Doctoral Thesis

Exponential Lower Bounds for Solving Infinitary Payoff Games and Linear Programs

Oliver Friedmann

Chair of Theoretical Computer Science
Department of Computer Science
Ludwig-Maximilians-University Munich

First advisor:	**Prof. Dr. Martin Hofmann**, University of Munich
Second advisor:	**Prof. Dr. Martin Lange**, University of Kassel
External examiner:	**Prof. Dr. Erich Grädel**, RWTH Aachen

Submitted:	**April 4th, 2011**
Defended:	**July 15th, 2011**

Abstract

Parity games form an intriguing family of infinitary payoff games whose solution is equivalent to the solution of important problems in automatic verification and automata theory. They also form a very natural subclass of *mean and discounted payoff games*, which in turn are very natural subclasses of turn-based *stochastic payoff games*. From a theoretical point of view, solving these games is one of the few problems that belong to the complexity class $\mathtt{NP} \cap \mathtt{coNP}$, and even more interestingly, solving has been shown to belong to $\mathtt{UP} \cap \mathtt{coUP}$, and also to PLS. It is a major open problem whether these game families can be solved in deterministic polynomial time.

Policy iteration is one of the most important algorithmic schemes for solving infinitary payoff games. It is parameterized by an *improvement rule* that determines how to proceed in the iteration from one policy to the next. It is a major open problem whether there is an improvement rule that results in a polynomial time algorithm for solving one of the considered game classes.

Linear programming is one of the most important computational problems studied by researchers in computer science, mathematics and operations research. Perhaps more articles and books are written about linear programming than on all other computational problems combined.

The *simplex* and the *dual-simplex* algorithms are among the most widely used algorithms for solving *linear programs* in practice. Simplex algorithms for solving linear programs are closely related to policy iteration algorithms. Like policy iteration, the simplex algorithm is parameterized by a *pivoting rule* that describes how to proceed from one basic feasible solution in the linear program to the next. It is a major open problem whether there is a pivoting rule that results in a (strongly) polynomial time algorithm for solving linear programs.

We contribute to both the policy iteration and the simplex algorithm by proving exponential lower bounds for several improvement resp. pivoting rules. For every considered improvement rule, we start by building 2-player *parity games* on which the respective policy iteration algorithm performs an exponential number of iterations. We then transform these 2-player games into 1-player *Markov decision processes*

which correspond almost immediately to concrete linear programs on which the respective simplex algorithm requires the same number of iterations. Additionally, we show how to transfer the lower bound results to more expressive game classes like payoff and turn-based stochastic games.

Particularly, we prove exponential lower bounds for the deterministic *switch all* and *switch best* improvement rules for solving games, for which no non-trivial lower bounds have been known since the introduction of Howard's policy iteration algorithm in 1960. Moreover, we prove exponential lower bounds for the two most natural and most studied randomized pivoting rules suggested to date, namely the *random facet* and *random edge* rules for solving games and linear programs, for which no non-trivial lower bounds have been known for several decades. Furthermore, we prove an exponential lower bound for the *switch half* randomized improvement rule for solving games, which is considered to be the most important multi-switching randomized rule. Finally, we prove an exponential lower bound for the most natural and famous history-based pivoting rule due to Zadeh for solving games and linear programs, which has been an open problem for thirty years.

Last but not least, we prove exponential lower bounds for two other classes of algorithms that solve parity games, namely for the *model checking algorithm* due to Stevens and Stirling and for the *recursive algorithm* by Zielonka.

Synopsis

This thesis provides an overview of our results, presenting new lower bounds for algorithms that solve infinitary payoff games as well as new lower bounds for the simplex algorithm for solving linear programs. In particular, it summarizes the main results of the following papers:

(1) O. Friedmann. Recursive Algorithm for Parity Games requires Exponential Time. In *Theoretical Informatics and Applications, Cambridge Journals, 2011*.

(2) O. Friedmann. An Exponential Lower Bound for the latest Deterministic Strategy Iteration Algorithms. In *Logical Methods in Computer Science, Selected Papers of the Conference LICS 2009*.

(3) O. Friedmann, T. Hansen and U. Zwick. Subexponential Lower Bounds for Randomized Pivoting Rules for Solving Linear Programs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC'11*, San Jose, CA, USA, 2011. Winner of the *Best Paper Award*.

(4) O. Friedmann. A Subexponential Lower Bound for Zadeh's Pivoting Rule for Solving Linear Programs and Games. In *Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization, IPCO'11*, New York, NY, USA, 2011. Awarded with *Zadeh's Prize*.

(5) O. Friedmann, T. Hansen and U. Zwick. A Subexponential Lower Bound for the Random Facet Algorithm for Parity Games. In *Proceedings of the Symposium on Discrete Algorithms, SODA'11*, San Francisco, CA, USA, 2011.

(6) O. Friedmann. The Stevens-Stirling-Algorithm for Solving Parity Games Locally Requires Exponential Time. In *International Journal of Foundations of Computer Science, Volume 21, Issue 3*, 2010.

(7) O. Friedmann. An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as we know it. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS'09*, Los Angeles, CA, USA, 2009. Winner of the *Kleene Award 2009*.

Acknowledgments

First of all, I want to thank my parents who have always supported me in every way. Without their help, I would not have had the chance to study for a doctoral degree. Equally, I wish to thank my non-academic friends for taking my numerous monologues on the subject of my thesis with great patience.

Also, I want to thank many colleagues who discussed various topics of this thesis with me and supplied me with very helpful suggestions: Martin Lange, Martin Hofmann, Uri Zwick, Thomas Dueholm Hansen, Markus Latte, Sven Schewe, Colin Stirling, Bernd Gärtner, Emo Welzl, Kousha Etessami, David Avis, Günter Ziegler, anyone I may have forgotten, and the anonymous referees of the publications listed above.

Further thanks go to Max Jakob, our system administrator at the Chair of Theoretical Computer Science in Munich, who always helped me to solve my technical needs with great kindness and who never complained about all the servers that I have crashed by running my compilations on them.

Finally, I want to thank Martin Hofmann and Martin Lange again for agreeing to act as my supervisors, as well as for their outstanding support and guidance during the last years. They always helped me to solve my numerous problems with extraordinary dedication and replied to all my countless email questions straight away providing me with all the help one could hope for.

Declaration

I hereby declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise, and that this work has not been submitted for any other degree or professional qualification.

My own contribution to the co-authored papers listed above can roughly be specified as 40% of (3) and 30% of (5). Three single papers, (3), (4) and (7), have been awarded with prizes.

Contents

1

A brief history of time

The field of theoretical computer sciences touches the disciplines of mathematical logic, automata theory and formal languages, graph theory, complexity theory, game theory, optimization theory, analysis of algorithms, and many more.

In this thesis, we consider *infinitary payoff games*, which are important subjects of algorithmic game theory on the one hand, and have some of its applications in the domain of modal logic and automata theory on the other hand. Additionally, we consider *linear programming*, which is probably one of the most important fields in convex optimization.

We mainly investigate the most important algorithm that solves infinitary payoff games, namely the *policy iteration* method under a complexity theoretical point of view, analyzing its worst-case runtime. Similarly, we investigate the worst-case runtime of the *simplex method* for linear programs.

Infinitary Payoff Games

We consider a variety of closely related classes of games in this thesis, which we like to call *infinitary payoff games*. These are zero-sum, perfect information games played by one or two players, and sometimes by an additional randomized player controlled by nature. The board is a directed, total graph, and each vertex of the graph belongs to one of the players.

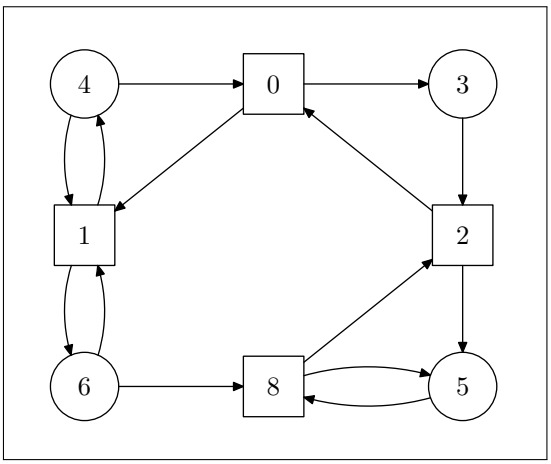
The game is played by putting a single token on one of the vertices (for instance, on a designated starting node), and moving it along an outgoing edge to a successor node. The player, to which the current node belongs to, decides, which outgoing edge

to take. If the current node is owned by the randomized player, then one outgoing edge is picked arbitrarily at random. This process continues ad infinitum, yielding an infinite sequence of nodes. It now depends on the specific class of games to determine, which payoff the players receive or who wins the infinite play.

It is the objective of each player to maximize his or her payoff, or to win against the other player. A player's *strategy* in a game is a plan of action for whatever situation might arise when playing against any opponent. A strategy specifies for each node owned by the player, which respective successor node is to take, and in general, this can depend on the whole *history* of the play up to that stage. If a strategy does not depend on the history, we say that the strategy is *positional*.

All games that we consider here are *positionally determined*, meaning that positional strategies suffice to answer the decision problems associated with the games. This is convenient for many reasons, for instance as the number of positional strategies is finite if the game has finite size.

Parity games are infinitary payoff two-player games played on directed graphs with integer priorities assigned to their vertices. The two players, called *even* and *odd*, construct an infinite path in the game graph. Even wins, if the largest priority that appears an infinite number of times on the path is even. Odd wins otherwise. A parity game might look as follows (circle nodes are owned by the even player):



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 problem of μ-calculus model

checking [EJ91, EJS93, Sti95, GTW02]. It is also at the core of various problems in computer-aided verification, namely validity checking for branching-time logics [FL10b, FLL10] and controller synthesis [VAW03].

Parity games form a very special subclass of *mean payoff games* [Pur95, EM79, GKK88, ZP96], which itself form a subclass of *discounted payoff games*, which in turn form a very special subclass of turn-based *stochastic games* [Con92, AM09], which we will also consider in this thesis. More general *stochastic games* were previously considered by Shapley [Sha53].

Another extremely important class of infinitary payoff "games" are *Markov decision processes*, named after Andrey Markov, providing a mathematical model for sequential decision making under uncertainty. The study of Markov decision processes started with the seminal work of Bellman [Bel57]. It can be seen as a special subclass of turn-based stochastic games in which only one player is really used. Markov decision processes have many applications in practice, for instance in robotics, automated control, economics, operations research and artificial intelligence.

Parity and related, more expressive game classes like payoff and stochastic games, are a very interesting subject on their own from a complexity theoretical point of view. While it is known that the decision problems corresponding to these games belong to $\mathtt{NP} \cap \mathtt{coNP}$ [EJS93, Pur95], and even to $\mathtt{UP} \cap \mathtt{coUP}$ [Jur98, ZP96], as well as to $\mathtt{PLS}$ [BM08], it is a major open problem whether any of these game families can be solved in polynomial time. Markov decision processes, on the other hand, *can* be solved in polynomial time by special techniques obtained from the domain of linear programming.

A variety of algorithms for solving parity games has been invented so far. The most prominent deterministic ones are the recursive algorithm by Zielonka [Zie98], the local μ-calculus model checker by Stevens and Stirling [SS98], Jurdziński's small progress measures algorithm [Jur00], the subexponential algorithm by Jurdziński, Paterson and Zwick [JPZ06] with a so-called big-step variant by Schewe [Sch07], as well as several variations of the policy iteration technique (see below), which is the only method that also applies to the other game classes.

This variety is owed to the theoretical challenge of answering the question whether parity (or any of these) games can be solved in polynomial time, rather than practical motivations. The currently best known upper bound on the deterministic solution of parity games is $\mathcal{O}(e \cdot n^{\frac{1}{3}p})$ due to Schewe's big-step algorithm [Sch07], where e is the number of edges, n is the number of nodes and p is the number of different priorities in the game.

Policy Iteration

The *strategy improvement*, *strategy iteration* or *policy iteration* technique is the most general approach that can be applied as a solving procedure for infinitary payoff games and related problems. It was introduced by Howard [How60] in 1960 for solving problems on Markov decision processes, and has been adapted by Hoffman and Karp in 1966 for solving nonterminating stochastic games [HK66]. Later, Condon adapted the algorithm for solving turn-based stochastic games [Con92], and Puri, Zwick and Paterson used the method to solve discounted and mean payoff games [Pur95, ZP96]. Finally, Jurdziński and Vöge formulated a *discrete* variant of the policy iteration algorithm for solving parity games [VJ00].

The beauty of the policy iteration technique lies in its simplicity. It is based on a (fixpoint-)iteration over a special finite subclass of strategies of the first player. In each iteration, the current strategy is mapped to a *valuation*. The valuation of a strategy allows us to decide *whether* the strategy is *optimal* for the first player, and if not, *how* we can improve the strategy to obtain a *better* one. An appealing feature is that we can compute valuations efficiently. In order to find an optimal strategy, which allows us to derive a solution for the game, we apply the following scheme, starting with an arbitrary strategy σ:

Algorithm 1 Policy Iteration

1: **while** σ is not optimal **do**
2: $\quad \sigma \leftarrow \text{Improve}(\sigma)$
3: **end while**

Policy iteration in fact describes a whole class of algorithms, as in general, there is more than one candidate strategy to proceed with in each iteration. The method

of choosing successor policies is called *improvement rule*. Under the assumption that we only consider efficient improvement rules, it follows that the computational complexity of policy iteration essentially only depends on the number of iterations, since a single iteration is carried out in deterministic polynomial time.

This leaves us with the question whether every improvement rule leads to a small number of iterations. Not very surprisingly, this is not the case. An example has been known for some time for which a sufficiently poor choice of a deterministic improvement rule causes an exponential number of iterations [BV07]. Then, we could ask whether it is even theoretically possible to obtain an improvement rule that results in a small number of iterations. Here, the answer is yes, and the easy proof is folklore. However, the proof does not reveal any insights on how to formulate such an improvement rule. Or putting it differently, the improvement rule that we could get from the proof is not efficiently computable itself.

A variety of improvement rules has been invented so far, which is probably owed to the theoretical challenge of finding an efficient policy iteration algorithm. Generally, there are *deterministic*, *randomized* and *memorizing* improvement rules. The most important ones, that are mentioned in the literature, are the deterministic SWITCH-ALL [VJ00] and SWITCH-BEST [Sch08], the randomized SWITCH-HALF [MS99], RANDOM-FACET [Kal92, Kal97, MSW96] and RANDOM-EDGE (folklore), and the memorizing LEAST-ENTERED [Zad80] rules. No non-trivial lower bounds on the worst-case complexity of all of these rules have been known until now.

As we will see, policy iteration is moreover closely related to an algorithm called *simplex method* for solving linear programs.

Linear Programming

Linear programming is one of the most important fields of optimization theory, and is very actively researched. Many economical and practical tasks can be expressed as a linear programming instance, and several subgoals of other optimization problems in computer science require linear programs to be solved.

The linear programming problem is to maximize (or minimize) a given *linear objective function* while satisfying some additional linear equalities and inequalities. Formally, a linear programming problem is to *maximize* an objective function

$$c_1 x_1 + \ldots + c_n x_n$$

subject to a number of linear (in)equalities, called *constraints*,

$$\begin{aligned} a_{1,1} x_1 + \ldots + a_{1,n} x_n &= b_1 \\ a_{2,1} x_1 + \ldots + a_{2,n} x_n &= b_2 \\ &\vdots \\ a_{m,1} x_1 + \ldots + a_{m,n} x_n &= b_m \end{aligned}$$

where all *coefficients* c_i, all b_i, as well as all coefficients $a_{i,j}$ are real numbers.

We will see that Markov decision processes can be formulated as linear programs, which, in fact, is also the reason why we can solve this class of games in polynomial time.

There are, essentially, three kinds of algorithms that solve linear programs. First, there is the *simplex algorithm*, which has been proposed by Dantzig [Dan63] in 1963. The exact complexity of this algorithm is unknown, and it is a major open problem to answer this question adequately. The other two algorithms for solving linear programs, namely the *ellipsoid method* by Khachiyan [Kha79] and the *interior-point method* by N. Karmarkar [Kar84], handle linear programs in polynomial time, however not in *strongly* polynomial time.

The difference between *strongly* and "normal" or *weakly* polynomial time is subtle, and it depends on the method that we apply for *measuring* the size of a given linear program. Clearly, the number of variables and the number of constraints should contribute *linearly* to the size. But how do we measure the coefficients? Classically, their magnitude and precession would also contribute to the size, as we would need to find an *encoding* of the involved coefficients. If the runtime of an algorithm can be polynomially bounded in terms of this measure, we would speak

of a weakly polynomial-time algorithm. On the other hand, if the runtime can be polynomially bounded by a measure that *only* depends on the number of variables and constraints, we would speak of strongly polynomial-time algorithm.

Although it is not even clear, whether (a variant of) the simplex method is a polynomial-time algorithm, it has the potential to be even a strongly polynomial-time algorithm, which is the reason why many people still consider this method.

Simplex Algorithm

The *simplex algorithm* is based on the observation that the space of points satisfying all constraints essentially is a convex polytope (disregarding some special cases), and the objective function has an optimal value on one of its vertices. Hence, the idea is to start with an arbitrary vertex, to check whether the objective function is optimal on that vertex, and if not, to improve to some adjacent vertex.

Similar to the policy iteration algorithm, the simplex method is parameterized by a *pivoting rule* that selects one of the eligible neighboring vertices. Again, the complexity of the simplex algorithm essentially only depends on the number of *pivoting steps*, i.e. on the number of visited vertices, since all other necessary operations can be performed in (strongly) polynomial time.

The question whether every pivoting rule results in a small number of pivoting steps has been refuted by Klee and Minty [KM72], shortly after Dantzig presented the simplex algorithm. But unlike policy iteration, it is a major open problem, whether it is even theoretically possible to have a small number of pivoting steps. This is known as the *Hirsch conjecture* (see e.g. [Dan63], pp. 160,168).

Many of the improvement rules for policy iteration can be phrased as pivoting rules for the simplex algorithm (and vice versa), as most of them are formulated abstractly enough. As for policy iteration, no non-trivial lower bounds on the worst-case complexity of many of these rules in the context of the simplex algorithm have been known until now.

This leaves us with the question whether there is a deeper connection between the policy iteration algorithm for infinitary payoff games and the simplex algorithm for linear programs. Indeed, we will see that Markov decision processes are the “missing link” that relates policy iteration and the simplex method in some meaningful way.

Our Contribution

We present *exponential* (i.e. of the form $2^{\Omega(n)}$) and *subexponential* (i.e. of the form $2^{\Omega(n^c)}$ for some $0<c<1$) lower bounds for essentially all policy iteration improvement rules and all simplex method pivoting rules with unresolved complexity status.

First, we consider parity game policy iteration, and construct explicit families of parity games on which the different improvement rules require exponential resp. subexponential time. It was hoped until now, that any of those would solve parity games in polynomial time.

Second, we show that all lower bound results obtained for parity game policy iteration can be transferred to more expressive game classes like mean and discounted payoff games, as well as turn-based stochastic games.

Third, we describe how our parity game constructions can be reshaped as Markov decision processes with the same implications for the policy iteration algorithm. This is not known to be possible in general, but we are able to at least translate our constructions.

Fourth, we formalize the relation between policy iteration for Markov decision processes and the simplex algorithm for induced linear programs, which allows us to transfer all applicable lower bound results to the domain of linear programming, solving problems that have been open for several decades. Note, however, that these results do not have any implications on the Hirsch conjecture.

Fifth, we show exponential lower bounds for the model checking algorithm due to Stevens and Stirling and for the recursive algorithm by Zielonka for solving parity games.

Outline

The thesis is organized as follows. In Chapter 2, we introduce the fundamentals of complexity theory and linear programming to fix our notation, and to provide the reader with the necessary background knowledge to follow our presentation.

In Chapter 3, we describe infinitary payoff games, their decision problems, important relations and reductions between the different game classes, and some notable observations.

In Chapter 4, we consider the policy iteration method for solving infinitary payoff games, and present our lower bound constructions. We also describe the relation to the simplex algorithm for linear programming.

For the sake of completeness, we describe the model checking algorithm and the recursive algorithm for solving parity games in Chapter 5, and prove exponential lower bounds.

We end in Chapter 6 with some concluding remarks and open problems. Some of the tedious proofs of Chapter 4 and Chapter 5 have been put into the appendix.

2
Preliminaries

We present the necessary mathematical foundations in this chapter that are required to follow the thesis.

First of all, we introduce the relevant part of complexity theory, one of the major fields of theoretical computer science, that tries to quantify the amount of computational resources – like time or space – that are required to execute a specific algorithm or to solve a specific task without explicitly stating how the task is to be solved. For instance, the well-known *bubble sort* algorithm is a procedure to sort a stack of cards, the general problem of sorting cards itself is a specific task that can be analyzed by complexity theory.

Then, we introduce the field of linear programming, which is a very important part of convex optimization. It subsumes all problems that can be expressed by maximization (or minimization) of a given *linear objective function* subject to linear constraints in the form of linear equalities and linear inequalities. In this thesis, we consider linear programs for two reasons. First, some of the policy iteration methods are based on the solution of related linear programs, and second, we will show that our lower bounds for infinitary payoff games can be transferred to lower bounds for one of the most important algorithms for solving linear programs, the *simplex algorithm*.

Some notation that we use in this thesis might not be common sense, therefore we explain all non-standard terms in the following once and for all.

Notation

For a binary relation $R \subseteq A \times B$ we write xRy as an abbreviation for $(x, y) \in R$, xR for $\{y \mid xRy\}$ and Ry for $\{x \mid xRy\}$.

Let $f : A \to B$ be a function. We call $\texttt{dom}(f) = \{x \in A \mid \exists y \in B.f(x) = y\}$ the *domain of* f and $\texttt{ran}(f) = \{y \in B \mid \exists x \in A.f(x) = y\}$ the *range of* f. By the *image of* $C \subseteq A$ *under* f, $f[C]$, we refer to the set $\{y \in B \mid \exists x \in C.f(x) = y\}$, and by the *inverse image of* $C \subseteq B$ *under* f, $f^{-1}[C]$, we refer to the set $\{x \in A \mid \exists y \in C.f(x) = y\}$.

Given a subset $C \subseteq A$, we denote the *restriction of* f *to* C by $f|_C$, being defined as follows:

$$f|_C : C \to B, \quad x \in C \mapsto f(x)$$

Given a binary relation $R \subseteq A \times B$ s.t. for every xRy and xRz it follows that $y = z$, we define the f*-update to* R, $f[R]$, by

$$f[R] : A \to B, \quad x \mapsto \begin{cases} f(x) & \text{if } xR = \emptyset \\ y & \text{if } xR = \{y\} \end{cases}$$

Note that the f-update to R and the image of C under f use the same notation. However, it will always be clear from the respective context to which operation we refer to. For a finite number of pairs $(x_1, y_1), \ldots, (x_n, y_n)$ with pairwise distinct x_i, we also apply the following notation for the f-update.

$$f[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n] : x \mapsto \begin{cases} y_i & \text{if } x = x_i \\ f(x) & \text{otherwise} \end{cases}$$

Let $k > 0$ be a natural number. We abbreviate that two natural numbers a and b are equal modulo k by the following equivalence relation.

$$a \equiv_k b \qquad \Longleftrightarrow \qquad (a \mod k) = (b \mod k)$$

Landau Symbols

The *Landau symbols* are a well-established way of specifying bounds on the asymptotic behavior of real-valued functions. The main motivation to use this notation in the analysis of algorithms or algorithmic problems is to capture the essential resource

consumption – in terms of time or space – of an algorithm without drawing too much attention to specific implementation details and their effects on the overall resource consumption.

This simplification is particularly useful as it directly corresponds to the idea of abstracting away from a specific machine. Instead of considering a concrete machine that executes some computational primitives with well-known resource consumptions, one applies a more general computational model that subsumes all specific machines that differ in their primitive resource consumptions by a constant factor.

Let now $f, g : \mathbb{R} \to \mathbb{R}$ be two functions. We say that f *is in* $\mathcal{O}(g)$, in terms $f \in \mathcal{O}(g)$ or $f = \mathcal{O}(g)$, iff there is some $x_0 \in \mathbb{R}$ and some $c > 0$ s.t. $|f(x)| \leq c \cdot |g(x)|$ for all $x > x_0$. Intuitively, $f \in \mathcal{O}(g)$ means that f is asymptotically bounded above by g (up to a constant factor). Disregarding c, we consider g to be an upper bound on f. For instance, $2 \cdot x^3 + 5 \cdot x^2 + 7 \in \mathcal{O}(x^3)$ as $\lim_{x\to\infty} \frac{2\cdot x^3+5\cdot x^2+7}{x^3} = 2$ and $10 \cdot x^2 \in \mathcal{O}(x^3)$ as $\lim_{x\to\infty} \frac{10\cdot x^2}{x^3} = 0$, but $x^4 \notin \mathcal{O}(x^3)$ as $\lim_{x\to\infty} \frac{x^4}{x^3} = \infty$

The corresponding lower bound is denoted as follows: We say that f *is in* $\Omega(g)$, in terms $f \in \Omega(g)$ or $f = \Omega(g)$, iff there is some $x_0 \in \mathbb{R}$ and some $c > 0$ s.t. $|f(x)| \geq c\cdot|g(x)|$ for all $x > x_0$. Intuitively, $f \in \Omega(g)$ means that f is asymptotically bounded below by g (up to a constant factor). It is not hard to see that $f \in \mathcal{O}(g)$ iff $g \in \Omega(f)$.

