4.2 (Reset level) *For every $F \subseteq E_0$, let:*

$$reset(F) = \max\left(\{0\} \cup \{i \in [n] \mid b_i^1 \in F \wedge a_i^1 \notin F\}\right) .$$

The next lemma gives a precise description of the optimal policy, given a functional set of edges.

Lemma 5.4.3 *Let $F \subseteq E_0$ be functional. Then π_F^* is an optimal policy for the subgraph G_F if and only if:*

(i) $b_i(\pi_F^*) = 1$ *iff* $b_i^1 \in F$ *and* $i \geq reset(F)$,

(ii) $a_i(\pi_F^*) = 1$ *iff* $a_i^1, b_i^1 \in F$ *and* $i > reset(F)$.

(iii) $u_i(\pi^*_F) = 1$ *iff* $b^1_i \in F$ *and* $i \geq reset(F)$.

(iv) $w_i(\pi^*_F) = 1$ *iff* $i \leq reset(F)$ *or* $a^1_i, b^1_i \in F$.

We again refer to the $u_1, \ldots, u_{n+1}$ vertices as the right lane, and to the $w_1, \ldots, w_{n+1}$ vertices as the left lane. Optimal policies should be understood as follows. For all $i > reset(F)$ it is not difficult to prove by induction that $\mathrm{val}_{\pi^*_F}(u_i) = \mathrm{val}_{\pi^*_F}(w_i)$. The edge b^1_i should then be used if possible for all $i \geq reset(F)$, and if $b_i(\pi^*_F) = 1$ then the positive reward N^{4i+6} outweighs the negative rewards $-N^{4i+5}$ and $-N^{4i+3}$, and the edges a^1_i, u^1_i, and w^1_i should be used if possible. For $i \leq reset(F)$ it is not difficult to prove that $\mathrm{val}_{\pi^*_F}(w_i) = -N^{4i+3} + \mathrm{val}_{\pi^*_F}(u_i)$. Hence, it is optimal to use $b_i(\pi^*_F) = a_i(\pi^*_F) = u_i(\pi^*_F) = 0$ and $w_i(\pi^*_F) = 1$ for $i < reset(F)$. I.e., we move via the right lane up to the reset level. For the left lane it is best to exit the lane as soon as possible below the reset level.

Formally, Lemma 5.4.3 is proved by showing that there are no improving switches with respect to π^*_F. It is not difficult to derive this from the discussion above, but we will not present a formal proof here. Lemma 5.5.5 in Section 5.5 can be viewed as a generalization of Lemma 5.4.3, and we refer to the proof of Lemma 5.5.5 for additional details.

The following lemma follows quite easily from Lemma 5.4.3. I.e., if π^*_F is not optimal for $G_{F\cup\{e\}}$, then e must be an improving switch with respect to π^*_F.

Lemma 5.4.4 *Let $F \subseteq E_0$ be functional. Then:*

(i) If $b^1_i \notin F$, then b^1_i is an improving switch with respect to π^*_F iff $i > reset(F)$.

(ii) If $a^1_i \notin F$, then a^1_i is an improving switch with respect to π^*_F iff $i = reset(F)$.

Consider now an application of RandomFacet on G_n, starting from π_0; the policy consisting of all the edges labelled 0. Suppose that the first edge chosen by the algorithm is b^1_i, for $i \in [n]$. The recursive call returns the optimal strategy $\pi' = \pi^*_{E_0\setminus\{b^1_i\}}$ for the subgraph without the edge b^1_i. Since $reset(E_0 \setminus \{b^1_i\}) = 0$, it follows from Lemma 5.4.3 that all levels, except for level i, are using all the 1-edges. I.e., $\mathfrak{b}_j = 1$ for $j \neq i$, and $\mathfrak{b}_i = 0$. By Lemma 5.4.4, it follows that b^1_i is an improving switch with respect to π', and the second recursive call is performed with $\pi'' = \pi'[b^1_i]$. Note that we now have $\mathfrak{b}_i = 1$.

Suppose that the next edge chosen in the recursive call is a_i^1. We now have $reset(E_0 \setminus \{a_i^1\}) = i$. Hence, for $\pi^{(3)} = \pi^*_{E_0 \setminus \{a_i^1\}}$ we have $\mathfrak{b}_j = 0$ for all $j < i$, and $\mathfrak{b}_j = 1$ for all $j \geq i$. By Lemma 5.4.4, it follows that a_i^1 is an improving switch with respect to $\pi^{(3)}$, and a second recursive call of the RandomFacet algorithm is performed with $\pi^{(4)} = \pi^{(3)}[a_i^1]$. In this case we again have $reset(E_0) = 0$, and we are back to a stable situation. Hence, the operations of the RandomFacet algorithm simulated the first level of the recursion for the randomized counter RandCount.

It is worth noting that after the sequence of changes described above, the left lane has not been updated accordingly for $\pi^{(4)}$. It will, however, be updated later at the lower levels of the recursion before anything important happens.

5.5 The lower bound construction

We next present the full lower bound construction for the RandomFacet algorithm. We define a family of graphs $G_{n,g,h,r}$ as shown schematically in Figure 5.4. Here n is the number of levels in the graph; the number bits in the simulated randomized bit-counter. g, h, and r are parameters for the construction that will be decided at a later time.

Note that in order to ensure the correct behavior it was crucial that b_i^1 was picked by the RandomFacet algorithm before a_i^1. It was also crucial that other 0-edges were not removed. To increase the probability of that happening we make use of *duplication*. A shaded rectangle with g or h in the bottom-left corner indicates that the corresponding subgraph is copied g or h times, respectively. For instance, b_i is copied gh times. For $1 \leq j \leq gh$ we refer to the j-th copy as $b_{i,j}$. See Figure 5.5 for an expanded view of this duplication. The A_i and a_i vertices are copied g times, and then the $a_{i,j}$ vertices, for $j \in [g]$, are copied h times. Note that the b_i and a_i vertices are copied the same number of times. It will be convenient to group the b_i vertices in the same way as the a_i vertices. For $j \in [g]$ and $k \in [h]$ we, thus, let $b_{i,j,k} = b_{i,(j-1)h+k}$.

The bold edges in Figure 5.4 indicate that the corresponding edges have been copied r times. We use multi edges for convenience. A similar graph without multi edges can easily be defined by introducing additional vertices.

Formally, $G_{n,g,h,r} = (V_0, V_R, E, c, p)$ is defined as follows.

$$\begin{aligned} V_0 &:= \{b_{i,j,k}, a_{i,j,k} \mid i \in [n], j \in [g], k \in [h]\} \cup \{u_i, w_i \mid i \in [n+1]\} \\ V_R &:= \{\mathrm{B}_i \mid i \in [n]\} \cup \{\mathrm{A}_{i,j} \mid i \in [n], j \in [g]\} \end{aligned}$$

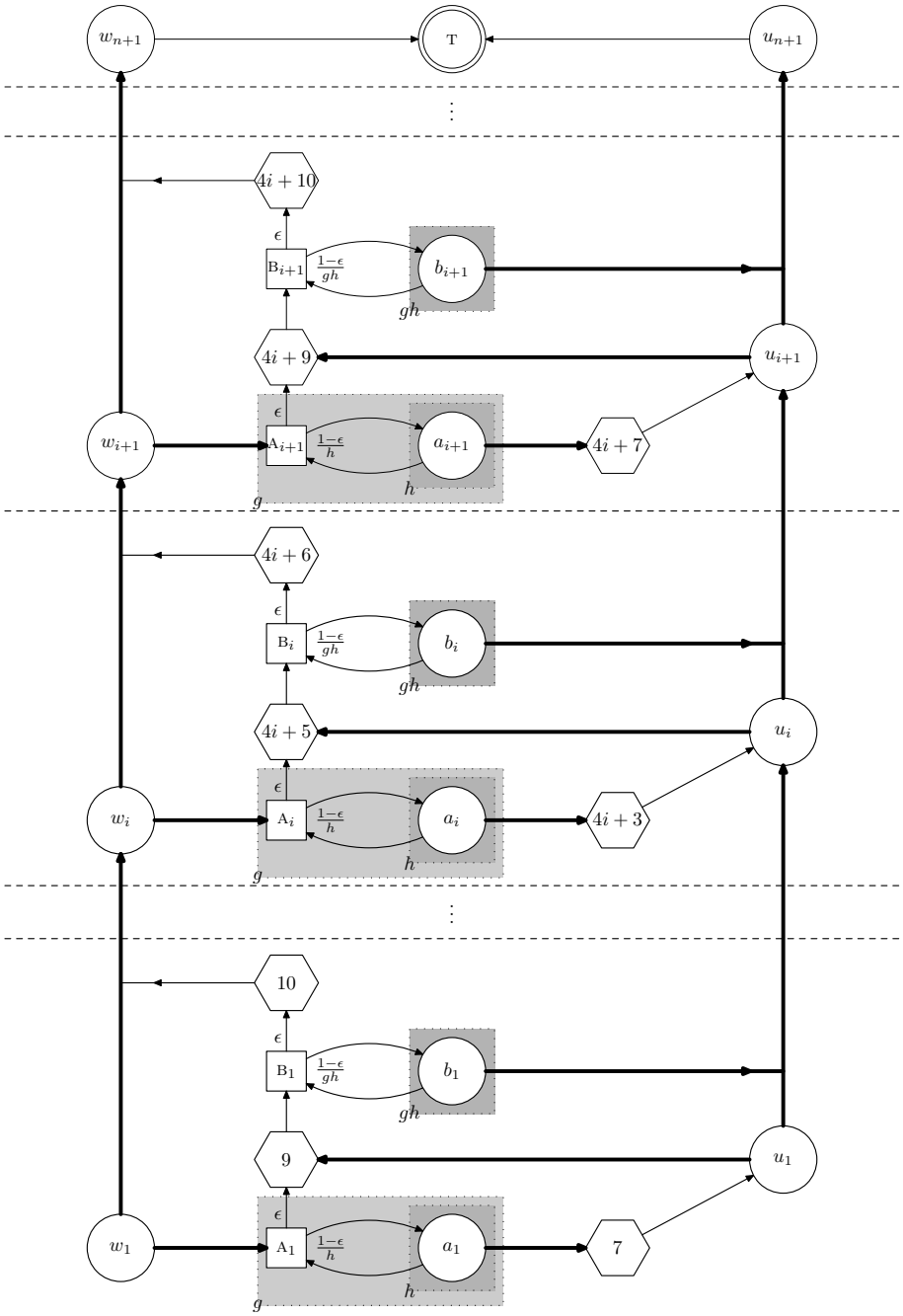


Figure 5.4: Lower bound MDP for the RandomFacet algorithm. The interpretation of the shaded rectangles is shown in Figure 5.5.

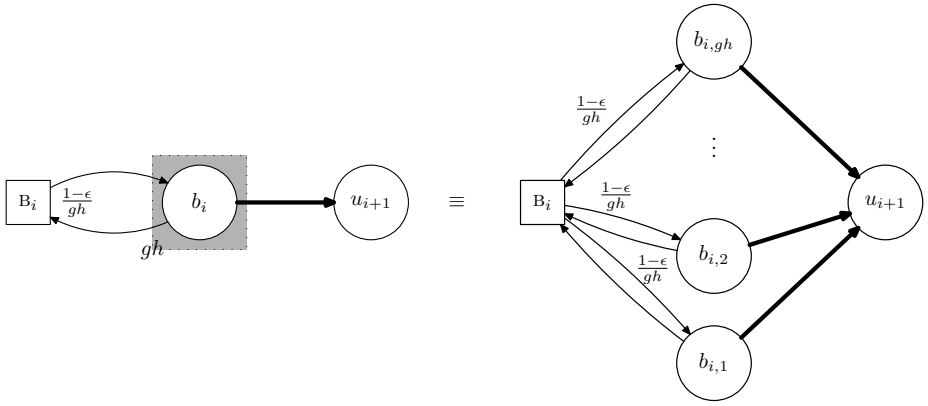


Figure 5.5: Duplication of a subgraph.

Vertex V_0	Successors in E	Priority	Multiplicity	Name
$a_{i,j,k}$	u_i	$4i+3$	r	$a^0_{i,j,k}$
	$\mathrm{A}_{i,j}$		1	$a^1_{i,j,k}$
$b_{i,j,k}$	u_{i+1}		r	$b^0_{i,j,k}$
	B_i		1	$b^1_{i,j,k}$
u_i	u_{i+1}		r	u^0_i
	B_i	$4i+5$	r	u^1_i
u_{n+1}	T		1	u^0_{n+1}
w_i	w_{i+1}		r	w^0_i
	$\mathrm{A}_{i,j}$		r	$w^{1,j}_i$
w_{n+1}	T		1	w^0_{n+1}
Vertex V_R	Successors in E_R	Priority	Probability	Name
$\mathrm{A}_{i,j}$	$a_{i,j,k}$		$\frac{1-\epsilon}{h}$	$\mathrm{A}^{0,k}_{i,j}$
	B_i	$4i+5$	ϵ	$\mathrm{A}^1_{i,j}$
B_i	$b_{i,j,k}$		$\frac{1-\epsilon}{gh}$	$\mathrm{B}^{0,j,k}_i$
	w_{i+1}	$4i+6$	ϵ	B^1_i

Table 5.1: Edges, priorities, multiplicities, names, and transition probabilities of $G_{n,g,h,r}$.

The edges connecting these vertices, and their respective priorities, multiplicities, and probabilities, are specified in Table 5.1. The edges are defined for all $i \in [n]$, $j \in [g]$, and $k \in [h]$. We use $N = 2$ and $\epsilon = N^{-(4n+8)}$. Recall that an edge with priority $p \in \mathbb{N}$ has reward $(-N)^p$. If no priority is specified the reward of the edge is zero. We also assign a name to every edge.

For every $i \in [n]$, we use the following abbreviations for sets of 1-edges:

$$\begin{aligned} \mathbf{a}^1_{i,j} &= \{a^1_{i,j,k} \mid k \in [h]\} \\ \mathbf{a}^1_{i} &= \{a^1_{i,j,k} \mid j \in [g], k \in [h]\} \\ \mathbf{b}^1_{i,j} &= \{b^1_{i,j,k} \mid k \in [h]\} \\ \mathbf{b}^1_{i} &= \{b^1_{i,j,k} \mid j \in [g], k \in [h]\} \end{aligned}$$

As before, we define the notion of *functional sets*.

Definition 5.5.1 (functional sets) *A subset $F \subseteq E_0$ is said to be* functional *if it contains at least one copy of each multi-edge.*

We define the *reset level* similarly to the way it was defined in Section 5.4. Since the b_i and a_i vertices have been copied we now operate with sets. We now interpret a policy π as a configuration $\mathfrak{b} \in \{0,1\}^n$ of a counter by saying that $\mathfrak{b}_i = 1$ if and only if $\mathbf{b}^1_i \subseteq \pi$. For $F \subseteq E_0$, we write:

$$\mathbf{a}^1_i \sqsubseteq F \quad \iff \quad \exists j \in [g] : \ \mathbf{a}^1_{i,j} \subseteq F$$

If it is not the case that $\mathbf{a}^1_i \sqsubseteq F$ we write $\mathbf{a}^1_i \not\sqsubseteq F$. I.e., if $\mathbf{a}^1_i \not\sqsubseteq F$ then no set $\mathbf{a}^1_{i,j}$ is completely contained in F.

Definition 5.5.2 (Reset level) *For every $F \subseteq E_0$ we define the* reset level *of F to be*

$$\mathit{reset}(F) = \max\left(\{0\} \cup \{i \in [n] \mid \mathbf{b}^1_i \subseteq F \ \wedge \ \mathbf{a}^1_i \not\sqsubseteq F\}\right)$$

Note that if $\mathit{reset}(F) = i$, then from $\textsc{B}_i$ we eventually move to w_{i+1}, obtaining the reward N^{4i+6}. On the other hand, for every $j \in [g]$ we move from $\textsc{A}_{i,j}$ to u_i with high probability, obtaining the negative reward $-N^{4i+3}$. As we shall see, all bits with index lower than i are reset in the optimal policy for the subgraph G_F.

For every functional subset $F \subseteq E_0$, we next define a set of policies Π^*_F for the subgraph G_F. We later show that Π^*_F is exactly the set of optimal policies in G_F. Before giving the definition of Π^*_F it will be helpful to define:

$$w_i(\pi) = \begin{cases} 0 & \text{if } w^0_i \in \pi \\ 1 & \text{if } w^{1,j}_i \in \pi, \text{ and } |\pi \cap \mathbf{a}^1_{i,j}| \geq |\pi \cap \mathbf{a}^1_{i,\ell}| \text{ for } \ell \in [g] \\ * & \text{otherwise} \end{cases}$$

Definition 5.5.3 (Π^*_F) *Let $F \subseteq E_0$ be functional. A policy π belongs to Π^*_F if and only if the following four requirements are satisfied:*

(i) $b^1_{i,j,k} \in \pi$ *iff* $b^1_{i,j,k} \in F$ *and* $i \geq reset(F)$.

(ii) $a^1_{i,j,k} \in \pi$ *iff* $a^1_{i,j,k} \in F$ *and* $\mathbf{b}^1_i \subseteq F$ *and* $i \geq reset(F)$.

(iii) $u^1_i \in \pi$ *iff* $\mathbf{b}^1_i \subseteq F$ *and* $i \geq reset(F)$.

(iv) $w_i(\pi) = 1$ *if* $i \leq reset(F)$, *or* $\mathbf{b}^1_i \subseteq F$ *and* $\mathbf{a}^1_i \sqsubseteq F$; *and* $w_i(\pi) = 0$ *otherwise.*

Note that the policies described by Definition 5.5.3 exactly correspond to the policies described by Lemma 5.4.3. Also note that if $\pi, \pi' \in \Pi^*_F$ then we can view π and π' as being equivalent. I.e., by renaming the vertices accordingly one policy can be turned into the other.

Consider a policy $\pi \in \Pi^*_F$, where F is functional. The edges $b^0_{i,j,k}$, $a^0_{i,j,k}$, u^0_i, u^1_i, w^0_i, and $w^{1,j}_i$, for $j \in [g]$, are always available since F is functional and these edges are multi-edges. Which copy of a multi-edge is chosen is immaterial. Note that if $i > reset(F)$ and $w_i(\pi') = 1$, then $w^{1,j}_i \in \pi$ for some j such that $\mathbf{a}^1_{i,j} \subseteq \pi$. I.e., in this case we move from w_i to B_i with probability 1 while getting the reward $-N^{4i+5}$. For $i = reset(F)$, w_i chooses the edge leading to the vertex $\mathrm{A}_{i,j}$ with most incoming edges in π. For $i < reset(F)$, the choice of w_i is immaterial since every choice leads to u_i while getting the reward $-N^{4i+3}$.

The next lemma describes all values of all vertices, given a policy $\pi \in \Pi^*_F$ w.r.t. a functional set F. We use $o(1)$ to denote a small positive term that involves ϵ. It will always be the case that $o(1)$ is less than 1. Let us note that it is not possible to pass through an edge with a priority more than once. In particular we have $|\mathrm{val}_\pi(v)| < N^{4n+7}$ for every policy π and every vertex $v \in V$. Hence, it will always be the case that $\epsilon|\mathrm{val}_\pi(v)| = o(1)$.