Binary Counting

Almost all lower bounds constructions of this thesis are ultimately based on binary counting. We introduce notation to succinctly describe binary numbers. It will be convenient for us to consider counter configurations with an *infinite* tape, where unused bits are zero. The set of n-bit configurations is formally defined as $\mathcal{B}_n = \{\mathfrak{b} \in \{0,1\}^\infty \mid \forall i > n : \mathfrak{b}_i = 0\}$.

We start with index one, i.e. $\mathfrak{b} \in \mathcal{B}_n$ is essentially a tuple $(\mathfrak{b}_n, \ldots, \mathfrak{b}_1)$, with $\mathfrak{b}_1$ being the least and $\mathfrak{b}_n$ being the most significant bit. By $\mathbf{0}$, we denote the configuration in which all bits are zero, and by $\mathbf{1}_n$, we denote the configuration in which the first n bits are one. We write $\mathcal{B} = \bigcup_{n>0} \mathcal{B}_n$ to denote the set of all counter configurations.

The *integer value* of a $\mathfrak{b} \in \mathcal{B}$ is defined as usual, i.e. $|\mathfrak{b}| := \sum_{i>0} \mathfrak{b}_i \cdot 2^{i-1} < \infty$. For two configurations $\mathfrak{b}, \mathfrak{b}' \in \mathcal{B}$, we induce the lexicographic linear ordering $\mathfrak{b} < \mathfrak{b}'$ by $|\mathfrak{b}| < |\mathfrak{b}'|$. It is well-known that $\mathfrak{b} \in \mathcal{B} \mapsto |\mathfrak{b}| \in \mathbb{N}$ is a bijection. For $\mathfrak{b} \in \mathcal{B}$ and $k \in \mathbb{N}$ let $\mathfrak{b} + k$ denote the unique $\mathfrak{b}'$ s.t. $|\mathfrak{b}'| = |\mathfrak{b}| + k$. If $k \leq |\mathfrak{b}|$, let $\mathfrak{b} - k$ denote the unique $\mathfrak{b}'$ s.t. $|\mathfrak{b}'| + k = |\mathfrak{b}|$.

Given a configuration $\mathfrak{b}$, we access the *i-next set bit* by $\nu_i^n(\mathfrak{b}) = \min(\{n+1\} \cup \{j \geq i \mid \mathfrak{b}_j = 1\})$, and the *$i$-next unset bit* by $\mu_i(\mathfrak{b}) = \min\{j \geq i \mid \mathfrak{b}_j = 0\}$.

2.1 Complexity Theory

Complexity theory is the field of theoretical computer science that tries to classify algorithmic problems according to their computational needs. It formalizes the idea of abstract machines that execute algorithms, and defines reasonable technicalities to specify the intrinsic computational requirements of an abstract machine that solves an algorithmic problem. These needs are phrased in terms of the asymptotic usage of memory space or the asymptotic number of steps that are required to solve the problem.

Both *logic* and complexity theory deal with algorithmic problems. A typical question in logic is whether a given problem is *decidable*, i.e. whether there is an algorithm that solves the problem. Although the decidability question plays a role in complexity theory as well, the major concern of complexity theory is to answer the question how *fast* a decidable problem can be solved. We only consider decidable problems here.

An *algorithmic problem* is the formal description of the relation between some input data describing the *problem instances* and some output data describing a *solution* to the given instance. For example, "sorting a list of natural numbers" could be seen as an algorithmic problem in which the set of problem instances is the set of lists of natural numbers and the corresponding solution of a given list of natural numbers would be the respective ordered version. The description of an algorithmic problem is purely theoretical and does neither favor nor specify a concrete algorithm that *solves* the problem.

One of the most important tasks of complexity theory is to assign different algorithmic problems that share common computational properties to so-called *complexity classes*. A complexity class usually specifies a *computational model*, a *computational paradigm*, and some *resource bounds*.

The *computational model* describes the abstract machine on which algorithms, based on a finite set of fundamental operations, can be executed by having access to some kind of memory. There are several reasonable computational models that can be applied when studying algorithmic problems from a theoretical point of view. The most commonly used model is, probably, the class of *Turing machines* (TM), named after its inventor Alan Turing. A Turing machine is a theoretical automaton that fulfills state transitions ranging over a finite number of states with the additional ability to have read and write access to a constant number of potentially unbounded tapes.

The *computational paradigm* explains how the execution of an algorithm is to be performed. We consider the following two paradigms:

- *deterministic paradigm*: the current computational state has at most one successive state, and
- *non-deterministic paradigm*: the current computational state can have more than one successive state.

Applying an algorithm (i.e. a Turing machine) to an input instance, results in a *run*, which is the complete trace of atomic steps that the abstract machine performed on the input instance to generate an output instance. Note that a deterministic algorithm has exactly one run on a given input instance, whereas a non-deterministic algorithm can have more than one run on a given input instance.

Along with the output instance, the Turing machine has to specify whether the run is *accepting* or *rejecting*. This is particularly important when applying a non-deterministic Turing machine to an input instance, because here it is possible that some runs are accepting and others are not. The output instance is only considered to be valid, if the corresponding run is accepting. Note that many problems are phrased as *decision problems* in which we do not expect any output instances.

Complexity theory considers the asymptotic *resource consumption* that is necessary to solve algorithmic problems. The most interesting resources are *time* and *space*. We only consider time in this thesis. Given a run r of a TM T, we let $\texttt{time}_T(r)$ denote the length of r. It is here that Landau symbols are particularly useful. Instead of presenting detailed resource-bounding functions f along with an algorithm that solves the respective problem, we usually apply the $\mathcal{O}(*)$-notation to specify asymptotic resource-bounds.

The analysis of some resource complexity is ultimately based on relating the *size* of an input instance to the resource consumption of an associated run. When considering concrete algorithmic problems like sorting cards or solving the *traveling salesman problem*, one usually leaves the details of the encoding of the problem implicit. This is reasonable most of the time, however, one needs to be careful in some occasions. Natural numbers, for instance, can be encoded as *unary numbers*, i.e. the encoding size of a number directly coincides with the number itself, or they can be encoded as *binary numbers*, i.e. the encoding size of a number is *logarithmic* in the number itself. Obviously, these two encodings result in quite different sizes of input instances. Hence, whenever implicit encodings raise such ambiguities, one has to be more specific about the respective encoding details.

Complexity theory investigates two kinds of questions. First, given a concrete algorithm that solves a concrete algorithmic problem, what are upper and lower bounds on the algorithmic complexity of the algorithm? Such bounds are usually expressed in terms of Landau symbols. The typical strategy to give a lower bound proof is to exhibit a family of input instances of increasing size that results in runs that can be bounded by the desired lower bound function. Upper bound proofs usually rely on some combinatorial analysis.

Second, given a concrete algorithmic problem, what are upper and lower bounds on the complexity of the algorithmic problem? Upper bounds are usually given by the construction of a concrete algorithm along with an upper bound on it. Lower bounds on an algorithmic problem again rely on combinatorial analysis most of the time.

Complexity Classes

Complexity theory is mainly concerned with the collection of algorithmic problems with common resource bounds, forming a *complexity class*. We assume some familiarity with the relationships between the important complexity classes NP and P, and briefly recap their informal definitions in the following:

- P: the class of all decision problems that can be solved by a polynomial-time bounded deterministic Turing machine,
- NP: the class of all decision problems that can be solved by a polynomial-time bounded non-deterministic Turing machine, that accepts an input iff it has an accepting run on it,
- coNP: the class of all decision problems that can be solved by a polynomial-time bounded non-deterministic Turing machine, that accepts an input, iff it has no rejecting run on it,
- UP: the class of all decision problems that can be solved by a polynomial-time bounded non-deterministic Turing machine, that accepts an input iff it has an accepting run on it, and that has no more than one accepting run on every input instance, and
- coUP: the class of all decision problems that can be solved by a polynomial-time bounded non-deterministic Turing machine, that accepts an input, iff it has no rejecting run on it, and that has no more than one rejecting run on every input instance.

Deterministic polynomial-time bounded algorithms are generally considered as being efficient. However, it is debatable whether an algorithm that runs in time $\Omega(n^{100})$ is really efficient in practice.

An alternative definition of NP-problems is known as the "guess and check" characterization: a problem is in NP iff given an input instance x and a potential solution y, it is easy (that is, in deterministic polynomial time) to verify whether the solution is actually correct.

It is a major open problem whether $\mathtt{NP} = \mathtt{P}$. The predominant opinion of the scientific community is that $\mathtt{P} \subsetneq \mathtt{NP}$. In fact, many theorems of nowadays complexity theory are based on the *assumption* that $\mathtt{NP} \neq \mathtt{P}$. Assuming this common hypothesis, it can be shown that there are problems in NP that are neither NP-complete nor contained in P [Lad75]. However, there are no known *natural* NP-problems that can be shown to reside between NP-complete and P problems, assuming that $\mathtt{NP} \neq \mathtt{P}$.

The main problems of this thesis – deciding the winner of certain infinitary payoff games (excluding Markov decision processes) – are some of the very rare natural combinatorial problems that are known to be in NP, coNP, UP and coUP, but that are not yet known to be in P. Due to the fact that these problems are contained in the formerly mentioned complexity classes, it is generally considered to be quite unlikely that they are complete for any of those. In fact, most scientists believe that these problems are actually contained in P. However, this question is still open.

Arithmetic Model

A very important computational model is the *arithmetic model* which allows to perform basic arithmetic operations like addition or multiplication in unit time, regardless of the size of the operands. This relates quite naturally to computers of our real life, since arithmetic operations are performed in essentially unit time by our CPUs.

We say that an algorithm runs in *strongly polynomial time* iff the time complexity is polynomially bounded by the *number of integers* of the input instance. Such algorithms can be easily converted into "normal" polynomial-time algorithms by replacing the unit time arithmetic operations by polynomial-time implementations.

An algorithm runs in *weakly polynomial time* on the other hand, iff it runs in polynomial time, but not in strongly polynomial time. Therefore, the runtime of the algorithm really depends on the integers rather than only on the number of integers.

In the field of combinatorial optimization, like linear programming and infinitary payoff game theory, strongly polynomial-time algorithms are generally desirable, as they only depend on the *combinatorial structure* of the problem rather than on complex arithmetic phenomena.

The Search Class PLS

Some algorithmic problems can be phrased as *search problems*, in which the algorithm starts with some initial solution to the given input instance, proceeds to some neighboring solution, until an optimal solution has been found. In general, this is a locally optimal solution. Papadimitriou, Yannakakis and Johnson [JPY88, PY88] were the first to define a rigorous complexity class that captures these kinds of problems.

Again, the complexity class of local search problems is defined w.r.t. input instances as words over a finite alphabet, so that we can easily measure the size of a given input instance. We leave the alphabets and lengths of words implicit here and just assume that we can assign a size to an input instance in a reasonable way.

More formally, a *polynomial local search problem (*PLS *problem)* is a tuple (D, S, F, N, c), where

- D is the set of *instances*,
- S is the set of *solutions*,
- $F : D \to 2^S$ is a function that assigns every instance a non-empty, finite set of solutions,
- $N : D \times S \to 2^S$ is a function that assigns every instance x and every solution $s \in F(x)$ a *neighborhood* $N(x, s) \subseteq F(x)$, and
- $c : D \times S \to \mathbb{R}$ is a function that assigns every instance x and every solution $s \in F(x)$ some *cost* $c(x, s)$

such that there are deterministic polynomial-time (in the size of the instance) algorithms for the following tasks, given an instance $x \in D$:

- the selection of an initial $s \in F(x)$,
- the computation of $c(x, s)$ for $s \in F(x)$,
- the decision, whether there is some $s' \in N(x, s)$ with $c(x, s') > c(x, s)$ for $s \in F(x)$, and the computation of such an s' in case it exists.

The goal is to compute, given an instance x, a solution $s \in F(x)$ with *locally maximal cost*, i.e. there is no $s' \in N(x, s)$ with $c(x, s') > c(x, s)$. The standard method for computing such a solution can be realized by Algorithm 2.

Algorithm 2 Polynomial Local Search

1: $s \leftarrow$ some element of $F(x)$
2: **while** there is some $s' \in N(x, s)$ with $c(x, s') > c(x, s)$ **do**
3: $\quad s \leftarrow s'$, where $s' \in N(x, s)$ with $c(x, s') > c(x, s)$
4: **end while**
5: **return** s

An appealing feature of polynomial local search problems is that they can be approximated to any factor in polynomial time (polynomial in the instance size and the approximation factor) [OPS04].

2.2 Linear Programming

Linear programming (LP) is one of the most important computational problems studied by researchers in computer science, mathematics and operations research. Though our understanding of linear programming improved vastly in the last 60 years, there are still many extremely intriguing and important open problems. Dozens of books and thousands of articles were written on linear programming.

The theorems that we present in this chapter are considered to be common knowledge, and cannot be attributed to a single author or a single publication. They can be found in any standard literature on linear programming, see, e.g., Chvátal [Chv83], Schrijver [Sch86], Matoušek and Gärtner [MG07], and Srinivasan [Sri08], and the references there in.

The linear programming problem is to maximize (or minimize) a given *linear objective function* subject to linear constraints in the form of linear equalities and linear inequalities.

More formally, the *standard form* is to *maximize* a linear objective function

$$f : \mathbb{R}_+^n \to \mathbb{R},\ (x_1, \ldots, x_n) \mapsto c_1 x_1 + \ldots + c_n x_n$$

subject to a number of linear (in)equalities, called *constraints*,

$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + \ldots + a_{2,n}x_n &= b_2 \\ &\vdots \\ a_{m,1}x_1 + \ldots + a_{m,n}x_n &= b_m \end{aligned}$$

where all $c_i \in \mathbb{R}$, $b_i \in \mathbb{R}$, as well as all $a_{i,j} \in \mathbb{R}$.

The most commonly used notation, however, expresses the problem in terms of matrix multiplications, i.e. an LP problem is to maximize $c^T x$ subject to $Ax = b$ where $x \in \mathbb{R}^n_+$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. We write $\mathtt{LP}_{\mathtt{max}}(c, A, b)$ to refer to the problem.

Given a linear program $L = \mathtt{LP}_{\mathtt{max}}(c, A, b)$ in standard form, we say that an $x \in \mathbb{R}^n_+$ is a *feasible solution* iff $Ax = b$. The *feasible region* of L is the set $P_L = \{x \in \mathbb{R}^n_+ \mid Ax = b\}$ of feasible solutions. We say that an LP L is *feasible* iff $P_L \neq \emptyset$, otherwise it is *infeasible*. A feasible solution $x^* \in P_L$ is *optimal* iff for all $x \in P_L$ we have: $c^T x^* \geq c^T x$. We say that an LP is *unbounded* iff for every $\lambda \in \mathbb{R}$, we have a feasible $x \in P_L$ s.t. $c^T x \geq \lambda$, otherwise it is *bounded*.

The following is sometimes known as the *weak fundamental theorem of linear programming*.

Theorem 2.1. *A linear programming problem is either* unbounded*,* infeasible *or* feasible *and* bounded*.*

It is easy to see that an optimal solution only exists, if the linear programming problem is feasible and bounded. *Solving* a linear programming problem means to decide whether it is unbounded, infeasible, or feasible and bounded, and to compute an optimal solution in the latter case.

Note that when *computing* an optimal solution, we assume that all real values are rationals. In fact, we can even assume that all values are integers, as the LP problem is equivalent to the problem in which every value has been multiplied by the lowest

common denominator of all values. The result for the modified problem can then be back-transformed to the original one by dividing by the lowest common denominator again.

There are several equivalent forms of linear programming problems that can be reduced to the original problem.

1. *Minimization* objectives $\min c^T x$ can be expressed as maximization objectives $\max -c^T x$.

2. Equality constraints – as in the standard form – can be expressed by two inequalities.

3. Inequality constraints $a_i^T x \leq b_i$ can be expressed by introducing so-called *slack variables* $s_i \in \mathbb{R}_+$ and replacing the inequality by the equation $a_i^T x + s_i = b_i$.

4. Non-positivity constrained variables $x_i \leq 0$ can be replaced by the term $-x_i'$ and the new variable $x_i' \geq 0$.

5. Unrestricted variables x_i can be replaced by the term $x_i = x_i^+ - x_i^-$ and two additional variables $x_i^+, x_i^- \geq 0$.

Geometry

The geometric interpretation of the linear constraints, i.e. the feasible region, is the space described by a convex polytope. Every linear objective function is both concave and convex, hence every local minimum resp. maximum is also a global one. Therefore, the solution of every LP problem either is the uniquely determined global maximum of the linear objective function in the feasible region or not existing, i.e. either unbounded or the whole polytope is infeasible.

Consider, for instance, the following LP problem.

$$\text{Maximize } 6x_1 + 5x_2 \text{ subject to } \begin{cases} x_1 + x_2 & \leq 5 \\ 3x_1 + 2x_2 & \leq 12 \end{cases}$$

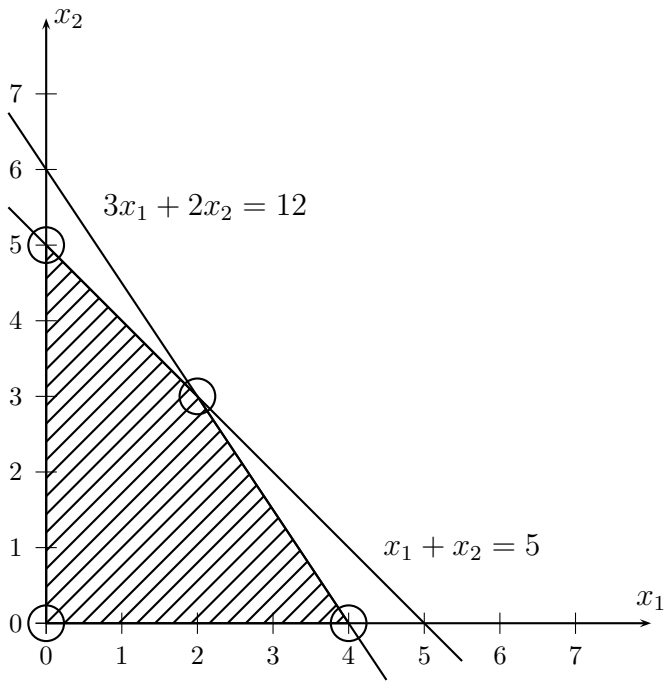


Figure 2.1: Geometry of the example

The optimal solution is $(x_1, x_2) = (2, 3)$ with value 27. See Figure 2.1 for the geometry of the LP problem. The encircled points mark the vertices of the polytope.

Formally, we say that $x \in P_L$ is a *vertex* of the polytope P_L iff for every vector $y \neq 0$, we have: $x + y \notin P_L$ or $x - y \notin P_L$. In other words, a vertex is a point of the polytope s.t. there is no proper line with x as the center that is completely included in the polytope.

Theorem 2.2. *Let $L = \mathtt{LP}_{\mathtt{max}}(c, A, b)$ be a bounded linear programming problem, and let $x \in P_L$. There is a vertex $x^* \in P_L$ s.t. $c^T x^* \geq c^T x$.*

Theorem 2.2 has the important consequence that there is always an optimal solution that corresponds to a vertex of the polytope. Note, however, that there may be optimal solutions that do not correspond to a vertex.

Corollary 2.3. *Every bounded and feasible linear programming problem has a vertex that is an optimal feasible solution.*

It is also not hard to see that we can always find a vertex in a linear program by minimizing a canonic linear objective function that is monotone in every variable of the program.

Corollary 2.4. *Every feasible linear program has a vertex.*

We need to introduce some notation to refer to submatrices. Let $A \in \mathbb{R}^{m \times n}$ be a matrix and $B = \{b_1, \ldots, b_k\}$ be a subset of column-indices $1 \leq b_1 < \ldots < b_k \leq n$. By $A|_B \in \mathbb{R}^{m \times k}$, we denote the submatrix of A that is restricted to the columns with indices in B, i.e. $(A|_B)_{i,j} = A_{i,b_j}$. For vectors $x \in \mathbb{R}^n$, we apply the same notation, i.e. $x|_B \in \mathbb{R}^k$ with $(x|_B)_j = x_{b_j}$.

Our next theorem describes the whole set of vertices in terms of the constraint matrix A. For a given vector $x \in \mathbb{R}^n$, let $B(x) = \{j \mid x_j > 0\}$.

Theorem 2.5. *Let $L = \mathtt{LP}_{\mathtt{max}}(c, A, b)$ be a linear programming problem and $x \in P_L$. Then x is a vertex iff $rank(A|_{B(x)}) = |B(x)|$, in other words iff $A|_{B(x)}$ has linearly independent columns.*

Bases

By Theorem 2.2, we know that the geometric characterizations of extreme points are, essentially, vertices. Here, we describe the *algebraic* characterization, using Theorem 2.5.

Given a linear programming problem $L = \mathtt{LP}_{\mathtt{max}}(c, A, b)$, we assume for the sake of the next paragraphs that $rank(A) = m$, i.e. the constraint matrix has full rank. Otherwise, there is either a redundant constraint in the system $Ax = b$ (which can be removed) or it has no solution at all. This particularly implies that $m \leq n$.

We say that $B \subseteq \{1, \ldots, n\}$ with $|B| = m$ is a *basis* of L iff $A|_B \in \mathbb{R}^{m \times m}$ is non-singular. By $\bar{B} = \{1 \leq j \leq n \mid j \notin B\}$, we denote the set of indices not included in B.

Let now $x \in P_L$ be a vertex. We say that a basis B is a *basic feasible solution* (w.r.t. x) iff $B(x) \subseteq B$. In this case, we call indices $j \in B$ *basic* (w.r.t. x) and indices $j \notin B$ *non-basic*.

As a consequence of Theorem 2.5, it is easy to see that every vertex $x \in P_L$ has a basic feasible solution. If $|B(x)| = m$, we are done. Otherwise, we can augment $B(x)$ with additional linearly independent columns (since $rank(A) = m$).

Lemma 2.6. *Every vertex has a basic feasible solution corresponding to it.*

It is possible that a vertex x has more than one basis. In this case, we call x *degenerate* and observe that $|B(x)| < m$. Note that a trivial upper bound on the number of bases for x is $\binom{m}{|B(x)|}$.

Given a basis B, it is easy to compute a corresponding vector $x \in \mathbb{R}^n$, namely by setting $x|_{\bar{B}} = 0$ and $x|_B = A|_B^{-1}b$. However, x is not necessarily feasible, i.e. it might well be the case that $x_i < 0$ for some index i.

We summarize the results in a main theorem, which is sometimes attributed as the *strong fundamental theorem of linear programming*.

Theorem 2.7. *Let L be a linear programming problem.*

1. *If L has no optimal solution, then L is either unbounded or infeasible.*
2. *If L has a feasible solution, then L has a basic feasible solution corresponding to a vertex.*
3. *If L is bounded and feasible, then L has a optimal basic feasible solution corresponding to a vertex.*

Duality

The notion of *duality* is an extremely important topic in linear programming. It is motivated by three related questions, namely finding upper bounds on the optimal solution of an LP problem, knowing when an optimal solution has been reached, and finding a measure for the distance to the optimal solution. The first will be captured by the *weak duality theorem*, the second by the *strong duality theorem*, and the last by the *complementary slackness theorem*.

First, we define the *dual* of a given linear programming problem. Let therefore $\mathtt{LP}_{\mathtt{max}}(c, A, b)$ be an LP problem in standard form, i.e. of the following form.

$$(P) \quad \texttt{max} \quad c^T x \qquad \text{s.t.} \qquad Ax = b, \quad x \geq 0$$

This problem is called the *primal* problem.

Suppose that we want to find an upper bound on the cost function $c^T x$. For every constraint, let $y_i \in \mathbb{R}$ denote a number. Now we multiply every constraint with the respective y_i, and sum up the result, ending up with $y^T Ax = b^T y$. Suppose further that every coefficient of x_i in $y^T Ax$ is greater or equal to c_j. Then, it must be the case that $y^T Ax \geq c^T x$, i.e. $b^T y$ is an upper bound on the cost function. In order to find the best upper bound, we therefore want to minimize $b^T y$ s.t. $A^T y \geq c$.

This is a linear programming problem again. More precisely, we want to solve the following linear programming problem, called the *dual* problem, formally given by:

$$(D) \quad \texttt{min} \quad b^T y \qquad \text{s.t.} \qquad A^T y \geq c$$

Note that if the primal has n variables and m constraints, then the dual has m variables and n constraints.

Recall our running example again.

$$\text{Maximize } 6x_1 + 5x_2 \text{ subject to } \begin{cases} x_1 + x_2 & \leq 5 \\ 3x_1 + 2x_2 & \leq 12 \end{cases}$$

Multiply the first constraint by 1 and the second by 2, for instance. Then, we end up with $7x_1 + 5x_2 \leq 29$; since $6x_1 + 5x_2 \leq 7x_1 + 5x_2$, it is immediate to see that 29 is an upper bound on the optimal solution of the cost function. The dual problem would look as follows.

$$\text{Minimize } 5y_1 + 12y_2 \text{ subject to } \begin{cases} y_1 + 3y_2 & \geq 6 \\ y_1 + 2y_2 & \geq 5 \end{cases}$$

The upper bound observation is formalized by the easy *weak duality theorem*.

Theorem 2.8. *Let x be a feasible solution to the primal maximization problem and let y be a feasible solution to the dual minimization problem. Then $c^T x \leq b^T y$.*

The weak duality theorem has important consequences. First, it is easy to see that coinciding feasible solutions imply the respective optimality. Second, the general structure of the primal problem is reflected in the dual and vice versa.

Lemma 2.9. *Let (P) be a primal* LP *problem and (D) be the dual.*

1. *Let x be a feasible solution to (P) and y be a feasible solution to (D). If $c^T x \geq b^T y$, then x is optimal to (P) and y is optimal to (D).*
2. *If (P) (resp. (D)) is unbounded, then (D) (resp. (P)) is infeasible.*
3. *If (P) (resp. (D)) is feasible and bounded, then (D) (resp. (P)) is feasible and bounded.*

Note it might well be the case, that both the primal and the dual are infeasible.

For the strong duality theorem, we need an important theorem of the alternative of linear equation systems, known as *Farkas Lemma*. It essentially states that if a system has no solution, then there is a witnessing vector that shows that there is no solution.

Lemma 2.10 (Farkas Lemma)**.** *Let A be a real matrix and b be a vector. Exactly one of the following systems has a solution.*

1. $Ax = b$, $x \geq 0$
2. $y^T A \geq 0$, $y^T b < 0$

An extremely important consequence is the *strong duality theorem* that tells us that optimal solutions of the primal and the dual always coincide. In other words, it suffices to solve the dual in order to obtain an optimal solution to the primal.

Theorem 2.11. *Let the primal and the dual problem be feasible, and let x^* resp. y^* be an optimal solution to the primal resp. the dual. Then $c^T x^* = b^T y^*$.*

Let now (P) be a primal maximization problem and (D) be a dual minimization problem. Let x be a feasible solution (P) and y be a feasible solution to (D). We

know by weak duality that $c^T x \leq b^T y$, and call the difference $b^T y - c^T x$ the *duality gap*. Some algorithms for solving linear programs operate on both the primal and the dual, and use the duality gap to measure their progress towards the optimal solution(s).

Given the primal in standard form, we have the following formulation for both the primal and the dual

$$\begin{array}{cccccc} (P) & \texttt{max} & c^T x & \text{s.t.} & Ax = b, & x \geq 0 \\ (D) & \texttt{min} & b^T y & \text{s.t.} & A^T y \geq c & \end{array}$$

In order to replace the inequalities in the dual by equalities again, we need to introduce so-called *slack variables* s_i for every dual constraint. In other words, we introduce a *slack vector* $s \in \mathbb{R}^n_+$ and formulate the dual as follows.

$$(D) \quad \texttt{min} \quad b^T y \qquad \text{s.t.} \qquad A^T y - s = c, \quad s \geq 0$$

The duality gap can then be written as $x^T s = b^T y - c^T x$.

The *complementary slackness theorem* summarizes the correspondence between optimal solutions for both the primal and the dual, and the slack variables.

Theorem 2.12. *Let x^* be feasible for the primal and (y^*, s^*) be feasible for the dual (with slack vector s^*). Then x^* is optimal to (P) and (y^*, s^*) is optimal to (D) iff $(x^*)^T s^* = 0$.*

Simplex Algorithm

The first algorithm that solves LP problems was developed by G. Dantzig [Dan63] in 1947, and is now well-known as the *simplex algorithm*. It is still among the most widely used algorithms for solving linear programs and performs extremely well in practice. It is an iterative algorithm that starts with an (arbitrary) feasible solution on one of the vertices of the polytope and then walks along the edges in such a way that the value of the objective function is non-decreasing until an optimum is found. See Figure 2.2 for a graphical depiction of the simplex algorithm.

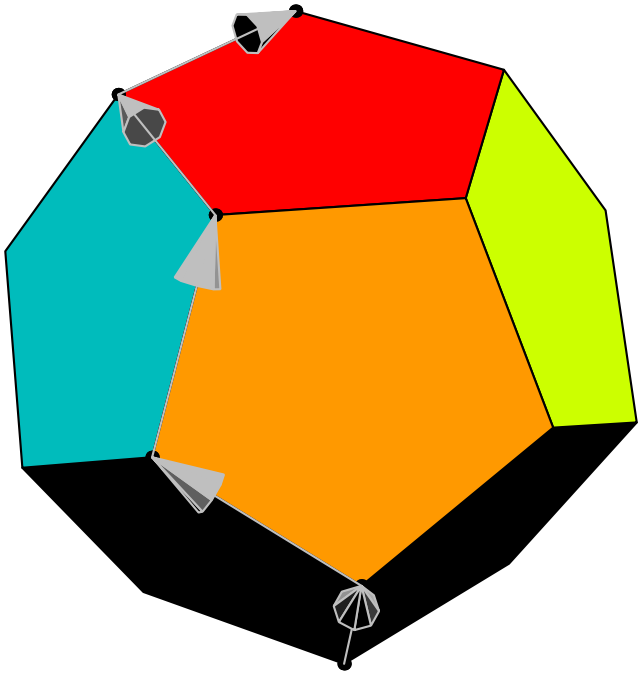


Figure 2.2: Simplex algorithm walking along the edges of an LP-polytope

The simplex algorithm essentially consists of three components: (1) finding an initial basic feasible solution, (2) identifying adjacent basic feasible solutions, and (3) identifying an optimal solution as optimal.

There is no particularly clever way of finding the initial basic feasible solution. A canonic way, for instance, is to add an unique additional helper variable to each constraint. A basic feasible solution can then be obtained by putting all additional helper variables in the basis.

Now assume that we are at a basic feasible solution with basis B. For any solution $x \in P_L$, we have $Ax = b$, i.e. $A|_B x|_B + A|_{\bar{B}} x|_{\bar{B}} = b$, i.e. $x|_B = A|_B^{-1} b - A|_B^{-1} A|_{\bar{B}} x|_{\bar{B}}$ (recall that $A|_B$ is non-singular because B is a basis). Observe that the associated value of the linear objective function can be rephrased as follows:

$$\begin{aligned} c^T x &= c|_B x|_B + c|_{\bar{B}} x|_{\bar{B}} \\ &= c|_B (A|_B^{-1} b - A|_B^{-1} A|_{\bar{B}} x|_{\bar{B}}) + c|_{\bar{B}} x|_{\bar{B}} \\ &= c|_B A|_B^{-1} b + (c|_{\bar{B}} - A|_B^{-1} A|_{\bar{B}}) x|_{\bar{B}} \end{aligned}$$

Note that the value of the objective function only depends on the values of the non-basic variables. Let now $j \in \bar{B}$. If $(c|_{\bar{B}} - A|_B^{-1} A|_{\bar{B}})_j$ is a positive coefficient, it follows that increasing the non-basic variable x_j results in an improved value of the objective function. Obviously $x|_B$ depends on $x|_{\bar{B}}$, and increasing x_j is only possible as long as all basic variables remain positive.

A single improvement step of the simplex algorithm therefore identifies a non-basic variable x_j with positive coefficient $(c|_{\bar{B}} - A|_B^{-1} A|_{\bar{B}})_j$, and increases it as much as possible while keeping $x|_B \geq 0$. This corresponds to setting one of the original basic variables x_k to zero. Then, the algorithm moves x_j to the basis and x_k to the non-basic variables, ending up with a new basic feasible solution with improved linear objective value.

An optimal solution can be easily identified by observing that there are no adjacent vertices with improved cost. However, it might be possible to end in a degenerate optimal solution (see below).

One of the most important characteristics of a simplex algorithm is the *pivoting rule* it employs. It determines which non-basic variable is to enter the basis at each iteration of the algorithm.

Recall that the algorithm may obtain degenerate basic feasible solutions. In this case, it is possible that we cannot increase a non-basic variable. We can still manage to replace a basic zero variable by a non-basic zero variable. However, it may happen that the algorithm cycles along non-improving edges without terminating. There are pivoting rules for which it is known that they avoid cycles, and hence obtain a solution to the LP problem after a finite number of iterations.

Essentially all *deterministic* pivoting rules are known to lead to an *exponential* number of pivoting steps on some LPs. This was first established by Klee and

Minty [KM72] for Dantzig's original pivoting rule. Similar results for many other rules were obtained by [Jer73], [AC78] and [GS79]. For a unified view of these constructions, see Amenta and Ziegler [AZ96].

There are two very important randomized pivoting rules for which the question, whether one of them admits a polynomial-time algorithm, has been open for decades, namely RANDOM-FACET [Kal92, Kal97, MSW96] and RANDOM-EDGE [GK07, BDF+95, GHZ98, GTW+03, BP07]. Another very interesting deterministic memorizing pivoting rule is Zadeh's LEAST-ENTERED rule [Zad80], for which no subexponential lower bound has been known. We will discuss all these pivoting rules in detail in Chapter 4 and provide concrete subexponential lower bound constructions for them.

From a combinatorial perspective, the simplex algorithm is based on walking along the edge-vertex graph of the LP polytope. It is not known whether there exists a pivoting rule that requires a polynomial number of pivoting steps on *any* linear program. This is, perhaps, the most important open problem 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 polytope is polynomial in the number of facets defining it.

An important conjecture in this context is the strong *Hirsch conjecture* which has recently been refuted by Santos [San10].

Conjecture 2.13 (Strong Hirsch Conjecture (see e.g. [Dan63], pp. 160,168))**.** *The diameter of the graph defined by an n-facet d-dimensional polytope is at most $n - d$.*

A weaker form is the so-called *polynomial Hirsch conjecture*, which is now the focus of the *polymath3* project.

Conjecture 2.14 (Polynomial Hirsch Conjecture)**.** *The diameter of the graph defined by an n-facet d-dimensional polytope is polynomial in n and d.*

The best upper bound known on the diameter is a quasi-polynomial bound (i.e., of the form $n^{\mathcal{O}(\log n)}$) obtained by Kalai and Kleitman [KK92].

Linear Programming as PLS problem

We note here that linear programs can be phrased as a PLS problem. The set of input instances obviously are the linear programs, the set of solutions w.r.t. a given linear program are all vertices of the corresponding polytope, the neighborhood of a vertex contains all adjacent vertices, and the cost function simply is the cost function of the linear program applied to the given vertex.

The simplex algorithm essentially *is* the standard algorithm for solving polynomial local search problems, providing us with the inclusion in PLS. However, as we will see, there are algorithms that solve linear programs even in (weakly) polynomial time.

Dual Simplex Algorithm

Next, we describe how a *dual simplex algorithm* operates by considering bases of constraints instead. Let (D) be a linear programming problem with n variables and m constraints.

$$(D) \quad \texttt{min} \quad c^T x \qquad \text{s.t.} \qquad A^T x \geq b$$

Let H be the set of linear constraints, i.e. $|H| = m$ and $F_H = \bigcap_{h \in H} \{x \mid x \text{ satisfies } h\}$ be the feasible region. Obviously, we have $P_D = F_H$. Let $v_H = \min_{x \in F_H} c^T x$, and define $v_H = \infty$ if $H = \emptyset$ and $v_H = -\infty$ if F_H does not contain a minimal x w.r.t. $c^T x$.

We say that a constraint h is *violated by* H iff $v_{H \cup \{h\}} > v_H$. A subset of constraints $B \subseteq H$ is an *H-basis* iff $v_H = v_B$ and for every $B' \subsetneq B$ we have $v_{B'} < v_B$.

We then have the following lemma. For details see, e.g., Chvátal [Chv83] and Schrijver [Sch86].

Lemma 2.15. *If $v_H < \infty$, then any H-basis $B \subseteq H$ contains exactly n constraints.*

Let B be an H-basis and $h \notin H$. Then BASIS$(B \cup \{h\})$ computes a basis of $B \cup \{h\}$. It is easy to see that this can be done in polynomial time.

This gives rise to an algorithm operating on the dual that removes constraints, recursively computes the optimum, and reinserts constraints that are violated. The algorithm known as RANDOM-FACET of Kalai [Kal92, Kal97] and of Matoušek *et al.* [MSW96] is a randomized implementation of this general approach. For pseudo-code, see Algorithm 3.

Algorithm 3 The RANDOM-FACET algorithm for linear programming

```
1:  procedure RANDOM-FACET(H,B)
2:      if H = B then
3:          return B
4:      else
5:          Choose h ∈ H \ B uniformly at random
6:          B' ← RANDOM-FACET(H \ {h}, B)
7:          if h is violated by B' then
8:              B'' ← BASIS(B' ∪ {h})
9:              return RANDOM-FACET(H, B'')
10:         else
11:             return B'
12:         end if
13:     end if
14: end procedure
```

Given a set of constraints H and a subset $B \subseteq H$ with $|B| = n$ and $v_B > -\infty$, RANDOM-FACET(H, B) computes a basis for H by recursion. Hence, given the full set H and an initial B, we have the following theorem:

Theorem 2.16 ([Kal92, Kal97, MSW96])**.** *Let (D) be a feasible and bounded linear programming problem, H be the set of constraints, $B \subseteq H$ with $|B| = n$ and $v_B > -\infty$. Then* RANDOM-FACET(H, B) *terminates and returns an H-basis.*

LP-type problems

The dual simplex algorithm has been abstracted to a more general setting by Sharir and Welzl in 1992 [SW92], which is now generally known as *LP-type problem*. The abstraction covers linear programming problems (corresponding to the dual simplex method) as well as finding optimal strategies in infinitary payoff games [Hal07] (corresponding to *some* policy iteration methods as we will see). Upper bounds on

the solution of LP-type problems therefore apply to some variants of policy iteration for concrete infinitary payoff games as well as to the dual simplex algorithms for concrete linear programs [Gär95].

We follow the notation of Amenta [Ame94] here. An *LP-type problem* is a pair (H, ω), where H is the *set of constraints* and $\omega : 2^H \to \mathbb{R} \cup \{\pm\infty\}$ is the *objective function*, that maps every subset of constraints $G \subseteq H$ to an element $\omega(G) \in \mathbb{R} \cup \{\pm\infty\}$. We require (H, ω) to satisfy the following two properties for all $F \subseteq G \subseteq H$:

- *Monotonicity*: $\omega(F) \leq \omega(G)$.
- *Locality*: Let $h \in H$ and $\omega(F) = \omega(G) \neq -\infty$. Then $\omega(F \cup \{h\}) > \omega(F)$ iff $\omega(G \cup \{h\}) > \omega(G)$.

A set $G \subseteq H$ is called *infeasible* if $\omega(G) = \infty$ and *feasible* otherwise, *unbounded* if $\omega(G) = -\infty$ and *bounded* otherwise. A subset of constraints $B \subseteq G \subseteq H$ is called G-*basis* iff $\omega(B) = \omega(G)$ and $\omega(B') < \omega(B)$ for every $B' \subsetneq B$. Again, we say that $h \in H$ is *violated* by $B \subseteq H$ iff $\omega(B \cup \{h\}) > \omega(B)$. An *LP-type algorithm* now computes a *basis* for a given set of constraints H. Clearly, the RANDOM-FACET rule of Algorithm 3 becomes applicable again to compute an H-basis for LP-type problems.

In the context of linear programming, H is the set of linear constraints and $\omega(G) = v_G$ is the cost of the minimal vertex in the feasible region of $G \subseteq H$.

Polynomial-time Algorithms

Although no polynomial versions of the simplex algorithm are known, linear programs can be solved in polynomial time using either the *ellipsoid* algorithm of Khachiyan [Kha79], or the *interior-point* algorithm of Karmarkar [Kar84].

The ellipsoid method has been presented by L. Khachiyan [Kha79] in 1979. It is an iterative algorithm that starts with an ellipsoid enclosing the optimal solution and then selects new ellipsoids with decreased volume containing the optimal solution

in every step. Although the ellipsoid method performs pretty badly in practice when compared to the simplex algorithm, Khachiyan was able to show as a major breakthrough result, that the algorithm converges in a polynomial number of steps, establishing the first polynomial-time algorithm that solves linear programming problems. For more on the ellipsoid algorithms and its combinatorial consequences, see Grötschel *et al.* [GLS88].

The interior-point method (and related *barrier methods*) was invented by N. Karmarkar [Kar84] in 1984. It also solves LP problems in polynomial time and generally has a much better performance in practice when compared to the ellipsoid method. Again, the algorithm is iterative and starts with an essentially arbitrary feasible point in the polytope and moves through the interior of the polytope towards the optimal solution, as opposed to the simplex algorithm that just follows the boundary of the feasible region. For more on interior-point algorithms, see Nesterov and Nemirovskii [NN94] and Ye [Ye97].

The ellipsoid and interior-point algorithms are polynomial, but 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. Furthermore, the ellipsoid and the interior-point algorithms have a strong numerical flavor, as opposed to the more combinatorial flavor of simplex algorithms. It is another major open problem whether there exists a strongly polynomial-time algorithm for solving linear programs. A polynomial pivoting rule for the simplex algorithm would also provide a positive answer to this open problem.

3

Game Theory

We introduce *infinitary payoff game theory*, which are *zero-sum perfect information graph games* played by one or two players, and sometimes by an additional randomized player controlled by nature.

We mainly consider parity games in this thesis. They are played on a directed graph that is partitioned into two node sets associated with two players; the nodes are labeled with natural numbers, called priorities. A play in a parity game is an infinite sequence of connected nodes whose winner is determined by the parity of the highest priority that occurs infinitely often, giving parity games their name.

The reason why parity games seem to be the most appropriate class of games, when trying to construct lower bound families, is that the effect of each node in a parity game is immediate: a higher priority dominates all lower priorities (in a play), no matter how many there are.

We also consider other infinitary payoff games, particularly mean payoff games, discounted payoff games, turn-based stochastic games and Markov decision processes. Markov decision processes provide a mathematical model for sequential decision making under uncertainty.

It is a major open problem whether any of the mentioned game classes – disregarding Markov decision processes – can be solved in polynomial time. Although Markov decision processes (MDPs) can be solved in polynomial time, we are still interested in obtaining lower bounds for the policy iteration algorithm for solving MDPs, as they directly relate to linear programming problems, enabling us to transfer the lower bounds to the simplex algorithm.

3.1 Infinitary Payoff Games

We introduce infinitary payoff *zero-sum perfect information graph games* in this chapter. All these game classes are played on a directed, total graph between one or two players, and sometimes even a randomization player, by moving a single token along the edges ad infinitum. We provide the common definitions for all infinitary payoff game classes here.

Graph Theory

Graph theory is an important field of discrete mathematics and should not be confused with graphs of functions and the like. It models pairwise relations between objects of a given universe.

A *directed graph* is a tuple $G = (V, E)$ where V is an arbitrary set and $E \subseteq V \times V$ is an arbitrary binary relation over V. Elements in V are called *nodes* and elements in E are called *edges*. See Figure 3.1 for an example of a directed graph. Nodes are depicted as circles and edges are drawn as arrows, pointing from one node to a successor node.

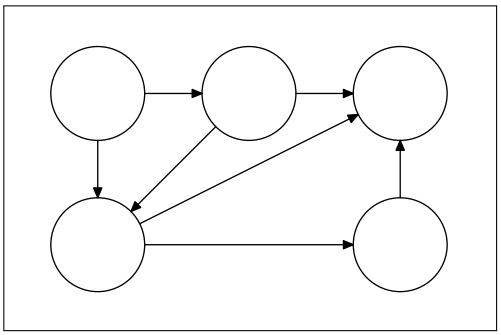


Figure 3.1: A directed graph

For a subset $U \subseteq V$, we write $G|_U$ to denote the graph *restricted to* U, i.e. $G|_U = (U, E \cap (U \times U))$. We write $G \setminus U$ to denote the graph *minus* U, i.e. $G \setminus U = G|_{V \setminus U}$.

We also use infix notation vEw instead of $(v, w) \in E$ and define the set of all *successors* of v as $vE := \{w \mid vEw\}$, as well as the set of all *predecessors* of w

as $Ew := \{v \mid vEw\}$. The *out-degree* resp. *in-degree* of a node v is the cardinality of its successor resp. predecessor set. A node v is called *source* resp. *sink* iff its predecessor resp. successor set is empty. A graph is called *total* iff it has no sinks.

From a complexity theoretic point of view, there are different approaches to measure the size of a graph. For general graphs, it is reasonable to count all nodes and edges, i.e. $|G| := |V| + |E|$, but for total graphs, it suffices to only count the edges, i.e. $|G| := |E|$, since by totally, we have that $|V| \leq |E|$ and hence $|V| + |E| \in \mathcal{O}(|E|)$.

A *finite path* is a sequence of nodes $\pi = v_0, \ldots, v_{k-1}$ s.t. $v_i E v_{i+1}$ for all $i < k-1$, and its *length* is denoted by $|\pi| := k$. We often write π_i to refer to v_i. Similarly, an *infinite path* is a sequence $\pi = v_0, v_1, \ldots$ s.t. $v_i E v_{i+1}$ for all i; we denote its *length* by $|\pi| := \infty$. Particularly, $\pi(|\pi| - 1)$ denotes the last node of a finite path π.

We call a (finite or infinite) path π *positional* iff reoccurring nodes are followed by the same successors every time, i.e. $\pi(i) = \pi(j)$ implies $\pi(i+1) = \pi(j+1)$ for every $i, j < |\pi| - 1$. A finite path π is called *tight* iff π is injective, i.e. no node occurs twice. A *cycle* is a tight path π s.t. $\pi(|\pi| - 1) E \pi(0)$. Note that every infinite positional path π can be partitioned into a finite tight path τ and a cycle ϱ s.t. $\pi = \tau\varrho^\omega$ (where ϱ^ω denotes the infinite repetition of ϱ). More concretely, a positional path can be written as follows:

$$\pi = v_1 \ldots v_k (w_1 \ldots w_l)^\omega$$

where $v_i \neq v_j$ for all $i \neq j$ and $w_i \neq w_j$ for all $i \neq j$. See Figure 3.2 for an example of a positional path.

Note that in general, this decomposition is not unique, since we allow the prefix part of the positional path to repeat some nodes of the cycle. We call τ the *path component* and ϱ the *cycle component* of any such decomposition.

Let $E^0 := \{(v, v) \in V\}$, $E^1 := E$ and $E^{i+1} := \{(v, w) \mid \exists u \in V : (vE^i u \text{ and } uEw)\}$. We refer to the *reflexive, transitive closure of* E by $E^* := \bigcup_{i \geq 0} E^i$, and to the *transitive closure of* E by $E^+ := \bigcup_{i > 0} E^i$.

The complexity of almost all graph-based algorithms rises with the cyclicity of graphs. We say that a non-empty subset of nodes $C \subseteq V$ is a *strongly connected*

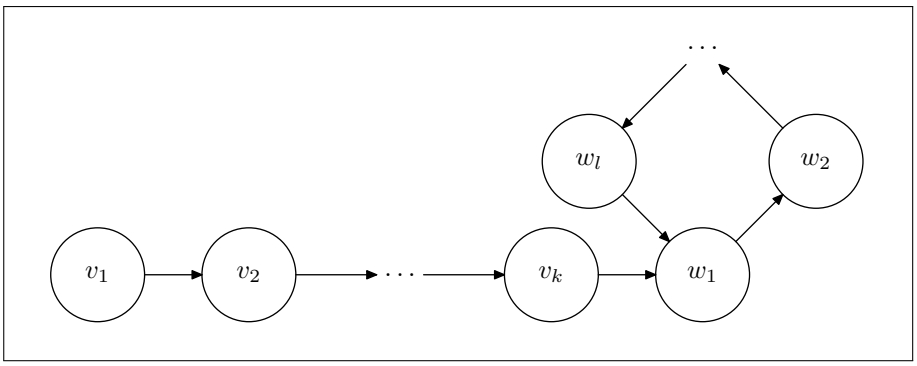


Figure 3.2: A positional path

component (SCC) iff uE^*v for every $u, v \in C$, i.e. if every node in C can reach every node in C. We say that an SCC C is *maximal* iff there is no superset $D \supsetneq C$ s.t. D is an SCC. An SCC C is called *proper* iff $|C| > 1$ or $C = \{v\}$ for some $v \in V$ with vEv.

Every finite graph $G = (V, E)$ admits a unique partitioning $C_1, \ldots, C_n$ into maximal strongly connected components. Tarjan's algorithm [Tar72], for instance, computes the decomposition into maximal SCCs in linear time.