Lemma 5.5.4 *Let $F \subseteq E_0$ be a functional set and let $\pi \in \Pi^*_F$. Then,*

(*i*) *If* $i > reset(F)$ *and* $\mathbf{b}_i^1 \subseteq F$ *then*

$$\begin{aligned}
\mathrm{val}_\pi(w_{i+1}) &= \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(\mathrm{B}_i) &= N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \\
\mathrm{val}_\pi(b_{i,j,k}) &= N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \\
\mathrm{val}_\pi(u_i) &= -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \\
\mathrm{val}_\pi(\mathrm{A}_{i,j}) &= -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \quad , \text{ if } \mathbf{a}_{i,j}^1 \subseteq F \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \quad , \text{ if } \mathbf{a}_{i,j}^1 \subseteq F \\
\mathrm{val}_\pi(w_i) &= -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1})
\end{aligned}$$

For $j \in [g]$ *with* $\mathbf{a}_{i,j}^1 \not\subseteq F$*:*

$$\begin{aligned}
\mathrm{val}_\pi(\mathrm{A}_{i,j}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) + o(1) \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) \quad , \text{ if } a_{i,j,k}^1 \notin F \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) + o(1) \quad , \text{ if } a_{i,j,k}^1 \in F
\end{aligned}$$

(*ii*) *If* $i > reset(F)$ *and* $\mathbf{b}_i^1 \not\subseteq F$ *then*

$$\begin{aligned}
\mathrm{val}_\pi(w_{i+1}) &= \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(\mathrm{B}_i) &= \mathrm{val}_\pi(u_{i+1}) + o(1) \\
\mathrm{val}_\pi(b_{i,j,k}) &= \mathrm{val}_\pi(u_{i+1}) \quad , \text{ if } b_{i,j,k}^1 \notin F \\
\mathrm{val}_\pi(b_{i,j,k}) &= \mathrm{val}_\pi(u_{i+1}) + o(1) \quad , \text{ if } b_{i,j,k}^1 \in F \\
\mathrm{val}_\pi(u_i) &= \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(\mathrm{A}_{i,j}) &= -N^{4i+3} + \mathrm{val}_\pi(u_{i+1}) - o(1) \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+3} + \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(w_i) &= \mathrm{val}_\pi(w_{i+1})
\end{aligned}$$

(iii) If $i = reset(F)$ then

$$\begin{aligned}
\mathrm{val}_\pi(w_{i+1}) &= \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(\mathrm{B}_i) &= N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \\
\mathrm{val}_\pi(b_{i,j,k}) &= N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \\
\mathrm{val}_\pi(u_i) &= -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) \\
\mathrm{val}_\pi(\mathrm{A}_{i,j}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) + o(1) \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) && , \text{ if } a^1_{i,j,k} \notin F \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) + o(1) && , \text{ if } a^1_{i,j,k} \in F \\
\mathrm{val}_\pi(w_i) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) + o(1)
\end{aligned}$$

(iv) If $i < reset(F)$ then

$$\begin{aligned}
\mathrm{val}_\pi(w_{i+1}) &\leq -N^{4i+3} + \mathrm{val}_\pi(u_{i+1}) + 1 \\
\mathrm{val}_\pi(\mathrm{B}_i) &= \mathrm{val}_\pi(u_{i+1}) - o(1) \\
\mathrm{val}_\pi(b_{i,j,k}) &= \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(u_i) &= \mathrm{val}_\pi(u_{i+1}) \\
\mathrm{val}_\pi(\mathrm{A}_{i,j}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) - o(1) \\
\mathrm{val}_\pi(a_{i,j,k}) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) \\
\mathrm{val}_\pi(w_i) &= -N^{4i+3} + \mathrm{val}_\pi(u_i) - o(1)
\end{aligned}$$

Proof: Let π be any policy, and let $\ell = |\pi \cap \mathbf{b}^1_i|$ for some $i \in [n]$. Furthermore, let $L = \frac{(gh-\ell)(1-\epsilon)}{gh}$. Then we have:

$$\mathrm{val}_\pi(\mathrm{B}_i) = \frac{\epsilon}{\epsilon + L}(N^{4i+6} + \mathrm{val}_\pi(w_{i+1})) + \frac{L}{\epsilon + L}\mathrm{val}_\pi(u_{i+1})$$

Let $x = \mathrm{val}_\pi(u_{i+1})$ and $y = N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$, then it follows that:

$$\mathrm{val}_\pi(\mathrm{B}_i) = \begin{cases} N^{4i+6} + \mathrm{val}_\pi(w_{i+1}) & \text{if } \mathbf{b}^1_i \subseteq \pi \\ \mathrm{val}_\pi(u_{i+1}) + o(1) & \text{if } \mathbf{b}^1_i \not\subseteq \pi \text{ and } x < y \\ \mathrm{val}_\pi(u_{i+1}) - o(1) & \text{if } \mathbf{b}^1_i \not\subseteq \pi \text{ and } x > y \end{cases} \tag{5.1}$$

Let $x = -N^{4i+3} + \mathrm{val}_\pi(u_i)$ and $y = -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i)$. Similar observations can be made for $\mathrm{A}_{i,j}$, for $i \in [n]$ and $j \in [g]$, such that we get:

$$\mathrm{val}_\pi(\mathrm{A}_{i,j}) = \begin{cases} -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i) & \text{if } \mathbf{a}^1_{i,j} \subseteq \pi \\ -N^{4i+3} + \mathrm{val}_\pi(u_i) + o(1) & \text{if } \mathbf{a}^1_{i,j} \not\subseteq \pi \text{ and } x < y \\ -N^{4i+3} + \mathrm{val}_\pi(u_i) - o(1) & \text{if } \mathbf{a}^1_{i,j} \not\subseteq \pi \text{ and } x > y \end{cases} \tag{5.2}$$

The lemma is then proved by backwards induction in i. For the basis we have $\mathrm{val}_\pi(w_{n+1}) = \mathrm{val}_\pi(u_{n+1}) = 0$. The lemma is proved by studying Figure 5.4 and using (5.1) and (5.2) when needed. We only present the proof for case (i). The remaining cases are proved analogously.

Let $i > \mathit{reset}(F)$ and assume that $\mathbf{b}_i^1 \subseteq F$. It follows by the induction hypothesis that $\mathrm{val}_\pi(w_{i+1}) = \mathrm{val}_\pi(u_{i+1})$. Using (5.1) we then get that $\mathrm{val}_\pi(\mathrm{B}_i) = N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$. Since $b_{i,j,k}^1 \in \pi$ for all $j \in [g]$ and $k \in [h]$ we get $\mathrm{val}_\pi(b_{i,j,k}) = \mathrm{val}_\pi(\mathrm{B}_i) = N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$. Similarly, we get that $\mathrm{val}_\pi(u_i) = -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i) = -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$. Note that:

$$-N^{4i+3} + \mathrm{val}_\pi(u_i) = -N^{4i+3} - N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i) < -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i)$$

Let $j \in [g]$. Recall that $\mathbf{a}_{i,j}^1 \subseteq \pi$ if and only if $\mathbf{a}_{i,j}^1 \subseteq F$. Using again (5.2) we get that if $\mathbf{a}_{i,j}^1 \subseteq \pi$, then $\mathrm{val}_\pi(\mathrm{A}_{i,j}) = -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i) = -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$. It follows also that $\mathrm{val}_\pi(a_{i,j,k}) = -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$, for $k \in [h]$. Since $w_i(\pi) = 1$ we also get that $\mathrm{val}_\pi(w_i) = -N^{4i+5} + N^{4i+6} + \mathrm{val}_\pi(w_{i+1})$, which completes the induction step. The only remaining case is when $\mathbf{a}_{i,j}^1 \not\subseteq F$, which implies that $\mathbf{a}_{i,j}^1 \not\subseteq \pi$. We again use (5.2), and the claim follows easily. This completes case (i). □

Lemma 5.5.5 *Let $F \subseteq E_0$ be functional. Let $\pi \in \Pi_F^*$ and $e = (u, v) \in E_0$ such that $\pi[e] \notin \Pi_F^*$. Then $\mathrm{val}_\pi(u) > c(u, v) + \mathrm{val}_\pi(v)$. I.e., e is not an improving switch with respect to π.*

Proof: To prove the lemma we go through all the cases of Lemma 5.5.4, compute for every vertex $u \in V_0$ the value obtained by using an edge $(u, v) \in F \setminus \pi$ for one step, and conclude that it is smaller. Below we show the resulting inequalities. On the left we have vertices $u \in V_0$, and on the right we have $c(u, v) + \mathrm{val}_\pi(v)$. The inequalities follow easily from Lemma 5.5.4. The only edges not considered below are of the form $w_i^{1,j} \notin \pi$, for $i = \mathit{reset}(F)$ and $j \in [g]$. For such edges the inequality follows from the fact that $w_i(\pi) = 1$, which means that $|\pi \cap \mathbf{a}_{i,j}^1| \geq |\pi \cap \mathbf{a}_{i,\ell}^1|$ for $\ell \in [g]$. Thus, the edge chosen at w_i minimizes the probability of going through u_i to B_i instead of going directly to B_i, which maximizes the reward.

(i) If $i > \mathit{reset}(F)$ and $\mathbf{b}_i^1 \subseteq F$ then

$$\begin{array}{rcll}
\mathrm{val}_\pi(b_{i,j,k}) & > & \mathrm{val}_\pi(u_{i+1}) & \\
\mathrm{val}_\pi(u_i) & > & \mathrm{val}_\pi(u_{i+1}) & \\
\mathrm{val}_\pi(a_{i,j,k}) & > & -N^{4i+3} + \mathrm{val}_\pi(u_i) & , \text{ if } a_{i,j,k}^1 \in F \\
\mathrm{val}_\pi(w_i) & > & \mathrm{val}_\pi(w_{i+1}) & \\
\mathrm{val}_\pi(w_i) & > & \mathrm{val}_\pi(\mathrm{A}_{i,j}) & , \text{ if } \mathbf{a}_{i,j}^1 \not\subseteq F
\end{array}$$

(*ii*) If $i > reset(F)$ and $\mathbf{b}_i^1 \not\subseteq F$ then

$$\begin{array}{rcll} \mathrm{val}_\pi(b_{i,j,k}) & > & \mathrm{val}_\pi(u_{i+1}) & , \text{ if } b_{i,j,k}^1 \in F \\ \mathrm{val}_\pi(u_i) & > & -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i) & \\ \mathrm{val}_\pi(a_{i,j,k}) & > & \mathrm{val}_\pi(\mathrm{A}_{i,j}) & \\ \mathrm{val}_\pi(w_i) & > & \mathrm{val}_\pi(\mathrm{A}_{i,j}) & , \text{ for all } j \in [g] \end{array}$$

(*iii*) If $i = reset(F)$ then

$$\begin{array}{rcll} \mathrm{val}_\pi(b_{i,j,k}) & > & \mathrm{val}_\pi(u_{i+1}) & \\ \mathrm{val}_\pi(u_i) & > & \mathrm{val}_\pi(u_{i+1}) & \\ \mathrm{val}_\pi(a_{i,j,k}) & > & -N^{4i+3} + \mathrm{val}_\pi(u_i) & , \text{ if } a_{i,j,k}^1 \in F \\ \mathrm{val}_\pi(w_i) & > & \mathrm{val}_\pi(w_{i+1}) & \end{array}$$

(*iv*) If $i < reset(F)$ then

$$\begin{array}{rcl} \mathrm{val}_\pi(b_{i,j,k}) & > & \mathrm{val}_\pi(\mathrm{B}_i) \\ \mathrm{val}_\pi(u_i) & > & -N^{4i+5} + \mathrm{val}_\pi(\mathrm{B}_i) \\ \mathrm{val}_\pi(a_{i,j,k}) & > & \mathrm{val}_\pi(\mathrm{A}_{i,j}) \\ \mathrm{val}_\pi(w_i) & > & \mathrm{val}_\pi(w_{i+1}) \end{array}$$

□

Corollary 5.5.6 *Let $F \subseteq E_0$ be functional. Then Π_F^* is exactly the set of optimal policies for G_F.*

Proof: Recall that policies $\pi, \pi' \in \Pi_F^*$ are equivalent such that $\mathrm{val}_\pi(u) = \mathrm{val}_{\pi'}(u)$ for all $u \in V_0$. It then follows from Lemma 5.5.5 that there are no improving switches with respect to policies $\pi \in \Pi_F^*$. Hence, it follows from Theorem 2.4.6 that every policy of Π_F^* is optimal.

To see that Π_F^* contains all the optimal policies, we use that the inequalities in Lemma 5.5.5 are strict. Let $\pi' \notin \Pi_F^*$. Then there exists a policy $\pi \in \Pi_F^*$, such that for every edge $e \in \pi' \setminus \pi$, $\pi[e] \notin \Pi_F^*$. I.e., if $\pi[e] \in \Pi_F^*$ then we may use $\pi[e]$ instead of π. Hence, every such edge e is strictly non-improving w.r.t. π, and since $\pi' = \pi[\pi' \setminus \pi]$ we get from Theorem 2.4.6 (when we view the MDP as a minimization problem) that at least one vertex has strictly lower value for π' than for π. Hence, π' is not optimal. □

The following lemma corresponds to Lemma 5.4.4.

Lemma 5.5.7 *Let $F \subseteq E_0$ be functional, and let $\pi \in \Pi_F^*$. Then*

(*i*) *If* $b^1_{i,j,k} \notin F$ *and* $i \geq reset(F)$*, then* $b^1_{i,j,k}$ *is an improving switch with respect to* π*.*

(*ii*) *If* $a^1_{i,j,k} \notin F$ *and* $i = reset(F)$*, then* $a^1_{i,j,k}$ *is an improving switch with respect to* π*.*

Proof: We first prove (*i*). Let $F' = F \cup \{b^1_{i,j,k}\}$. It is not difficult to see that $i \geq reset(F')$. By Definition 5.5.3 and Corollary 5.5.6 every policy of $\Pi^*_{F'}$ must contain $b^1_{i,j,k}$. Thus, $\pi \notin \Pi^*_{F'}$, and by Theorem 2.4.6 there must be an edge in F' which is an improving switch with respect to π. Since no edge in F is an improving switch, $b^1_{i,j,k}$ must be an improving switch with respect to π.

The proof of (*ii*) is similar. Let $F' = F \cup \{a^1_{i,j,k}\}$. Since $\mathbf{b}^1_i \subseteq F$ we again have $i \geq reset(F')$, and $\pi \notin \Sigma_{F'}$, and it again follows that $a^1_{i,j,k}$ must be an improving switch with respect to π. □

5.6 The lower bound for RandomFacet

In this section we let $G = G_{n,g,h,r}$. We also let π_0 be the policy composed solely of 0-edges.

First, we need some definitions that allow us to identify interesting events along a potential path. Let therefore $P = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ be a potential path. Note that the edges $e_1, e_2, \ldots, e_k$ on P are all distinct. We define $\sigma_P : E_0 \to [k] \cup \{\infty\}$ such that for all $e \in E_0$:

$$\sigma_P(e) \;=\; \begin{cases} \ell & \text{if } e \text{ appears along } P, \text{ and } e = e_\ell \\ \infty & \text{otherwise} \end{cases}$$

I.e., $\sigma_P(e)$ is the index of e in P, if it appears in P, and ∞ otherwise. Let M be the set of multi-edges. We view every element $\mathbf{e} \in M$ as a set $\mathbf{e} = \{e^1, e^2, \ldots, e^r\}$ of r identical edges. We now define:

$$\begin{aligned}
\forall i \in [n], j \in [g]: \quad \sigma_P(\mathbf{b}^1_{i,j}) &= \min_{k \in [h]} \sigma_P(b^1_{i,j,k}) \\
\forall i \in [n]: \quad \sigma_P(\mathbf{b}^1_{i}) &= \min_{j \in [g]} \sigma_P(\mathbf{b}^1_{i,j}) \\
\forall i \in [n], j \in [g]: \quad \sigma_P(\mathbf{a}^1_{i,j}) &= \min_{k \in [h]} \sigma_P(a^1_{i,j,k}) \\
\forall i \in [n]: \quad \sigma_P(\mathbf{a}^1_{i}) &= \max_{j \in [g]} \sigma_P(\mathbf{a}^1_{i,j}) \\
\forall \mathbf{e} \in M: \quad \sigma_P(\mathbf{e}) &= \max_{j \in [r]} \sigma_P(e^j)
\end{aligned}$$

The index $\sigma_P(\mathbf{b}_i^1)$ refers to the first occurrence of an edge from $\mathbf{b}_i^1$; the set of edges used to define the i-th bit. Note that removing such an edge corresponds to *disabling* the bit. I.e., if $\mathbf{b}_i^1 \not\subseteq \pi$ then $\mathfrak{b}_i = 0$. The index $\sigma_P(\mathbf{a}_i^1)$ refers to the first occurrence of an edge from $\mathbf{a}_{i,j}^1$ such that edges from $\mathbf{a}_{i,j'}^1$, for $j' \neq j$, have already appeared. In particular, $\sigma_P(\mathbf{a}_i^1)$ is the minimum index ℓ for which $\mathbf{a}_i^1 \not\sqsubseteq E_0 \setminus \{e_1, \ldots, e_\ell\}$. The index $\sigma_P(\mathbf{e})$ for $\mathbf{e} \in M$ refers to the last occurrence of a copy of the edge e.

Next, we define potential paths in which the random selection of edges is well-behaved.

Definition 5.6.1 (Good paths) *Let $P = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ be a potential path. We say that P is* good *if and only if:*

(i) $\sigma_P(\mathbf{e}) = \infty$ *for all* $\mathbf{e} \in M$.

(ii) $\sigma_P(\mathbf{b}_i^1) \leq \sigma_P(\mathbf{a}_i^1)$ *for all* $i \in [n]$.

The following lemma follows immediately from Definition 5.6.1.

Lemma 5.6.2 *Let $P = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ be a good path, and let $T = (U, D, F, \pi, e, d, u_0)$ be a computation tree. Assume that $P \in T$, and let $u_0, \ldots, u_k \in U$ be the nodes corresponding to P in T. Then $F = F(u_{\ell-1}) \setminus \{e_\ell\}$ is functional for all $\ell \in [k]$.*

Definition 5.6.3 (Candidate paths, canonical paths) *Let $n \geq i_1 > i_2 > \cdots > i_t \geq 1$, and let $P = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ be a good path that satisfies:*

(i) $\sigma_P(\mathbf{b}_{i_p}^1) \leq \sigma_P(\mathbf{a}_{i_p}^1) \leq \sigma_P(\mathbf{b}_{i_q}^1)$ *for all* $1 \leq p < q \leq t$.

(ii) For every $\ell \in [k]$, $d_\ell = R$ if and only if

$$\ell \in \{\sigma_P(\mathbf{b}_{i_p}^1) \mid p \in [t]\} \cup \{\sigma_P(\mathbf{a}_{i_p}^1) \mid p \in [t-1]\} .$$

If $k \leq \sigma_P(\mathbf{b}_{i_t}^1)$ then we say that P is an $(i_1, i_2, \ldots, i_t)$-candidate path. *If $k = \sigma_P(\mathbf{b}_{i_t}^1)$ then we say that P is* $(i_1, i_2, \ldots, i_t)$-canonical. *We let $canon_G(i_1, i_2, \ldots, i_t)$ be the set of all $(i_1, i_2, \ldots, i_t)$-canonical paths for G.*

Note that every $(i_1, i_2, \ldots, i_t)$-canonical path is also an $(i_1, i_2, \ldots, i_t)$-candidate path. Also note that every canonical path ends with an arc going right. In particular, we have $canon_G(i_1, i_2, \ldots, i_t) \subseteq paths(G)$. Furthermore, if $(i_1, i_2, \ldots, i_t)$ is different from $(j_1, j_2, \ldots, j_s)$, then $canon_G(i_1, i_2, \ldots, i_t)$

and $canon_G(j_1, j_2, \ldots, j_s)$ are disjoint. Let $S_n = 2^{[n]}$ be the set of all subsets of $[n]$, and recall that we denote $appear_{E_0,\pi_0}(P)$ by $appear(P)$. It follows that:

$$\sum_{P \in paths(G)} \Pr\left[appear(P)\right] \;\geq\; \sum_{s \in S_n} \sum_{P \in canon_G(s)} \Pr\left[appear(P)\right]$$

For our lower bound we are interested in the probability of the appearance of canonical paths, and in order to estimate such probabilities it will be helpful to study candidate paths.

The policies that are relevant for our analysis, have a very special structure described by the following two definitions.

Definition 5.6.4 (k-stable) *A policy π is* k-stable *w.r.t. $F \subseteq E_0$, for $k \in [n+1]$, if and only if:*

(i) For all $i \geq k$, $\mathbf{b}_i^1 \cap F \subseteq \pi$.

(ii) For all $i < k$, $\mathbf{b}_i^1 \cap \pi = \emptyset$.

(iii) For all $i \in [n]$, $\mathbf{a}_i^1 \cap F \subseteq \pi$ if $\mathbf{b}_i^1 \subseteq \pi$, and $\mathbf{a}_i^1 \cap \pi = \emptyset$ otherwise.

Definition 5.6.5 (k-unstable) *A policy π is* k-unstable *w.r.t. $F \subseteq E_0$, for $k \in [n]$, if and only if:*

(i) For all $i \in [n]$, $\mathbf{b}_i^1 \cap F \subseteq \pi$.

(ii) $\mathbf{a}_k^1 \cap \pi = \emptyset$.

(iii) For all $i \in [n] \setminus \{k\}$, $\mathbf{a}_i^1 \cap F \subseteq \pi$ if $\mathbf{b}_i^1 \subseteq \pi$, and $\mathbf{a}_i^1 \cap \pi = \emptyset$ otherwise.

Note that the initial policy π_0 is $(n+1)$-stable with respect to E_0. Also note that if π is k-stable (or k-unstable) w.r.t. F, and $e \in F \setminus \pi$, then π is also k-stable (or k-unstable) w.r.t. $F \setminus \{e\}$. Let $F \subseteq E_0$ be functional, and assume that $reset(F) = 0$. Let $e \in F$ be an edge, let $\pi \in \Pi^*_{F \setminus \{e\}}$, and let $\pi' = \pi[e]$. If $reset(F \setminus \{e\}) = k$ then $e \in \mathbf{a}_k^1$ and π' is k-stable. If $e \in \mathbf{b}_k^1$ then π' is k-unstable.

Lemma 5.6.6 *Let $P = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ be an $(i_1, i_2, \ldots, i_t)$-candidate path. Let $T = (U, D, F, \pi, e, d, u_0)$ be a computation tree. Assume that $P \in T$, and let $u_0, \ldots, u_k \in U$ be the nodes corresponding to P in T. Define $i_0 := n + 1$ and $\sigma_P(\mathbf{a}_{i_0}^1) := 1$. Then for all $\ell \in [k]$:*

(*i*) *For all* $p \in [t]$, *if* $\sigma_P(\mathbf{a}^1_{i_{p-1}}) \leq \ell < \sigma_P(\mathbf{b}^1_{i_p})$ *then* $\pi(u_\ell)$ *is* i_{p-1}*-stable w.r.t.* $F(u_\ell)$.

(*ii*) *For all* $p \in [t-1]$, *if* $\sigma_P(\mathbf{b}^1_{i_p}) \leq \ell < \sigma_P(\mathbf{a}^1_{i_p})$ *then* $\pi(u_\ell)$ *is* i_p*-unstable w.r.t.* $F(u_\ell)$.

Furthermore, $reset(F(u_\ell)) = 0$.

Proof: To shorten notation, define $b^p = \sigma_P(\mathbf{b}^1_{i_p})$, for $p \in [t]$, and $a^p = \sigma_P(\mathbf{a}^1_{i_p})$, for $0 \leq p < t$. Note that $a^0 \leq b^1 \leq a^1 \leq \cdots \leq b^{t-1} \leq a^{t-1} \leq b^t$.

We prove the lemma by induction in p. For $p = 1$ and $1 = a^0 \leq \ell < b^1$ we have $i_{p-1} = i_0 = n+1$ and $\pi(u_\ell) = \pi_0$, and (*i*) is satisfied. Recall that if $d(u_{\ell-1}, u_\ell) = L$ then $F(u_\ell) = F(u_\ell) \setminus \{e_\ell\}$. Since P is a good path and $reset(F(u_0)) = 0$ it follows from Definition 5.6.1 (*ii*) that $reset(F(u_\ell)) = 0$.

We now consider (*ii*). Let $p \in [t-1]$. Then for all $b^p \leq \ell < a^p$ we have $\pi(u_\ell) = \pi(u_{b^p}) = \pi(((u_{b^p-1})_L)^R)[e_{b^p}]$. Since P is a good path it follows from Lemma 5.6.2 that $F = F(u_{b^p-1}) \setminus \{e_{b^p}\}$ is functional, and since $\pi' = \pi(((u_{b^p-1})_L)^R)$ is optimal for G_F we get from Corollary 5.5.6 that $\pi' \in \Pi^*_F$. Furthermore, we know by induction that $reset(F) = 0$. It follows from Definition 5.5.3 that $\pi(u_\ell) = \pi'[e_{b^p}]$ is i_p-unstable w.r.t. $F(u_\ell)$. We again get that $reset(F(u_\ell)) = 0$ because $reset(F(u_{b^p})) = 0$ and P satisfies Definition 5.6.1 (*ii*).

We next consider (*i*). Let $p \in [t]$. Then for all $a^{p-1} \leq \ell < b^p$ we have $\pi(u_\ell) = \pi(u_{a^{p-1}}) = \pi(((u_{a^{p-1}-1})_L)^R)[e_{a^{p-1}}]$. Since P is a good path it follows from Lemma 5.6.2 that $F = F(u_{a^{p-1}-1}) \setminus \{e_{a^{p-1}}\}$ is functional, and since $\pi' = \pi(((u_{a^{p-1}-1})_L)^R)$ is optimal for G_F we get from Corollary 5.5.6 that $\pi' \in \Pi^*_F$. Since $reset(F \cup \{e_{a^{p-1}}\}) = 0$ by induction, and $\mathbf{a}^1_{i_{p-1}} \not\sqsubseteq F$, we have $reset(F) = i_{p-1}$. It follows from Definition 5.5.3 that $\pi(u_\ell) = \pi'[e_{a^{p-1}}]$ is i_{p-1}-stable w.r.t. $F(u_\ell)$. We again get that $reset(F(u_\ell)) = 0$ because $reset(F(u_{a^{p-1}})) = 0$ and P satisfies Definition 5.6.1 (*ii*). □

Lemma 5.6.7 *Let* $P = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$ *be an* $(i_1, i_2, \ldots, i_t)$*-candidate path. Let* $T = (U, D, F, \pi, e, d, u_0)$ *be a computation tree. Assume that* $P \in T$, *and let* $u_0, \ldots, u_k \in U$ *be the nodes corresponding to* P *in* T. *Let* u_{k+1} *be a child of* u_k, *such that* $(u_k, u_{k+1}) \in D$, *and let* $e_{k+1} = e(u_k, u_{k+1})$. *Define potential paths* $P_d = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k), (e_{k+1}, d) \rangle$, *for* $d \in \{L, R\}$. *Then neither* P_L *nor* P_R *is an* $(i_1, i_2, \ldots, i_t)$*-candidate path if* P_L *has one of the following properties:*

(*i*) $\sigma_{P_L}(\mathbf{e}) = k+1$, *for any* $\mathbf{e} \in M$.

(*ii*) $\sigma_{P_L}(\mathbf{a}^1_i) = k+1$ *and* $\sigma_{P_L}(\mathbf{b}^1_i) = \infty$, *for any* $i \in [n]$.

(*iii*) $\sigma_{P_L}(\mathbf{b}^1_{i_p}) = k+1$ *and* $\sigma_{P_L}(\mathbf{b}^1_{i_q}) = \infty$, *where* $q < p$.

Assume that P_L does not satisfy (i), (ii), or (iii). If $\sigma_{P_L}(\mathbf{a}^1_{i_p}) = k+1$, for $p \in [t-1]$, or $\sigma_{P_L}(\mathbf{b}^1_{i_p}) = k+1$, for $p \in [t]$, then P_R is an $(i_1, i_2, \ldots, i_t)$-candidate path, and $P_R \in T$; otherwise P_L is an $(i_1, i_2, \ldots, i_t)$-candidate path, and $P_L \in T$.

Proof: It follows immediately from Definition 5.6.3 that P_L and P_R are not $(i_1, i_2, \ldots, i_t)$-candidate paths if either (*i*), (*ii*), or (*iii*) is satisfied. For the remainder of the proof we assume that neither (*i*), nor (*ii*), nor (*iii*) is satisfied.

Assume that $\sigma_{P_L}(\mathbf{a}^1_{i_p}) = k+1$, for $p \in [t-1]$. Because of (*ii*) we know that $\sigma_{P_L}(\mathbf{b}^1_{i_p}) \leq k$, and since P is an $(i_1, i_2, \ldots, i_t)$-candidate path we know that $\sigma_{P_L}(\mathbf{b}^1_{i_{p+1}}) = \infty$. It follows that P_R is an $(i_1, i_2, \ldots, i_t)$-candidate path. Furthermore, we know from Lemma 5.6.6 (*ii*) that $reset(F(u_k)) = 0$, and, hence, $reset(F(u_k) \setminus \{e_{k+1}\}) = i_p$. We also know from Lemma 5.6.2 that $F(u_k) \setminus \{e_{k+1}\}$ is functional. It then follows from Lemma 5.5.7 (*ii*) that $e_{k+1} \in \mathbf{a}^1_{i_p}$ is an improving switch with respect to $\pi(((u_k)_L)^R) \in \Sigma^*_{F(u_k)\setminus\{e_{k+1}\}}$. Hence, u_k has a right child, and $P_R \in T$.

Assume next that $\sigma_{P_L}(\mathbf{b}^1_{i_p}) = k+1$, for $p \in [t]$. Because of (*iii*) we know that $\sigma_{P_L}(\mathbf{b}^1_{i_{p-1}}) \leq k$. Furthermore, since P is an $(i_1, i_2, \ldots, i_t)$-candidate path we know that $\sigma_{P_L}(\mathbf{b}^1_{i_{p+1}}) = \infty$. Assume for the sake of contradiction that $\sigma_P(\mathbf{a}^1_{i_{p-1}}) = \infty$. Then it follows from Lemma 5.6.6 (*ii*) that $\mathbf{b}^1_{i_p} \cap F(u_k) \subseteq \pi(u_k)$, but since $e_{k+1} \in \mathbf{b}^1_{i_p}$ and $e_{k+1} \in F(u_k) \setminus \pi(u_k)$ this is a contradiction. Hence, $\sigma_P(\mathbf{a}^1_{i_{p-1}}) \leq k$. We, thus, have $\sigma_{P_R}(\mathbf{a}^1_{i_{p-1}}) < \sigma_{P_R}(\mathbf{b}^1_{i_p}) = k+1 < \sigma_{P_R}(\mathbf{b}^1_{i_{p+1}})$. It follows that P_R is an $(i_1, i_2, \ldots, i_t)$-candidate path. We know from Lemma 5.6.6 (*i*) that $reset(F(u_k)) = 0$, and, hence, $reset(F(u_k) \setminus \{e_{k+1}\}) = 0$. We also know from Lemma 5.6.2 that $F(u_k) \setminus \{e_{k+1}\}$ is functional. It then follows from Lemma 5.5.7 (*i*) that $e_{k+1} \in \mathbf{b}^1_{i_p}$ is an improving switch with respect to $\pi(((u_k)_L)^R) \in \Sigma^*_{F(u_k)\setminus\{e_{k+1}\}}$. Hence, u_k has a right child, and $P_R \in T$.

Finally, assume that $e_{k+1} \notin \mathbf{a}^1_{i_p}$, for $p \in [t-1]$, and $e_{k+1} \notin \mathbf{b}^1_{i_p}$, for $p \in [t]$. Since neither (*i*), nor (*ii*), nor (*iii*) is satisfied, we immediately get that P_L is an $(i_1, i_2, \ldots, i_t)$-candidate path. Also, since the left child of u_k exists we get that $P_L \in T$. This completes the proof. □

Let us note that if a computation tree T contains two $(i_1, i_2, \ldots, i_t)$-candidate paths, then, because of Definition 5.6.3 (*ii*), one must be a prefix

of the other. It follows that every computation tree T contains a unique $(i_1, i_2, \ldots, i_t)$-candidate path of maximal length (possibly the empty path). We let $P_T(i_1, i_2, \ldots, i_t)$ be the unique $(i_1, i_2, \ldots, i_t)$-candidate path in T of maximal length. We also let $P'_T(i_1, i_2, \ldots, i_t)$ be the extension of $P_T(i_1, i_2, \ldots, i_t)$ by one arc going left. More precisely, let

$$P = P_T(i_1, i_2, \ldots, i_t) = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle \ ,$$

and let $u_0, \ldots, u_k \in U$ be the nodes corresponding to P in T. Since P is an $(i_1, i_2, \ldots, i_t)$-candidate path, u_k has a left child u_L. We then have

$$P'_T(i_1, i_2, \ldots, i_t) = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k), (e(u_k, u_L), L) \rangle \ .$$

Note that T contains an $(i_1, i_2, \ldots, i_t)$-canonical path if and only if $P_T(i_1, i_2, \ldots, i_t)$ is $(i_1, i_2, \ldots, i_t)$-canonical. Also note that it is not possible for T to contain two different $(i_1, i_2, \ldots, i_t)$-canonical paths, since, due to Definition 5.6.3, both paths would be required to end at the same time, meaning that the paths would be identical. Hence, if P and P' are both $(i_1, i_2, \ldots, i_t)$-canonical, then $\mathit{appear}(P)$ and $\mathit{appear}(P')$ are two disjoint events.

In the following let $n \geq i_1 > i_2 > \cdots > i_t \geq 1$ be given.

Definition 5.6.8 (Canonical events) *We let $\mathit{appear}(i_1, i_2, \ldots, i_t)$ be the event that the computation tree generated by* RandomFacet(E_0, π_0) *contains an $(i_1, i_2, \ldots, i_t)$-canonical path. I.e.,*

$$\mathit{appear}(i_1, i_2, \ldots, i_t) \ = \bigcup_{P \in \mathit{canon}_{G_{F,\pi}}(i_1, i_2, \ldots, i_t)} \mathit{appear}(P)$$

Let T be a random computation tree generated by RandomFacet(E_0, π_0). If $P = P_T(i_1, i_2, \ldots, i_t)$ is not $(i_1, i_2, \ldots, i_t)$-canonical, then we know from Lemma 5.6.7 that $P' = P'_T(i_1, i_2, \ldots, i_t)$ must satisfy either (i), (ii), or (iii) from Lemma 5.6.7. Let k be the length of P. We define corresponding events as follows:

- (i) For every $\mathbf{e} \in M$, let $\mathit{Bad}_1(\mathbf{e})$ be the event that P is not $(i_1, i_2, \ldots, i_t)$-canonical, and that $\sigma_{P'}(\mathbf{e}) = k + 1$.
- (ii) For every $i \in [n]$, let $\mathit{Bad}_2(i)$ be the event that P is not $(i_1, i_2, \ldots, i_t)$-canonical, that $\sigma_{P'}(\mathbf{a}^1_i) = k + 1$, and that $\sigma_{P'}(\mathbf{b}^1_i) = \infty$.
- (iii) For every $p \in [t]$, let $\mathit{Bad}_3(p)$ be the event that P is not $(i_1, i_2, \ldots, i_t)$-canonical, $\sigma_{P_L}(\mathbf{b}^1_{i_q}) = k + 1$ for some $q > p$, and that $\sigma_{P'}(\mathbf{b}^1_{i_p}) = \infty$.

We let $Good_1(\mathbf{e})$, $Good_2(i)$, and $Good_3(p)$ be the complements of $Bad_1(\mathbf{e})$, $Bad_2(i)$, and $Bad_3(p)$, respectively. We also define

$$Bad_1 = \bigcup_{\mathbf{e}\in M} Bad_1(\mathbf{e}) , \quad Bad_2 = \bigcup_{i\in[n]} Bad_2(i) , \quad Bad_3 = \bigcup_{p\in[t]} Bad_3(p) ,$$

and let $Good_1$, $Good_2$, and $Good_3$ be the complements of Bad_1, Bad_2, and Bad_3, respectively.

When estimating $\Pr[appear(i_1, i_2, \ldots, i_t)]$ we may think of the call to RandomFacet(E_0, π_0) as a randomized process that either generates an $(i_1, i_2, \ldots, i_t)$-canonical path, or fails due to Bad_1, Bad_2, or Bad_3. I.e., we imagine terminating the process as soon as one of these situations occur. Note that if this process terminates at a node u of the computation tree, then u is part of $P' = P'_T(i_1, i_2, \ldots, i_t)$. In particular, we may restrict our attention to edges picked at nodes along the path P'. Note also that the events Bad_1, Bad_2, and Bad_3 are disjoint. I.e., Bad_1 occurs when picking a multi-edge, Bad_2 occurs when picking an edge from some $\mathbf{a}^1_i$, and Bad_3 occurs when picking an edge from some $\mathbf{b}^1_i$. Finally, let us note that the process may be viewed as working in phases, with a new phase starting every time we follow a right arc along the $(i_1, i_2, \ldots, i_t)$-candidate path. In the $(2p-1)$-st phase, for $p \in [t]$, the policy assigned to the corresponding nodes of T is i_{p-1}-stable, and in the $(2p)$-th phase, for $p \in [t-1]$, the policy assigned to the corresponding nodes of T is i_p-unstable.

The main technical lemma of this section, from which the desired lower bound on the expected number of steps performed by the RandomFacet algorithm easily follows, is the following lemma. To get the lemma, the parameters g, h, and r are chosen appropriately.

Lemma 5.6.9 *For every $t \le \sqrt{n}$ and every $n \ge i_1 > i_2 > \cdots > i_t \ge 1$ we have*

$$\Pr[appear(i_1, i_2, \ldots, i_t)] \ge \frac{1}{2\,t!} .$$

Most of the remainder of this section is devoted to the proof of Lemma 5.6.9. Since Bad_1 and Bad_2 are disjoint we get:

$$\begin{aligned}
&\Pr[appear(i_1, i_2, \ldots, i_t)] = \\
&\Pr[Good_1 \wedge Good_2 \wedge Good_3] = \\
&\Pr[Good_3] \Pr[Good_1 \wedge Good_2 \mid Good_3] = \\
&\Pr[Good_3] (1 - \Pr[Bad_1 \mid Good_3] - \Pr[Bad_2 \mid Good_3])
\end{aligned}$$

Lemma 5.6.10 $\Pr[Good_3] \geq \frac{1}{t!}$

Proof: The events $Good_3(p)$ and $Good_3(q)$ are in general not disjoint for $p \neq q$. We therefore use that:

$$\Pr\left[Good_3\right] \;=\; \prod_{p=1}^{t} \Pr\left[Good_3(p) \mid Good_3(p-1), \ldots, Good_3(1)\right] \;.$$

We then prove that $\Pr[Good_3(p) \mid Good_3(p-1), \ldots, Good_3(1)] \geq \frac{1}{t-p}$ for every $p \in [t]$.

Let $P' = P'_T(i_1, i_2, \ldots, i_t)$, and let ℓ be the index of the first edge from $\cup_{q=p}^{t} \mathbf{b}^1_{i_q}$ appearing in P'. If no such index exists we get the event $Good_3(p)$. Assume that ℓ exists. Since the sets $\mathbf{b}^1_{i_q}$ all have the same size we get that $\ell = \sigma_{P'}(\mathbf{b}^1_{i_p})$ with probability $\frac{1}{t-p}$. In this case we again get the event $Good_3(p)$. The lemma follows. □

In order to bound the probabilities of $\Pr[Bad_1 \mid Good_3]$ and $\Pr[Bad_2 \mid Good_3]$), we can now consider a modified process which is guaranteed not to fail because of the third condition. Whenever an edge from $\mathbf{b}^1_{i_1}, \mathbf{b}^1_{i_2}, \ldots, \mathbf{b}^1_{i_t}$ is chosen for the first time, we make sure that it is the right one. Note that $\Pr[Bad_1 \mid Good_3] \geq \Pr[Bad_1]$, and similarly for Bad_2. I.e., if the process fails with the event Bad_3, then it does not fail with the event Bad_1 or the event Bad_2.

Lemma 5.6.11 *For every multi-edge* $\mathbf{e} \in M$*, we have* $\Pr[Bad_1(\mathbf{e}) \mid Good_3] \leq n^{-2}$*, assuming* $h = 2(g+1)t + r$ *and* $r = \Theta(\log n)$*.*

Proof: Let $P' = P'_T(i_1, i_2, \ldots, i_t) = \langle (e_1, d_1), (e_2, d_2), \ldots, (e_k, d_k) \rangle$. We introduce the short-hand notation $a^{p,j} = \sigma_{P'}(\mathbf{a}^1_{i_p,j})$, for $p \in [t-1]$ and $j \in [g]$, and $b^p = \sigma_{P'}(\mathbf{b}^1_{i_p})$, for $p \in [t]$. Observe that if $a^{p,j} < \infty$, for every $p \in [t-1]$ and $j \in [g]$, and $b^p < \infty$, for every $p \in [t]$, then $P_T(i_1, i_2, \ldots, i_t)$ is $(i_1, i_2, \ldots, i_t)$-canonical, and we do not get the event $Bad_1(\mathbf{e})$.

Let $\ell \in [k]$. We say that $\mathbf{a}^1_{i_p,j}$ is *active* after ℓ steps if $a^{p,j} > \ell$. I.e., if $\mathbf{a}^1_{i_p,j}$ is active after ℓ steps then no edge from $\mathbf{a}^1_{i_p,j}$ has appeared in $\{e_1, \ldots, e_\ell\}$. Similarly, $\mathbf{b}^1_{i_p}$ is *active* after ℓ steps if $b^j > \ell$. We define the set of active edges as:

$$active(\ell) \;=\; \Big(\bigcup_{p,j : a^{p,j} > \ell} \mathbf{a}^1_{i_p,j} \Big) \cup \Big(\bigcup_{p : b^p > \ell} \mathbf{b}^1_{i_p} \Big)$$

Note that if $active(k) = \emptyset$, then following the discussion above, $P_T(i_1, i_2, \ldots, i_t)$ is $(i_1, i_2, \ldots, i_t)$-canonical, and we do not get the event $Bad_1(\mathbf{e})$.

Consider now every index $\ell \in [k]$ for which we either have $e_\ell \in \mathbf{e}$ or $e_\ell \in \mathit{active}(\ell-1)$. I.e., we either pick an instance of the multi-edge under consideration, or we pick an active edge. Clearly, we can pick an instance of the multi-edge at most r times. Since every time we pick an active edge the corresponding set of edges becomes inactive, we can pick an active edge at most $(g+1)t$ times. We claim that every time we pick one of these two types of edges, we pick a multi-edge with probability at most $\frac{r}{h+r}$. Indeed, there are at most r copies of the multi-edge available to be picked. On the other hand, we know from Lemma 5.6.6 that at least one active set $\mathbf{a}^1_{i_p,j}$ or $\mathbf{b}^1_{i_p}$ is not part of $\pi(u_{\ell-1})$. Since the size of every such set is at least h the claim follows.

The event $Bad_1(e)$ occurs only if we pick r copies of $\mathbf{e}$ before $(g+1)t$ active edges. Let X be the sum of $(g+1)t+r$ Bernoulli trials, each with a 'success' probability of $p = r/(h+r)$. $Bad_1(e)$ occurs only if $X \geq r$. We now apply Chernoff. $\mu = E[X] = \frac{r}{h+r}((g+1)t+r)$. As $h = 2(g+1)t+r$, we then have $\mu = \frac{r}{2}$. Then,

$$\Pr[X \geq r] = \Pr[X \geq 2\mu] \leq \left(\frac{\mathrm{e}}{4}\right)^{\mu} = \left(\frac{\mathrm{e}}{4}\right)^{r/2} < n^{-2}.$$

It is possible that there are fewer than $(g+1)t+r$ trials, which only lowers the probability. □

Lemma 5.6.12 *For every $i \in [n]$ we have*

$$\Pr[Bad_2(i) \mid Good_3] \leq \frac{(g!)^2}{(2g)!} < 2^{-g}\ .$$

Proof: Let $P' = P'_T(i_1, i_2, \ldots, i_t) = \langle (e_1,d_1),(e_2,d_2),\ldots,(e_k,d_k)\rangle$, and let $i \in [n]$ be given. Let $u_0, \ldots, u_k$ be the nodes in T corresponding to P'.

The event $Bad_2(i)$ occurs if $\sigma_{P'}(\mathbf{a}^1_{i,j}) \leq k$, for every $j \in [g]$, and $\sigma_{P'}(\mathbf{b}^1_{i,j}) = \infty$, for every $j \in [g]$. In particular, we must have $\mathbf{b}^1_i \subseteq F(u_\ell)$ for every $0 \leq \ell < k$. It follows from Lemma 5.6.6 that if $e_\ell \in \mathbf{a}^1_i$, for $\ell \in k$, then both $\mathbf{a}^1_i \cap \pi(u_{\ell-1}) = \emptyset$ and $\mathbf{b}^1_i \cap \pi(u_{\ell-1}) = \emptyset$. Since $|\mathbf{b}^1_{i,j}| = |\mathbf{a}^1_{i,j}| = h$ for every $j \in [g]$, it follows that the w-th time, for $w \in [g]$, we pick an edge from a new set $\mathbf{a}^1_{i,j}$ without picking an edge from $\mathbf{b}^1_i$, this happens with probability $\frac{g-w+1}{2g-w+1}$. Hence,

$$\Pr[Bad_2(i) \mid Good_3] \ \leq\ \prod_{w=1}^{g} \frac{g-w+1}{2g-w+1} \ =\ \prod_{w=1}^{g} \frac{w}{w+g} \ =\ \frac{(g!)^2}{(2g)!} \ <\ 2^{-g}\ .$$

□

We now pick $g = \Theta(\log n)$ and $r = \Theta(\log n)$. To satisfy the assumption for Lemma 5.6.11, we need to have $h = 2(g+1)t + r$, i.e. $h = \Theta(\sqrt{n}\log n)$. We are now ready to prove Lemma 5.6.9.