Theorem 3.1 ([Tar72])**.** *Every graph $G = (V, E)$ can, in time $\mathcal{O}(|E|)$, be partitioned into maximal SCCs $C_1, \ldots, C_n$ with $V = \bigcup_{i \leq n} C_i$ and $C_i \cap C_j = \emptyset$ for all $i \neq j$.*

Let $G = (V, E)$ be a graph and $C_1, \ldots, C_n$ be a decomposition into maximal strongly connected components. There is a topological ordering $\rightarrow$ on these SCCs which is defined as $C_i \rightarrow C_j$ iff $i \neq j$ and there are $u \in C_i$, $v \in C_j$ with uEv. In other words, $C_i \rightarrow C_j$ if there is a connecting edge from C_i to C_j between distinct SCCs C_i and C_j. An SCC C is called *final* if there is no SCC C' s.t. $C \rightarrow C'$. Note that every finite graph must have at least one final SCC.

Graph Games

We consider *zero-sum perfect information graph games* in this thesis. All these game classes are played on a directed, total graph, called the *game graph* or the *arena*. A single pebble is placed on one of the nodes – sometimes a designated *starting node* – and moved along an outgoing edge to a successor node. This process continues ad infinitum, yielding an infinite path π, called *play* in the context of graph games.

The main difference between all game classes that we consider in this thesis is the number and kinds of players that can participate in moving the pebble in an instance of a game. We consider one- and two-player game classes, as well as their probabilistic extensions in which we also have a *randomization* player, controlled by nature. We therefore have either one or two "normal" players and sometimes a probabilistic player. Such games are often referred to as 1-player, 1.5-player, 2-player and 2.5-player games, where the ".5" refers to the existence of a probabilistic player. Nevertheless, the number of player *entities* in a 2.5-player game is three.

Let $G = (V, E)$ be the *underlying graph* of the game. The set of nodes V, depending on the game class, is partitioned into node sets for the respective player entities. There are several different names for the players, and we fix the following ones for the rest of the thesis. The first *normal* player is usually called *player 0* or the *maximizer*, the second *normal* player is similarly called *player 1* or the *minimizer*. The player that is controlled by nature is usually called the *randomizer*, *randomization player*, *probabilistic player* or simply the *average player*.

More concretely, the set of nodes is partitioned into V_0, V_1 and V_R in a 2.5-player game, to denote the set of nodes controlled by player 0, player 1 and the randomizer respectively. Given a node $v \in V$, we say that v is *controlled* or *owned* by some player if it is contained in the respective set. This player is called the controller, owner or *chooser* of the node. See Figure 3.3 for a graphical depiction of nodes of all three player entities: nodes owned by player 0 are drawn as circles, nodes owned by player 1 are drawn as rectangles, and nodes controlled by the average player are shown as diamonds. Given a two-player game G, we say that G *is a one-play game*, iff there is a player i s.t. for every $v \in V_i$, we have $|vE| = 1$.

Note that in other contexts, the normal players switch roles, i.e. the first player is the minimizer while the second player is the maximizer. The first player is sometimes called player 1 and the second player is then called player 2.

We write $E_i = E \cap (V_i \times V)$ to denote the set of player i controlled edges, and likewise $E_R = E \cap (V_R \times V)$ to denote the set of edges controlled by randomization.

In probabilistic games, we have an edge labeling function $p : E_R \to [0, 1]$ that assigns to each outgoing edge e of the randomizer a probability $p(e)$ s.t.

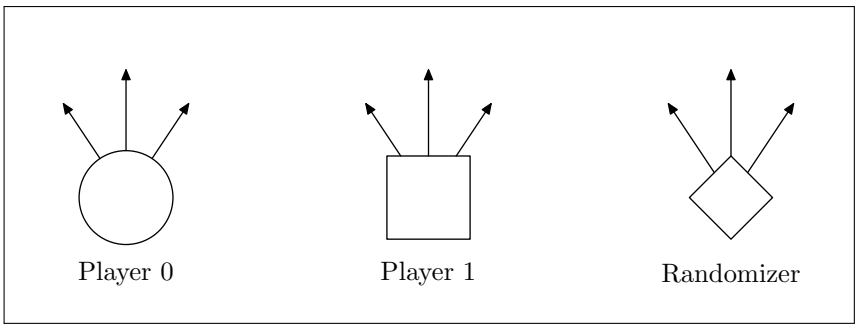


Figure 3.3: Players in infinitary payoff games

$\sum_{u \in vE} p(v,u) = 1$ for every $v \in V_R$. See Figure 3.4 for a graphical depiction of the edge labeling of a randomization node.

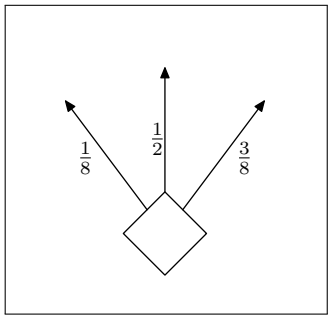


Figure 3.4: Average node with edge probabilities

We now describe how a game is played between the different players. Starting in a node $v_0 \in V$, they construct a path through the graph as follows. If the construction so far has yielded a finite sequence $v_0 \dots v_n$ and $v_n \in V_i$ (with $i \in \{0,1\}$) then player i selects a $w \in v_n E$, and the play continues with the sequence $v_0 \dots v_n w$. If $v_n \in V_R$, a successor $w \in v_n E$ is chosen arbitrarily with probability $p(v_n, w)$, and the play continues with $v_0 \dots v_n w$ as well.

Strategies

A player's strategy in a game is an exhaustive plan of action for whatever situation might arise when playing against any opponent. It determines the action the player will take at any stage of the game, possibly depending on the finite history of play up to that stage.

In the context of *Markov decision processes*, the common term for a strategy is *policy*; we will use both terms synonymously.

Formally, a (general) *strategy* for player i is a function $\sigma : V^*V_i \to V$, s.t. for all sequences $v_0 \dots v_n$ with $v_{j+1} \in v_j E$ for all $j = 0, \dots, n-1$, and all $v_n \in V_i$, we have: $\sigma(v_0 \dots v_n) \in v_n E$. That is, a strategy for player i assigns to every finite path through G that ends in V_i a successor of the ending node.

A play $v_0 v_1 \dots$ *conforms* to a strategy σ for player i if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ then $v_{j+1} = \sigma(v_0 \dots v_j)$. Intuitively, conforming to a strategy means to always make those choices that are prescribed by the strategy.

Given a strategy σ and a *partial* strategy τ, we write $\sigma[\tau]$ to denote the *τ-update of σ*, being defined by $\sigma[\tau](v) = \tau(v)$ if $v \in \mathtt{dom}(\tau)$ and $\sigma[\tau](v) = \sigma(v)$ otherwise.

A strategy σ for player i is called *positional*, *memory-less* or *history-free* if for all $v_0 \dots v_n \in V^*V_i$ and all $w_0 \dots w_m \in V^*V_i$ we have: if $v_n = w_m$ then $\sigma(v_0 \dots v_n) = \sigma(w_0 \dots w_m)$. That is, the value of the strategy on a finite path only depends on the last node on that path.

Positional strategies are much simpler objects than general strategies, and preferable when solving related decision problems. We will see that all infinitary payoff games are *positionally determined*, meaning that we can restrict ourselves to positional strategies.

We identify positional strategies therefore with functions $\sigma : V_i \to V$. Similarly, a play $v_0 v_1 \dots$ *conforms* to a positional strategy σ for player i if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ then $v_{j+1} = \sigma(v_j)$. We denote the set of positional strategies of player i by $\mathcal{S}_i(G)$.

Let σ be a positional strategy for player 0 and let $v \in V$ be a node. In the context of one-player games, there is exactly one positional path that starts in v and conforms to σ. Formally, we denote the *induced play* by $\pi_{\sigma,v}$. Similarly, in the context of a two-player game, let τ additionally be a positional strategy for player 1. Again, there is exactly one positional path that starts in v and conforms to both σ and τ. It will also be denoted by $\pi_{\sigma,\tau,v}$.

A positional strategy σ for player i induces a *strategy subgame* $G|_\sigma := (V, E|_\sigma)$ where $E|_\sigma := \{(u,v) \in E \mid u \in V_i \Rightarrow \sigma(u) = v\}$. A strategy subgame $G|_\sigma$ is

basically the same game as G with the restriction that whenever σ provides a strategy decision for a node $u \in V_i$, all transitions from u but $\sigma(u)$ are no longer accessible.

3.2 Parity Games

Parity games (see e.g. [EJS93]) are infinite-duration perfect information two-player games played on directed total graphs with natural numbers, called *priorities*, assigned to their vertices. The two players, called player 0 or *even*, and player 1 or *odd*, construct an infinite path in the game graph. Even wins if the largest priority that appears an infinite number of times on the path is even. Odd wins otherwise.

Parity games are an interesting object of study in computer science, and the theory of formal languages and automata in particular, for (at least) the following reasons:

- They are closely related to other games of infinite duration like payoff games and turn-based stochastic games [Jur98, Pur95, Sti95], see the following subchapters.
- They are at the core of other important problems in computer science, for instance, solving a parity game, i.e., determining which of the two players has a *winning strategy*, is known to be polynomial-time equivalent to model checking for the modal μ-calculus [EJS93, Sti95].
- They arise in complementation problems for tree automata [GTW02, EJ91] and in emptiness as well as word problems for various kinds of (alternating) automata [EJ91].
- Controller synthesis problems can be reduced to satisfiability problems for branching-time logics [VAW03] which in turn require the solving of parity games, because of determinizations of Büchi word automata into parity automata [Pit06, KW08].
- Solving a parity game is one of the rare problems that belong to the complexity classes PLS [BM08], NP $\cap$ coNP, and even to UP $\cap$ coUP [Jur98]. It is a

tantalizing open problem whether parity games can be solved in polynomial time.

- There are many, structurally different algorithms that solve parity games. This variety is owed to the theoretical challenge of answering the question whether parity games can be solved in polynomial time.

Technicalities

Formally, a parity game is a tuple $G = (V, V_0, V_1, E, \Omega)$ where (V, E) forms a directed, total graph, partitioned into the sets of the two players V_0 and V_1; $\Omega : V \to \mathbb{N}$ is the *priority function* that assigns to each node a natural number, called the *priority* of the node. We write $|\Omega|$ for the index $ind(G)$ of the parity game, i.e. the number of different priorities assigned to its nodes. See Figure 3.5 for a graphical depiction of a parity game: every node is labeled with its priority.

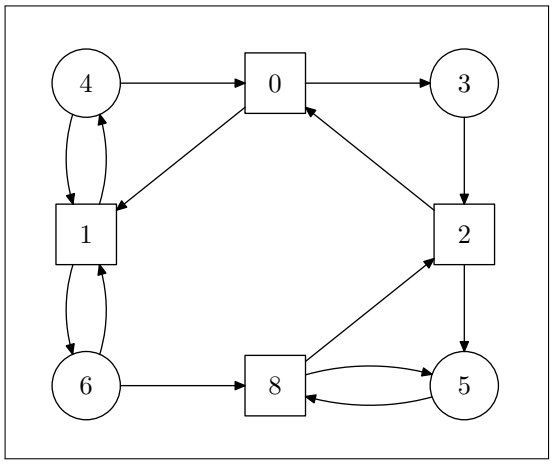


Figure 3.5: A parity game

We will restrict ourselves to finite parity games. The name *parity game* is due to the fact that the winner of a play is determined according to the parities (even or odd) of the priorities occurring in that play. It has a unique winner given by the *parity* of the largest priority that occurs infinitely often. Player 0 wins if this priority is even, and player 1 wins if it is odd.

More formally, every infinite play has a unique *winner* given by the *parity* of the greatest priority that occurs infinitely often. The winner of the play $v_0v_1v_2\ldots$ is player i iff $\max\{p \mid \forall j \in \mathbb{N}\, \exists k \geq j : \Omega(v_k) = p\} \equiv_2 i$ (recall that $i \equiv_2 j$ holds iff $|i - j| \mod 2 = 0$). That is, player 0 tries to make an even priority occur infinitely often without any greater odd priorities occurring infinitely often, player 1 attempts the converse.

Consider for instance the parity game of Figure 3.5, and assume that both player 0 and player 1 play by positional strategies σ and τ that are described by "select the successor with the highest priority". Starting in the node labeled with 4, we end up in the following positional play.

$$\pi_{\sigma,\tau,4} = 4, 1, 6, (8, 5)^\omega$$

Obviously, the largest priority that occurs infinitely often is 8, hence player 0 wins this play.

Technically, we are considering so-called *max-parity games*. There is also the *min-parity* variant, in which the winner is determined by the parity of the *least* priority occurring infinitely often. On finite graphs, these two variants are equivalent in the sense that a max-parity game $G = (V, V_0, V_1, E, \Omega)$ can be converted into a min-parity game $G' = (V, V_0, V_1, E, \Omega')$ whilst preserving important notions like winning regions, strategies, etc.; simply let p be an even upper bound on all the priorities $\Omega(v)$ for every $v \in V$. Then define $\Omega'(v) := p - \Omega(v)$. This construction also works the other way round, i.e. in order to transform a min-parity into a max-parity game.

Now that we know the *objective* of the two players, we can express the fact that a player can win every play starting from a certain node by the notion of *winning strategies*. A strategy σ for player i is a *winning strategy* in node v if player i wins every play that begins in v and conforms to σ. Given a set of nodes U, we say that σ is a *winning strategy on* U iff σ is a winning strategy in every node $v \in U$. We say that player i *wins* the game G starting in v iff player i has a winning strategy for G starting in v.

Let U be a set of nodes. If we a have positional winning strategy σ_v for player i for every node $v \in U$, then we have one single positional winning strategy for player i on U.

Lemma 3.2. *Let G be a parity game, $U \subseteq V$ and $i \in \{0, 1\}$. Let σ_v be a positional winning strategy for player i starting in v for every $v \in U$. There is a single positional winning strategy for player i on U.*

Proof. A single positional winning strategy σ on U can be constructed as follows. Let $U = \{v_1, \ldots, v_n\}$. Let U_i be the set of nodes $v \in U$ s.t. σ_{v_i} is a positional winning strategy for player i starting in v. Clearly, $v_i \in U_i$.

Let now $\sigma(v) = \sigma_{v_i}(v)$ where $i = \min\{j \mid v \in U_j\}$. Obviously, σ is a positional strategy on U. We need to show that σ is a winning strategy for player i.

Let $v \in U$ and let π be a play starting in v conforming to σ. Let $i_1, i_2, \ldots$ be the indices corresponding to the choices of σ. It is easy to see that $i_1 \geq i_2 \geq \ldots$, hence we end up in a single reoccurring positional winning strategy σ_{v_i}. □

With G we associate two sets $W_0, W_1 \subseteq V$, called the *winning sets*, such that W_i is the set of all nodes v s.t. player i wins the game G starting in v.

Clearly, we must have $W_0 \cap W_1 = \emptyset$ for otherwise assume that there is a node v such that both players have winning strategies σ and τ for G starting in v. Then there is a unique play π that starts in v and conforms to both σ and τ. However by definition, π is won by both players, and therefore the maximal priority occurring infinitely often would have to be both even and odd.

On the other hand, it is not obvious that every node should belong to either of W_0 or W_1. However, this is indeed the case and known as *determinacy*: a player has a winning strategy for a node iff the opponent does not have a winning strategy for that node. But before we can prove that, we need some additional definitions.

Consider the parity game of Figure 3.5 once again. It is easy to see that $W_0 = \{4, 1, 6\}$ with positional winning strategy $\sigma(4) = 1$ and $\sigma(6) = 1$, and that $W_1 = \{0, 8, 3, 2, 5\}$ with positional winning strategy $\tau(0) = 3$, $\tau(2) = 0$, and $\tau(8) = 2$.

Attractors and Dominions

A set $U \subseteq V$ is said to be i-closed iff player i can force any play starting in U to stay within U. This means that player $1-i$ must not be able to leave U, but player i must always have the choice to remain inside U:

$$\forall v \in U : \left(v \in V_{1-i} \Rightarrow vE \subseteq U \right) \text{ and } \left(v \in V_i \Rightarrow vE \cap U \neq \emptyset \right)$$

Note that W_0 is 0-closed and W_1 is 1-closed.

A set $U \subseteq V$ is called an *i-dominion* iff U is i-closed and the induced subgame is won by player i. In other words, an i-closed set U is an i-dominion iff player i wins $G|_U$. Particularly, we have that $U \subseteq W_i$. Clearly, W_0 is a 0-dominion and W_1 is a 1-dominion. That is, an i-dominion U covers the idea of a region in the game graph that is won by player i by forcing player $1-i$ to stay in U.

Let $U \subseteq V$, $i \in \{0, 1\}$. The *i-attractor* of U is the least set W s.t. $U \subseteq W$ and whenever $v \in V_i$ and $vE \cap W \neq \emptyset$, or $v \in V_{1-i}$ and $vE \subseteq W$ then $v \in W$. Hence, the i-attractor of U contains all nodes from which player i can move "towards" U and player $1-i$ must move "towards" U. Let $Attr_i(G, U) = W$ denote the i-attractor of U.

Attractors will play an important role in the *recursive solving procedure* by Zielonka [Zie98] described in Chapter 5.1, because they can efficiently be computed using breadth-first search on the inverse graph underlying the game. At the same time, it is possible to construct an *attractor strategy* which is a positional strategy in a reachability game. Following this strategy guarantees player i to reach a node in U eventually, regardless of the opponent's choices.

Define, for all $k \in \mathbb{N}$:

$$\begin{aligned}
Attr_i(G,U)^0 &:= U \\
Attr_i(G,U)^{k+1} &:= Attr_i(G,U)^k \cup \{v \in V_i \mid \exists w \in Attr_i(G,U)^k \text{ s.t. } vEw\} \\
&\quad \cup \{v \in V_{1-i} \mid \forall w : vEw \Rightarrow w \in Attr_i(G,U)^k\} \\
Attr_i(G,U) &:= \bigcup_{k \in \mathbb{N}} Attr_i(G,U)^k
\end{aligned}$$

Note that any attractor on a finite game is necessarily finite, and the approximation defined above thus terminates after at most $|V|$ many steps. It is also not difficult to see that the attractor can be computed in time $\mathcal{O}(|W \cup \bigcup_{v \in W} Ev|)$. The corresponding attractor strategy – which is a partial strategy defined on $W \setminus U$ – is defined as $\tau^{Attr}(v) = w$ if there is $k > 0$ s.t. $v \in (V_i \cap Attr_i(G,U)^k) \setminus Attr_i(G,U)^{k-1}$ and $w \in Attr_i(G,U)^{k-1} \cap vE$. Note that the choice of w is not unique, but any w with the prescribed property will suffice. We will write $Attr_i(G,U) = (W,\tau)$, where τ is a corresponding attractor strategy, whenever we want to refer to an attractor strategy.

An important property that has been noted before [Zie98, Sti95] is that removing the i-attractor of any set of nodes from a game will still result in a total game graph.

Lemma 3.3 ([Zie98, Sti95])**.** *Let $G = (V, V_0, V_1, E, \Omega)$ be a parity game and $U \subseteq V$. Let $V' := V \setminus Attr_i(G,U)$. Then $G|_{V'}$ is again a parity game (with its underlying graph being total).*

In other words $V \setminus Attr_i(G,U)$ is $(1-i)$-closed; if additionally U is an i-dominion then $Attr_i(G,U)$ also is an i-dominion. This yields a general procedure for solving parity games: find a dominion in the game graph that is won by one of the two players, build its attractor, and investigate the complement subgame.

Lemma 3.4. *Let G be a parity game, U be an i-dominion and $W = Attr_i(G,U)$. Then W is an i-dominion as well. If additionally σ as a winning strategy on U, and τ is an attractor strategy on W, then $\sigma[\tau]$ is a winning strategy on W.*

Proof. Let U be an i-dominion and $W = Attr_i(G,U)$. It is easy to see that W is i-closed. We need to show that $W \subseteq W_i$. Let $v \in W$. If $v \in U$, we are done by assumption. If $v \in W \setminus U$, we can force every play starting in v to end up in a node $u \in U$ by following the attractor strategy. Since u is won by player i, so is v. □

Positional Determinacy

Parity games enjoy positional determinacy meaning that for every node v in the game either $v \in W_0$ or $v \in W_1$ [EJ91]. Additionally, if player i wins $v \in W_i$ with a strategy σ_v, then there is also a positional winning strategy σ'_v for player i; hence,

we restrict ourselves to positional strategies for the rest of the thesis. By Lemma 3.2, we have that player i has a single positional winning strategy on W_i.

Parity games lie in the third level of the Borel hierarchy, and are as such determined [Mar75]. However, it can also be shown directly that parity games are positionally determined by induction using attractors.

Theorem 3.5 ([Mar75, GH82, EJ91])**.** *Let $G = (V, V_0, V_1, E, \Omega)$ be a parity game. Then $W_0 \cap W_1 = \emptyset$ and $W_0 \cup W_1 = V$. Additionally, for $v \in W_i$, player i has a positional winning strategy starting in v.*

Proof. We prove this theorem by induction on $|G|$. If $V = \emptyset$, we are done. Let $V \neq \emptyset$. Let U_0 be the set of all nodes $v \in V$ s.t. player 0 has a positional winning strategy starting in v. It is easy to see that $Attr_i(G, U_0) = U_0$ and $U_0 \subseteq W_0$.

If $U_0 \neq \emptyset$, let $G' = G \setminus U_0$. By induction hypothesis, we have that $W'_0 \cup W'_1 = V \setminus U_0$ with positional winning strategies for all nodes in $V \setminus U_0$. It is easy to see that $W'_0 = \emptyset$, $W_0 = U_0$ and $W_1 = W'_1$.

If $U_0 = \emptyset$, we show that $W_1 = V$. Let p be the largest priority occuring in G, and let $P = \{v \in V \mid \Omega(v) = p\}$. Let $i = p \mod 2$, $A = Attr_i(G, P)$ and $G' = G \setminus A$. By induction hypothesis, we have that $W'_0 \cup W'_1 = V \setminus A$ with positional winning strategies for all nodes in $V \setminus A$. It is easy to see that $W'_0 = \emptyset$ and $W'_1 = V \setminus A$.

If $i = 1$, is is not hard to see that player 1 wins on $W'_1 \cup A$ by using the positional winning strategy of the induction hypothesis on W'_1, an arbitrary positional strategy on P and an attractor strategy on A. Then we have that every play that visits P infinitely often is won by player 1, because the greatest occurring priority is p. Otherwise, the play eventually stays in W'_1, and is therefore won by induction hypothesis.

If $i = 0$, we need to show that player 1 wins on $W'_1 \cup A$. It is not hard to see that player 1 wins on W'_1 by using the positional winning strategy of the induction hypothesis on W'_1. Let $A' = Attr_1(G, W'_1)$. It is easy to see that player 1 wins on A' using an attractor strategy. Note that $A' \neq \emptyset$, since otherwise player 0 would have a positional winning strategy on A, being impossible since $U_0 = \emptyset$. Let now $G'' = V \setminus A'$. By induction hypothesis, we have that $W''_0 \cup W''_1 = V \setminus A'$ with

positional winning strategies for all nodes in $V \setminus A'$. It is easy to see that $W_0'' = \emptyset$ and $W_1'' = V \setminus A'$. Note that player 1 wins in G on W_1'' by using the induction hypothesis strategy. □

Decision Problems

The problem of *solving* a parity game is to compute W_0 and W_1, as well as corresponding winning strategies σ_0 and σ_1 for the players on their respective winning regions. This is generally known as *solving a parity game globally*.

The problem of *solving a parity game locally* is to decide, for a given node v, whether or not v belongs to W_0. Note that it may not be necessary to visit all nodes of a game in order to answer this question. Consider the following trivial example. The node under consideration belongs to player 0, has even priority and an edge to itself. Hence, this edge represents a winning strategy for player 0 in this node. A depth-first search can find this loop whilst leaving the rest of the game unexplored.

Clearly, the local and global problem are interreducible, the global one solves the local one for free, and the global one is solved by calling the local one $|V|$ many times. But neither of these indirect methods is particularly clever. Thus, there are algorithms for the global, and other algorithms for the local problem (see below).

Note that if we only have one player in the game, it is not too hard to solve parity games in polynomial time. See Algorithm 4 for a pseudo-code specification.

Lemma 3.6. *Let G be a one-player parity game. W_0 and W_1 can be computed in polynomial time.*

Proof. It is easy to see that Algorithm 4 terminates after polynomial time. Let i^* be the player with real choices. We show the correctness by induction on $|G|$. The basis is trivial.

Let therefore p, i, U, A, W_0' and W_1' as specified in the algorithm. If $i = i^*$, it follows that $A = V$, and player i^* can win the whole game by using an attractor strategy and an arbitrary strategy on U. Hence, we have $W_{1-i}' = \emptyset$ and return $W_i = V$.