Proof: The events $Bad_1(\mathbf{e})$ are disjoint for different multi-edges $\mathbf{e} \in M$. The number of edges that have been copied are $|M| = n(2gh + g + 3) = O(n^{3/2}\log^2 n)$. We, thus, get from Lemma 5.6.11 that:

$$\Pr\left[Bad_1 \mid Good_3\right] \;=\; \sum_{\mathbf{e}\in M} \Pr\left[Bad_1(\mathbf{e}) \mid Good_3\right] \;\leq\; |M|\, n^{-2} \leq \frac{1}{4}\,.$$

Similarly, the events $Bad_2(i)$, for $i \in [n]$, are disjoint, and we get from Lemma 5.6.12 that:

$$\Pr\left[Bad_2 \mid Good_3\right] \;=\; \sum_{i\in[n]} \Pr\left[Bad_2(i) \mid Good_3\right] \;\leq\; n\, 2^{-g} \leq \frac{1}{4}\,.$$

Using Lemma 5.6.10 it follows that:

$$\begin{aligned}
&\Pr\left[appear(i_1, i_2, \ldots, i_t)\right] \;= \\
&\Pr\left[Good_3\right]\left(1 - \Pr\left[Bad_1 \mid Good_3\right] - \Pr\left[Bad_2 \mid Good_3\right]\right) \;\geq \\
&\frac{1}{t!}\left(1 - \frac{1}{4} - \frac{1}{4}\right) \;=\; \frac{1}{2\,t!}\,.
\end{aligned}$$

□

Theorem 5.6.13 *The expected number of improving switches performed by the* RandomFacet *algorithm, when run on the game G_n with initial policy σ_0, is at least $\mathrm{e}^{\Omega(\sqrt{n})}$.*

Proof: From Lemma 5.2.2 and Lemma 5.6.9 we get that:

$$\begin{aligned}
f(E_0, \pi_0) \;&=\; \sum_{P\in paths(G)} \Pr\left[appear(P)\right] \\
&\geq\; \sum_{s\in S_n} \sum_{P\in canon_G(s)} \Pr\left[appear(P)\right] \\
&\geq\; \sum_{(i_1,i_2,\ldots,i_t)} \Pr\left[appear(i_1, i_2, \ldots, i_t)\right] \\
&\geq\; \frac{1}{2}\frac{1}{t!}\binom{n}{t}\,,
\end{aligned}$$

where $t = \sqrt{n}$. The claim follows because $\frac{1}{2}\frac{1}{t!}\binom{n}{t} = \mathrm{e}^{(2-o(1))\sqrt{n}}$, as

$$\frac{1}{t!}\binom{n}{t} = \frac{n!}{(t!)^2(n-t)!} \geq \frac{(n-\sqrt{n})^{\sqrt{n}}}{((\sqrt{n})!)^2} = \frac{\left(n\left(1-\frac{1}{\sqrt{n}}\right)\right)^{\sqrt{n}}}{((\sqrt{n})!)^2}$$
$$\sim \frac{\frac{n^{\sqrt{n}}}{\mathrm{e}}}{2\pi\sqrt{n}\left(\frac{n}{\mathrm{e}^2}\right)^{\sqrt{n}}} = \frac{\mathrm{e}^{2\sqrt{n}}}{2\pi\mathrm{e}\sqrt{n}}\,.$$

□

Since the MDP $G_{n,g,h,r}$ contains $N = \Theta(ngh) = \Theta(n^{3/2}\log^2 n)$ vertices and $M = \Theta(nghr) = \Theta(n^{3/2}\log^3 n)$ edges, as $g = \Theta(\log n)$, $h = \Theta(\sqrt{n}\log n)$ and $r = \Theta(\log n)$, we can derive the following corollary:

Corollary 5.6.14 *There are MDPs with N states and $O(N \log N)$ actions for which the expected number of switches performed by the* RandomFacet *algorithm is* $\mathrm{e}^{\Omega(N^{1/3}/\log^{2/3} N)}$.

Since $G_{n,g,h,r}$ satisfies the stopping condition we get the following corollary for linear programming.

Corollary 5.6.15 *There are linear programs in standard form with N equations and $O(N \log N)$ variables for which the expected number of improving pivots performed by the* RandomFacet *algorithm is* $\mathrm{e}^{\Omega(N^{1/3}/\log^{2/3} N)}$.

5.7 The modified RandomFacet algorithm

We next consider a modified variant of the RandomFacet algorithm, which we will refer to as the RandomFacet* algorithm. The RandomFacet* algorithm is shown in Figure 5.6. This variant receives, as a third argument, a *permutation function* $\sigma : F \to \mathbb{N}$ that assigns to each edge of F a *distinct* natural number. σ may be viewed as the inverse of a permutation of F, except that when we remove edges from F there is no need to update the permutation. Instead of choosing a *random* edge e from $F \setminus \sigma$, the algorithm now chooses the edge e of $F \setminus \sigma$ with the *smallest permutation index* $\sigma(e)$. Let $\Sigma(G)$ be the set of all permutation functions with respect to G with range $\{1, 2, \ldots, |E_0|\}$. Unless explicitly stated otherwise, we initialize the RandomFacet* algorithm with a uniformly random permutation function from $\Sigma(G)$.

Algorithm 8: RandomFacet*(F, π, σ)

```
if F = π then
    return π;
else
    e ← argmin_{e∈F\π} σ(e) ;
    π' ← RandomFacet*(F \ {e}, π, σ) ;
    if e is an improving switch with respect to π' then
        π'' ← π'[e] ;
        return RandomFacet*(F, π'', σ);
    else
        return π';
```

Figure 5.6: Variant of the RandomFacet algorithm

It is natural to ask what the expected running time of the modified RandomFacet algorithm is when it is given a uniformly random permutation function from $\Sigma(G)$ as input. Note that the RandomFacet algorithm may be viewed as the RandomFacet* algorithm, except that it is given a new uniformly random permutation function every time it is called. The RandomFacet* algorithm is, thus, related to the RandomFacet algorithm, but the expected running times of the two algorithms are not the same. To see this consider the following simple example. Let $F = \{e_1, e_2, e_3\}$, and let π be a policy such that $e_1 \in \pi$ and $e_2, e_3 \notin \pi$. Let $\sigma : F \to \{1, 2, 3\}$ be a uniformly random permutation function. Then the RandomFacet* algorithm picks e_2 and e_3 with equal probability as the first edge. However, in the recursive call, e_1 now has probability $2/3$ of having the lowest index, meaning that the resulting permutation function is not uniformly random.

In the paper Friedmann, Hansen, and Zwick from SODA 2011 [41] we proved a lower bound of subexponential form for the RandomFacet* algorithm for parity games. We then *incorrectly* claimed that the expected running time of the RandomFacet* algorithm was the same as the expected running time of the RandomFacet algorithm. In this section we present the lower bound for the RandomFacet* algorithm from [41]. It uses exactly the same construction as the lower bound for the RandomFacet algorithm, but the analysis is different. The main idea is again to show that, with high probability, a run of the RandomFacet* algorithm can be interpreted as a run of the randomized counter RandCount. More precisely,

Algorithm 9: RANDCOUNT*(N, σ)

```
if N ≠ ∅ then
    i ← argmin_{i∈N} σ(i);
    RandCount*(N \ {i}, σ) ;
    for j ∈ [i − 1] do 𝔟(j) ← 0 ;
    𝔟(i) ← 1 ;
    RandCount*(N ∩ [i − 1], σ) ;
```

Figure 5.7: A variant of the randomized counter.

we use a variant of RANDCOUNT that takes a fixed permutation function $\sigma : [n] \to [n]$ of the bits as input, and uses σ to decide which bits to increment. We let $\Sigma([n])$ be the set of permutation functions for $[n]$. Note that $|\Sigma([n])| = n!$. The modified randomized counter, RANDCOUNT*, is shown in Figure 5.7. The following lemma shows that the expected number of counting steps of RANDCOUNT and RANDCOUNT* is the same when RANDCOUNT* is given a uniformly random permutation function. Recall that $g(n)$ is the expected number of times RANDCOUNT$([n])$ sets a bit to 1. We let $g^*([n], \sigma)$ be the number of times RANDCOUNT*$([n], \sigma)$ sets a bit to 1, and $g^*(n)$ be the expected number of times RANDCOUNT*$([n], \sigma)$ sets a bit to 1 when σ is uniformly random.

Lemma 5.7.1 $g(n) = g^*(n)$.

Proof: The lemma is proved by using induction and linearity of expectation. Clearly $g(0) = g^*(0) = 0$. We see that:

$$\begin{aligned}
g^*(n) &= \frac{1}{n!} \sum_{\sigma \in \Sigma([n])} g^*([n], \sigma) \\
&= \frac{1}{n} \sum_{i \in [n]} \frac{1}{(n-1)!} \sum_{\sigma \in \Sigma([n] \setminus \{i\})} g^*([n] \setminus \{i\}, \sigma) + 1 + g^*([i-1], \sigma) \\
&= \frac{1}{n} \sum_{i \in [n]} g^*(n-1) + 1 + g^*(i-1) \\
&= g(n-1) + 1 + \frac{1}{n} \sum_{i \in [n]} g(i-1) \;=\; g(n) \,.
\end{aligned}$$

□

Let $G_{n,g,h,r} = (V_0, V_R, E, c, p)$ be defined as in Section 5.5. We start by defining some properties of permutation functions that will allow us to get a good lower bound. Let $\sigma : E_0 \to \{1, \ldots, |E_0|\}$ be a permutation function. Recall that M is the set of multi-edges, with $\mathbf{e} \in M$ being a set $\mathbf{e} = \{e^1, e^2, \ldots, e^r\}$ of r identical edges. Define:

$$\begin{aligned}
\forall i \in [n], j \in [g]: \quad \sigma(\mathbf{b}^1_{i,j}) &= \min_{k \in [h]} \sigma(b^1_{i,j,k}) \\
\forall i \in [n]: \quad \sigma(\mathbf{b}^1_i) &= \min_{j \in [g]} \sigma(\mathbf{b}^1_{i,j}) \\
\forall i \in [n], j \in [g]: \quad \sigma(\mathbf{a}^1_{i,j}) &= \min_{k \in [h]} \sigma(a^1_{i,j,k}) \\
\forall i \in [n]: \quad \sigma(\mathbf{a}^1_i) &= \max_{j \in [g]} \sigma(\mathbf{a}^1_{i,j}) \\
\forall \mathbf{e} \in M: \quad \sigma(\mathbf{e}) &= \max_{j \in [r]} \sigma(e^j)
\end{aligned}$$

Definition 5.7.2 (Good permutation) *Let $\sigma \in \Sigma(G_{n,g,h,r})$. Then σ is said to be* good *if and only if:*

(i) $\sigma(\mathbf{a}^1_{i,j}) < \sigma(\mathbf{e})$ *for all* $\mathbf{e} \in M$, $i \in [n]$, *and* $j \in [g]$.

(ii) $\sigma(\mathbf{b}^1_i) < \sigma(\mathbf{a}^1_i)$ *for all* $i \in [n]$.

The first requirement of Definition 5.7.2 ensures that at least one copy of every multi-edge $\mathbf{e} \in M$ is available before $\mathbf{a}^1_i$ (or $\mathbf{b}^1_i$) is disabled. Recall that for the analysis of the RandomFacet algorithm we wanted $\mathbf{b}^1_i$ to be disabled before $\mathbf{a}^1_i$. This property will be ensured by the second requirement of Definition 5.7.2.

Lemma 5.7.3 *Let $\sigma \in \Sigma(G_{n,g,h,r})$ be chosen uniformly at random. Then σ is good with probability $p_{n,g,h,r}$, where:*

$$p_{n,g,h,r} \geq 1 - n^2 g(2gh + g + 3)\frac{h!\, r!}{(h+r)!} - n\frac{(g!)^2}{(2g)!}\ .$$

Proof: Let $\mathbf{e} \in M$, $i \in [n]$, and $j \in [g]$ be given. Recall that $\mathbf{e}$ contains r copies of a multi-edge. Similarly, $\mathbf{a}^1_{i,j}$ contains h edges. Let us consider the event that σ is not good because $\sigma(\mathbf{a}^1_{i,j}) > \sigma(\mathbf{e})$. This means that all the edges of $\mathbf{a}^1_{i,j}$ have higher indices than all the edges of $\mathbf{e}$. Since σ is uniformly random this happens with probability:

$$\Pr\left[\sigma(\mathbf{a}^1_{i,j}) > \sigma(\mathbf{e})\right] = \frac{h!\, r!}{(h+r)!}$$

Since $|M| = n(2gh + g + 3)$, it follows that σ fails to satisfy Definition 5.7.2 (i) with probability at most $n^2 g(2gh + g + 3)\frac{h!\,r!}{(h+r)!}$.

We next bound the probability that σ fails to satisfy Definition 5.7.2 (ii). Let $i \in [n]$ be given. Since $|\mathbf{b}^1_{i,j}| = |\mathbf{a}^1_{i,j}| = h$ for all $j \in [g]$ the situation is the same as above. I.e., we may view σ as defining a uniformly random permutation of $S = \{\mathbf{b}^1_{i,1}, \ldots, \mathbf{b}^1_{i,g}, \mathbf{a}^1_{i,1}, \ldots, \mathbf{a}^1_{i,g}\}$. It follows that:

$$\Pr\left[\sigma(\mathbf{b}^1_i) > \sigma(\mathbf{a}^1_i))\right] \;=\; \frac{(g!)^2}{(2g)!}$$

Hence, the probability that σ fails to satisfy Definition 5.7.2 (ii) is at most $n\frac{(g!)^2}{(2g)!}$. The statement of the lemma follows. □

Note that $\frac{(g!)^2}{(2g)!} \leq 2^{-g}$. We will use $g = h = r = 3\log n$, such that $p_{n,g,h,r} \geq \frac{1}{2}$.

Definition 5.7.4 (Induced permutation function) *Let $\sigma \in \Sigma(G_{n,g,h,r})$ be a permutation function, then the* induced permutation function $\hat{\sigma} : [n] \to [n]$ *is defined as the unique permutation function that satisfies $\hat{\sigma}(i) < \hat{\sigma}(j)$ if and only if $\sigma(\mathbf{b}^1_i) < \sigma(\mathbf{b}^1_j)$, for all $i, j \in [n]$.*

Definition 5.7.5 (Active bits) *Let $F \subseteq E_0$ be a subset of edges, and let $p \in [n+1]$. We define the set of* active bits below p for F *as:*

$$\mathit{active}(F, p) = \{i \in [p-1] \mid \mathbf{b}^1_i \subseteq F\}$$

We let $f^*(F, \pi, \sigma)$ be the number of improving switches performed by RandomFacet$^*(F, \pi, \sigma)$. We also let $f^*(F, \pi)$ be $f^*(F, \pi, \sigma)$ when σ is picked uniformly at random. Recall that the terms p-stable and p-unstable were defined in definitions 5.6.4 and 5.6.5, respectively.

Lemma 5.7.6 *Let $\sigma \in \Sigma(G_{n,g,h,r})$ be a good permutation function, let $F \subseteq E_0$ be functional, let $\pi \subseteq F$ be a policy, let $p \in [n+1]$, and assume that $\mathit{reset}(F) = 0$. Assume furthermore that either π is p-stable w.r.t. F, or π is p-unstable (and $p \leq n$) w.r.t. F and $\mathbf{b}^1_p \subseteq F$. Let $N = \mathit{active}(F, p)$. Then $f^*(F, \pi, \sigma) \geq g^*(N, \hat{\sigma})$.*

Proof: The proof of the lemma is by induction in the size of N. If $N = \emptyset$, then the number of counting steps of RandCount$^*(N, \hat{\sigma})$ is zero, and the statement is clearly true.

Let $e = \operatorname{argmin}_{e \in F} \sigma(e)$, let $i = \operatorname{argmin}_{i \in N} \hat{\sigma}(i)$, and let $F' = F \setminus \{e\}$. Since σ is a good permutation we know that for $N \neq \emptyset$, F' is also functional. For the induction step we consider three cases:

(*i*) $e \in \mathbf{b}_i^1$.

(*ii*) $e \in \mathbf{a}_p^1$ and $\mathbf{a}_p^1 \not\sqsubseteq F'$.

(*iii*) $e \notin \mathbf{b}_i^1$, and either $e \notin \mathbf{a}_p^1$ or $\mathbf{a}_p^1 \sqsubseteq F'$.

Case (*i*)**:** Since $e \in F \setminus \pi$ it must be the case that π is p-stable, since otherwise $\mathbf{b}_\ell^1 \cap F \subseteq \pi$ for all $\ell \in [n]$. Observe that $reset(F') = 0$, and that π is p-stable w.r.t. F'. Furthermore, $active(F', p) = N \setminus \{i\}$. It follows by the induction hypothesis that for the first recursive call we have $f^*(F \setminus \{e\}, \pi, \sigma) \geq g^*(N \setminus \{i\}, \hat{\sigma})$. Let $\pi' \in \Pi^*_{F\setminus\{e\}}$ be the policy returned by RandomFacet$^*(F \setminus \{e\}, \pi, \sigma)$. It follows from Definition 5.5.3, Corollary 5.5.6, and Lemma 5.5.7 that e is an improving switch with respect to π', and that $\pi''[e]$ is i-unstable w.r.t. F. Note also that $\mathbf{b}_i^1 \subseteq F$. Hence, we get from the induction hypothesis that for the second recursive call we have $f^*(F, \pi'', \sigma) \geq g^*(N \cap [i-1], \hat{\sigma})$. The induction step follows.

Case (*ii*)**:** Since $e \in \mathbf{a}_p^1$ it must be the case that π is p-unstable w.r.t. F. Since $\mathbf{b}_p^1 \subseteq F$ and $reset(F) = 0$, it follows that $reset(F') = p$. Let $\pi' \in \Pi^*_{F\setminus\{e\}}$ be the policy returned by RandomFacet$^*(F \setminus \{e\}, \pi, \sigma)$. It follows from Definition 5.5.3, Corollary 5.5.6, and Lemma 5.5.7 that e is an improving switch with respect to π', and that $\pi''[e]$ is p-stable w.r.t. F. Hence, we get for the second recursive call that $f^*(F, \pi'', \sigma) \geq g^*(N, \hat{\sigma})$, and the induction step follows.

Case (*iii*)**:** Since $e \notin \mathbf{b}_i^1$ where $i = \operatorname{argmin}_{i \in N} \hat{\sigma}(i)$, we have $e \notin \mathbf{b}_\ell^1$ for any $\ell \in N$. It follows that $active(F', p) = N$. Furthermore, since σ is a good permutation we also have $\mathbf{a}_\ell^1 \sqsubseteq F'$ for any $\ell \in N$. Also, since π is either p-stable or p-unstable w.r.t. F we do not have $e \in \mathbf{a}_\ell^1$ for any $\ell > p$ with $\mathbf{b}_\ell^1 \subseteq F$. We also do not have $e \in \mathbf{b}_\ell^1$ for any $\ell \geq p$. In particular, $\mathbf{b}_p^1 \subseteq F'$ if $\mathbf{b}_p^1 \subseteq F$. Finally we know that $\mathbf{a}_p^1 \not\sqsubseteq F'$. It follows that $reset(F') = 0$. Since removing e from F does not change whether π is p-stable or p-unstable, we get that we can apply the induction hypothesis to the first recursive call: $f^*(F \setminus \{e\}, \pi, \sigma) \geq g^*(N, \hat{\sigma})$. The induction step follows.

This completes the proof. □

Theorem 5.7.7 *The expected number of improving switches performed by the* RandomFacet* *algorithm for $G = G_{n,g,h,r}$, with $g = h = r = 3\log n$, starting with π_0 and a uniformly random permutation function is* $\mathrm{e}^{\Omega(\sqrt{n})}$.

Proof: If $\sigma \in \Sigma(G)$ is picked uniformly at random then we know from Lemma 5.7.3 that σ is good with probability at least 1/2. Moreover, the

induced permutation function $\hat{\sigma}$ is also uniformly random since every set $\mathbf{b}_i^1$ has the same size. Since π_0 is $(n{+}1)$-stable it follows from Lemma 5.7.6 that $f^*(E_0, \pi_0) \geq g^*(n)$. From Lemma 5.7.1 we know that $g^*(n) = g(n)$, and from Lemma 5.3.2 we know that $g(n) = \mathrm{e}^{\Omega(\sqrt{n})}$. □

Since the MDP $G_{n,g,h,r}$ contains $N = \Theta(n \log^2 n)$ vertices and $M = \Theta(n \log^3 n)$ edges, we can derive the following corollary:

Corollary 5.7.8 *There are MDPs with N states and $O(N \log N)$ actions for which the expected number of switches performed by the* RandomFacet* *algorithm is $\mathrm{e}^{\Omega(\sqrt{N}/\log N)}$.*

Since $G_{n,g,h,r}$ satisfies the stopping condition we get the following corollary for linear programming.

Corollary 5.7.9 *There are linear programs in standard form with N equations and $O(N \log N)$ variables for which the expected number of improving pivots performed by the* RandomFacet* *algorithm is $\mathrm{e}^{\Omega(\sqrt{N}/\log N)}$.*

5.8 The Randomized Bland's rule algorithm

Bland's rule for the simplex algorithm works by repeatedly performing improving pivots with the available non-basic variable that has the smallest index. A corresponding PolicyIteration algorithm can be defined for MDPs: in each step the improving switch with the smallest index is performed. I.e., the algorithm takes as input a permutation (function) $\sigma : E_0 \to \{1, \ldots, |E_0|\}$ of the edges, and the first improving switch according to this permutation is chosen. We again let $\Sigma(G)$ be the set of all permutation functions with range $\{1, \ldots, |E_0|\}$. We refer to the resulting PolicyIteration algorithm as the Bland's rule algorithm. The Bland's rule algorithm is shown in Figure 5.8. For consistency, we perform the improving switch with the largest index instead of the improving switch with the smallest index. When the permutation of edges is chosen uniformly at random we refer to the Bland's rule algorithm as the Randomized Bland's rule algorithm. The Randomized Bland's rule algorithm is also referred to as the "one-permutation" variant of the RandomFacet algorithm by Matoušek [86]. In this section we prove a lower bound for the expected number of iterations performed by the Randomized Bland's rule algorithm.

The Bland's rule algorithm is, in fact, very similar to the modified RandomFacet algorithm. To see this it is helpful to consider a recursive formulation of the Bland's rule algorithm, as shown in Figure 5.8.

Algorithm 10: BLAND'S RULE(E_0, π, σ)

```
k ← |E_0|;
while k ≥ 1 do
    if σ^{-1}(k) is an improving switch w.r.t. π then
        π ← π[σ^{-1}(k)];
        k ← |E_0|;
    else
        k ← k − 1;
return π;
```

Figure 5.8: The BLAND'S RULE algorithm.

Algorithm 11: BLAND'S RULE$^*(F, \pi, \sigma)$

```
if F = ∅ then
    return π;
else
    e ← argmin_{e∈F} σ(e);
    π' ← BLAND'S RULE*(F \ {e}, π, σ) ;
    if e is an improving switch w.r.t. π' then
        π'' ← π'[e] ;
        return BLAND'S RULE*(F, π'', σ);
    else
        return π';
```

Figure 5.9: A recursive formulation of the BLAND'S RULE algorithm.

Note that the only difference between the RandomFacet* algorithm and the Bland's rule* algorithm is that the Bland's rule* algorithm can pick e from all of F and not only from $F \setminus \pi$. As a consequence we do not get the invariant that $\pi \subseteq F$ for the Bland's rule* algorithm. The set of edges currently considered by the algorithm is, thus, $F \cup \pi$. The algorithm may then drop edges from π when performing improving switches. The following lemma describes the guarantee we get about the policy returned by Bland's rule*(F, π, σ). It follows from the lemma that Bland's rule*(E_0, π, σ) returns an optimal policy.

Before stating the lemma we introduce some notation.

Definition 5.8.1 (Fixed edges) *Let $F \subseteq E_0$ and let π be a policy. We say that an edge $e \in \pi$ is* fixed *for π with respect to F if $\mathrm{val}_\pi(v) \geq \mathrm{val}_{\pi'}(v)$ for every policy $\pi' \subseteq F \cup \pi \setminus \{e\}$ and every vertex $v \in V$. We let $fixed(\pi, F) \subseteq \pi$ be the set of fixed edges for π with respect to F.*

Note that if $fixed(\pi, F) = \pi$ then π is optimal for $G_{F \cup \pi}$.

Lemma 5.8.2 *Let $F \subseteq E_0$ be a set of edges, let $\pi \subseteq E_0$ be a policy, and let $\sigma \in \Sigma(G)$ be a permutation function, and let $\pi^* =$ Bland's rule*(F, π, σ). Then there are no improving switches in F with respect to π^*. I.e., π^* is optimal for $G_{F \cup \pi^*}$.*

Proof: We prove the lemma by induction. If either $F = \emptyset$ or $fixed(\pi, F) = \pi$ then clearly the lemma is true. Let F and π be given. For the induction hypothesis we assume that the lemma is true for F' and π' with $|F'| < |F|$, or with $|F'| = |F|$ and $|fixed(\pi', F')| > |fixed(\pi, F)|$.

Let $e = \operatorname{argmin}_{e \in F} \sigma(e)$, and $\pi' =$ Bland's rule*$(F \setminus \{e\}, \pi, \sigma)$. By the induction hypothesis there are no improving switches in $F \setminus \{e\}$ w.r.t. π', and if e is not an improving switch w.r.t. π' we are done.

Assume that e is an improving switch w.r.t. π', let $\pi'' = \pi'[e]$, and let $\pi^* =$ Bland's rule*(F, π'', σ). Since π' is optimal for $G_{F \cup \pi'}$, and e is an improving switch w.r.t. π', e is fixed for π'' with respect to F. Observe also that e was not fixed for π with respect to F, since at least one vertex has higher value for π'' than for π. It follows that $|fixed(\pi'', F)| > |fixed(\pi, F)|$, and we get from the induction hypothesis that there are no improving switches in F with respect to π^*. □

The following lemma shows that the Bland's rule algorithm and the Bland's rule* algorithm are, indeed, equivalent.

Lemma 5.8.3 *Let $\pi_0 \subseteq E_0$ be a policy and $\sigma \in \Sigma(G)$ be a permutation function. Then* Bland's rule(E_0, π_0, σ) *and* Bland's rule$^*(E_0, \pi_0, \sigma)$ *perform exactly the same sequence of improving switches.*

Proof: Observe that the Bland's rule algorithm repeatedly performs the improving switch e with the largest index $\sigma(e)$. Define $F_k = \{e \in E_0 \mid \sigma(e) > k\}$. In particular, if π is the current policy and e is an improving switch w.r.t. π, then the Bland's rule algorithm performs the switch e if and only if no edge $e' \in F_{\sigma(e)}$ is an improving switch w.r.t. π.

Let $e^1, \ldots, e^N$ be the sequence of improving switches performed by the call Bland's rule$^*(E_0, \pi_0, \sigma)$, and let $\pi^0, \pi^1, \ldots, \pi^N$ be the corresponding sequence of policies. I.e., $\pi^\ell = \pi^{\ell-1}[e^\ell]$ for $\ell \in [N]$. We must, thus, show for all $\ell \in [N]$ that no edge $e \in F_{\sigma(e^\ell)}$ is an improving switch w.r.t. $\pi^{\ell-1}$.

This follows from the fact that the Bland's rule* algorithm always removes the edge e with the smallest index $\sigma(e)$. In particular, we have $\pi^{\ell-1} =$ Bland's rule$^*(F_{\sigma(e^\ell)}, \pi^k, \sigma)$, for some $k \leq \ell - 1$. It follows from Lemma 5.8.2 that no edge $e \in F_{\sigma(e^\ell)}$ is an improving switch w.r.t. $\pi^{\ell-1}$. This completes the proof. □

5.8.1 Lower bound

To prove a lower bound for the Randomized Bland's rule algorithm we use the same construction $G_{n,g,h,r}$ as we did for the RandomFacet algorithm and for the RandomFacet* algorithm. For the analysis we consider the recursive Bland's rule* algorithm. As in Section 5.7 for the RandomFacet* algorithm, the main idea is to show that, with high probability, a run of the Randomized Bland's rule algorithm can be interpreted as a run of the randomized counter RandCount. More precisely, we use the variant of RandCount that takes a fixed permutation (function) of the bits as input, and uses this permutation (function) to decide which bits to increment. This modified randomized counter, RandCount*, was shown in Figure 5.7.

Before analyzing Bland's rule*, it will be helpful to introduce an additional definition.

Definition 5.8.4 (Fixed levels) *Let $F \subseteq E_0$, let π be a policy, and let $p \in [n]$. We say that π is* fixed from a_p w.r.t. F *if every edge $e \in \pi \cap (\mathbf{a}_i^0 \cup \mathbf{a}_i^1 \cup \mathbf{b}_i^0 \cup \mathbf{b}_i^1)$, for $i \geq p$, is fixed. Similarly, we say that π is* fixed from b_p w.r.t. F *if π is fixed from a_{p+1} w.r.t. F, and every edge $e \in \pi \cap (\mathbf{b}_p^0 \cup \mathbf{b}_p^1)$, is fixed.*

The following lemma follows immediately from Definition 5.8.4.

Lemma 5.8.5 *If π is fixed from a_p w.r.t. $F \subseteq E_0$, and π' is obtained from π be repeatedly performing improving switches, then π' is also fixed from a_p w.r.t. F. Moreover, $\mathbf{b}_i^1 \cap \pi = \mathbf{b}_i^1 \cap \pi'$ and $\mathbf{a}_i^1 \cap \pi = \mathbf{a}_i^1 \cap \pi'$, for $i \geq p$. The corresponding statement holds if π is fixed from b_p w.r.t. F.*

Lemma 5.8.6 *Let $F \subseteq E_0$ be functional, and let $p \in [n]$. Assume that $reset(F) < p$, $\mathbf{b}_p^1 \subseteq F$, and $\mathbf{a}_p^1 \sqsubseteq F$. Furthermore, let $e \in F$, $\pi \in \Pi^*_{F\setminus\{e\}}$, and $\pi' = \pi[e]$. If $e \in \mathbf{b}_p^1$, then π' is p-unstable and fixed from b_p w.r.t. F. Similarly, if $e \in \mathbf{a}_p^1$ and $\mathbf{a}_p^1 \not\sqsubseteq F \setminus \{e\}$, then π' is p-stable and fixed from a_p w.r.t. F.*

Proof: We first make some general observations. Let $e \in \mathbf{b}_p^1 \cup \mathbf{a}_p^1$, and $\pi \in \Pi^*_{F\setminus\{e\}}$. Since there is no way to reach the i-th level of the construction from the $(i+1)$-th level, it follows that adding or removing the edge e in the p-th level does not affect which choices are optimal at the higher levels. Since every edge is fixed for π w.r.t. $F \setminus \{e\}$, it follows that edges $e' \in \pi \cap (\mathbf{a}_i^0 \cup \mathbf{a}_i^1 \cup \mathbf{b}_i^0 \cup \mathbf{b}_i^1)$, for $i > p$, are fixed for π' w.r.t. F. By an analogous argument it follows that if $e \in \mathbf{a}_p^1$, then every edge $e' \in \pi \cap \mathbf{b}_p^1$ is fixed for π' w.r.t. F.

Consider first the case when $e \in \mathbf{b}_p^1$. It follows from Definition 5.5.3 and Corollary 5.5.6 that π' is p-unstable w.r.t. F. Moreover, using e is optimal for G_F, and the choices of vertices reachable from e in π are optimal for G_F. Hence, we also get that every edge $e' \in \pi' \cap \mathbf{b}_p^1$ is fixed for π' w.r.t. F. For edges $e' \in \pi' \cap \mathbf{b}_p^0$ there is no choice for the corresponding vertices.

The case when $e \in \mathbf{a}_p^1$ and $\mathbf{a}_p^1 \not\sqsubseteq F \setminus \{e\}$ is similar. It follows from Definition 5.5.3 and Corollary 5.5.6 that π' is p-stable w.r.t. F, and by the same argument as above we also get that every edge $e' \in \pi' \cap \mathbf{a}_p^1$ is fixed for π' w.r.t. F. For edges $e' \in \pi' \cap \mathbf{a}_p^0$ there is no choice for the corresponding vertices. □

Let $\sigma \in \Sigma(G_{n,g,h,r})$ be a permutation function. We use the same definitions of $\sigma(\mathbf{b}_i^1)$ and $\sigma(\mathbf{a}_i^1)$ as in Section 5.7. For every permutation function $\sigma \in \Sigma(G_{n,g,h,r})$, and every integer $k \in \{1, \ldots, |E_0| + 1\}$, define:

$$F(\sigma, k) \;=\; \{e \in E_0 \mid \sigma(e) \geq k\}$$

The following lemma follows immediately from Definitions 5.5.1 and 5.7.2.

Lemma 5.8.7 *If σ is a good permutation function, then $F(\pi, k+1)$ is functional for $k \in \{\sigma(\mathbf{b}_i^1), \sigma(\mathbf{a}_i^1) \mid i \in [n]\}$.*

Lemma 5.8.8 *Let $\sigma \in \Sigma(G)$ be a good permutation function, let $k \in \{1, \ldots, |E_0|+1\}$, and let π be p-unstable and fixed from b_p w.r.t. $F(\sigma, k) \cup \pi$. Assume that $\mathbf{b}_p^1 \subseteq \pi$, that $\mathbf{a}_p^1 \sqsubseteq F(\sigma, k)$, and that $reset(F(\sigma, k) \cup \pi) \leq p$. Furthermore, let $\pi^* =$ BLAND'S RULE$^*(F(\sigma, k), \pi, \sigma)$. Then $\pi^* \cap \mathbf{b}_i^1 \subseteq F(\sigma, k)$ for all $i < p$.*

Proof: Let $k' = \sigma(\mathbf{a}_p^1)$. From the assumption of the lemma we know that $k' \geq k$. By repeatedly entering the first branch of the recursion we get to a recursive call of the form BLAND'S RULE$^*(F(\sigma, k'), \pi, \sigma)$.

The policy returned from the next first recursive call is then $\pi' =$ BLAND'S RULE$^*(F(\sigma, k'-1), \pi, \sigma) \in \Sigma^*_{F(\sigma,k'-1) \cup \pi'}$. I.e., we know from Lemma 5.8.7 that $F(\sigma, k'-1)$ is functional. Since π is p-unstable and fixed from b_p, we get from Lemma 5.8.5 that $\mathbf{b}_i^1 \cap \pi' = \mathbf{b}_i^1 \cap \pi$ for all $i \geq p$. Since $reset(F(\sigma, k) \cup \pi) \leq p$ it follows that $reset(F(\sigma, k'-1) \cup \pi') = p$. It follows from Definition 5.5.3 that $\mathbf{b}_i^1 \cap \pi' = \emptyset$ for all $i < p$. Since $\pi^* \subseteq F(\sigma, k) \cup \pi'$ the statement of the lemma follows. □

We let $h^*(F, \pi, \sigma)$ be the number of improving switches performed by BLAND'S RULE$^*(F, \pi, \sigma)$. We also let $h^*(F, \pi)$ be $h^*(F, \pi, \sigma)$ when σ is picked uniformly at random. Recall that the terms p-stable and p-unstable were defined in definitions 5.6.4 and 5.6.5, respectively. The induced permutation function $\hat{\sigma}$ was defined in Definition 5.7.4, and the set of active bits, $active(F, p)$, was defined in Definition 5.7.5. The following lemma corresponds to Lemma 5.7.6 for the RANDOMFACET* algorithm. The proof of the lemma is also similar to the proof of Lemma 5.7.6.

Lemma 5.8.9 *Let $\sigma \in \Sigma(G_{n,g,h,r})$ be a good permutation function, let π be a policy, let $k \in \{1, \ldots, |E_0| + 1\}$, let $F = F(\sigma, k) \cup \pi$, and let $p \in [n+1]$. Assume that $reset(F) = 0$, and that either π is p-stable and fixed from a_p w.r.t. F; or π is p-unstable (and $p \leq n$) and fixed from b_p w.r.t. F, $\mathbf{b}_p^1 \subseteq F$, and $\mathbf{a}_p^1 \sqsubseteq F(\sigma, k)$. Let $N = active(F(\sigma, k), p)$. Then $h^*(F, \pi, \sigma) \geq g^*(N, \hat{\sigma})$.*

Proof: The proof of the lemma is by induction in the size of N. If $N = \emptyset$, then the number of counting steps of RANDCOUNT$^*(N, \hat{\sigma})$ is zero, and the statement is clearly true.

Let $e = \text{argmin}_{\,e \in F(\sigma,k)}\, \sigma(e) = \sigma^{-1}(k)$, let $i = \text{argmin}_{\,i \in N}\, \hat{\sigma}(i)$, and let $F' = (F(\sigma, k) \setminus \{e\}) \cup \pi = F(\sigma, k+1) \cup \pi$. Since σ is a good permutation we know from Lemma 5.8.7 that for $N \neq \emptyset$, $F(\sigma, k+1)$ is functional. For the induction step we consider four cases:

(*i*) $e \in \mathbf{b}_i^1$ and $e \notin \pi$.

(*ii*) $e \in \mathbf{b}_i^1$ and $e \in \pi$.

(*iii*) $k = \sigma(\mathbf{a}_p^1)$ and π is p-unstable w.r.t. F.

(*iv*) $e \notin \mathbf{b}_i^1$, and either $k \neq \sigma(\mathbf{a}_p^1)$ or π is p-stable w.r.t. F.

Case (*i*)**:** Since $e \notin \pi$ it must be the case that π is p-stable w.r.t. F, since otherwise $\mathbf{b}_\ell^1 \cap F(\sigma, k) \subseteq \pi$ for all $\ell \in [n]$. Observe that $reset(F') = 0$, and that π is p-stable and fixed from a_p w.r.t. F'. Furthermore, $active(F(\sigma, k+1), p) = N \setminus \{i\}$. It follows by the induction hypothesis that for the first recursive call we have $h^*(F(\sigma, k+1), \pi, \sigma) \geq g^*(N \setminus \{i\}, \hat{\sigma})$.

Let $\pi' = \text{BLAND'S RULE}^*(F(\sigma, k+1), \pi, \sigma)$. Then by Lemma 5.8.2 we have $\pi' \in \Pi^*_{F(\sigma,k+1)\cup\pi'}$. Since π is fixed from a_p w.r.t. F, and therefore also $F' \subseteq F$, and since $reset(F) = 0$, we get from Lemma 5.8.5 that $reset(F(\sigma, k+1)\cup\pi') < p$. We also have $F(\sigma, k+1)\cup\pi' \subseteq F(\sigma, k+1)\cup\pi = F'$, and since $\mathbf{b}_\ell^1 \cap \pi = \mathbf{a}_\ell^1 \cap \pi = \emptyset$ for all $\ell < p$, it follows that $reset(F(\sigma, k+1) \cup \pi') = 0$. We then get from Lemma 5.8.6 that $\pi'' = \pi'[e]$ is i-unstable and fixed from b_i w.r.t. $F(\sigma, k) \cup \pi'$, and, hence, also w.r.t. $F(\sigma, k) \cup \pi'' \subseteq F(\sigma, k)\cup\pi'$. Furthermore, we have $reset(F(\sigma, k)\cup\pi'') = 0$, $\mathbf{b}_i^1 \subseteq F(\sigma, k)\cup\pi''$, and $\mathbf{a}_i^1 \sqsubseteq F(\sigma, k)$. We get from the induction hypothesis that for the second recursive call we have $h^*(F(\sigma, k), \pi'', \sigma) \geq g^*(N \cap [i-1], \hat{\sigma})$. The induction step follows.

Case (*ii*)**:** Since $e \in \pi$ it must be the case that π is p-unstable w.r.t. F. Observe that $reset(F') = 0$, and that π is p-unstable and fixed from b_p w.r.t. F'. Furthermore, $active(F(\sigma, k+1), p) = N \setminus \{i\}$. It follows by the induction hypothesis that for the first recursive call we have $h^*(F(\sigma, k+1), \pi, \sigma) \geq g^*(N \setminus \{i\}, \hat{\sigma})$.

Let $\pi' = \text{BLAND'S RULE}^*(F(\sigma, k+1), \pi, \sigma)$. Then by Lemma 5.8.2 we have $\pi' \in \Pi^*_{F(\sigma,k+1)\cup\pi'}$. Since π is fixed from b_p w.r.t. F, and therefore also $F' \subseteq F$, and since $reset(F) = 0$, we get from Lemma 5.8.5 that $reset(F(\sigma, k+1) \cup \pi') \leq p$. Furthermore, since $\mathbf{a}_p^1 \sqsubseteq F(\sigma, k+1)$ we have $reset(F(\sigma, k+1) \cup \pi') \neq p$. Finally, it follows from Lemma 5.8.8 that $reset(F(\sigma, k+1)\cup\pi') = 0$. The rest of the proof for case (*ii*) then proceeds as the proof for case (*i*).

Case (*iii*)**:** Let $\pi' = \text{BLAND'S RULE}^*(F(\sigma, k+1), \pi, \sigma) \in \Pi^*_{F(\sigma,k+1)\cup\pi'}$. Since π is p-unstable w.r.t. F we have $\mathbf{a}_p^1 \cap \pi = \emptyset$, which implies that $\mathbf{a}_p^1 \not\sqsubseteq F(\sigma, k+1) \cup \pi'$. Since $\mathbf{b}_p^1 \subseteq F$, and π is fixed from b_p w.r.t. F, we get from Lemma 5.8.5 that $\mathbf{b}_p^1 \subseteq F(\sigma, k+1) \cup \pi'$. Moreover, since $reset(F) = 0$ it follows that $reset(F(\sigma, k+1)\cup\pi') = p$. We then get from Lemma 5.8.6 that

$\pi'' = \pi'[e]$ is p-stable and fixed from a_p w.r.t. $F(\sigma, k) \cup \pi'$, and therefore also $F(\sigma, k) \cup \pi''$. Furthermore, we have $reset(F(\sigma, k) \cup \pi'') = 0$. From the induction hypothesis we then know that for the second recursive call we have $h^*(F(\sigma, k), \pi'', \sigma) \geq g^*(N, \hat{\sigma})$, which completes the induction step.

Case (iv)**:** Since $e \notin \mathbf{b}_i^1$ where $i = \operatorname{argmin}_{i \in N} \hat{\sigma}(i)$, we have $e \notin \mathbf{b}_\ell^1$ for any $\ell \in N$. It follows that $active(F(\sigma, k+1), p) = N$. We claim that $reset(F') = 0$. Indeed, the only way we can get $reset(F') > 0$ is if $e \in \mathbf{a}_\ell^1$, for $\ell \in [n]$, where $\mathbf{b}_\ell^1 \subseteq F$, $\mathbf{a}_\ell^1 \sqsubseteq F$ and $\mathbf{a}_\ell^1 \not\sqsubseteq F'$. However, since σ is a good permutation we have $e \notin \mathbf{a}_\ell^1$ for any $\ell \in N$. Also, since π is either p-stable or p-unstable w.r.t. F we have $\mathbf{a}_\ell^1 \cap F = \mathbf{a}_\ell^1 \cap F'$ for all $\ell > p$ with $\mathbf{b}_\ell^1 \subseteq F'$. Finally, since either $k \neq \sigma(\mathbf{a}_p^1)$ or π is p-stable w.r.t. F, we have $reset(F') \neq p$. We conclude that $reset(F') = 0$. Since removing e from F does not change whether π is p-stable, fixed from a_p, p-unstable, or fixed from b_p w.r.t. F, we get that we can apply the induction hypothesis to the first recursive call: $f^*(F \setminus \{e\}, \pi, \sigma) \geq g^*(N, \hat{\sigma})$. The induction step follows.