Algorithm 4 Algorithm for solving single player games

```
1:  procedure SOLVE(G)
2:      if V_G = ∅ then
3:          (W_0, W_1) ← (∅, ∅)
4:          return (W_0, W_1)
5:      else
6:          p ← max{Ω_G(v) | v ∈ V_G}
7:          i ← p mod 2
8:          U ← {v ∈ V_G | Ω_G(v) = p}
9:          A ← Attr_i(G, U)
10:         (W'_0, W'_1) ← SOLVE(G \ A)
11:         if W'_{1−i} = ∅ then
12:             (W_i, W_{1−i}) ← (V_i, ∅)
13:             return (W_0, W_1)
14:         else
15:             B ← Attr_{1−i}(G, W'_{1−i})
16:             (W_i, W_{1−i}) = (V_G \ B, B)
17:             return (W_0, W_1)
18:         end if
19:     end if
20: end procedure
```

If $i \neq i^*$, let $B \leftarrow Attr_{1-i}(G, W'_{1-i})$. It is easy to see that W'_{1-i} is won by player i^*, and hence so is B. It remains to show that $W_i \supseteq V \setminus B$. By contradiction assume that there is a node $v \in V \setminus B$ s.t. $v \in W_{i^*}$. Let σ be a corresponding positional winning strategy.

First observe that any play starting in v conforming to σ cannot reach B since B is an i^*-attractor. If this play visits A infinitely often, it must visit U infinitely often as well and hence is won by player i. If it visits A only finitely often, we have that the play eventually stays in W'_i. But we know by induction hypothesis, that such a play must be won by i. □

It is a major open problem whether full two-player parity games can be solved in polynomial time.

Theorem 3.7 ([EJS93])**.** *Solving parity games is in* NP ∩ coNP.

Proof. It suffices to show that solving parity games locally is in NP ∩ coNP. Let therefore $v \in V$. We need to decide whether $v \in W_0$. First, we show the inclusion

in NP by a guess-and-check argument. Guess a positional strategy σ for player 0. Check that σ is a winning strategy for player 0 starting in v. By Lemma 3.6, we know that this can be done in polynomial time.

For the inclusion in coNP, we check for player 1 that every positional strategy is not a winning strategy. By Theorem 3.5, we know that this implies $v \in W_0$. □

It has also been shown that solving parity games belongs to $\mathtt{UP} \cap \mathtt{coUP}$ [Jur98]. This is not as surprising as it looks. We will see in Chapter 4 that policy iteration enables us to equip strategies with a linear ordering that allows us to check in polynomial time whether a given strategy is optimal; additionally, we have that optimal strategies coincide with a winning strategy, if one exists. Therefore, the inclusion in $\mathtt{UP} \cap \mathtt{coUP}$ is easy to see.

Theorem 3.8 ([Jur98])**.** *Solving parity games is in* $\mathtt{UP} \cap \mathtt{coUP}$.

Furthermore it has been shown that solving parity games belongs to PLS. Again, we will see in Chapter 4 that policy iteration can be modeled as a PLS algorithm.

Theorem 3.9 ([BM08])**.** *Solving parity games is in* PLS.

Algorithms

There are many algorithms that solve parity games. This variety is owed to the theoretical challenge of answering the question whether parity games can be solved in polynomial time. There are four structurally different classes of solving algorithms. Let n denote the number of nodes, e the number of edges and p the number of different priorities in a game.

1. First, there is the *recursive algorithm* due to Zielonka [Zie98], that is based on a decomposition into subgames with recursion on the number of nodes and priorities. It admits an almost trivial upper bound $\mathcal{O}(n^p)$. We show an explicit exponential lower bound given by the Fibonacci series in Chapter 5.1. It should be noted, however, that the recursive algorithm is one of the best performing methods in practice [FL09].

There is a variation on the recursive algorithm given by Jurdziński, Paterson and Zwick [JPZ06] that improves the trivial upper bound to $n^{\sqrt{n}}$, which is based on an exhaustive search for small dominions, interrelated with the original algorithm.

2. Second, there is the *small progress measures algorithm* due to Jurdziński [Jur00], which is based on an iterative increase of lexicographically ordered tuples of priority occurrences until a fixpoint is reached. Jurdziński already proves an exponential upper bound of $\mathcal{O}(p \cdot e \cdot \left(\frac{n}{p}\right)^{\lfloor 0.5 \cdot p \rfloor})$ and an exponential lower bound of $\Omega(\left\lceil \frac{n}{p} \right\rceil^{\lceil 0.5 \cdot p \rceil})$ in his paper.

   There is a variation by Schewe [Sch07], his so-called *big-step algorithm*. Like the variation by Jurdziński, Paterson and Zwick [JPZ06], it searches for small dominions with subsequent decomposition via the original recursive algorithm. However, the search for small dominions is performed by small progress measures iteration here. It admits the currently best known upper bound on the deterministic solution of parity games, namely $\mathcal{O}(e \cdot n^{\frac{1}{3}p})$.

   We only mention the small progress measures algorithm for completeness here, since lower and upper bounds are already known quite precisely.

3. Third, there is an algorithm for solving parity games locally by Stevens and Stirling [SS98], to which we will refer to as the *model checking algorithm*. In fact, it is directly defined as a model checking problem in [SS98]; since μ-calculus model-checking and solving parity games are interreducible problems, we will study the model checking algorithm directly as a local parity game solving algorithm here.

   It is based on exploring the given game depth-first, detecting cycles and backtracking subsequently. It has a trivial upper bound of $\mathcal{O}(n^n)$. We show an explicit exponential lower bound of $\Omega(1.5^n)$ in Chapter 5.2 for this algorithm.

4. The *strategy improvement*, *strategy iteration* or *policy iteration* technique is the most general approach that can be applied as a solving procedure for parity games and related game classes. It was introduced by Howard [How60] for solving problems on Markov decision processes and has been adapted by

several other authors for solving nonterminating stochastic games [HK66], turn-based stochastic games [Con92], discounted and mean payoff games [Pur95, ZP96], as well as parity games [VJ00, Sch08].

Strategy iteration is an algorithmic scheme that is parameterized by an *improvement rule* which defines how to select a successor strategy in the iteration process. The exact runtime complexity highly depends on the applied improvement rule. The main focus of this thesis is to show exponential and subexponential lower bounds for all major improvement rules. See Chapter 4 for a rigorous treatment of the rules and the corresponding lower bounds.

Remarks

We end this chapter with some remarks on seemingly simpler computational problems than computing the winning sets along with positional winning strategies. The motivation for all this is to allow solving algorithms to restrict the class of considered parity games without compromising the generality.

First, an algorithm can make some of the following assumptions whenever it is convenient:

1. We can assume that the given parity game is a single strongly connected component. It is easy to see that a general parity game can be solved in a bottom-up fashion, starting with the terminal strongly connected components, then computing the attractors of the winning sets of the solved terminal SCCs, and finally removing the attractors from the game. Then, continue with the rest. See [FL09] for a detailed treatment.

2. We can assume that every priority in a parity game occurs only once, or we can assume that between two different even (resp. odd) priorities, there must be an occurrence of an odd (resp. even) priority. Again, see [FL09] for a detailed treatment.

3. We can assume that a parity game is *alternating*, meaning that every player i node is only connected to nodes of player $1-i$. Every parity game can be

transformed into an alternating parity game by adding additional nodes that have only one outgoing edge and negligible priorities. Winning strategies and winning sets essentially coincide.

4. We can assume that the out-degree of all nodes is bounded by two. Again, this can be achieved by replacing two out-going edges of a player i controlled node, that has more than two edges, by one edge going to an additional node with negligible priority, that has the two original out-going edges.

Second, it suffices to compute the winning sets, without giving positional winning strategies. Assume that we have an algorithm PARTITION that partitions a given parity game into the winning sets for both players. We can then show the following:

Theorem 3.10. *If* PARTITION *runs in polynomial time, then there is an algorithm that computes positional winning strategies in polynomial time.*

Proof. Given a parity game $G = (V, V_0, V_1, E, \Omega)$ and the winning sets W_0 and W_1, define the set of *ambiguous choices* as follows.

$$F(G) = \{(v, w) \in E \mid |vE| > 1 \text{ and } (v \in V_0 \cap W_0 \text{ or } v \in V_1 \cap W_1)\}$$

The algorithm that computes the positional winning strategies can be described as follows.

If $F(G) = \emptyset$, it follows that there is exactly one positional strategy for player 0 and one positional strategy for player 1 on the respective winning sets. Obviously, these must be winning strategies as well.

If $F(G) \neq \emptyset$, let W_0 and W_1 be the winning sets for G. Let $e \in F(G)$ be an arbitrary edge; consider $G' = G \setminus \{e\}$ and compute the winning sets W_0' and W_1' for G'. If $W_0 = W_0'$ (and $W_1 = W_1'$), it must be the case that there are winning strategies contained in G that do not use e. We continue with G'.

If otherwise $W_0 \neq W_0'$ (and $W_1 \neq W_1'$), it follows that e belongs to a winning strategy. Let $e = (v, w)$ and consider $G'' = G \setminus (vE \setminus \{e\})$. Obviously, the winning sets of G'' and G coincide. Continue with G''. □

The converse holds, in some sense, as well. Given an arbitrary positional strategy σ for player i, we can easily determine the largest dominion in a game on which σ is a winning strategy by applying Lemma 3.6.

Third, we show that winning sets as such do not give a lot of insight into what happens in a game. More formally, we show that, given an arbitrary game G, we can transform it into a game G' s.t. either $W'_0 = V_{G'}$ or $W'_1 = V_{G'}$. Given a winning strategy for G', we can easily find a dominion in G.

Theorem 3.11. *Let* STRATEGY *be a polynomial-time algorithm for computing winning strategies on parity games that are won completely by one of the two players. Then, there is a polynomial-time algorithm for solving general parity games.*

Proof. Let $G = (V, V_0, V_1, E, \Omega)$ be a parity game. W.l.o.g. assume that G is alternating. Let p be the largest priority occurring in G. Let v^* be an unused node name, i.e. $v^* \notin V$. Let $i = p \mod 2$. Consider the following game $G' = (V', V'_0, V'_1, E', \Omega')$ where

- $V' = V \cup \{v^*\}$,
- $V'_i = V_i \cup \{v^*\}$,
- $V'_{1-i} = V_{1-i}$,
- $E' = E \cup (V_i \times \{v^*\}) \cup (\{v^*\} \times V)$, and
- $\Omega' = \Omega[v^* \mapsto p + 1]$.

First, we show that G' is completely won by one of the two players. Let W_0 and W_1 be the winning sets of G.

If $W_i \neq \emptyset$, let σ be a positional winning strategy on W_i. We now define a positional winning strategy σ' on V'_i. Let $w \in W_i$ be arbitrary. Then $\sigma' = \sigma[v^* \mapsto w][V_i \setminus W_i \mapsto v^*]$. In other words, every node outside of W_i controlled by player i moves to v^*, and v^* moves into the W_i. Hence, G' is completely won by player i.

If $W_i = \emptyset$, we know that G is completely won by player $1-i$. Assume that G' is not completely won by player $1-i$, i.e. there is an i-dominion D. It must be the case

that $v^* \in D$, and every i-winning strategy has to go through v^* infinitely often. But then the largest priority is $p+1$ which is won by player $1-i$. Contradiction.

Second, we show that there is a polynomial-time algorithm for solving general parity games. By Lemma 3.6, it suffices to show that we can find a dominion in G.

We apply STRATEGY on G' and get a positional winning strategy for one of the two players.

If we get a winning strategy for player i, it is easy to see that the winning set W_i' contains an i-dominion D that is an i-dominion on G as well. We can find the dominion be the following algorithm. Fix the i-winning strategy σ on G', i.e. consider $G'' = G'|_\sigma$. Now decompose G'' into strongly connected components (see Theorem 3.1). Take a terminal SCC D. It is easy to see that $v^* \notin D$, since otherwise player $1-i$ could win a play in D. Hence, D is an i-dominion even in G.

If we get a winning strategy for player $1-i$, it must be the case that this is also a winning strategy on G. □

We can even be more restrictive and assume that we already know which player wins the whole game.

Corollary 3.12. *Let* EVENSTRATEGY *be a polynomial-time algorithm for computing winning strategies for player 0 on parity games that are won completely by player 0. Then, there is a polynomial-time algorithm for solving general parity games.*

Proof. Let EVENSTRATEGY be a polynomial-time algorithm for computing winning strategies for player 0 on parity games that are won completely by player 0.

First, it is easy to see that we can transform EVENSTRATEGY to a polynomial-time algorithm ODDSTRATEGY that assumes the given parity game to be completely won by player 1.

Let now DECIDEEVEN resp. DECIDEODD be the algorithm of Theorem 3.11 parameterized with EVENSTRATEGY resp. ODDSTRATEGY.

Let p be a polynomial upper bound on EVENSTRATEGY and ODDSTRATEGY, hence there is a polynomial upper bound q on DECIDEEVEN and DECIDEODD by Theorem 3.11.

We now execute both DECIDEEVEN and DECIDEODD for at most time q. Since the given parity game must be won by one of the two players, one of the two routines has to terminate in time and to return the correct answer. We can easily check which answer is correct, if we get two, by Lemma 3.6. □

3.3 Related Games

We describe payoff games [EM79, GKK88, ZP96] with different kinds of outcome criteria here, particularly the limiting average and the discounted reward criterion. Then, we define Markov decision processes [Bel57] and present turn-based stochastic games [Sha53].

Payoff Games

Formally, a *payoff game* [EM79, GKK88, ZP96] is a tuple $G = (V, V_0, V_1, E, r)$ where (V, E) forms a directed, total graph, partitioned into the sets of the two players V_0 and V_1; $r : E \to \mathbb{R}$ is the *reward function* that assigns to each edge an *immediate reward*. See Figure 3.6 for a graphical depiction of a payoff game: every edge is labeled with its immediate reward.

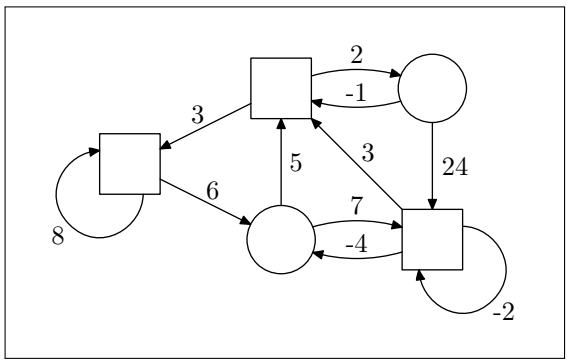


Figure 3.6: A payoff game

The two players in a payoff game have opposing roles again, but this time, they try to maximize (player 0) resp. minimize (player 1) the *outcome* of a play. The

outcome of a play is, essentially, a function R that maps a play π to some value in $\mathbb{R}$; there are different reasonable ways to define the outcome of a given play π.

Consider, for instance, the payoff game of Figure 3.6, and assume that both player 0 and player 1 play by positional strategies that yield the following immediate rewards:

$$\pi = 6, 5, (2, -1)^\omega$$

First, there is the *limiting average criterion* that sums up all immediate rewards and builds the average. Formally, let π be a play. The *limiting average* of π, $\tilde{R}(\pi)$, is defined as follows:

$$\tilde{R}(\pi) = \liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} r(\pi(i), \pi(i+1))$$

In our example, the limiting average would be 0.5. A payoff game with the limiting average criterion is often called *(deterministic) mean payoff game (MPG)*.

Second, there is the *discounted reward criterion* that is based on a *discount factor* $0 < \lambda < 1$. Formally, let π be a play. The *discounted reward* of π (w.r.t. λ), $R_\lambda(\pi)$, is defined as follows:

$$R_\lambda(\pi) = (1-\lambda) \sum_{i=0}^{\infty} \lambda^i \cdot r(\pi(i), \pi(i+1))$$

In our example, the discounted reward would be, depending on λ, as follows:

$$(1-\lambda)(6+5\lambda) + \frac{2\lambda^2 - \lambda^3}{1+\lambda}$$

A payoff game with discounted reward criterion is often called *discounted payoff game (DPG)*. We sometimes write G_λ for a payoff game G, to denote that we apply the discounted reward criterion to G with discount factor λ.

Discount factors have interesting economical interpretations. When $\lambda \to 1$, the value of the discounted game tends to the value of the mean payoff game.

Depending on the definition of the outcome R, we define the *value* of a game, as a function $\mathcal{V}$ that maps every state of the game to some value in $\mathbb{R}$. Let σ be a

player 0 strategy and τ be a player 1 strategy. First, we define the value associated with the pair of strategies.

$$\mathcal{V}[R]^{\tau}_{\sigma}(v) = R(\pi_{\sigma,\tau,v})$$

In other words, the reward of a pair of strategies maps every node to the outcome of the unique play associated with v, σ and τ.

Second, we define the *maximizer value* resp. *minimizer value* of a node v as the largest resp. smallest outcome that maximizer resp. minimizer can secure using a certain strategy, no matter what the other player is doing.

$$\mathcal{V}[R]_{\sigma}(v) = \inf_{\tau} \mathcal{V}[R]^{\tau}_{\sigma}(v)$$

$$\mathcal{V}[R]^{\tau}(v) = \sup_{\sigma} \mathcal{V}[R]^{\tau}_{\sigma}(v)$$

Third, we define the *value of the game* $\mathcal{V}[R]$ by the following equation whenever it exists.

$$\mathcal{V}[R](v) = \sup_{\sigma} \mathcal{V}[R]_{\sigma}(v) = \inf_{\tau} \mathcal{V}[R]^{\tau}(v)$$

Additionally we say that the value of the game can be *secured by optimal positional strategies* iff there is a positional strategy $\sigma^* \in \mathcal{S}_0(G)$ and a positional strategy $\tau^* \in \mathcal{S}_1(G)$ s.t. we have

$$\mathcal{V}[R](v) = \mathcal{V}[R]_{\sigma^*}(v) = \mathcal{V}[R]^{\tau^*}(v)$$

Such strategies are said to be optimal strategies.

The following very important theorem for payoff games tells us that the value always exists and that it can always be secured by optimal positional strategies.

Theorem 3.13 ([EM79])**.** *Let G be a payoff game. The following holds:*

1. *$\mathcal{V}[\tilde{R}]$ exists and can be secured by optimal positional strategies.*
2. *Let $0<\lambda<1$. $\mathcal{V}[R_\lambda]$ exists and can be secured by optimal positional strategies.*

The *decision problem* here is, given some vector $x \in \mathbb{R}^n$, to decide whether the value of the game is greater (resp. less) or equal to x. The problem of *solving* a payoff game is to compute the values of the game along with optimal positional strategies. Note that, given the values of the game, one can induce corresponding optimal positional strategies by choosing locally consistent edges.

Given a one-player payoff game G, we can again solve the game in polynomial time via solving a related linear program.

Theorem 3.14 ([KL93, Kar78])**.** *Let G be a one-player payoff game. The values can be computed in polynomial time.*

Like for parity games, it is a tantalizing open problem whether payoff games can be solved in polynomial time.

Theorem 3.15 ([ZP96, Pur95])**.** *Solving payoff games is in* $\texttt{NP} \cap \texttt{coNP}$.

Again, we can also show that solving payoff games belongs to $\texttt{UP} \cap \texttt{coUP}$. We will see in Chapter 4 that policy iteration enables us to equip strategies with a linear ordering that allows us to check in polynomial time whether a given strategy is optimal. Therefore, the inclusion in $\texttt{UP} \cap \texttt{coUP}$ is easy to see.

Theorem 3.16 ([Jur98, ZP96])**.** *Solving payoff games is in* $\texttt{UP} \cap \texttt{coUP}$.

As for parity games, we will see in Chapter 4 that policy iteration for payoff games can be modeled as a PLS algorithm.

Theorem 3.17. *Solving payoff games is in* PLS.

One-player payoff games can be solved, as we will see, by linear programming, i.e. particularly in polynomial time. General payoff games are usually solved by policy iteration (see Chapter 4). Another algorithm for solving general payoff games is the so-called *value iteration*, see for instance [CH08].

Markov Decision Processes

Markov decision processes (MDP) 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, to machine learning, artificial intelligence, economics and game theory. The study of MDPs started with the seminal work of Bellman [Bel57]. More general *stochastic games* were previously considered by Shapley [Sha53]. For a thorough treatment of MDPs, see the books of Howard [How60], Derman [Der70], Puterman [Put94] and Bertsekas [Ber01].

An MDP is composed of a finite set of *states*. Each state has a set of *actions* associated with it. Each action has an immediate *reward* and a probability distribution according to which the next state of the process is determined if this action is taken. In each time unit, the *controller* of the MDP has to choose an action associated with the current state of the process. The goal of the controller is to maximize the long-term *outcome* again.

We consider Markov decision processes as 1.5-player games in our framework of infinitary payoff games. Hence, states are nodes owned by player 0, actions are edges controlled by player 0, leading to nodes of the average player, having a probability distribution on the edges.

Formally, a *Markov decision process* is a tuple $G = (V, V_0, V_R, E, r, p)$ where (V, E) forms a directed, total graph, partitioned into the sets of the 0-player V_0 and the average player V_R; $r : E \to \mathbb{R}$ is the *reward function* that assigns to each edge an *immediate reward*, and $p : E_R \to [0, 1]$ is a probability distribution that assigns to each outgoing edge e of the randomizer a probability $p(e)$ s.t. $\sum_{u \in vE} p(v, u) = 1$ for every $v \in V_R$. See Figure 3.7 for a graphical depiction of a Markov decision process. Note that in this figure, edges of the randomized player have no cost at all.

The different criteria that work for payoff games can be applied here as well. However, we have to deal with the probabilistic player now. Given a node v and a strategy σ for player 0, let $\Omega_{v,\sigma}$ denote the set of plays π that start in v and are consistent with σ. A probability space and a probability measure over these plays can be induced by using cylinder sets of finite paths. Given a finite path π, the associated

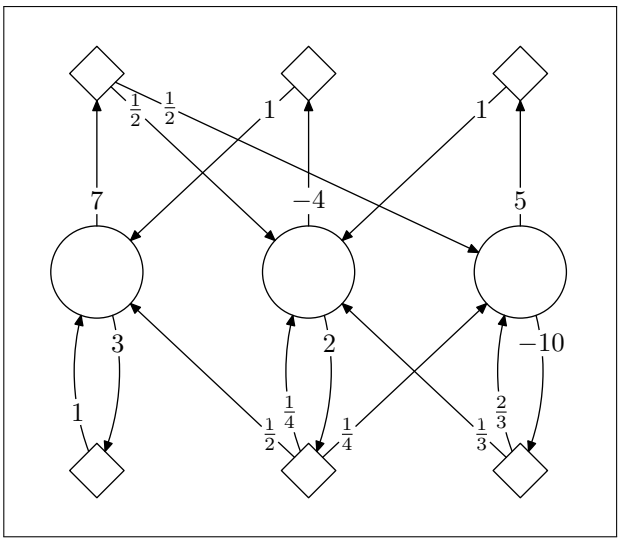


Figure 3.7: A Markov Decision process

cylinder set $C(\pi)$ contains all infinite completions of π. We define the probability of a cylinder set $C(\pi)$ by the probability of choosing the common finite prefix:

$$\mathbb{P}(C(\pi)) = \prod_{0<i<|\pi|, \pi(i-1)\in V_R} p(\pi(i-1), \pi(i))$$

The induced σ-algebra on the class of cylinder sets $C(\pi)$ s.t. $\pi(0) = v$ and π conforms to σ, yields that there is a unique extension to a probability measure $\mathbb{P}_{v,\sigma}$ on the σ-algebra [ADD00]. Given an (measurable) outcome function R that assigns a number in $\mathbb{R}$ to a play π, we can define the *expected outcome* $\mathbb{E}_{v,\sigma}[R]$ as the expectation of R on the σ-algebra in the $\Omega_{v,\sigma}$ universe.

We can define the *value* of a game again, this time using the expected outcome instead. Let σ be a player 0 strategy. We define the *maximizer value* of a node v as the expected outcome conforming to the strategy.

$$\mathcal{V}[R]_\sigma(v) = \mathbb{E}_{v,\sigma}[R]$$

Then, we define the *value of the game* $\mathcal{V}[R]$ again by the following equation, whenever it exists:

$$\mathcal{V}[R](v) = \sup_\sigma \mathcal{V}[R]_\sigma(v)$$

Additionally, we say that the value of the game can be *secured by an optimal positional strategy* iff there is a positional strategy $\sigma^* \in \mathcal{S}_0(G)$ s.t.

$$\mathcal{V}[R](v) = \mathcal{V}[R]_{\sigma^*}(v)$$

Again, we have the important theorem that tells us that the value always exists and that it can always be secured by an optimal positional strategy.

Theorem 3.18 ([Put94])**.** *Let G be a Markov decision process. The following holds:*

1. *$\mathcal{V}[\tilde{R}]$ exists and can be secured by an optimal positional strategy.*
2. *Let $0<\lambda<1$. $\mathcal{V}[R_\lambda]$ exists and can be secured by an optimal positional strategy.*

The decision problem and the notion of solving are exactly defined as before. It is well-known that Markov decision processes can be solved in polynomial time via general linear programs. Ye [Ye05] has even presented a strongly polynomial time algorithm for solving discounted MDPs using interior point methods when the discount factor is fixed.