This completes the proof. □

The following theorem is then proved in the same way as Theorem 5.7.7. Note that $E_0 = F(\sigma, 1)$ for any permutation function σ.

Theorem 5.8.10 *The expected number of improving switches performed by the* Randomized Bland's rule *algorithm for $G = G_{n,g,h,r}$, with $g = h = r = 3 \log n$, starting with π_0 is $\mathrm{e}^{\Omega(\sqrt{n})}$.*

Since the MDP $G_{n,g,h,r}$ contains $N = \Theta(n \log^2 n)$ vertices and $M = \Theta(n \log^3 n)$ edges, we can derive the following corollary:

Corollary 5.8.11 *There are MDPs with N states and $O(N \log N)$ actions for which the* Randomized Bland's rule *algorithm performs $\mathrm{e}^{\Omega(\sqrt{N}/\log N)}$ expected improving switches.*

Since $G_{n,g,h,r}$ satisfies the stopping condition we get the following corollary for linear programming.

Corollary 5.8.12 *There are linear programs in standard form with N equations and $O(N \log N)$ variables for which the expected number of pivots performed by the* Randomized Bland's rule *algorithm is $\mathrm{e}^{\Omega(\sqrt{N}/\log N)}$.*

Chapter 6

Deterministic Markov decision processes

6.1 Introduction

Howard's PolicyIteration algorithm [63] is one of the most widely used algorithms for solving Markov decision processes (MDPs). The complexity of Howard's algorithm in this setting was unresolved for almost 50 years. In 2010, Fearnley [34], building on results of Friedmann [38], showed that there are MDPs on which Howard's algorithm requires exponential time. In another breakthrough, Ye [121] showed that Howard's algorithm is strongly polynomial when applied to *discounted* MDPs, with a fixed discount ratio. Hansen, Miltersen, and Zwick [59] improved some of the bounds of Ye and extended them to the 2-player case. The results of Ye and of Hansen *et al.* were presented in sections 2.2.4 and 3.4.2.

Weighted directed graphs may be viewed as *Deterministic* MDPs (DMDPs) and solving such DMDPs for the average reward criterion is essentially equivalent to finding minimum mean-cost cycles (MMCCs) in such graphs. Howard's algorithm can thus be used to solve this purely combinatorial problem. The complexity of Howard's algorithm in this setting is an intriguing open problem. Fearnley's [34] exponential lower bound seems to depend in an essential way on the use of stochastic actions, so it does not extend to the deterministic setting. Similarly, Ye's [121] and Hansen, Miltersen, and Zwick's [59] polynomial upper bounds depend in an essential way on the MDPs being *discounted* and do not extend to the non-discounted (average reward) case.

The MMCC problem is an interesting problem that has various applica-

tions. It generalizes the problem of finding a negative cost cycle in a graph. It is also used as a subroutine in algorithms for solving other problems, such as min-cost flow algorithms, (See, e.g., Goldberg and Tarjan [52].)

There are several polynomial time algorithms for solving the MMCC problem. Karp [74] gave an $O(mn)$-time algorithm for the problem, where m is the number of edges and n is the number of vertices in the input graph. Young *et al.* [122] gave an algorithm whose complexity is $O(mn + n^2 \log n)$. Although this is slightly worse, in some cases, than the running time of Karp's algorithm, the algorithm of Young *et al.* [122] behaves much better in practice.

Dasdan [26] experimented with many different algorithms for the MMCC problem, including Howard's algorithm. He reports that Howard's algorithm usually runs much faster than Karp's algorithm, and is usually almost as fast as the algorithm of Young *et al.* [122]. A more thorough experimental study of MMCC algorithms was conducted by Georgiadis *et al.* [51].[1]

Understanding the complexity of Howard's algorithm for MMCCs is interesting from both the applied and theoretical points of view. Howard's algorithm for MMCC is an extremely simple and natural combinatorial algorithm, similar in flavor to the Bellman-Ford algorithm for finding shortest paths [7, 8, 37] and to Karp's [74] algorithm. Yet, its analysis seems to be elusive. Howard's algorithm also has the advantage that it can be applied to the more general problem of finding a cycle with a *minimum cost-to-time ratio* (see, e.g., Megiddo [90, 91]).

Howard's algorithm works in iteration. Each iteration takes $O(m)$ time. It is trivial to construct instances on which Howard's algorithm performs n iterations. (Recall that n and m are the number of vertices and edges in the input graph.) Madani [83] constructed instances on which the algorithm performs $2n - O(1)$ iterations. No graphs were known, however, on which Howard's algorithm performed more than a *linear* number of iterations. In this chapter we present the results of Hansen and Zwick [60]. We construct the first graphs on which Howard's algorithm performs $\Omega(n^2)$ iterations, showing, in particular, that there are instances on which its running time is $\Omega(n^4)$, an order of magnitude *slower* than the running times of the algorithms of Karp [74] and Young *et al.* [122]. Our lower bounds on the complexity of Howard's algorithm do not undermine the usefulness of Howard's algorithm, as the instances used in our quadratic lower bound are

[1] Georgiadis *et al.* [51] claim that Howard's algorithm is not robust. From personal conversations with the authors of [51] it turns out, however, that the version they used is substantially different from Howard's algorithm [63].

very unlikely to appear in practice.

For any fixed discount factor $\gamma < 1$, we also show how these lower bounds can be transferred to corresponding discounted MDPs. I.e., we show that there exist weighted directed graphs for which Howard's algorithm, with the discounted reward criterion and discount factor γ, performs $\Omega(n^2)$ iterations. The upper bound by Hansen, Miltersen, and Zwick [59] for this case is $O(\frac{m}{1-\gamma} \log \frac{n}{1-\gamma})$. The lower bound is therefore tight up to a logarithmic factor when γ is fixed.

In Hansen and Zwick [60] we also construct n-vertex outdegree-2 graphs on which Howard's algorithm performs $2n - O(1)$ iterations. (Madani's [83] examples used $\Theta(n^2)$ edges.) This example is interesting as it shows that the number of iterations performed may differ from the number of edges in the graph by only an additive constant. We refer to Hansen and Zwick [60][2] for this result.

Our examples still leave open the possibility that the number of iterations performed by Howard's algorithm is always at most m, the number of edges. (The graphs on which the algorithm performs $\Omega(n^2)$ iterations also have $\Omega(n^2)$ edges.) We conjecture that this is always the case:

Conjecture 6.1.1 *The number of iterations performed by Howard's algorithm, when applied to a weighted directed graph, is at most the number of* edges *in the graph.*

6.2 Howard's PolicyIteration algorithm

We next describe the specialization of Howard's algorithm for *deterministic* MDPs with the average reward criterion, i.e., for finding Minimum Mean-Cost Cycles. For a description of Howard's algorithm for general MDPs, see Chapter 2.

Recall that, using the graphical representation of MDPs (Definition 2.1.3), a deterministic MDP can be represented as a weighted directed graph $G = (V, E, c)$, where $c : E \to \mathbb{R}$. We assume that each vertex has a unique *serial number* associated with. We also assume, without loss of generality, that each vertex $v \in V$ has at least one outgoing edge. Since we study the MMCC problem we interpret $c(u, v)$ as the cost of the edge (u, v), and not the reward. I.e., the controller of the MDP tries to minimize the incurred costs.

[2] A full version of the paper is available at `http://cs.au.dk/~tdh/papers/policy.pdf`.

If $C = v_0 v_1 \dots v_{k-1} v_0$ is a cycle in G, we let $\text{val}(C) = \frac{1}{k} \sum_{i=0}^{k-1} c(v_i, v_{i+1})$, where $v_k = v_0$, be its *mean cost*. The vertex on C with the smallest serial number is said to be the *head* of the cycle. Our goal is to find a cycle C that minimizes $\text{val}(C)$.

A *policy* π is a mapping $\pi : V \to V$ such that $(v, \pi(v)) \in E$, for every $v \in V$. Note that following the definition of policies for general MDPs a policy would map each vertex to an outgoing edge. We may assume, however, that there is only one edge from any vertex u to any other vertex v, such that the two definitions are equivalent. A policy π, defines a subgraph $G_\pi = (V, E_\pi)$, where $E_\pi = \{(v, \pi(v)) \mid v \in V\}$. As the outdegree of each vertex in G_π is 1, we get that G_π is composed of a collection of disjoint directed cycles with directed paths leading into them.

Given a policy π, we assign to each vertex $v_0 \in V$ a value $\text{val}_\pi(v_0)$ and a potential $\text{pot}_\pi(v_0)$. Values and potential were defined for general MDPs in Definition 2.3.2. We present here a specialization to deterministic MDPs (weighted directed graphs). Let $P_\pi(v_0) = v_0 v_1, \dots$ be the infinite path defined by $v_i = \pi(v_{i-1})$, for $i > 0$. This infinite path is composed of a finite path P leading to a cycle C which is repeated indefinitely. We will use the convention that P ends at the head of C. I.e., the finite path does not end as soon as it reaches C, and as a consequence all paths leading to the same cycle C ends at the same vertex. If $v_r = v_{r+k}$ is the vertex that is the head of C, then $P = v_0 v_1 \dots v_r$ and $C = v_r v_{r+1} \dots v_{r+k}$. We now define

$$\begin{aligned} \text{val}_\pi(v_0) &= \text{val}(C) = \frac{1}{k} \sum_{i=0}^{k-1} c(v_{r+i}, v_{r+i+1}), \\ \text{pot}_\pi(v_0) &= \sum_{i=0}^{r-1} (c(v_i, v_{i+1}) - \text{val}(C)). \end{aligned}$$

In other words, $\text{val}_\pi(v_0)$ is the mean cost of C, the cycle into which $P_\pi(v_0)$ is absorbed, while $\text{pot}_\pi(v_0)$ is the *distance* from v_0 to v_r, the head of this cycle, when the mean cost of the cycle is subtracted from the cost of each edge. It is easy to check that values and potentials satisfy the following equations:

$$\begin{aligned} \text{val}_\pi(v) &= \text{val}_\pi(\pi(v)) \,, \\ \text{pot}_\pi(v) &= c(v, \pi(v)) - \text{val}_\pi(v) + \text{pot}_\pi(\pi(v)) \,. \end{aligned}$$

Note that our definition of potentials does not exactly match the equations in Theorem 2.3.3 for general MDPs. For this to be the case we should require that $\sum_{v \in C} \text{pot}_\pi(v) = 0$ for every cycle C of G_π, instead of $\text{pot}_\pi(v_r) = 0$

where v_r is the head of C. We use this alternative definition of potentials for simplicity. We will briefly argue below that Howard's PolicyIteration algorithm also works with these modified potentials. For additional details see Puterman [101].

The *reduced cost* of an edge $(u, v) \in E$ is defined as the pair:

$$\bar{\mathbf{c}}^{\pi}(u, v) \;=\; (\,\mathrm{val}_{\pi}(v) - \mathrm{val}_{\pi}(u)\;,\; c(u, v) - \mathrm{val}_{\pi}(u) + \mathrm{pot}_{\pi}(v) - \mathrm{pot}_{\pi}(u)\,)\;.$$

Note that this matches Definition 2.3.7. We compare such pairs lexicographically. I.e., $(x_1, y_1) < (x_2, y_2)$ if and only if $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$. Note that $\bar{\mathbf{c}}^{\pi}(u, \pi(u)) = (0, 0)$. An edge (u, v) is an improving switch if and only if $\bar{\mathbf{c}}^{\pi}(u, v) < (0, 0)$.

Howard's algorithm starts with an arbitrary policy π and keeps *improving* it. If π is the current policy, then the next policy π' produced by the algorithm is defined by

$$\pi'(u) \;=\; \operatorname*{argmin}_{v:(u,v)\in E} \bar{\mathbf{c}}^{\pi}(u, v)\;. \tag{6.1}$$

In other words, for every vertex the algorithm selects the outgoing edge with the lowest reduced cost. (In case of ties, the algorithm favors edges in the current policy.) When $\pi' = \pi$, the algorithm stops. The algorithm is presented for general MDPs in Section 2.3.1. Let us note that (6.1) can be simplified to:

$$\pi'(u) \;=\; \operatorname*{argmin}_{v:(u,v)\in E} (\,\mathrm{val}_{\pi}(v)\;,\; c(u, v) - \mathrm{val}_{\pi}(u) + \mathrm{pot}_{\pi}(v)\,)\;, \tag{6.2}$$

since $\mathrm{val}_{\pi}(u)$ and $\mathrm{pot}_{\pi}(u)$ are constant when taking the minimum over edges leaving u.

Let us note that the modified potentials we use can be obtained from the potentials described by Theorem 2.3.3 in the following way. For each cycle C of G_{π}, for some policy π, we add a suitable constant to the potentials of the vertices of C and all vertices on paths leading to C. This can only change the behavior of Howard's PolicyIteration algorithm if two cycles of G_{π} have the same value, but different constants are added to the potentials of each cycle. This situation does not occur for our lower bound constructions. It can be shown that Howard's PolicyIteration algorithm also works with the modified potentials. For details, see Puterman [101].

The following theorem is an interpretation of Theorem 2.3.9 for Howard's PolicyIteration algorithm for deterministic MDPs.

Theorem 6.2.1 *Let π be a policy.*

(*i*) *Suppose that* π' *is obtained from* π *by a policy improvement step. Then, for every* $v \in V$ *we have* $(\mathrm{val}_{\pi'}(v), \mathrm{pot}_{\pi'}(v)) \leq (\mathrm{val}_{\pi}(v), \mathrm{pot}_{\pi}(v))$. *Furthermore, if* $\pi'(v) \neq \pi(v)$, *then* $(\mathrm{val}_{\pi'}(v), \mathrm{pot}_{\pi'}(v)) < (\mathrm{val}_{\pi}(v), \mathrm{pot}_{\pi}(v))$.

(*ii*) *If* π *is not modified by an improvement step, then* $\mathrm{val}_{\pi}(v)$ *is the minimum mean weight of any cycle reachable from* v *in* G. *Furthermore by following edges of* π *from* v *we get into a cycle of this minimum mean weight.*

Each iteration of Howard's algorithm takes only $O(m)$ time and is not much more complicated than an iteration of the Bellman-Ford algorithm.

6.3 A quadratic lower bound

We next construct a family of weighted directed graphs for which the number of iterations performed by Howard's algorithm is *quadratic* in the number of vertices. More precisely, we prove the following theorem:

Theorem 6.3.1 *Let* n *and* m *be even integers, with* $\frac{5n}{3} \leq m \leq \frac{n^2}{9} + \frac{4n}{3}$. *There exists a weighted directed graph with* n *vertices and* m *edges on which Howard's algorithm performs* $m - n + 1$ *iterations.*

All policies generated by Howard's algorithm, when run on the instances of Theorem 6.3.1, contain a *single* cycle, and hence all vertices have the same *value*. Edges are therefore selected for inclusion in the improved policies based on *potentials*. (Recall that potentials are essentially adjusted distances.) The main idea behind our construction, which we refer to as the *dancing cycles* construction, is the use of cycles of very large costs, so that Howard's algorithm favors long, i.e., containing many edges, paths to the cycle of the current policy attractive, delaying the discovery of better cycles.

Given a graph and a sequence of policies, it is possible check, by solving an appropriate *linear program*, whether there exist costs for which Howard's algorithm generates the given sequence of policies. Experiments with a program that implements this idea helped us obtain the construction presented below.

For simplicity, we first prove Theorem 6.3.1 for $m = \frac{n^2}{9} + \frac{4n}{3}$. We later note that the same construction works when removing pairs of edges, which gives the statement of the theorem.

6.3.1 The Construction

For every n we construct a weighted directed graph $G_n = (V, E, c)$, on $|V| = 3n$ vertices and $|E| = n^2 + 4n$ edges, and an initial policy π^0 such that Howard's algorithm performs n^2+n+1 iterations on G_n when it starts with π^0.

The graph G_n is shown in Figure 6.1 for $n = 4$. The graph G_n is composed of two symmetric parts. To highlight the symmetry we let $V = \{v_n^1, \ldots, v_1^1, v_1^0, \ldots, v_n^0\}$. Note that the set of vertices is split in two, according to whether the superscript is 0 or 1. In order to simplify notation when dealing with vertices with different superscripts, we sometimes refer to v_1^0 as v_0^1 and to v_1^1 as v_0^0. The set of edges is:

$$E = \{(v_i^1, v_j^1), (v_i^0, v_j^0) \mid 1 \leq i \leq n,\ i-1 \leq j \leq n\}.$$

The edge (v_1^1, v_1^0) plays a special role, and it should be interpreted as a path of length $n+1$. I.e., we introduce an additional n vertices for this path. Since there is only one edge leaving each of these vertices, every policy must contain these edges. We view the path as a single edge in order to simplify notation.

It will be helpful to assign indices to some of the edges. For this purpose we define the following set of triplets:

$$T_n = \{(i, j, s) \mid 1 \leq i \leq j \leq n \wedge 0 \leq s \leq 1\}\ .$$

We also define a bijection $f : T_n \to \{1, \ldots, n^2 + n\}$ such that $f(\ell_1, r_1, t_1) < f(\ell_2, r_2, t_2)$ if and only if (ℓ_1, r_1, t_1) is lexicographically smaller than (ℓ_2, r_2, t_2). It is not difficult to see that:

$$f(i, j, s) \ = \ (2n - i)(i - 1) + 2(j - 1) + s + 1\ .$$

For every $(i, j, s) \in T_n$ we then say that the edge (v_i^s, v_j^s) has index $f(i, j, s)$. The costs of the edges are then defined as follows:

$$c(v_i^s, v_j^s) = \begin{cases} n^{f(i,j,s)} & \text{if } (i, j, s) \in T_n \\ 0 & \text{otherwise} \end{cases}$$

I.e., the costs of the edges grow exponentially with the indices of the edges. The exponents of the costs of edges for which we defined indices are shown in Figure 6.1. If an edge has cost 0 it is indicated by the number 0.

Let us note that the costs presented here have been chosen to simplify the analysis. It is possible to define smaller costs, but assuming $c(v_i^s, v_{i-1}^s) = 0$

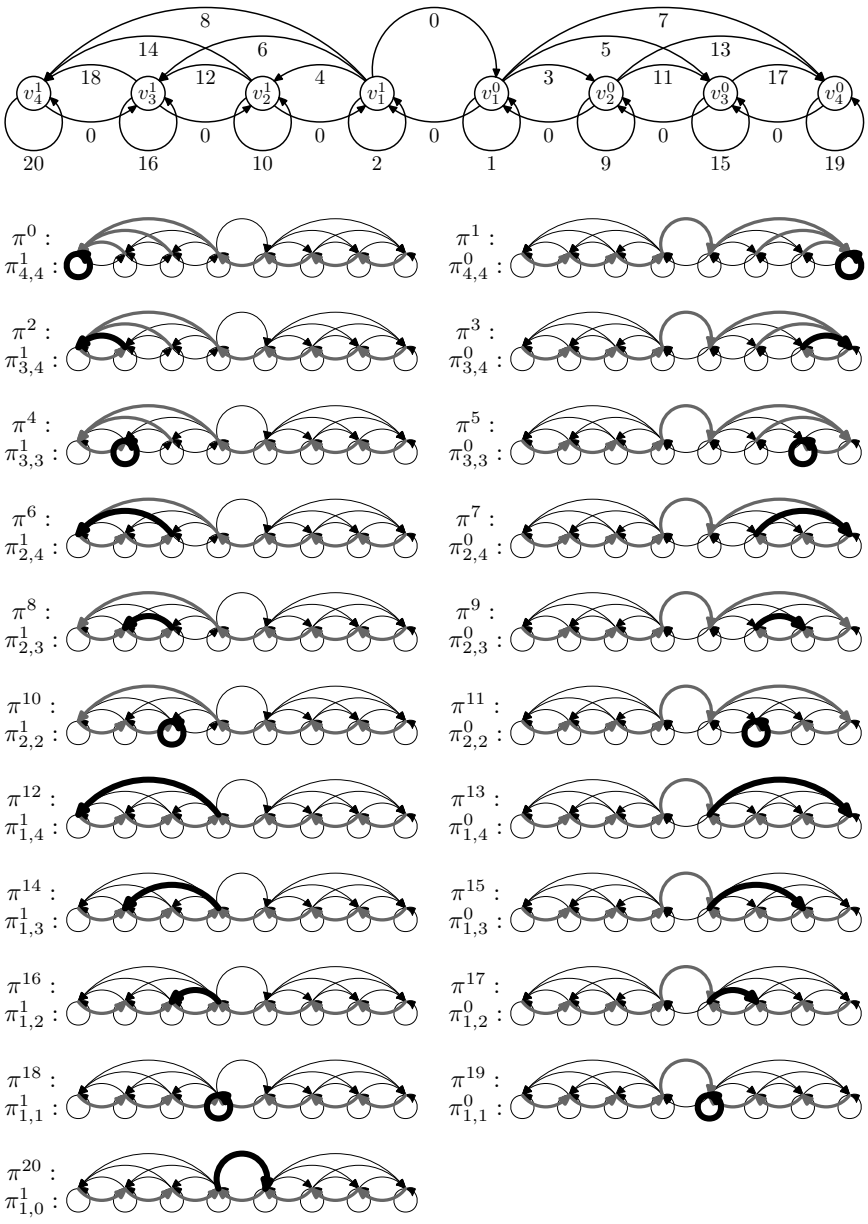


Figure 6.1: G_4 and the corresponding sequence Π_4. Π_4 is shown in left-to-right order. Policies $\pi^{f(\ell,r,s)} = \pi^t_{\ell,r}$ are shown in bold, with the edge with index $f(\ell, r, s)$ being highlighted. Numbers below edges define costs. 0 means 0, $k > 0$ means n^k. The edge (v_1^1, v_1^0) represents a path of length $n + 1$ and cost 0.

for $s \in \{0,1\}$ and $1 \leq i \leq n$ (which can always be enforced using a potential transformation), one can show that integral costs must be exponential in n.