Theorem 3.19 ([Put94, Ye05])**.** *Solving Markov decision processes is in* P.

Markov decision processes are usually solved by policy iteration (see Chapter 4) and *value iteration*, see for instance [Put94]. Again, we will see in Chapter 4 that policy iteration for MDPs can be modeled as a PLS algorithm.

Theorem 3.20. *Solving Markov decision processes is in* PLS.

Turn-based Stochastic Games

Turn-based stochastic payoff games form an even more general family of games that can be seen as a combination of Markov decision processes and payoff games. A turn-based stochastic payoff game [Sha53, AM09] is a tuple $G = (V, V_0, V_1, V_R, E, r, p)$, where V_0 and V_1 are the sets of vertices controlled by players 0 and 1, V_R is the set of randomization vertices, controlled by nature, p is a function that assigns a probability

to each out-going edge of the randomizer, and $r : E \to \mathbb{R}$ is a function that assigns an immediate reward to each edge of the graph.

The two players again construct an infinite path in the game graph. When the last vertex reached is in V_i, where $i \in \{0, 1\}$, player i chooses the next edge. When the last vertex reached is in V_R, the next edge is chosen according to the probabilities specified by p. The two players try to maximize, respectively minimize, the long term *expected outcome* average reward per turn.

As with Markov decision processes, we define the *expected outcome* $\mathbb{E}_{v,\sigma,\tau}[R]$, here with respect to a pair of strategies σ and τ:

$$\mathcal{V}[R]^{\tau}_{\sigma}(v) = \mathbb{E}_{v,\sigma,\tau}[R] \qquad \mathcal{V}[R]_{\sigma}(v) = \inf_{\tau} \mathcal{V}[R]^{\tau}_{\sigma}(v) \qquad \mathcal{V}[R]^{\tau}(v) = \sup_{\sigma} \mathcal{V}[R]^{\tau}_{\sigma}(v)$$

$$\mathcal{V}[R](v) = \sup_{\sigma} \mathcal{V}[R]_{\sigma}(v) = \inf_{\tau} \mathcal{V}[R]^{\tau}(v)$$

Again we say that the value of the game can be *secured by optimal positional strategies* iff there is a positional strategy $\sigma^* \in \mathcal{S}_0(G)$ and a positional strategy $\tau^* \in \mathcal{S}_1(G)$ s.t. we have

$$\mathcal{V}[R](v) = \mathcal{V}[R]_{\sigma^*}(v) = \mathcal{V}[R]^{\tau^*}(v)$$

Theorem 3.21 ([Sha53])**.** *Let G be a turn-based stochastic payoff game. The following holds:*

1. *$\mathcal{V}[\tilde{R}]$ exists and can be secured by optimal positional strategies.*
2. *Let $0<\lambda<1$. $\mathcal{V}[R_\lambda]$ exists and can be secured by optimal positional strategies.*

As with payoff games, we have that solving turn-based stochastic payoff games in polynomial time is still an open problem.

Theorem 3.22 ([Con92])**.** *Solving payoff games is in* NP ∩ coNP *and in* UP ∩ coUP.

Turn-based stochastic payoff games are usually solved by policy iteration, see Chapter 4. Moreover, policy iteration shows that solving turn-based stochastic payoff games can be modeled as a PLS algorithm.

Theorem 3.23. *Solving turn-based stochastic payoff games is in* PLS.

We note there are also so-called *simple stochastic games* (see e.g. [ZP96, Con92]), another class of 2.5-player games. It deviates from the class of our infinitary payoff games in having no cost associated with the edges and two sink nodes 0 and 1. The objective of the maximizer is to reach the 1-sink, while the objective of the minimizer is to reach the 0-sink. We assume simple stochastic games to be *halting*, i.e. starting in a node v and playing according to any pair of strategies almost surely ends in one of the two sinks. The *value of a node* here is the probability of reaching the 1-sink when player 0 and player 1 play optimal. Again, the values of a game exist and there are optimal positional strategies [Con92]. We note without going into details that there is a direct correspondence between simple stochastic games and turn-based stochastic games that can be used to transfer our lower bounds to simple stochastic games [ZP96, Con92] as well.

3.4 Relations and Reductions

We describe here how the different infinitary payoff game classes are related by giving well-known, explicit procedures on how to reduce a game of one class into a corresponding game of another class, if possible. By *reduce*, we mean that the translation can be performed by a deterministic polynomial-time algorithm, and that the solution to the translated game can be back-transformed to a solution of the original game (again in deterministic polynomial time). Particularly, we consider the following reductions:

1. Parity games can be reduced to mean payoff games [Pur95],
2. Mean payoff games can be reduced to discounted payoff games [ZP96],
3. Every payoff game can be reduced to turn-based stochastic games (trivially),
4. Markov decision processes can be reduced to turn-based stochastic games (trivially),

5. Deterministic MDPs can be reduced to mean payoff games (trivially),
6. Markov decision processes can be reduced to linear programming problems; we will consider this correspondence in detail in Chapter 4.5, and
7. all infinitary payoff games can be cast as an LP-type problem; we will see this in Chapter 4.1.

See Figure 3.8 for a summary of the most important relations. It is easy to see that having two players in a game raises the difficulty in obtaining polynomial-time decision procedures quite a lot.

From Parity Games to Mean Payoff Games

Parity games can be easily reduced to mean payoff games, following the construction of Puri [Pur95]. Let $G = (V, V_0, V_1, E, \Omega)$ be a parity game. The *induced mean payoff game* $H = (V, V_0, V_1, E, r)$ operates on the same graph and induces the reward function r as follows:

$$r : (v, w) \mapsto (-|V|)^{\Omega(v)}$$

The essential idea is that even priorities are mapped to exponentially large positive numbers while odd priorities are mapped to exponentially large negative numbers. By this, one can ensure that the occurrence of an edge associated with a certain priority has essentially the same effect on the play as the original priority.

Player 0 has a winning strategy from vertex v in G if and only if player 0 has a strategy that guarantees a *positive* outcome starting from v in G'. The correctness of the reduction follows from the fact that a cycle in the game graph has a positive mean value if and only if the largest priority on one of its vertices is even.

Theorem 3.24 ([Pur95])**.** *Let G be a parity game, H be the induced mean payoff game, and let σ and τ be optimal positional strategies in H. The following holds:*

1. $W_0 = \{v \in V \mid \mathcal{V}[\tilde{R}](v) \geq 0\}$,

2. $W_1 = \{v \in V \mid \mathcal{V}[\tilde{R}](v) < 0\}$,

3. *σ is a winning strategy for player 0 on W_0 in G, and*

4. *τ is a winning strategy for player 1 on W_1 in G.*

Parity games thus form a very well-behaved subfamily of mean payoff games.

From Mean Payoff Games to Discounted Payoff Games

Mean payoff games can be reduced to discounted payoff games by specifying a discount factor that is sufficiently close to 1. Given a mean payoff game $G = (V, V_0, V_1, E, r)$, we say that a discount factor λ is *large enough* iff

$$\lambda \geq 1 - \frac{1}{4 \cdot |V|^3 \cdot \max\{|r(v)| \mid v \in V\}}$$

Let v be a node in a mean payoff game G and let λ be large enough. Zwick and Paterson [ZP96] show that the value $\mathcal{V}[\tilde{R}](v)$ can be essentially bounded by $\mathcal{V}[R_\lambda](v)$, i.e. more precisely:

$$|\mathcal{V}[R_\lambda](v) - \mathcal{V}[\tilde{R}](v)| \leq \frac{1-\lambda}{2|V|^2(1-\lambda)}$$

It follows that $\mathcal{V}[\tilde{R}](v)$ can be obtained from $\mathcal{V}[R_\lambda](v)$ by rounding to the nearest rational with a denominator less than $|V|$.

Theorem 3.25 ([ZP96])**.** *Let G be a mean payoff game, let λ be a large enough discount factor and let σ and τ be optimal positional strategies w.r.t. $\mathcal{V}[R_\lambda]$. Then σ and τ are also optimal positional strategies w.r.t. $\mathcal{V}[\tilde{R}]$.*

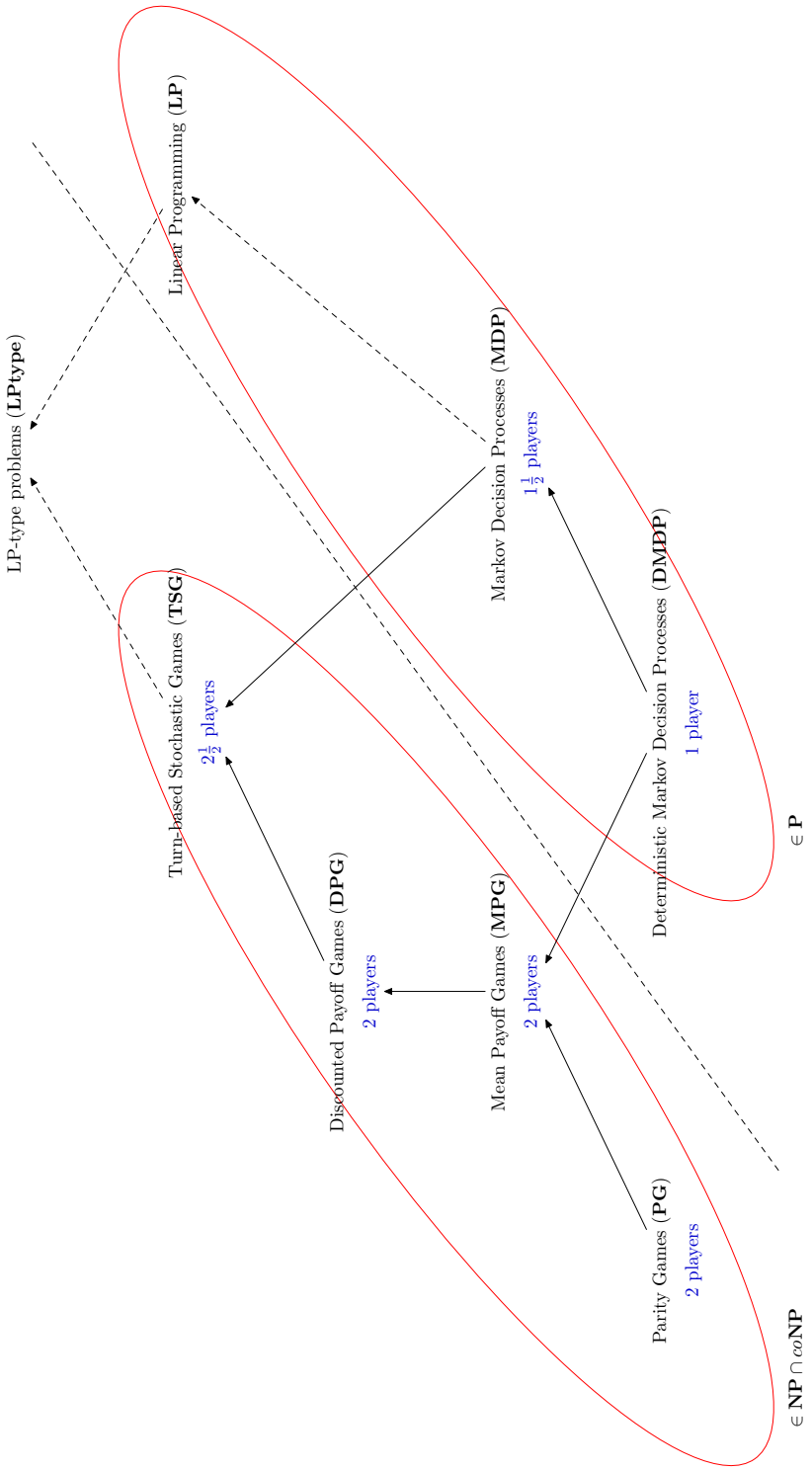


Figure 3.8: Reductions and Complexity

4

Lower Bounds for Strategy Iteration

The *strategy improvement*, *strategy iteration* or *policy iteration* technique is the most general approach that can be applied as a solving procedure for infinitary payoff games and related problems, such as interval inequality systems [GS07], static analysis [CGG+05] and many others. It was introduced by Howard [How60] for solving problems on Markov decision processes and has been adapted by several other authors for solving nonterminating stochastic games [HK66], simple stochastic games [Con92], discounted and mean payoff games [Pur95, ZP96] as well as parity games [VJ00].

Strategy iteration is an algorithmic scheme that is parameterized by an *improvement rule*, which defines how to select a successor strategy in the iteration process. The runtime of strategy iteration is known to depend crucially on the applied improvement rule. An example has been known for some time for which a sufficiently poor choice of a single-switch rule causes an exponential number of iterations of the strategy improvement algorithm [BV07]. It is a major open problem, whether there is a polynomial-time computable improvement rule, that results in a polynomial number of iterations in the worst case.

However, our contribution to this field is to show that all major improvement rules of the literature have instances on which they require subexponential or exponential time. Particularly, we show that SWITCH-ALL (which appears all over the literature), SWITCH-BEST [Sch08], RANDOM-FACET [Kal92, Kal97, MSW96], RANDOM-EDGE (which appears all over the literature), SWITCH-HALF [MS99], and LEAST-ENTERED [Zad80] require subexponential or exponential time on parity games and on all other classes of infinitary payoff games (whenever the respective rule is applicable).

Some of the mentioned rules can be cast as pivoting rules for the simplex algorithm, in fact, the RANDOM-FACET, RANDOM-EDGE and LEAST-ENTERED rules have been formulated *originally* as parameterizations of the simplex method. We show that the simplex algorithm for solving linear programs, when parameterized with one of the applicable rules, requires subexponential or exponential time as well.

The rest of this chapter is organized as follows. We start with an introduction to policy iteration, and define and explain all terms that we require to reason about the lower bound constructions. Then, we describe all major improvement rules appearing in the literature, present the known results and relate it to our contributions. We explain our general strategy of constructing and proving lower bounds, and show how to transfer results from one class, say parity games, to other infinitary payoff games via the notion of sink parity games, or to linear programming, via Markov decision process and their relation to LPs. We conclude the chapter with the constructions of our lower bounds.

4.1 General Framework

We introduce the fundamentals of strategy iteration in this chapter. Although the strategy iteration algorithm shares many similarities with the linear programming simplex method, we refrain from describing strategy iteration in a framework that is general enough to subsume the simplex algorithm as well, for reasons of clarity and comprehensibility. We will discuss the similarities and differences at the end of this chapter.

Let now G be an infinitary payoff game played on the graph (V, E). The strategy improvement algorithms are based on iterating over strategies of one player, usually player 0, until a final optimal strategy has been found. In order to reduce unnecessary formalisms, we will leave the game G implicit in the following definitions, whenever the context is clear.

Valuations

In general, we have a *totally ordered set* $(\mathcal{U}, \preceq)$ of *node valuations* and a map $\Xi_\sigma : V \to \mathcal{U}$ that assigns, given a positional player 0 strategy σ, to a node v a node

valuation. The whole map Ξ_σ is called *game valuation*. We extend the ordering on node valuations to an ordering on nodes w.r.t. a given strategy as follows:

$$v \preceq_\sigma u \qquad :\Longleftrightarrow \qquad \Xi_\sigma(v) \preceq \Xi_\sigma(u)$$

We will see that the game valuations of the infinitary payoff games of this thesis can be computed in polynomial time. We assume that as an axiom for the rest of this chapter.

(SI1) The game valuation of a strategy is polynomial-time computable.

Based on node valuations, we define the pre-ordered set of *game valuations* $(\mathcal{V}, \trianglelefteq)$ by setting $\mathcal{V} = \{\Xi : V \to \mathcal{U}\}$; the pre-order $\trianglelefteq$ is induced by applying the total order on the node valuations component-wise.

$$\Xi \trianglelefteq \Xi' \qquad :\Longleftrightarrow \qquad \Xi(v) \preceq \Xi'(v) \text{ for all } v \in V$$

Obviously, $\Xi \triangleleft \Xi'$ iff $\Xi \trianglelefteq \Xi'$ and $\Xi \neq \Xi'$. We extend the pre-order on game valuations to a pre-order on strategies by setting $\sigma \trianglelefteq \sigma'$ iff $\Xi_\sigma \trianglelefteq \Xi_{\sigma'}$. In general, the valuation vectors Ξ_σ and $\Xi_{\sigma'}$ may be incomparable.

We say that a strategy σ is *optimal* iff for every strategy σ' we have $\sigma' \trianglelefteq \sigma$. A crucial property of game valuations is that optimal strategies exist.

(SI2) There is an $\trianglelefteq$-optimal strategy σ.

Game valuations should be thought of as a tight description of the performance of the respective player 0 strategy. A key idea of strategy iteration is to use the game valuation of a given strategy σ to guide the search for an improved strategy σ', that is closer to the optimal strategy.

Let $e \in E_0$ be a player 0 edge. We say that e is an *improving edge* or *improving switch* w.r.t. a strategy σ iff $\sigma \triangleleft \sigma[e]$. An appealing feature of policy iteration algorithms is, that determining whether e constitutes an improving switch with respect to σ can be done *without* evaluating $\sigma[e]$, as we will see.

Next, we define the *set of improving switches* as the set of edges that are improving w.r.t. a strategy σ:

$$I_\sigma = \{e \in E_0 \mid \sigma \lhd \sigma[e]\}$$

The set of improving switches has very important properties that describe how the given strategy can be modified, to result in an improved strategy. Particularly, if the set of improving switches is empty, we have found the optimal strategy.

(SI3) If $I_\sigma = \emptyset$ then σ is optimal.

We say that a strategy σ is *improvable* iff $I_\sigma \neq \emptyset$.

We can use improving switches to find an improved strategy as well. Particularly, as we have $\sigma \lhd \sigma[e]$ with $e \in I_\sigma$ by definition; but even better, strategy iteration allows to apply so-called *multi-switches*.

We say that a subset $I \subseteq I_\sigma$ is *applicable* iff $(v, u), (v, w) \in I$ implies $u = w$. In other words, an applicable subset of improving switches does not contain two different edges originating from the same node. Such a combination of switches is called *multi-switch*. Policy iteration satisfies the following condition:

(SI4) Let $I \subseteq I_\sigma$ be a non-empty applicable set. Then $\sigma \lhd \sigma[I]$.

Strategy Iteration

The algorithm starts with some initial strategy σ_0 and generates an improving sequence $\sigma_0, \sigma_1, \ldots, \sigma_l$ of strategies, ending with an optimal strategy σ_l. Every strategy in this sequence is obtained by switching an applicable subset of the respective set of improving switches.

The initial strategy σ_0, often denoted by ι, can usually be an arbitrary strategy, but some variants of the strategy iteration require ι to fulfill some special properties, see, for instance, [Sch08].

There is no particularly clever way to select the initial strategy. We therefore see policy iteration as an algorithm that receives the game *and* the initial strategy as

Algorithm 5 Policy iteration

1: $\sigma \leftarrow \iota$
2: **while** σ is improvable **do**
3: $\quad I \leftarrow$ non-empty applicable subset of I_σ
4: $\quad \sigma \leftarrow \sigma[I]$
5: **end while**
6: **return** σ

input instance, and computes the optimal strategy starting with the given input, see Algorithm 5.

An execution trace of the strategy iteration is called *run*, and defined to be the sequence of strategies that occurred in the iteration. Formally, $r = \sigma_0, \sigma_1, \ldots, \sigma_l$ is called a *run on G starting with σ_0*.

Lemma 4.1. *Let G be game. Strategy iteration on G terminates and returns an optimal strategy.*

Improvement Rules and Diameter

We now formalize the notion of an *improvement rule* or *pivoting rule*, that selects a non-empty applicable subset, and applies it to the given strategy. Formally, an *improvement rule* is a map IMPR-RULE : $\mathcal{S}_0(G) \to \mathcal{S}_0(G)$ s.t. for every improvable strategy σ, there is an non-empty applicable set $I \subseteq I_\sigma$ s.t. IMPR-RULE$(\sigma) = \sigma[I]$. Then, policy iteration can be realized by Algorithm 6.

Algorithm 6 Policy iteration with improvement rule

1: $\sigma \leftarrow \iota$
2: **while** σ is improvable **do**
3: $\quad \sigma \leftarrow$ IMPR-RULE(σ)
4: **end while**
5: **return** σ

Since we assume that the single operations of policy iteration can be performed in polynomial time, it immediately follows that the runtime complexity directly depends on the number of iterations. There is no improvement rule known for which we have a polynomial upper bound. In fact, we will show for all major improvement rules that they have subexponential or exponential lower bounds.

Nevertheless, one could ask the question whether it is theoretically possible to have a polynomial-time admitting improvement rule? For that reason, we define the *diameter* of an infinitary payoff game. Let $R(G, \sigma)$ denote the set of policy iteration runs on G starting with σ. The diameter of G is then defined as follows:

$$\operatorname{diam}(G) = \max_{\sigma \in \mathcal{S}_0(G)} \min_{r \in R(G,\sigma)} |r|$$

We now show that the diameter of infinitary payoff games is *linear* in the worst case. This is a very important fact about policy iteration, particularly when comparing it to the simplex algorithm for solving linear programs. There, we have the (polynomial) Hirsch conjecture, saying that the diameter of any linear program is polynomial in the worst case. But there is probably no easy way to show this, and it might even be the case that the diameter of some linear programs is superpolynomial.

We formulate the proof of linear diameter by specifying an improvement rule that enforces linearly many iterations in the worst case.

Lemma 4.2. *Let G be an infinitary payoff game. There is an improvement rule* SWITCH-LIN *s.t. policy iteration requires at most $|V|$ many iterations.*

Proof. Let $G = (V, E)$ be the underlying graph of an infinitary payoff game and let σ^* be an $\trianglelefteq$-optimal strategy. We define the improvement rule SWITCH-LIN as follows:

$$\text{SWITCH-LIN}(\sigma)(v) := \begin{cases} \sigma^*(v) & \text{if } (v, \sigma^*(v)) \in I_\sigma \\ \sigma(v) & \text{otherwise} \end{cases}$$

We show that SWITCH-LIN is indeed an improvement rule and that the strategy iteration parameterized with SWITCH-LIN requires at most $|V_0|$ iterations on G in one go by verifying that

$$M(\sigma) \subsetneq V_0 \qquad \Longrightarrow \qquad M(\sigma) \subsetneq M(\text{SWITCH-LIN}(\sigma))$$

for all σ where $M(\sigma) = \{v \in V_0 \mid \sigma(v) = \sigma^*(v)\}$.

Let σ be a strategy s.t. $M(\sigma) \subsetneq V_0$. Since $M(\sigma) \subseteq M(\text{SWITCH-LIN}(\sigma))$ holds by definition, we simply need to show that there is at least one node $v \in V_0$ with $\sigma(v) \neq \sigma^*(v)$ and $(v, \sigma^*(v)) \in I_\sigma$. Consider the game $G' = (V, F)$ where

$$F = \{(v, w) \in E \mid v \notin V_0 \text{ or } \sigma(v) = w \text{ or } \sigma^*(v) = w\}$$

It is easy to see that σ^* is an $\trianglelefteq$-optimal strategy w.r.t. G'. As σ is not optimal, there must be at least one proper improvement edge $(v, w) \in I_\sigma^{G'}$. By definition of G', it follows that $\sigma(v) \neq w$ and $\sigma^*(v) = w$. □

Corollary 4.3. *The diameter of an infinitary payoff game is linear in the number of nodes in the worst case.*

Counter Strategies

If we have a two- or 2.5-player game, there is another interpretation of the game valuation associated with a given strategy σ. Here, we start with a *profile valuation* that takes a player 0 strategy σ and a player 1 strategy τ and computes an associated *profile valuation* $\Xi_{\sigma,\tau} \in \mathcal{V}$. Recall that it is the goal of player 0 to maximize the valuation of his or her strategies, and player 1 tries to accomplish the converse. Hence, we *define* the game valuation in this context as the worst profile valuation conforming to σ:

$$\Xi_\sigma(v) = \min_{\prec}\{\Xi_{\sigma,\tau}(v) \mid \tau \in \mathcal{S}_1(G)\}$$

Let σ be a player 0 strategy and τ be a player 1 strategy. We say that τ is an *optimal counterstrategy* against σ iff $\tau(v) \preceq_\sigma u$ for every $(v, u) \in E_1$. An important property of policy iteration for two- and 2.5-player games is, that optimal counterstrategies exist.

(SI5) Let σ be a player 0 strategy. There is an optimal counterstrategy τ s.t. $\Xi_\sigma(v) = \Xi_{\sigma,\tau}(v)$ for every node v.

This particularly implies that, given a game valuation, we can find an associated optimal counterstrategy, and we will see that the converse also holds in the concrete settings. Given a player 0 strategy σ, let τ_σ denote an (arbitrary but fixed) optimal counterstrategy against σ.