We next describe a sequence of policies Π_n of length n^2+n+1. We will later show that Howard's algorithm generates this sequence of policies when run on G_n. For $(\ell, r, t) \in T_n$, and for $(\ell, r, t) = (1, 0, 1)$, we define a policy $\pi^t_{\ell,r}$:

$$\pi^t_{\ell,r}(v^s_i) = \begin{cases} v^s_{i-1} & \text{for } s \neq t \text{ or } i > \ell \\ v^s_r & \text{for } s = t \text{ and } i = \ell \\ v^s_n & \text{for } s = t \text{ and } i < \ell \end{cases}$$

The policy $\pi^t_{\ell,r}$ contains a single cycle $v^t_r v^t_{r-1} \dots v^t_\ell v^t_r$ which is determined by its *defining* edge $e^t_{\ell,r} = (v^t_\ell, v^t_r)$. As shown in Figure 6.1, all vertices to the left of v^t_ℓ, the *head* of the cycle, choose an edge leading furthest to the right, while all the vertices to the right of v^t_ℓ choose an edge leading furthest to the left.

We let $N = n^2 + n$. The sequence $\Pi_n = (\pi^k)_{k=0}^N$ is defined as follows. For $0 \leq k < N$, we let $\pi^k = \pi^t_{\ell,r}$ such that $k = N - f(\ell, r, t)$. Furthermore, we let $\pi^N = \pi^1_{1,0}$. We can now write:

$$\Pi_n = \left(\pi^k\right)_{k=0}^{N} = \left(\left(\left(\pi^{1-t}_{n-\ell,n-r}\right)_{t=0}^{1}\right)_{r=0}^{\ell}\right)_{\ell=0}^{n-1}, \pi^1_{1,0}$$

We refer to Figure 6.1 for an illustration of G_4 and the corresponding sequence Π_4.

6.3.2 Proof of lower bound

We will next prove that the sequence of policies generated by Howard's PolicyIteration algorithm for G_n starting from π^0 is Π_n.

Let $(\ell, r, t) \in T_n$ be given, and let $k = N - f(\ell, r, t)$. The only cycle in $\pi^k = \pi^t_{\ell,r}$ is $C^t_{\ell,r} = v^t_r v^t_{r-1} \dots v^t_\ell v^t_r$, and we let v^t_ℓ be the head of $C^t_{\ell,r}$. As $c(v^s_i, v^s_{i-1}) = 0$, for $1 \leq i \leq n$ and $s \in \{0,1\}$, we have

$$\text{val}(C^t_{\ell,r}) \;=\; \frac{c(v^t_\ell, v^t_r)}{r - \ell + 1}\,.$$

Let $\mu_k = \text{val}(C^t_{\ell,r})$ be the value of every state for the policy π^k. As $c(v^t_\ell, v^t_r) = n^{N-k}$ and all cycles in our construction are of size at most n we have

$$n^{N-k-1} \;\leq\; \mu_k \;\leq\; n^{N-k}\,.$$

In order to prove that the policies generated by Howard's PolicyIteration algorithm are the ones given by Π_n, we must show that (6.2) is satisfied for every iteration. Since there is only one cycle for each of the relevant policies, the values of all the states are the same, and it suffices to show that:

$$\pi^{k+1}(u) \;=\; \operatorname*{argmin}_{v:(u,v)\in E} \; c(u,v) - \mu_k + \mathrm{pot}_{\pi^k}(v) \quad , \quad \forall u \in V \; . \tag{6.3}$$

Note that the definition of potentials shows that:

$$\mathrm{pot}_{\pi}(u) \;=\; c(u,\pi(u)) - \mu_k + \mathrm{pot}_{\pi^k}(\pi(u))$$

Recall that $\mathrm{pot}_{\pi^k}(v)$ is the distance from v to v_ℓ^t in the graph G_{π^k} when μ_k is subtracted from the cost of every edge. The expression in (6.3) can, thus, be interpreted as the distance from u to v_ℓ^t when moving along the edge (u,v) and then continuing in G_{π^k}, paying at each step the cost of the edge minus μ_k.

For every vertex $u \in V$, let $\mathit{steps}(u,k)$ be the number of edges on the path from u to v_ℓ^t in G_{π^k}. Furthermore, let $\mathit{cost}(u,k)$ be the sum of the costs on the path from u to v_ℓ^t in G_{π^k}. It follows that:

$$\mathrm{pot}_{\pi^k}(u) \;=\; \mathit{cost}(u,k) - \mathit{steps}(u,k)\mu_k \; , \tag{6.4}$$

It is not difficult to check that (v_ℓ^t, v_r^t) is the most expensive edge in π^k, and that on the path from u to v_ℓ^t in G_{π^k} only one edge has cost different from 0. Furthermore, no edge in π^k has index $N-k-1$. (The edge with index $N-k-1$ is the defining edge for π^{k+1}.) It follows that:

$$0 \;\le\; \mathit{cost}(u,k) \;\le\; n^{N-k-2} \tag{6.5}$$

The following lemma shows that no edge more expensive than (v_ℓ^t, v_r^t) is going to be an improving switch, and that it is better not to use (v_ℓ^t, v_r^t) if there is an available edge with lower cost.

Lemma 6.3.2 *Let $k \in \{0,\ldots,N-1\}$, let $u \in V$ be any vertex, and let $(u,v) \in E$ be an edge such that $c(u,v) \ge n^{N-k+1}$. Then:*

$$c(u,v) - \mu_k + \mathrm{pot}_{\pi^k}(v) > \mathrm{pot}_{\pi^k}(u)$$

I.e., (u,v) is not an improving switch. Furthermore, if $c(u,\pi(u)) = n^{N-k}$ and $c(u,v) \le n^{N-k-1}$ then:

$$c(u,v) - \mu_k + \mathrm{pot}_{\pi^k}(v) < \mathrm{pot}_{\pi^k}(u)$$

Proof: We first assume that $c(u,v) = n^{N-k+1}$, and let $(\ell', r', t') \in T_n$ be the triple such that $k-1 = N - f(\ell', r', t')$. Then $u = v_{\ell'}^{t'}$ and $v = v_{r'}^{t'}$. Furthermore, $\mathrm{pot}_{\pi^k}(v) = -(r'-\ell')\mu_k + \mathrm{pot}_{\pi^k}(u)$. It follows that:

$$\mathrm{pot}_{\pi^k}(u) - \mathrm{pot}_{\pi^k}(v) + \mu_k = (r'-\ell'+1)\mu_k < (r'-\ell'+1)\mu_{k-1} = n^{N-k+1}$$

It follows that (u,v) is not an improving switch.

Next, consider the case when $c(u,v) \geq n^{N-k+2}$. Let $q = \mathit{steps}(v,k) - \mathit{steps}(u,k)$. From (6.4) we then have:

$$\begin{aligned} \mathrm{pot}_{\pi^k}(u) - \mathrm{pot}_{\pi^k}(v) + \mu_k &= \mathit{cost}(u,k) - \mathit{cost}(v,k) + (q+1)\mu_k \\ &\leq n^{N-k-2} + (q+1)n^{N-k} \\ &\leq n^{N-k-2} + n^{N-k+1} \ . \end{aligned}$$

Since $n^{N-k+2} > n^{N-k-2} + n^{N-k+1}$, this completes the proof of the first part of the lemma.

Next, assume that $c(u,\pi(u)) = n^{N-k}$ and $c(u,v) \leq n^{N-k-1}$. In particular, we must have $v \neq u = v_\ell^t$, and therefore $\mathrm{pot}_{\pi^k}(u) = 0$. Furthermore, no edge leaving v_ℓ^t has cost n^{N-k-1}, so we must have $c(u,v) \leq n^{N-k-2}$. Since $\mathit{steps}(v,k) > 0$ we then get $\mathrm{pot}_{\pi^k}(v) = \mathit{cost}(v,k) - \mathit{steps}(v,k)\mu_k < 0$. It follows that:

$$c(u,v) - \mu_k + \mathrm{pot}_{\pi^k}(v) \ < \ n^{N-k-2} - n^{N-k-1} \ < \ 0 \ ,$$

which completes the proof. □

It follows from Lemma 6.3.2 that all edges more expensive than the defining edge $e_{\ell,r}^t$ are too expensive to enter policies appearing after $\pi^k = \pi_{\ell,r}^t$. Also, if an edge cheaper than $e_{\ell,r}^t$ is available from v_ℓ^t, then $e_{\ell,r}^t$ will not appear in later iterations. Define:

$$W(u,k) = \{v \in V \mid (u,v) \in E \wedge c(u,v) \leq n^{N-k-1}\} \ .$$

Then $W(u,k)$ defines the edges emanating from u that have not been ruled out by Lemma 6.3.2.

Lemma 6.3.3 *Let $k \in \{0,\ldots,N-1\}$, let $u \in V$ be any vertex, and let $v_1, v_2 \in W(u,k)$ be two vertices with $\mathit{steps}(v_1,k) > \mathit{steps}(v_2,k)$. Then:*

$$c(u,v_2) - \mu_k + \mathrm{pot}_{\pi^k}(v_2) > c(u,v_1) - \mu_k + \mathrm{pot}_{\pi^k}(v_1) \ .$$

I.e., (u,v_1) is preferred over (u,v_2).

Proof: Define $q = steps(v_1, k) - steps(v_2, k)$, and observe that:

$$\begin{aligned} \mathrm{pot}_{\pi^k}(v_1) - \mathrm{pot}_{\pi^k}(v_2) &= cost(v_1, k) - cost(v_2, k) - q\mu_k \\ &\leq cost(v_1, k) - cost(v_2, k) - qn^{N-k-1} \ . \end{aligned}$$

We consider two cases. Either $c(u, v_1) \leq n^{N-k-2}$ or $c(u, v_1) = n^{N-k-1}$. In the first case we get the two inequalities:

$$\begin{aligned} c(u, v_2) - c(u, v_1) &\geq -n^{N-k-2} \ , \\ cost(v_1, k) - cost(v_2, k) - qn^{N-k-1} &\leq n^{N-k-2} - n^{N-k-1} \ . \end{aligned}$$

It follows that $c(u, v_2) - c(u, v_1) > pot_{\pi^k}(v_1) - \mathrm{pot}_{\pi^k}(v_2)$ (for $n > 2$).

Let $(\ell', r', t') \in T_n$ be the triple such that $k + 1 = N - f(\ell', r', t')$. If $c(u, v_1) = n^{N-k-1}$ then $u = v^{t'}_{\ell'}$ and $v_1 = v^{t'}_{r'}$. It follows that $cost(v_1, k) - cost(v_2, k) = 0$, and that either $c(u, v_2) > 0$, $q > 1$, or $\mu_k > n^{N-k-1}$. In all three cases we get the desired inequality. □

Using Lemma 6.3.2 and Lemma 6.3.3 it is not difficult to show that Howard's PolicyIteration algorithm generates the sequence of policies Π_n when run on G_n. This can be done with a case analysis that studies for different groups of vertices how the policy is updated. Such a partition into cases is shown in Figure 6.2.

Note that $W(u, k)$ only contains edges with indices lower than $N - k$. Thus, with each new iteration one less edge is available. Notice also that for some vertex v^s_i, the larger the index of an edge leaving v^s_i is, the closer the edge moves to v^s_n.

Let $\pi^t_{\ell,r}$ be the current policy, and let $k = f(\ell, r, t)$. We next consider the cases shown in Figure 6.2, and show that the correct transitions are made. The main observation used in all the cases is that, for some vertex $u \in V$, if $\mathrm{argmax}_{\ v \in W(u,k)} steps(v, k)$ is achieved by a unique vertex v', then we know from lemmas 6.3.2 and 6.3.3 that the next policy π' generated by Howard's PolicyIteration algorithm satisfies $\pi'(u) = v'$. We must, thus, show that $\pi^{k+1}(u) = v'$. We only present the case analysis for case 1, $t = 1$. The remaining cases are handled analogously.

Case (1.1): Consider v^s_i for $i > \ell$ and $s \in \{0, 1\}$. The only element in $W(v^s_i, k)$ is v^s_{i-1}, and it follows from Lemma 6.3.2 that $\pi^{k+1}(v^s_i) = v^s_{i-1}$ corresponds to the edge picked by Howard's PolicyIteration algorithm.

Cases (1.2) and (1.3): Consider v^0_i for $i \leq \ell$. From lemmas 6.3.2 and 6.3.3, we know that Howard's algorithm picks $v \in W(v^0_i, k)$ such that $steps(v, k)$ is maximized. We get that $\pi^{k+1}(v^0_\ell) = v^0_r$ (case 1.2), and $\pi^{k+1}(v^0_i) = v^0_n$ for $i < \ell$ (case 1.3), are the choices made by Howard's algorithm.

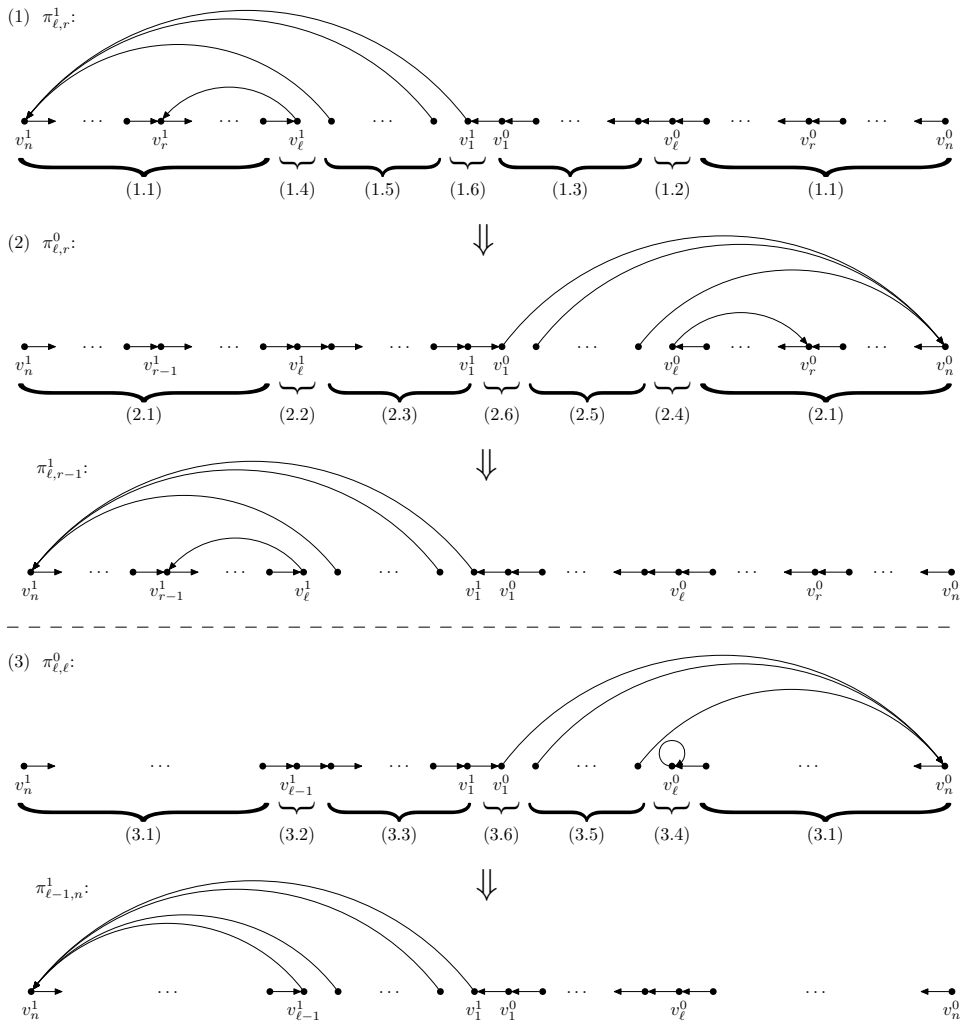


Figure 6.2: Policies of transitions (1) $\pi^1_{\ell,r}$ to $\pi^0_{\ell,r}$, (2) $\pi^0_{\ell,r}$ to $\pi^1_{\ell,r-1}$, and (3) $\pi^0_{\ell,\ell}$ to $\pi^1_{\ell-1,n}$. Vertices of the corresponding sub-cases have been annotated accordingly.

Case (1.4): Consider v^1_ℓ. We see that:

$$\pi^{k+1}(v^1_\ell) \;=\; v^1_{\ell-1} \;=\; \operatorname*{argmax}_{v \in W(v^1_\ell,k)} \mathit{steps}(v,k)\;.$$

Hence, we know from lemmas 6.3.2 and 6.3.3 that Howard's algorithm picks the desired edge.

Case (1.5): Consider v_i^1 for $\ell < i < 1$. We have

$$v_{i-1}^1 \in \operatorname*{argmax}_{v \in W(v_i^1,k)} \mathit{steps}(v,k) \ ,$$

but v_{i-1}^1 is not the only vertex achieving this number of steps. There may be several ways of getting to v_n^1 in two steps. Since edges emanating from v_j^1, for $j > i-1$, have larger indices than edges emanating from v_{i-1}^1, it is not difficult to see, however, that $\pi^{k+1}(v_i^1) = v_{i-1}^1$ is the cheapest choice.

Case (1.6): Consider v_1^1. Recall that the edge (v_1^1, v_1^0) represents a path of length $n+1$ and cost 0 to v_1^0. We, thus, get that

$$\pi^{k+1}(v_1^1) \ = \ v_1^0 \ = \ \operatorname*{argmax}_{v \in W(v_1^1,k)} \mathit{steps}(v,k) \ ,$$

is the choice made by Howard's algorithm. Let us note that we need (v_1^1, v_1^0) to represent a path of length $n+1$ exactly when $\ell = 1$. In all other cases it would suffice to use a single edge.

This completes the case analysis.

Let us note that for any $r \in [n]$ and $1 < \ell \leq r$, if the edges (v_ℓ^0, v_r^0) and (v_ℓ^1, v_r^1) are removed from G_n, then Howard's algorithm skips $\pi_{\ell,r}^1$ and $\pi_{\ell,r}^0$, but otherwise Π_n remains the same. This can be repeated any number of times, essentially without modifying the proof given in this section, thus giving us the statement of Theorem 6.3.1.

6.4 Discounted deterministic MDPs

We next show how our lower bound for Howard's PolicyIteration algorithm for deterministic MDPs with the average reward criterion can be modified to work for the discounted reward criterion with a fixed discount. See Section 2.2 for a more extensive introduction to the discounted reward criterion.

We use the same terminology as in Section 6.2, except that we define values and reduced costs differently. Let G be a weighted directed graph, let $\gamma < 1$ be a discount factor, and let π be a policy. Then the values of the vertices are defined as the unique solution to:

$$\forall v \in V: \quad \mathrm{val}_\pi(v) \ = \ c(v, \pi(v)) + \gamma \mathrm{val}_\pi(\pi(v)) \ .$$

Let $v_0 \in V$, and let $P_\pi(v_0) = v_0 v_1, \ldots$ be the infinite path defined by $v_i = \pi(v_{i-1})$, for $i > 0$. This infinite path is composed of a finite path P leading

to a cycle C which is repeated indefinitely. If $v_r = v_{r+k}$ is the vertex that is the head of C, then $P = v_0 v_1 \dots v_r$ and $C = v_r v_{r+1} \dots v_{r+k}$. The value of v_0 for π can then also be computed as:

$$\mathrm{val}_\pi(v_0) \;=\; \Big[\sum_{i=0}^{r-1} \gamma^i c(v_i, v_{i+1})\Big] + \gamma^r \frac{\sum_{i=0}^{k-1} \gamma^i c(v_{r+i}, v_{r+i+1})}{1-\gamma^k} \;.$$

The reduced cost of an edge $(u, v) \in E$ for π is:

$$\bar{\mathbf{c}}^\pi(u, v) \;=\; c(u, v) + \gamma \mathrm{val}_\pi(v) - \mathrm{val}_\pi(u) \;.$$

Let π' be the policy obtained from π after one iteration of Howard's policy iteration algorithm. Then for all $u \in V$:

$$\pi'(u) \;=\; \operatorname*{argmin}_{v:(u,v)\in E} \bar{\mathbf{c}}^\pi(u, v) \;=\; \operatorname*{argmin}_{v:(u,v)\in E} c(u, v) + \gamma \mathrm{val}_\pi(v) \;,$$

with ties broken in favor of edges in π. For details on Howard's PolicyIteration algorithm see Section 2.2.2.

6.4.1 A quadratic lower bound

The following theorem is the analog of Theorem 6.3.1 for the average reward criterion. Note that it is essential that the discount factor $\gamma < 1$ is fixed, since the costs used to define the graph mentioned in the theorem depends on γ.

Theorem 6.4.1 *Let $\gamma < 1$ be a fixed discount factor, and let n and m be even integers with $\frac{5n}{3} \le m \le \frac{n^2}{9} + \frac{4n}{3}$. There exists a weighted directed graph with n vertices and m edges on which Howard's algorithm performs $m-n+1$ iterations for the discounted reward criterion with discount factor γ.*

The presentation of the proof of Theorem 6.4.1 will rely heavily on Section 6.3. We use the same lower bound construction for Theorem 6.4.1 as we did for Theorem 6.3.1, except that the costs are defined differently. I.e., for every $n \in \mathbb{N}$, we define a graph $G_n = (V, E, c)$. We let V, E, T_n, and $f : T_n \to \{1, \dots, n^2 + n\}$ be defined as in Section 6.3.1. The costs are then defined as:

$$c(v_i^s, v_j^s) = \begin{cases} \Gamma^{f(i,j,s)} & \text{if } (i, j, s) \in T_n \\ 0 & \text{otherwise} \end{cases}$$

where $\Gamma = \frac{n}{\gamma^{3n}}$. Note that in the limit for $\gamma \to 1$ we get the same costs as in Section 6.3.1.