Discrete Strategy Iteration

In this paragraph, we introduce a concrete policy iteration algorithm for parity games, known as *discrete strategy iteration* due to Jurdziński and Vöge [VJ00]. It can be seen as a refined version of Puri's strategy iteration for discounted payoff games [Pur95] (see below), that can also be used to solve parity games by reduction [ZP96, VJ00]. The advantage of the discrete strategy iteration over the Puri's on parity games is, that discrete policy iteration omits the use of high-precision rational numbers and is therefore much more efficient in practice.

Discrete strategy iteration requires a total ordering $<$, called *relevance ordering*, on all nodes that is consistent with priorities, i.e. $\Omega(v) < \Omega(w)$ implies $v < w$. This is equivalent to assuming that all nodes have different priorities. Recall that every transformation of priorities, that preserves the original ordering of different priorities, has the same winning sets and winning strategies as the original game. For clarity of presentation, we assume therefore that the priority assignment function is injective.

The algorithm is based on profile valuations as defined in the previous paragraph. Let σ be a player 0 strategy, τ be player 1 strategy, and v be a node. Recall that there is exactly one positional play that starts in v and conforms to σ and τ. Such a play can be uniquely written as follows:

$$\pi_{\sigma,\tau,v} = v_1 \dots v_k (w_1 \dots w_l)^\omega$$

with $v_1 = v$, $v_i \neq w_1$ for all $1 \leq i \leq k$ and $\Omega(w_1) > \Omega(w_j)$ for all $1 < j \leq l$. Note that the uniqueness follows from the fact that all nodes on the cycle have different priorities and we choose w_1 to be the node with highest priority.

Discrete strategy iteration relies on a more abstract description of such a play $\pi_{\sigma,\tau,v}$. In fact, we only consider the *dominating cycle node* w_1, the set of *more relevant nodes* – i.e. all v_i with $\Omega(v_i) > \Omega(w_1)$ – on the path to the cycle node, and the *length* k of the path leading to the cycle node. More formally, the profile valuation is defined as follows:

$$\Xi_{\sigma,\tau}(v) := (w_1, \{v_i \mid \Omega(v_i) > \Omega(w_1)\}, k)$$

We refer to w_1 as the *cycle component*, to the second as the *path component*, and to k as the *length component* of the node valuation. In other words, a *node valuation* here, is a triple describing the most important parts of a positional play.

In order to compare node valuations with each other, we introduce a total (lexicographic) ordering on the set of node valuations. For that reason, we need to define a total ordering $\prec$ on the second component of node valuations – i.e. on subsets of V – first. To compare two different sets M and N of nodes, we order all nodes lexicographically according to their priority.

To determine which set of nodes is better w.r.t. $\prec$, one considers the node v with the highest priority that occurs in only one of the two sets. The set containing v is greater than the other if and only if v has even priority.

More formally, we say that a node v is the *most significant difference* between M and N iff $v \in M \triangle N$ and for all other $w \in M \triangle N$ we have $\Omega(v) > \Omega(w)$, where $M \triangle N$ denotes the symmetric difference of both sets.

Let now M and N be different sets and v be the most significant difference. Then define:

$$M \prec N \qquad \text{iff} \qquad v \in N \text{ and } v \text{ even, or } v \in M \text{ and } v \text{ odd}$$

We are now able to extend the total ordering on sets of nodes to node valuations by a lexicographic ordering.

$$(u, M, e) \prec (v, N, f) \;:\Longleftrightarrow\; \begin{cases} (-1)^{\Omega(u)}\Omega(u) < (-1)^{\Omega(v)}\Omega(v), \text{ or} \\ u = v \text{ and } M \prec N, \text{ or} \\ u = v \text{ and } M = N \text{ and } e < f \text{ and } u \text{ odd, or} \\ u = v \text{ and } M = N \text{ and } e > f \text{ and } u \text{ even.} \end{cases}$$

The motivation behind this ordering is a lexicographic measurement of the profitability of a positional play: the most important part of a positional play is the cycle in which the play eventually ends in, and here, it is the priority of the dominating cycle node that defines the profitability for player 0. The term $(-1)^{\Omega(u)}\Omega(u)$ is known

as the *reward* of a node u, and essentially results in the following ordering on the priorities:

$$\ldots < 7 < 5 < 3 < 1 < 0 < 2 < 4 < 6 < \ldots$$

The second important part is the loopless path that leads to the dominating cycle node. Here, we measure the profitability of a loopless path by a *lexicographic* ordering on the *relevancy* of the nodes on path, applying the *reward* ordering on each component in the lexicographic ordering. Finally, we consider the length, and the intuition behind the definition is that, assuming we have an even-priority dominating cycle node, it is better to reach the cycle fast whereas it is better to stay as long as possible out of the cycle otherwise.

Game valuations, improving switches and counterstrategies are exactly defined as before. Particularly, one can think of game valuations here as follows: for a fixed strategy σ of player 0 and a node v, the associated valuation essentially states which is the worst cycle that can be reached from v conforming to σ as well as the worst loopless path leading to that cycle (also conforming to σ).

Theorem 4.4 ([VJ00, Vög00])**.** *Assume parity game context.*

(SI1) The game valuation of a strategy is polynomial-time computable.

(SI2) There is an $\trianglelefteq$-optimal strategy σ.

(SI3) If $I_\sigma = \emptyset$ then σ is optimal.

(SI4) Let $I \subseteq I_\sigma$ be a non-empty applicable set. Then $\sigma \lhd \sigma[I]$.

(SI5) Let σ be a player 0 strategy. There is an optimal counterstrategy τ s.t. $\Xi_\sigma(v) = \Xi_{\sigma,\tau}(v)$ for every node v.

The set of improving switches can now be determined without evaluating $\sigma[e]$ for every switch e. In fact, we have the following theorem:

Theorem 4.5 ([VJ00, Vög00])**.** *Let σ be a strategy. Then:*

$$I_\sigma = \{(v, w) \in E_0 \mid \sigma(v) \prec_\sigma w\}$$

It remains to show how an optimal strategy coincides with winning sets W_i and winning strategies for both players in the parity game. First, we define winning sets w.r.t. a given strategy σ as follows:

$$W_i(\sigma) = \{v \mid \Xi_\sigma(v) = (w, \_, \_) \text{ and } \Omega(w) \equiv_2 i\}$$

We have the following theorem that allows us to derive a lower bound on the winning set of player 0, given an arbitrary strategy σ:

Theorem 4.6 ([VJ00, Vög00])**.** *Let G be a parity game and σ be a player 0 strategy. Then $W_0(\sigma) \subseteq W_0$ and σ is a winning strategy on $W_0(\sigma)$.*

A similar theorem holds true for player 1 if we have found an optimal strategy for player 0.

Theorem 4.7 ([VJ00, Vög00])**.** *Let G be a parity game and σ be an optimal player 0 strategy. Then $W_1(\sigma) \subseteq W_1$ and τ_σ is a winning strategy on $W_1(\sigma)$.*

Corollary 4.8. *Let G be a parity game and σ be an optimal player 0 strategy. Then σ is a winning strategy on $W_0 = W_0(\sigma)$, and τ_σ is a winning strategy on $W_1 = W_1(\sigma)$.*

We note that there are discrete strategy iteration variants that handle nodes with the same priority differently. Instead of introducing an arbitrary ordering on such nodes, it is also possible to value them equally and use multisets for the path component along with a similar lexicographic ordering. See, for instance, [Sch08].

We conclude this paragraph with the question whether we can solve single-player parity games in a polynomial number of iterations. To answer that question, we define the most natural multi-switch rule, called SWITCH-ALL here (and discuss it in detail later).

SWITCH-ALL:	Apply the best local improvement in every node simultaneously.

More formally, it holds for every strategy σ, every player 0 node v and every $w \in vE$ that $w \preceq_\sigma \text{SWITCH-ALL}(\sigma)(v)$.

We have the following theorem that tells us that one-player parity games can be solved efficiently by strategy iteration:

Theorem 4.9 ([Vög00])**.** *Let G be an one-player parity game. Policy iteration with the* SWITCH-ALL *rule requires polynomially many iterations in the worst case.*

Example of Discrete Strategy Iteration

Discrete strategy iteration is understood best with a concrete example. Consider therefore the parity game of Figure 4.1.

Policy iteration starts with an initial strategy σ_0. Let that be $\sigma_0(4) = 0$ and $\sigma_0(6) = 8$. The reader can check that the best cycle for player 1 is $(3, 2, 0)^\omega$ with dominating cycle node 3, since the only higher odd priority is 5, which is not dominating on any cycle. We see that player 1 can even force every play to end up in the cycle dominated by 3, hence, the cycle component of every node will be 3. Particularly, the counterstrategy τ_0 is $\tau_0(1) = 4$, $\tau_0(0) = 3$, $\tau_0(2) = 0$, and $\tau_0(8) = 2$. See Figure 4.1 for a graphical depiction of that setting; bold edges indicate the strategies.

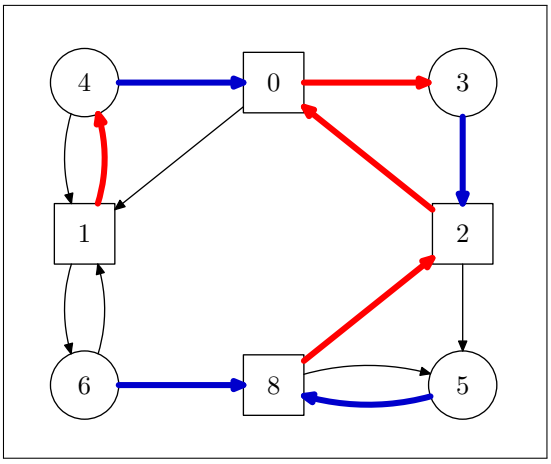


Figure 4.1: Discrete Iteration Example, Strategy 0

The positional paths and valuations of nodes 1 and 0 are, for instance, as follows:

$$\pi_{\sigma_0,\tau_0,1} = 1, 4, 0, (3, 2, 0)^\omega \qquad \Xi_{\sigma_0}(1) = (3, \{4\}, 3)$$

$$\pi_{\sigma_0,\tau_0,0} = 0, (3, 2, 0)^\omega \qquad \Xi_{\sigma_0}(8) = (3, \emptyset, 1)$$

There is a single improving edge here, namely $(4, 1)$. By Theorem 4.5, we know that it suffices to compare the valuation of 0 with the valuation of 1 to determine whether $(4, 1)$ is an improving switch. As both valuations have the same cycle component, we compare the two path components. The most significant difference is 4, which is an even priority, and hence, the path component containing the 4 is better.

Therefore, we switch to $\sigma_1 = \sigma_0[(4, 1)]$. The reader can check that the cycle components remain the same in the new counterstrategy τ_1, however, player 1 is forced to alter the choice $\tau_0(1) = 4$ to $\tau_1(1) = 6$, i.e. we have $\tau_1 = \tau_0[(1, 6)]$. See Figure 4.2 for a graphical depiction of that setting.

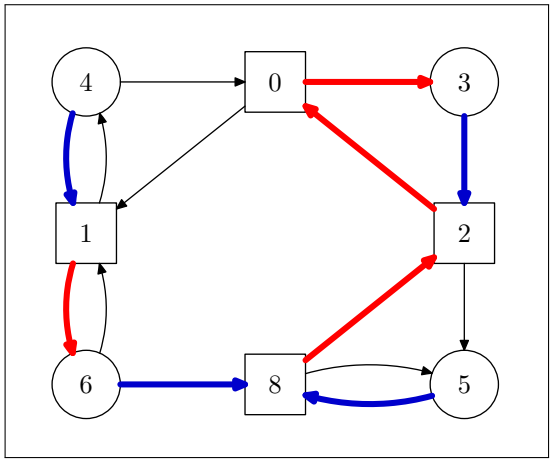


Figure 4.2: Discrete Iteration Example, Strategy 1

The positional paths and valuations of nodes 1 and 8 are, for instance, as follows:

$$\pi_{\sigma_0,\tau_0,1} = 1, 6, 8, 2, 0, (3, 2, 0)^\omega \qquad \Xi_{\sigma_0}(1) = (3, \{8, 6\}, 5)$$

$$\pi_{\sigma_0,\tau_0,8} = 8, 2, 0, (3, 2, 0)^\omega \qquad \Xi_{\sigma_0}(8) = (3, \{8\}, 3)$$

There is a single improving edge here, namely $(6, 1)$, as the valuation of 1 is better than the valuation of 8, because on the path to the dominating cycle node 3, we have priority 6 additionally on the path starting with 1.

Therefore, we switch to $\sigma_2 = \sigma_1[(6, 1)]$. The reader can check that player 1 now has lost the ability to reach 3 from 4, 1 and 6, see Figure 4.3.

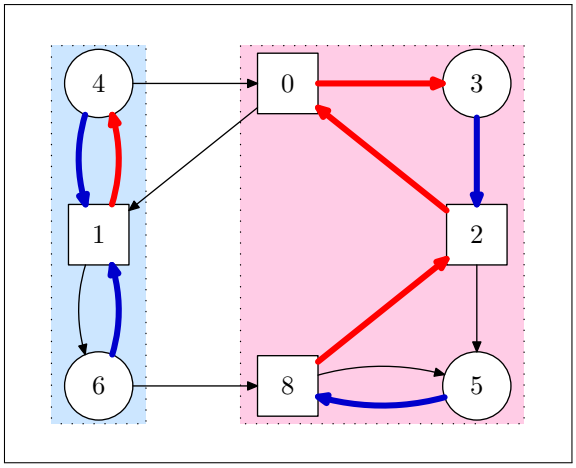


Figure 4.3: Discrete Iteration Example, Strategy 2

Particularly, there are only even-priority dominated cycles on the left side of the game. It is not hard to see that there are no improving switches anymore, hence, σ_2 is optimal, hence, a winning strategy on the left part of the game and τ_2 is a winning strategy for player 1 on the right part of the game.

Strategy Iteration on Payoff Games

Let G_λ be a discounted payoff game (including stochastic ones and therefore including Markov decision processes). Policy iteration algorithms in this context were first developed by Howard [How60] who used them to solve infinite-horizon MDPs. They were adapted for the solution of two-player games by many researchers, see, e.g., [HK66],[Con92],[Pur95].

Again, we need to explain how to obtain a game valuation, given a player 0 strategy σ. We use the value function $\mathcal{V}[R_\lambda]_\sigma$ to define game valuations, i.e. we use $\Xi_\sigma := \mathcal{V}[R_\lambda]_\sigma$, and the set of node valuations is $(\mathbb{R}, <)$.

Theorem 4.10 ([How60, Pur95, Con92])**.** *Assume (stochastic) discounted payoff game context.*

(SI1) The game valuation of a strategy is polynomial-time computable.

(SI2) There is an $\trianglelefteq$-optimal strategy σ.

(SI3) *If $I_\sigma = \emptyset$ then σ is optimal.*

(SI4) *Let $I \subseteq I_\sigma$ be a non-empty applicable set. Then $\sigma \lhd \sigma[I]$.*

(SI5) *Let σ be a player 0 strategy. There is an optimal counterstrategy τ s.t. $\Xi_\sigma(v) = \Xi_{\sigma,\tau}(v)$ for every node v.*

Particularly note that *optimality* in the context of policy iteration means that an optimal strategy σ and a corresponding optimal counterstrategy secure the value of the game in the sense of Chapter 3.3.

Also note that game valuations can be computed in polynomial time by Theorem 3.19. We note that they can even be computed in strongly polynomial time by applying the algorithm of Madani, Thorup and Zwick [MTZ10].

An appealing feature of policy iteration algorithms is again that determining whether e constitutes an improving switch with respect to σ can be done *without* evaluating $\sigma[e]$. An edge $e = (v, u)$ is an improving switch if and only if $(1 - \lambda)r(v, u) + \lambda\,\Xi_\sigma(u) > \Xi_\sigma(v)$.

We next move from discounted reward payoff games to mean payoff games, i.e. payoff with the limiting average reward criterion. We can define game valuations as before by simply using the associated value. Unfortunately, in the non-discounted case, it does not hold in general that if σ is not optimal then there exist at least one switch that *strictly* improves the value. To remedy the situation, we need to define *potentials* as follows:

For concreteness, assume first that the payoff game is deterministic. Let σ and τ be strategies of players 0 and 1. Let $v_0 v_1 \ldots$ be the infinite path that conforms to σ and τ. We define the *value* $\textsc{val}_{\sigma,\tau}(v_0)$ of this play to be $\lim_{n\to\infty} \frac{1}{n} \sum_{j=0}^{n-1} r(v_j, v_{j+1})$. The infinite path $v_0 v_1 \ldots$ is composed of finite path P leading to a cycle $C = u_0 u_1 \ldots u_k$, where $u_k = u_0$, which is repeated an infinite number of times, and $\textsc{val}_{\sigma,\tau}(v_0)$ is simply the average reward $\frac{1}{k} \sum_{j=0}^{k-1} r(u_j, u_{j+1})$ of the cycle C.

Let $u = u(C)$ be a fixed vertex on the cycle C, e.g., the vertex with the smallest index. Let $v_0 v_1 \ldots v_\ell = u$ be the finite prefix of the infinite path $v_0 v_1 \ldots$ that ends with the first visit to u. We define the *potential* $\textsc{pot}_{\sigma,\tau}(v_0)$ to be $\sum_{j=0}^{\ell-1}(r(v_j, v_{j+1}) -$

$\text{VAL}_{\sigma,\tau}(v_0)$), i.e., the total reward on the path leading to u, when $\text{VAL}_{\sigma,\tau}(v_0)$ is subtracted from the reward of each edge. Finally, we define the *valuation* $\Xi_{\sigma,\tau}(v_0)$ to be the pair $(\text{VAL}_{\sigma,\tau}(v_0), \text{POT}_{\sigma,\tau}(v_0))$. We compare valuations *lexicographically*, i.e., $\Xi_{\sigma,\tau}(v_0) \prec \Xi_{\sigma',\tau'}(v_0)$ if and only if $\text{VAL}_{\sigma,\tau}(v_0) < \text{VAL}_{\sigma',\tau'}(v_0)$, or $\text{VAL}_{\sigma,\tau}(v_0) = \text{VAL}_{\sigma',\tau'}(v_0)$ and $\text{POT}_{\sigma,\tau}(v_0) < \text{POT}_{\sigma',\tau'}(v_0)$.

With these slightly more complicated valuations, Theorem 4.10 becomes valid again, and policy iteration algorithms can therefore be used to solve deterministic mean payoff games.

Note the definition of values and potentials can be generalized to the stochastic setting. For instance, in the context of general Markov decision processes, the *values* $\text{VAL}_\sigma(u)$ and *potentials* $\text{POT}_\sigma(u)$ of the vertices under σ are defined as the unique solutions of the following set of linear equations:

$$\text{VAL}_\sigma(u) = \begin{cases} \text{VAL}_\sigma(v) & \text{if } u \in V_0 \text{ and } \sigma(u) = v \\ \sum_{v:(u,v)\in E_R} p(u,v)\,\text{VAL}_\sigma(v) & \text{if } u \in V_R \end{cases}$$

$$\text{POT}_\sigma(u) = \begin{cases} r(u,v) - \text{VAL}_\sigma(v) + \text{POT}_\sigma(v) & \text{if } u \in V_0 \text{ and } \sigma(u) = v \\ \sum_{v:(u,v)\in E_R} p(u,v)\,\text{POT}_\sigma(v) & \text{if } u \in V_R \end{cases}$$

together with the condition that $\text{POT}_\sigma(u)$ sum up to 0 on each irreducible recurrent class of the Markov chain defined by σ.

Complexity Status

It is not hard to see that the policy iteration technique allows to show that solving infinitary payoff games is in PLS. The instances are the game instances, the sets of associated solutions are the positional strategies of player 0, the neighborhood of a positional strategy is the set of strategies that is obtained by switching an applicable non-empty subset of switches, and the cost function assigns a real-valued representation of the valuation to a strategy.

For discounted reward infinitary payoff games, a real-valued representation of the valuation can be obtained by $\sum_{v\in V} \Xi_\sigma(v)$. We already know that all other infinitary

payoff games can be reduced to discounted reward infinitary payoff games, hence, there is a real-valued representation for all other kinds of game classes as well.

For the inclusion in PLS, it remains to check that an initial strategy can be selected in polynomial time (trivial), that a valuation can be computed in polynomial time (axiom (SI1)), that we can decide whether $I_\sigma = \emptyset$ in polynomial time, and that we can compute an improved strategy otherwise in polynomial time (easy, as we can determine the set of improving switches efficiently).

Theorem 4.11. *Solving infinitary payoff games is in* PLS.

For the inclusion in UP and coUP, we need to see that the pre-order on strategies can be (artificially) extended to a partial ordering on strategies, because then, there is exactly one optimal strategy. Recall that we have a total ordering on node valuations, and that we extended it to a pre-order on nodes by:

$$v \preceq_\sigma u \qquad :\iff \qquad \Xi_\sigma(v) \preceq \Xi_\sigma(u)$$

But we can also extend this to a total ordering on nodes by breaking ties between two nodes arbitrarily, for instance by comparing their indices. In fact, some authors define strategy iteration this way right from the start [VJ00].

Theorem 4.12. *Solving infinitary payoff games is in* UP $\cap$ coUP.

Abstractions

The policy iteration technique allows to cast the problem of solving infinitary payoff games as LP-type problem [Hal07]. Recall that an LP-type problem is a pair (H, ω), where H is the *set of constraints* and $\omega : 2^H \to \mathbb{R} \cup \{\pm\infty\}$ is the *objective function*, that maps every subset of constraints $A \subseteq H$ to an element $\omega(A) \in \mathbb{R} \cup \{\pm\infty\}$.

Given an infinitary payoff game G, we define $H = E_0$ to be the set of player 0 controlled edges. Next, we need to define the objective function ω for every $A \subseteq E_0$ as follows: We say that A is *proper* iff for every $v \in V_0$ there is some $w \in vE$ s.t. $(v, w) \in A$. For every proper A, let σ_A denote a player 0 strategy that is optimal

in the game $G|_A$, i.e. in the subgame in which player 0 can only use edges from A. We define the objective function, using a real-valued representation of the valuation again, by

$$\omega(A) := \begin{cases} \sum_{v \in V} \Xi_{\sigma_A}(v) & \text{if } A \text{ is proper} \\ -\infty & \text{otherwise} \end{cases}$$

It is easy to see that the monotonicity and locality conditions are satisfied by this definition.

There is another abstraction of policy iteration called *acyclic unique sink orientations* (AUSOs), that corresponds to infinitary payoff games with out-degree limited by two. For information on AUSOs, see [SW01, Gär02, GS06]. The problem of finding the optimal strategy in an infinitary payoff game with n nodes is modeled by a directed graph in which vertices correspond to strategies and edges correspond to an (applicable) multi-switch. There is also a generalization of AUSOs for the non-binary case, called *grid unique sink orientations* (GridUSOs), see [GMR08].

Comparison to the Simplex Algorithm

The simplex algorithm for solving linear programs and the policy iteration method for solving infinitary payoff games share several properties. Both are fixpoint iteration algorithms, their complexity depends crucially on the number of iterations, there are improvement resp. pivoting rules that determine in every iteration how to proceed, and they can be abstracted to LP-type problems or polynomial local search.

But there are also important differences. In infinitary payoff game policy iteration, we have the *comparability* of switches, meaning that we are allowed to apply any edge change to a strategy and obtain a comparable strategy, i.e. $\sigma \trianglelefteq \sigma[e]$ or $\sigma[e] \trianglelefteq \sigma$ for every strategy σ and every player 0 controlled edge e. Switching an improving *edge* (v, w) is realized by replacing $(v, \sigma(v))$ in the strategy by (v, w).

In linear programming, this corresponds to letting an improving non-basic variable enter the basis while a basic variable, that has been reduced to zero, leaves the basis. However, variables cannot be partitioned in a non-trivial way such that the pair of entering and leaving variable is always contained in one set of the partition.

In policy iteration, such a partition is trivially given by grouping together edges with the same source node.

This difference has important consequences. First, we cannot prove directly that the diameter of a linear program is small, while that is an easy proof for infinitary payoff games. This is a major open problem and known as the Hirsch conjecture. Second, if we have more than one improving variable in a linear program, it is not true in general that applying multiple variables at once results in a basic feasible solution with improved cost.

Nevertheless, many improvement rules for infinitary payoff game policy iteration can be cast as pivoting rules for the simplex algorithm and vice versa. The notion of a *single improving switch* then corresponds to an improving edge in the domain of policy iteration and to an improving variable in the domain of the simplex algorithm.

4.2 Improvement Rules

We give a brief introduction into all important improvement rules and their differences. We outline known upper and lower bound results, and include our contribution to the latter matter.

Improvement rules can be generally classified according to four properties. Some of these properties have immediate consequences for the applicability of the respective improvement rule, i.e. in which policy iteration contexts the rule is eligible to be applied.

1. First, there is the *switching* property, specifying whether the improvement rule is allowed to apply more than one improving switch at at time. Rules that apply exactly one improving switch are called *single-switching* rules, and otherwise *multi-switching* rules.