We also define the sequence of policies $\Pi_n = (\pi^k)_{k=0}^N$, where $N = n^2 + n$, as in Section 6.3.1. Recall that for $(\ell, r, t) \in T_n$ and $k = N - f(\ell, r, t)$ we have $\pi^k = \pi^t_{\ell,r}$. The goal of the proof is again to show that Howard's PolicyIteration algorithm generates Π_n when run on G_n starting from π^0.

Let $(\ell, r, t) \in T_n$ be given, and let $k = N - f(\ell, r, t)$. Then v^t_ℓ is the head of the cycle $C^t_{\ell,r} = v^t_r v^t_{r-1} \dots v^t_\ell v^t_r$ in G_{π^k}. Furthermore, we have:

$$\mathrm{val}_{\pi^k}(v^t_\ell) \;=\; \frac{c(v^t_\ell, v^t_r)}{1-\gamma^{r-\ell+1}} \;\le\; \frac{1}{1-\gamma}\Gamma^{N-k} \,.$$

Observe also that:

$$\begin{aligned}\mathrm{val}_{\pi^k}(v^t_\ell) \;&\ge\; \frac{1}{1-\gamma^n}\Gamma^{N-k} \;=\; \frac{1}{(1-\gamma)\sum_{i=0}^{n-1}\gamma^i}\Gamma^{N-k} \\ &>\; \frac{1}{(1-\gamma)n}\Gamma^{N-k} \;=\; \frac{1}{(1-\gamma)\gamma^{3n}}\Gamma^{N-k-1}\end{aligned}$$

Let $v_0 \in V$, and let $P_{\pi^k}(v_0) = v_0 v_1, \dots$ be the path defined by $v_i = \pi^k(v_{i-1})$, for $i > 0$. Define $\mathit{steps}(v_0, k)$ to be the number of steps from v_0 to v^t_ℓ in G_{π^k}. I.e., $v_r = v^t_\ell$ where $r = r(v_0, k) = \mathit{steps}(v_0, k)$. We sometimes use r instead of *steps* to get more compact expressions. Define:

$$\mathit{cost}(v_0, k) = \Big[\sum_{i=0}^{r-1} \gamma^i c(v_i, v_{i+1})\Big] \,.$$

Then we have:

$$\mathrm{val}_{\pi^k}(v_0) \;=\; \mathit{cost}(v_0, k) + \gamma^{r(v_0,k)}\mathrm{val}_{\pi^k}(v^t_\ell) \,.$$

Recall that (v^t_ℓ, v^t_r) is the most expensive edge in π^k. Also, on the path from u to v^t_ℓ in G_{π^k}, for any $u \in V$, only one edge has cost different from 0, and this edge has index at most $N - k - 2$. I.e., $\mathit{cost}(u, k) \le \Gamma^{N-k-2}$ for all $u \in V$. For any vertex $u \in V \setminus \{v^t_\ell\}$ we then have:

$$\mathrm{val}_{\pi^k}(u) \;\le\; \Gamma^{N-k-2} + \gamma^{r(u,k)}\frac{1}{1-\gamma}\Gamma^{N-k} \;\le\; \frac{1}{1-\gamma}\Gamma^{N-k} \,.$$

The following two lemmas correspond to lemmas 6.3.2 and 6.3.3. Their proofs are also very similar to their counterparts for the average reward criterion.

Lemma 6.4.2 *Let $k \in \{0, \ldots, N-1\}$, let $u \in V$ be any vertex, and let $(u,v) \in E$ be an edge such that $c(u,v) \geq \Gamma^{N-k+1}$. Then:*

$$c(u,v) + \gamma \mathrm{val}_{\pi^k}(v) > \mathrm{val}_{\pi^k}(u)$$

I.e., (u,v) is not an improving switch. Furthermore, if $c(u,\pi(u)) = \Gamma^{N-k}$ and $c(u,v) \leq \Gamma^{N-k-1}$ then:

$$c(u,v) + \gamma \mathrm{val}_{\pi^k}(v) < \mathrm{val}_{\pi^k}(u)$$

Proof: We first assume that $c(u,v) = \Gamma^{N-k+1}$, and let $(\ell', r', t') \in T_n$ be the triple such that $k-1 = N - f(\ell', r', t')$. Then $u = v_{\ell'}^{t'}$ and $v = v_{r'}^{t'}$. Furthermore, $\mathrm{val}_{\pi^k}(v) = \gamma^{r'-\ell'} \mathrm{val}_{\pi^k}(u)$. It follows that:

$$\begin{aligned}
\mathrm{val}_{\pi^k}(u) - \gamma \mathrm{val}_{\pi^k}(v) &= (1 - \gamma^{r'-\ell'+1}) \mathrm{val}_{\pi^k}(u) \\
&\leq \frac{1-\gamma^{r'-\ell'+1}}{1-\gamma} \Gamma^{N-k} \\
&< n\Gamma^{N-k} < \Gamma^{N-k+1} = c(u,v) \,,
\end{aligned}$$

and, hence, (u,v) is not an improving switch.

Next, consider the case when $c(u,v) \geq \Gamma^{N-k+2}$. Let $D = cost(u,k) - \gamma cost(v,k) \leq \Gamma^{N-k-2}$. We have:

$$\begin{aligned}
\mathrm{val}_{\pi^k}(u) - \gamma \mathrm{val}_{\pi^k}(v) &= D + (\gamma^{r(u,k)} - \gamma^{r(v,k)+1}) \mathrm{val}_{\pi^k}(v_\ell^t) \\
&\leq \Gamma^{N-k-2} + \frac{1-\gamma^n}{1-\gamma} \Gamma^{N-k} \\
&< \Gamma^{N-k-2} + n\Gamma^{N-k} \\
&< \Gamma^{N-k+2} \leq c(u,v)
\end{aligned}$$

Next, assume that $c(u,\pi(u)) = \Gamma^{N-k}$ and $c(u,v) \leq \Gamma^{N-k-1}$. In particular, we must have $u = v_\ell^t$. Furthermore, no edge leaving v_ℓ^t has cost Γ^{N-k-1}, so we must have $c(u,v) \leq \Gamma^{N-k-2}$. We get:

$$\begin{aligned}
\mathrm{val}_{\pi^k}(u) - \gamma \mathrm{val}_{\pi^k}(v) &= (1 - \gamma^{r(v,k)+1}) \mathrm{val}_{\pi^k}(v_\ell^t) - \gamma cost(v,k) \\
&\geq \frac{1-\gamma^{r(v,k)+1}}{(1-\gamma)\gamma^{3n}} \Gamma^{N-k-1} - \Gamma^{N-k-2} \\
&> \Gamma^{N-k-1} - \Gamma^{N-k-2} \\
&\geq \Gamma^{N-k-2} \geq c(u,v) \,.
\end{aligned}$$

This completes the proof. □

For every $u \in V$, we again define the edges emanating from u that have not been ruled out by Lemma 6.4.2:

$$W(u,k) = \{v \in V \mid (u,v) \in E \wedge c(u,v) \leq \Gamma^{N-k-1}\} .$$

Lemma 6.4.3 *Let $k \in \{0,\ldots,N-1\}$, let $u \in V$ be any vertex, and let $v_1, v_2 \in W(u,k)$ be two vertices with $steps(v_1,k) > steps(v_2,k)$. Then:*

$$c(u,v_2) + \gamma \mathrm{val}_{\pi^k}(v_2) > c(u,v_1) + \gamma \mathrm{val}_{\pi^k}(v_1) .$$

I.e., (u,v_1) is preferred over (u,v_2).

Proof: We consider two cases. Either $c(u,v_1) \leq \Gamma^{N-k-2}$ or $c(u,v_1) = \Gamma^{N-k-1}$. Let $D = cost(v_1,k) - cost(v_2,k) \leq \Gamma^{N-k-2}$. For the first case we observe that:

$$\begin{aligned}
\gamma(\mathrm{val}_{\pi^k}(v_1) - \mathrm{val}_{\pi^k}(v_2)) &= \gamma D + (\gamma^{r(v_1,k)+1} - \gamma^{r(v_2,k)+1})\mathrm{val}_{\pi^k}(v_\ell^t) \\
&< \Gamma^{N-k-2} - (1-\gamma)\gamma^{3n}\mathrm{val}_{\pi^k}(v_\ell^t) \\
&\leq \Gamma^{N-k-2} - \Gamma^{N-k-1} \\
&\leq -\Gamma^{N-k-2} \leq c(u,v_2) - c(u,v_1) .
\end{aligned}$$

Assume next that $c(u,v_1) = \Gamma^{N-k-1}$. Let $(\ell',r',t') \in T_n$ be the triple such that $k+1 = N - f(\ell',r',t')$. It follows that $u = v_{\ell'}^{t'}$, $v_1 = v_{r'}^{t'}$, and $\mathrm{val}_{\pi^k}(v_1) = \gamma^q \mathrm{val}_{\pi^k}(v_2)$ where $q = steps(v_1,k) - steps(v_2,k)$. Hence, we get:

$$\begin{aligned}
\gamma(\mathrm{val}_{\pi^k}(v_1) - \mathrm{val}_{\pi^k}(v_2)) &= \gamma(\gamma^q - 1)\mathrm{val}_{\pi^k}(v_2) \\
&< -\frac{1-\gamma^q}{(1-\gamma)\gamma^{3n}}\Gamma^{N-k-1} \\
&< -\frac{q}{\gamma^{3n}}\Gamma^{N-k-1} < -\Gamma^{N-k-1} \\
&\leq c(u,v_2) - c(u,v_1) .
\end{aligned}$$

□

Using lemmas 6.4.2 and 6.4.3 we proceed as we did in Section 6.3.2 with lemmas 6.3.2 and 6.3.3. Since the remainder of the proof of Theorem 6.4.1 is essentially the same as it was for the average reward criterion, we do not present it here.

Bibliography

[1] I. Adler, R. Karp, and R. Shamir. A simplex variant solving an m times d linear program in $O(\min(m^2, d^2))$ expected number of pivot steps. *Journal of Complexity*, 3(4):372–387, 1987.

[2] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the ACM*, 32(4):871–895, 1985.

[3] N. Amenta and G. Ziegler. Deformed products and maximal shadows of polytopes. In *Advances in Discrete and Computational Geometry, Amer. Math. Soc., Providence, Contemporary Mathematics 223*, pages 57–90, 1996.

[4] D. Andersson and P. Miltersen. The complexity of solving stochastic games on graphs. In *Proc. of 20th ISAAC*, pages 112–121, 2009.

[5] D. Avis and V. Chvátal. Notes on Bland's pivoting rule. In *Polyhedral Combinatorics*, volume 8 of *Mathematical Programming Studies*, pages 24–34. Springer, 1978.

[6] J. Balogh and R. Pemantle. The Klee-Minty random edge chain moves with linear speed. *Random Structures & Algorithms*, 30(4):464–483, 2007.

[7] R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[8] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[9] D. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, second edition, 2001.

[10] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[11] H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In *STACS*, pages 663–674, 2003.

[12] H. Björklund and S. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.

[13] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

[14] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[15] K. Borgwardt. Probabilistic analysis of the simplex method. In J. Lagarias and M. Todd, editors, *Mathematical Developments Arising from Linear Programming*, Contemporary Mathematics, pages 21–34. American Mathematical Society, 1991.

[16] E. Boros, K. Elbassioni, M. Fouz, V. Gurvich, K. Makino, and B. Manthey. Stochastic mean payoff games: smoothed analysis and approximation schemes. In *Proc. of 38th ICALP*, pages 147–158, 2011.

[17] A. Broder, M. Dyer, A. Frieze, P. Raghavan, and E. Upfal. The worst-case running time of the random simplex algorithm is exponential in the height. *Inf. Process. Lett.*, 56(2):79–81, 1995.

[18] K. L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer, 1960.

[19] V. Chvátal. *Linear Programming*. Series of Books in the Mathematical Sciences. W.H. Freeman, 1983.

[20] B. A. Cipra. The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News*, 33(2), 2000.

[21] K. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995.

[22] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[23] A. Condon. On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–71, 1993.

[24] G. Dantzig. *Linear programming and extensions*. Princeton University Press, 1963.

[25] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5:183–195, 1955.

[26] A. Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Trans. Des. Autom. Electron. Syst.*, 9(4):385–418, 2004.

[27] C. Daskalakis and C. Papadimitriou. Continuous local search. In *Proc. of 22nd SODA*, pages 790–804, 2011.

[28] G. de Ghellinck. Les problemes de decisions sequentielles. *Cahiers Centre Etudes Rech. Operationnelle*, 2:161–179, 1960.

[29] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[30] C. Derman. *Finite state Markov decision processes*. Academic Press, 1972.

[31] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

[32] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Symposium on Foundations of Computer Science, FOCS'91*, pages 368–377. IEEE Computer Society Press, 1991.

[33] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *International Conference on Computer-Aided Verification, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[34] J. Fearnley. Exponential lower bounds for policy iteration. In *Proc. of 37th ICALP*, pages 551–562, 2010.

[35] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[36] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[37] L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[38] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proc. of 24th LICS*, pages 145–156, 2009.

[39] O. Friedmann. *Exponential Lower Bounds for Solving Innitary Payoff Games and Linear Programs*. PhD thesis, Ludwig-Maximilians-University Munich, 2011.

[40] O. Friedmann. A subexponential lower bound for zadeh's pivoting rule for solving linear programs and games. In *Proc. of 15th IPCO*, pages 192–206, 2011.

[41] O. Friedmann, T. D. Hansen, and U. Zwick. A subexponential lower bound for the random facet algorithm for parity games. In *Proc. of 22nd SODA*, pages 202–216, 2011.

[42] O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *Proc. of 43rd STOC*, pages 283–292, 2011.

[43] O. Friedmann and M. Lange. Solving parity games in practice. In *Proc. of 7th ATVA*, pages 182–196, 2009.

[44] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24:1018–1035, 1995.

[45] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002.

[46] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures and Algorithms*, 20(3):353–381, 2002.

[47] B. Gärtner, M. Henk, and G. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, 18(3):349–372, 1998.

[48] B. Gärtner and V. Kaibel. Two new bounds for the random-edge simplex-algorithm. *SIAM J. Discrete Math.*, 21(1):178–190, 2007.

[49] B. Gärtner and L. Rüst. Simple stochastic games and P-matrix generalized linear complementarity problems. In *Proc. of 15th FCT*, pages 209–220, 2005.

[50] B. Gärtner, F. Tschirschnitz, E. Welzl, J. Solymosi, and P. Valtr. One line and n points. *Random Structures & Algorithms*, 23(4):453–471, 2003.

[51] L. Georgiadis, A. Goldberg, R. Tarjan, and R. Werneck. An experimental study of minimum mean cycle algorithms. In *Proc. of 11th ALENEX*, pages 1–13, 2009.

[52] A. Goldberg and R. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.

[53] D. Goldfarb and W. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277 – 285, 1979.

[54] M. Goldwasser. A survey of linear programming in randomized subexponential time. *SIGACT News*, 26(2):96–104, 1995.

[55] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

[56] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.

[57] V. Gurvich, A. Karzanov, and L. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28:85–91, 1988.

[58] N. Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007.

[59] T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. In *Proc. of 2nd ICS*, pages 253–263, 2011.

[60] T. D. Hansen and U. Zwick. Lower bounds for howard's algorithm for finding minimum mean-cost cycles. In *Proc. of 21st ISAAC*, pages 415–426, 2010.

[61] A. Hoffman and R. Karp. On nonterminating stochastic games. *Management Science*, 12:359–370, 1966.

[62] R. Hollanders, J.-C. Delvenne, and R. M. Jungers. The complexity of policy iteration is exponential for discounted markov decision processes. Unpublished manuscript, available at `http://perso.uclouvain.be/romain.hollanders/docs/CDC12_HollandersDelvenneJungers.pdf`, 2012.

[63] R. Howard. *Dynamic programming and Markov processes*. MIT Press, 1960.

[64] R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973.

[65] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[66] M. Jurdziński. Small progress measures for solving parity games. In *Symposium on Theoretical Aspects of Computer Science, STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[67] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

[68] M. Jurdzinski and R. Savani. A simple P-matrix linear complementarity problem for discounted games. In *Proc. of 4th CiE*, pages 283–293, 2008.

[69] G. Kalai. The diameter of graphs of convex polytopes and f-vector theory. *Dimacs Series in Discrete Math.*, 4:387–411, 1991.

[70] G. Kalai. A subexponential randomized simplex algorithm (extended abstract). In *Proc. of 24th STOC*, pages 475–482, 1992.

[71] G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997.

[72] G. Kalai and D. Kleitman. Quasi-polynomial bounds for the diameter of graphs and polyhedra. *Bull. Amer. Math. Soc.*, 26:315–316, 1992.

[73] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[74] R. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

[75] J. Kelner and D. Spielman. A randomized polynomial-time simplex algorithm for linear programming. In *Proc. of 38th STOC*, pages 51–60, 2006.

[76] L. Khachiyan. A polynomial time algorithm for linear programming. *Doklady Akademii Nauk SSSR, n.s.*, 244(5):1093–1096, 1979. English translation in *Soviet Math. Dokl. 20*, 191–194.

[77] T. Kitahara and S. Mizuno. A bound for the number of different basic solutions generated by the simplex method. *Mathematical Programming*, 2011. In press, `http://dx.doi.org/10.1007/s10107-011-0482-y`.

[78] V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities III*, pages 159–175. Academic Press, New York, 1972.

[79] V. Lifschitz and B. Pittel. The number of increasing subsequences of the random permutation. *Journal of Combinatorial Theory, Series A*, 31(1):1–20, 1981.

[80] M. Littman. *Algorithms for sequential decision making*. PhD thesis, Brown University, Department of Computer Science, 1996.

[81] M. Littman, T. Dean, and L. Kaelbling. On the complexity of solving markov decision problems. In *Proc. of the 11th UAI*, pages 394–402, 1995.

[82] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117(1):151–155, 1995.

[83] O. Madani. Personal communication, 2008.

[84] A. S. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, 1960.

[85] Y. Mansour and S. Singh. On the complexity of policy iteration. In *Proc. of the 15th UAI*, pages 401–408, 1999.

[86] J. Matoušek. Lower bounds for a subexponential optimization algorithm. *Random Structures and Algorithms*, 5(4):591–608, 1994.

[87] J. Matoušek and B. Gärtner. *Understanding and using linear programming*. Springer, 2007.

[88] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4-5):498–516, 1996.

[89] J. Matoušek and T. Szabó. RANDOM EDGE can be exponential on abstract cubes. *Advances in Mathematics*, 204(1):262–277, 2006.

[90] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.

[91] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[92] U. Meister and U. Holzbaur. A polynomial time bound for Howard's policy improvement algorithm. *OR Spektrum*, 8:37–40, 1986.

[93] M. Melekopoglou and A. Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing*, 6(2):188–192, 1994.

[94] M. Mitzenmacher and E. Upfal. *Probability and computing, Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[95] K. G. Murty. Computational complexity of parametric linear programming. *Mathematical Programming*, 19:213–219, 1980.

[96] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming*. SIAM, 1994.

[97] A. Neyman and S. Sorin, editors. *Stochastic Games and Applications*, volume 570 of *NATO Science Series C: Mathematical and Physical Sciences*. Springer, 2003.

[98] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[99] V. Petersson and S. Vorobyov. A randomized subexponential algorithm for parity games. *Nord. J. Comput.*, 8(3):324–345, 2001.

[100] A. Puri. *Theory of Hybrid Systems and Discrete Event Simulation*. PhD thesis, University of California, Berkeley, 1995.

[101] M. Puterman. *Markov decision processes*. Wiley, 1994.

[102] S. Rao, R. Chandrasekaran, and K. Nair. Algorithms for discounted games. *Journal of Optimization Theory and Applications*, pages 627–637, 1973.

[103] J. Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical Programming*, 40:59–93, 1988.

[104] L. Rüst. *The P-Matrix Linear Complementarity Problem – Generalizations and Specializations*. PhD thesis, Swiss Federal Institure of Technology, Zürich, 2007.

[105] F. Santos. A counterexample to the hirsch conjecture. *CoRR*, abs/1006.2814v1, 2010.

[106] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. of 17th CSL*, pages 369–384, 2008.

[107] N. Schmitz. How good is Howard's policy improvement algorithm? *Mathematical Methods of Operations Research*, 29:315–316, 1985.

[108] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[109] I. Schurr and T. Szabó. Jumping doesn't help in abstract cubes. In *Proc. of 11th IPCO*, pages 225–235, 2005.

[110] L. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. U.S.A.*, 39:1095–1100, 1953.

[111] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *STACS*, pages 569–579, 1992.

[112] S. Smale. On the average number of steps of the simplex method of linear programming. *Mathematical Programming*, 27(3):241–262, 1983.

[113] D. Spielman and S. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[114] A. Stickney and L. Watson. Digraph models of bard-type algorithms for the linear complementarity problem. *Mathematics of Operations Research*, 3(4):322–333, 1978.

[115] C. Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[116] T. Szabo and E. Welzl. Unique sink orientations of cubes. In *Proc. of 42nd FOCS*, pages 547–555, 2001.

[117] M. Todd. Polynomial expected behavior of a pivoting algorithm for linear complementarity and linear programming problems. *Mathematical Programming*, 35(2):173–192, 1986.

[118] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In *Proc. of 12th CAV*, volume 1855 of *LNCS*, pages 202–215, 2000.

[119] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Wiley-Interscience, 1997.

[120] Y. Ye. A new complexity result on solving the Markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005.

[121] Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Math. Oper. Res.*, 36(4):593–603, 2011.

[122] N. Young, R. Tarjan, and J. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.

[123] N. Zadeh. What is the worst case behavior of the simplex algorithm? Technical Report 27, Department of Operations Research, Stanford, 1980.

[124] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.

[125] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.