   Note that multi-switching rules only apply to infinitary payoff games, but not to the simplex algorithm for linear programming. Recall that we are only allowed to let one variable enter the basis, which corresponds to performing a single switch.

2. Second, there is the *method* of obtaining the improving switches that are to be applied. We say that an improvement rule is *combinatorial* if it only considers the set of improving switches and potentially the *ordering* of the current valuation of the single nodes. Otherwise, we call the rule *structurally involved*.

   Note that combinatorial improvement rules are favorable, because they apply almost immediately to all considered settings (infinitary payoff games, linear programming), or even to policy iteration settings that have not been discovered yet. Structurally involved rules, on the other hand, usually rely heavily on the specific structure of the problem class they are tied for.

3. Third, there is the computational *model* of selecting improving switches, namely *deterministic* or *probabilistic* rules. We will see that it makes a huge difference in obtaining lower bounds when dealing with probabilistic instead of deterministic rules.

4. Fourth, there is the *memory* property, specifying whether the improvement rule manages additional data structures that are used to store information about the history of the policy iteration run. An improvement rule that requires additional persistent memory is called *memorizing* or *history-based*, and otherwise *oblivious* improvement rule.

The complexity of the simplex algorithm and the policy iteration algorithm is directly related to the number of iterations it requires to find the optimum. Hence, the complexity relies greatly on the applied improvement rule.

Upper bounds for an improvement rule are usually obtained by *combinatorial arguments*, and preferably in the most abstract setting possible, that is, formulated in the AUSO or LP-type world. By proving an upper bound in the most abstract setting possible, the bound immediately transfers to all subsumed concrete settings. For instance, an upper bound in the LP-type problem world directly transfers to infinitary payoff games and linear programming problems. Note that we do not contribute upper bounds in this thesis.

Lower bounds, are usually obtained by *explicit constructions*, and preferably in the least abstract setting possible, that is, formulated in the parity game or Markov

decision process world. By proving a lower bound in the least abstract setting possible, it (more or less) immediately transfers to all other settings by which it is subsumed. For instance, a lower bound in the parity game world transfers to the payoff game world under some circumstances (as we will see).

On the other hand, a lower bound in the AUSO world does not relate to any concrete setting, i.e. if we have an exponential lower bound for a particular improvement rule in the abstract setting of the AUSO world, we cannot transfer this result to, for instance, parity games. In other words, it is still possible that a certain improvement rule solves parity games in polynomial time that requires exponential time in a more abstract setting.

Note that no subexponential or exponential lower bounds in concrete settings for any of the considered improvement rules have been known before this work. The only known lower bounds have been formulated in abstract settings.

In the following, we describe the important improvement rules of the literature, the known results and our contributions. We start with deterministic rules, followed by probabilistic rules, and conclude the overview with history-based rules.

Deterministic Rules

An example has been known for some time for which a sufficiently poor choice of a deterministic single-switch rule causes an exponential number of iterations of the strategy improvement algorithm [BV07]. However, there is no particularly clever way to define a deterministic oblivious single-switching rule, which is why they are not considered to be a good candidate for a polynomial-time admitting improvement rule. A similar observation holds true for the simplex algorithm [KM72].

The SWITCH-ALL or *locally optimizing rule* (see Table 4.1) is generally considered to be the most natural choice for an improvement rule. Consider the case in which all player 0 nodes have at most out-degree two. This implies that a node either has no improving edge or exactly one improving edge, namely the one which is not chosen by the current strategy.

This improvement rule is obviously a multi-switching rule and therefore not applicable for linear programs.

SWITCH-ALL	
Authors	appears all over the literature
Description	Apply every improving edge simultaneously.
Properties	multi-switching, combinatorial, deterministic, oblivious
Applicability	all infinitary payoff games
Upper Bound	$2^n/n$ (Mansour & Singh [MS99])
Abs. Lower Bound	$2^{n/2}$ (Schurr & Szabó [SS05])
Con. Lower Bound	$2^{\Omega(n)}$ (Friedmann)

Table 4.1: Summary of the Switch All improvement rule

SWITCH-BEST	
Authors	Schewe [Sch08]
Description	Apply the best possible combination of switches.
Properties	multi-switching, structurally involved, deterministic, oblivious
Applicability	all deterministic infinitary payoff games
Upper Bound	1.72^n (Schurr & Szabó [SS05])
Abs. Lower Bound	$2^{n/2}$ (Schurr & Szabó [SS05])
Con. Lower Bound	$2^{\Omega(n)}$ (Friedmann)

Table 4.2: Summary of the Switch Best improvement rule

The SWITCH-BEST or *globally optimizing rule* [Sch08] (see Table 4.2) computes a globally optimal successor strategy in the sense that the associated valuation is the best under all allowed successor strategies. The main difference between the locally optimizing policy and the globally optimizing policy is that the latter takes

cross-effects of improving switches into account. It is aware of the impact of any combination of profitable edges, in contrast to the locally optimizing policy that only sees the local valuations, but not the effects.

However, this rule relies on the explicit structure of the problem and is formulated for parity games and deterministic payoff games.

We present an explicit construction of parity games on which SWITCH-ALL and SWITCH-BEST policy iteration require exponential time, and relate the results to the other classes of infinitary payoff games.

Probabilistic Rules

Kalai [Kal92, Kal97] and Matoušek, Sharir and Welzl [MSW96] devised *randomized* pivoting rules (see Table 4.3) that never require more than an expected *subexponential* number of pivoting steps to solve any linear program. Their algorithms can, in fact, be used to solve a more general class of problems, particularly infinitary payoff games.

RANDOM-FACET

Authors	Kalai [Kal92, Kal97]; Matoušek, Sharir & Welzl [MSW96]
Description	Compute optimal strategy by recursive exclusion of unused single edges.
Properties	single-switching, combinatorial, probabilistic, oblivious, recursive
Applicability	all infinitary payoff games, linear programming
Upper Bound	$2^{\mathcal{O}(\sqrt{n})}$ (Kalai [Kal92])
Abs. Lower Bound	$2^{\Omega(\sqrt{n})}$ (Matoušek [Mat94])
Con. Lower Bound	$2^{\Omega(\sqrt{n}/\log(n))}$ (Friedmann, Hansen, Zwick)

Table 4.3: Summary of the Random Facet improvement rule

The improvement rule is formulated as an *recursive* optimization algorithm without any additional memory. It is well-known that recursion can be simulated by iteration *with memory* – here, of course, by policy fixpoint iteration. In other words, if we are strict, we could say that this improvement rule is memorizing, however, in the recursive formulation, it is oblivious.

Perhaps the most natural randomized pivoting rule is RANDOM-EDGE (see Table 4.4), which among all improving switches chooses one uniformly at random. The upper bounds currently known for RANDOM-EDGE are still exponential (see Gärtner and Kaibel [GK07], for additional results regarding RANDOM-EDGE, see [BDF+95, GHZ98, GTW+03, BP07]). RANDOM-EDGE is also applicable in a much wider abstract setting. Matoušek and Szabó [MS06] showed that it can be subexponential on AUSOs.

RANDOM-EDGE

Authors	appears all over the literature
Description	Apply a single improving switch arbitrarily at random.
Properties	single-switching, combinatorial, probabilistic, oblivious
Applicability	all infinitary payoff games, linear programming
Upper Bound	–
Abs. Lower Bound	$2^{\Omega(\sqrt[3]{n})}$ (Matoušek & Szabó [MS06])
Con. Lower Bound	$2^{\Omega(\sqrt[4]{n})}$ (Friedmann, Hansen, Zwick)

Table 4.4: Summary of the Random Edge improvement rule

We give an explicit constructions of (different) parity games and Markov decision processes on which RANDOM-FACET and RANDOM-EDGE policy iteration require subexponential time, and relate the results to the other classes of infinitary payoff games and to linear programming.

Another important randomized improvement rule is SWITCH-HALF [MS99] (see Table 4.5), which applies every improving switch with probability $1/2$, assuming the binary case, i.e. in which every node has out-degree limited by two.

SWITCH-HALF	
Authors	Mansour, Singh [MS99]
Description	Apply every improving switch with probability $1/2$.
Properties	multi-switching, combinatorial, probabilistic, oblivious
Applicability	all infinitary payoff games
Upper Bound	1.72^n (Mansour & Singh [MS99])
Abs. Lower Bound	–
Con. Lower Bound	$2^{\Omega(\sqrt[4]{n})}$ (Friedmann, Hansen, Zwick)

Table 4.5: Summary of the Switch Half improvement rule

We explain how the lower bound construction for RANDOM-EDGE transfer to all (non-recursive) oblivious randomized multi-switch improvement rules.

Memorizing Rules

There is one famous memorizing improvement rule that has entered the folklore of convex optimization. Also known as the LEAST-ENTERED rule (see Table 4.6), Zadeh's pivoting method [Zad80] belongs to the family of memorizing improvement rules, which among all improving switches chooses one which has been switched least often.

We give an explicit construction of parity games and Markov decision processes on which LEAST-ENTERED policy iteration requires subexponential time, and relate the results to the other classes of infinitary payoff games and to linear programming.

LEAST-ENTERED	
Authors	Zadeh [Zad80]
Description	Apply a switch that has been switched least often.
Properties	single-switching, combinatorial, deterministic, memorizing
Applicability	all infinitary payoff games, linear programming
Upper Bound	–
Abs. Lower Bound	–
Con. Lower Bound	$2^{\Omega(\sqrt{n})}$ (Friedmann)

Table 4.6: Summary of the Least Entered improvement rule

4.3 Lower Bound Proof Plan

We give the reader a complete outline of our proof technique for lower bounds in this chapter, and go through all major steps from a high-level point of view. All lower bound constructions are based on the following steps:

1. We construct a family of parity games that provides a lower bound for the respective improvement rule. We restrict ourselves in the construction to a very special form of parity games, called *sink parity games*, which has no disadvantages (as we will see) when trying to construct (sub)exponential lower bounds. Also, we try to use player 1 as rarely as possible.

   We think of parity game strategy iteration as a deterministic (due to the deterministic nature of both players) computational model in which we can implement different functional structures. All lower bound constructions are based on the implementation of a variant of a binary counter.

   Proving the constructions correct is then equal to showing that the sequence of strategies simulates the binary counter, or at least counts "good enough" with high probability when applying randomized pivoting rules.

2. We transfer the lower bound result for parity games to more expressive game classes like mean payoff games, discounted payoff games, turn-based stochastic games and the like.

   We show in general that policy iteration on sink parity games behaves exactly like policy iteration on the more expressive game classes, when the sink parity game is reduced to them by the standard reductions of Chapter 3.4. We prove this correspondence independently of the applied improvement rule.

3. We transfer the lower bound results to Markov decision processes. Unfortunately, there is no standard reduction from parity games to MDPs.

   In order to obtain MDPs from our parity games, we need to get rid of player 1. However, we know already that player 1 is essential for obtaining (sub)exponential lower bounds, see Lemma 4.9.

   Can we use the randomization player of MDPs to simulate the behavior of player 1? This does not seem to be possible in general. However, as we will see, we use player 1 only in a very special role, and therefore can replace player 1 by the randomization player.

   We failed to prove that this translation works in general, i.e. independently of the applied improvement rule. Therefore, we show that the translation from parity games to Markov decision processes operates as desired for every construction once again.

4. We transfer our lower bounds to the simplex algorithm for solving linear programs. Here, we show in general that the the simplex algorithm on linear programs, that are induced by our lower bound Markov decision processes, behaves exactly the same as policy iteration on the original games.

We would like to stress that most of our intuition about the lower bound constructions was obtained by thinking in terms of parity games. Thinking in terms of MDPs seems harder, and we doubt whether we could have obtained our results by thinking directly in terms of linear programs.

Parity Game Strategy Iteration

The reason why parity games seem to be the most appropriate class of games, when trying to construct a worst-case family for any class of infinitary payoff games, is, that the effect of each node in a parity game is immediate: a higher priority dominates all lower priorities (in a play), no matter how many there are.

A similar observation holds true for discrete strategy iteration as well, with its seemingly artificial structure of node valuations, that splits the positional play associated with the current strategy σ and the best response counterstrategy τ_σ into three components of decreasing importance: the dominating cycle node, the (more relevant nodes on the) path leading to the cycle, and the length of the path.

Consider the node valuations from a complexity theorist's point of view. There are only linearly many different values for the first and third component, while there are exponentially many for the second. Suppose that we have a run of the strategy iteration algorithm of exponential length. This particularly implies that we need to have a partial run of exponential length in which the cycle component never changes (this observation is similar to the pigeonhole principle).

It follows from a designer's point of view that there is no real benefit in actually using different cycle nodes. Hence, our basic layout of a game exploiting exponential behavior consists of a complex structure leading to one single loop – the only cycle node that will occur in valuations. In this setting, the strategy iteration algorithm is just improving the paths leading to the cycle node. In other words, we can forget about the first component of valuations.

Now consider the third component, the length of the path leading to the dominating cycle node. It essentially measures the number of nodes on the path that are less relevant than the dominating cycle node. The fewer less relevant nodes than the single dominating cycle node we have, the more nodes can be included in the path component, which seems to be better for the design of games that enforce exponentially many iterations. In fact, we will assign the *least* priority in the game to the dominating cycle node, implying that there are *no* nodes that are less relevant.

This particularly implies that the length component equals the cardinality of the path components all the time, and hence, we can forget about the length component

as well. In other words, we design games in which we only care about the path component, which contains all nodes on the path to the single dominating cycle node. The priority of the single loop that is used in the games will be 1. We will call these games *sink parity games*.

There is another reason why sink parity games are particularly favorable: we can show in this context that policy iteration for more expressive infinitary payoff games, like payoff games, behaves exactly the same as policy iteration for sink parity games, when we reduce them to payoff games. This allows us to transfer lower bounds for sink parity game policy iteration immediately to policy iteration for more expressive infinitary payoff games.

We can now ask ourselves how a subexponential lower bound construction has to look like. By Theorem 4.9, we know already that we need two players, at least for some improvement rules. Consider why one-player parity games can be solved in polynomially many iterations. In fact, they can be solved in a linear number of iterations in the setting of sink parity games. All the algorithm performs here is acyclic path optimization. There can be other cycles in the game besides the single loop that we end up in, but they will not be used (by assumption that we have a sink parity game), which means that there will be no improving edge to close another cycle.

Main Gadget

Although strategy iteration on sink parity games seems to be acyclic path optimization, this is not entirely correct. Consider a strategy σ, an improved strategy σ', and the corresponding optimal counterstrategies τ and τ'. Obviously, $G|_{\sigma,\tau}$ and $G|_{\sigma',\tau'}$ have the same single cycle, since G is a sink parity game. However, if we consider $G|_{\sigma',\tau}$, i.e. the game conforming to the improved strategy σ' and the old counterstrategy τ, it might be the case that we encounter *intermediate cycles*.

Such intermediate cycles need to have dominating cycle nodes that give a better reward to player 0, as player 1 denies staying in these cycles with τ' leading to the original dominating cycle node again.

The main gadget that is used to enforce exponentially many iterations in every lower bound construction, exploits intermediate cycles, and is called the *simple cycle gadget*, see Figure 4.4.

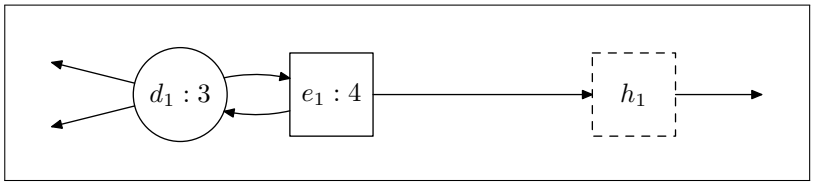


Figure 4.4: Simple Cycle

First, note that the cycle is dominated by player 0, as the highest priority on the cycle is 4. If player 0 decides to use the edge (d_1, e_1), i.e. to move into the cycle, then player 1 will use as counterstrategy the *escape edge* (e_1, h_1) in order to avoid staying in the player 0 dominated cycle with priority 4. However, if player 0 decides to use an edge going out of the cycle, it might be the case that player 1 decides to use the edge (e_1, d_1) in the counterstrategy.

We can use this structure to *hide* the effect of the outgoing player 1 edge (e_1, h_1). As long as player 0 is pointing out of the cycle, strategy iteration is not able to "see" the valuation associated with the node h_1 by looking at e_1 (when player 1 is using the edge (e_1, d_1)).

Let M be the path component of the valuation of d_1 pointing out of the cycle and let N be the path component of the valuation of e_1, assuming that e_1 is pointing to d_1. Then $N = \{e_1\} \cup M$, i.e. (d_1, e_1) is an improving edge, but the local improvement is very small, namely only $e_1 \in N \triangle M$. However, by moving to e_1, the new valuation of e_1 actually will be $\{e_1\} \cup Q$ where Q is the path component of h_1 (as player 1 is forced to leave the cycle).

Assume now that we use the standard improvement rule SWITCH-ALL that applies the best local improvement in every node simultaneously. Consider the example of Figure 4.5; bold edges indicate the current strategy of player 0 and the associated counterstrategy. Assume further that d, e, and the single sink parity game loop have the lowest priorities in the whole game.

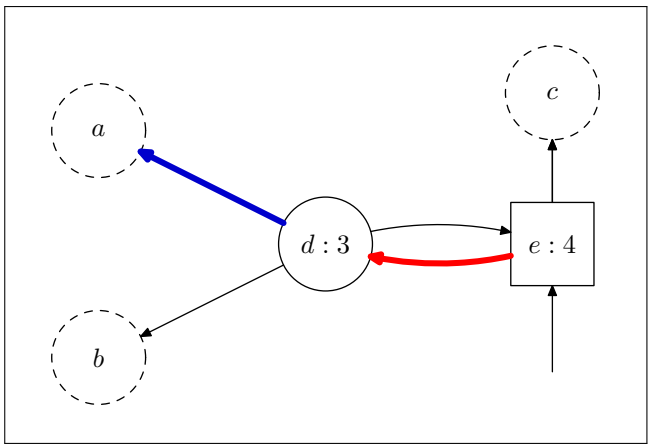


Figure 4.5: Simple Cycle Example, Strategy 0

We assume that $a \prec_\sigma b$, and that $b \prec\!\!\prec c$, i.e. c has a much butter valuation than b. Obviously, $a \prec_\sigma e$, but only because of the very low priority 4. The effect of c cannot be observed by player 0 at the moment by considering the valuations of the neighboring nodes. Hence, we have $e \prec_\sigma b$, and apply the improving switch (d, b) by the SWITCH-ALL rule, resulting in Figure 4.6.

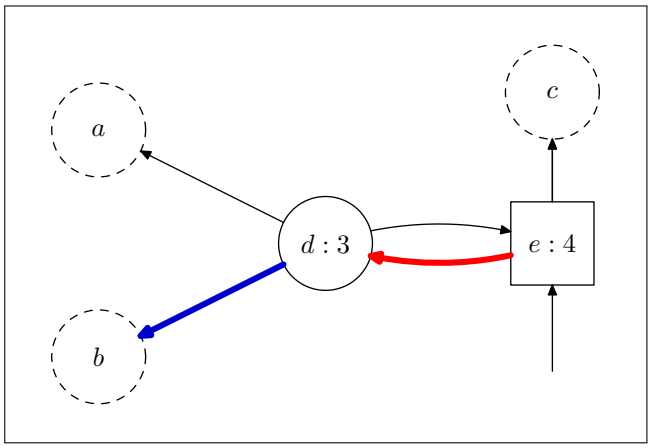


Figure 4.6: Simple Cycle Example, Strategy 1

It might be the case that now $b \prec_\sigma a$, which would lead strategy iteration to end up in Figure 4.5 again. In other words, we can postpone the update of player 0 to move into the cycle for as long as we can provide external nodes like a and b with (slightly) improved valuations in each iteration. The profitability of c can be arbitrarily high.

Finally assume that (d, e) is the only remaining improving edge. Then, we would end up in Figure 4.7, forcing player 1 to finally leave the cycle.

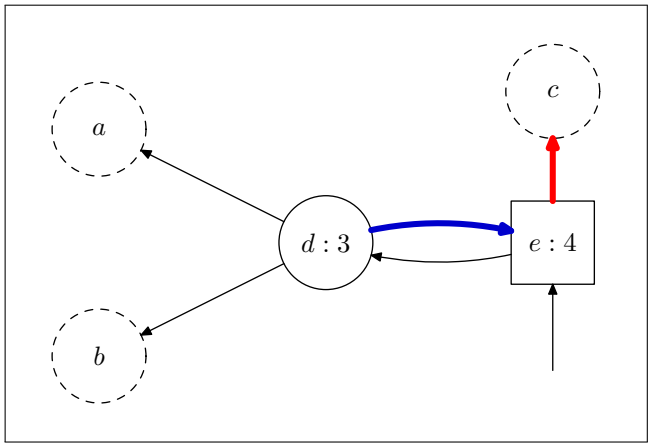


Figure 4.7: Simple Cycle Example, Strategy 2

Now node d gets the valuation associated with c. Note that we will be able to reuse such cycles again by valuating a node like a or b better than c for a single iterations. Such cycles will be the only structure in which we really use player 1 controlled nodes. There are some variations of the simple cycles (as we will see), but the essential structure, that hides one single escape edge that might be extremely profitable to player 0, remains the same.

Relation to Markov Decision Processes

The only structure that we need to translate, when moving from parity games to Markov decision processes, are player 1 controlled nodes. In our lower bound games, the only role that player 1 has, is to hide the effect of some escape node. In other words, the valuation of a player 1 node e_1 should be essentially equal to the valuation of the player 0 node d_1, as long as player 0 moves out of the cycle, and if player 0 moves into the cycle, the valuation of e_1 should be essentially equal to the valuation of the escape node h_1.

This effect can be simulated by a randomization node that moves to the escape node with extremely (that is, inversely exponentially) low probability ε. See Fig-

ure 4.8 for a more complicated cycle setting and the correspondence between player 1 controlled cycles in parity games and randomization controlled cycles in MDPs.

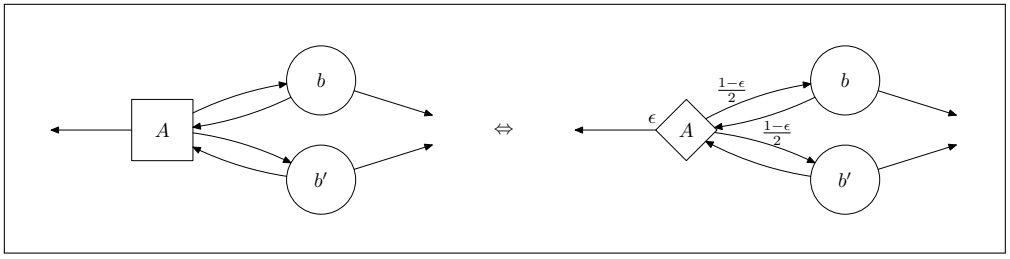


Figure 4.8: Conversion of a vertex controlled by player 1 to a randomization vertex

First, assume that both cycles attached to the node A are closed, i.e. player 0 moves to A from b and b'. Although the randomized node circles through the cycles with very high probability (without accumulating any rewards), it eventually moves out to the escape node, resulting in the same valuation as the valuation of the escape node itself, reflecting exactly the behavior of the cycle structure in parity games.

Second, assume that a cycle is open, i.e. one of the V_0-controlled nodes of the cycle decides to move out of the cycle to some *reset node*. Now, the randomized node moves into the cycle with very large probability and therefore leaves the cycle to the reset node with high probability as well. The resulting valuation of the randomized node essentially matches the valuation of the reset node, again reflecting the behavior of the cycle structure in parity games.

Relation to Linear Programming

The conditions for optimal values (and potentials) in a Markov decision process can be formulated as a linear program in which variables correspond to values and constraints to edges, and vice versa by duality.

In order to transfer the lower bounds for Markov decision processes to the simplex algorithm for solving linear programs, we simply need to show that (1) basic feasible solutions in the induced linear program correspond to strategies in the original game, and that (2) adjacent basic feasible solutions with improved cost correspond to strategies that have been improved by a single improving switch. This

allows us to transfer lower bounds for Markov decision processes immediately to the simplex algorithm, assuming that the respective improvement rule can be formulated accordingly as a pivoting rule.

A similar observation holds true for pivoting algorithms that operate on the dual linear program (e.g. RANDOM-FACET).

4.4 Sink Game Relations

We define *sink parity games* in this chapter. Sink parity games have a very special structure that consists of a game graph that leads to a single node looping to itself. We allow the game to contain other cycles. However, they should never occur in a run of the policy iteration algorithm, meaning that in the subgraph related to a player 0 strategy and an optimal counterstrategy, it holds that every play ends in the single loop.

This seemingly simple structure is expressive enough to construct lower bound games for parity game strategy iteration. As a bonus, we show that lower bounds based on sink parity games can be directly transferred to payoff games.

All technically tedious proofs have been put into Appendix A.1.

Sink Parity Games

Every approach trying to construct a parity game family of polynomial size that requires (sub)exponentially many iterations to be solved by strategy iteration (no matter which rule the algorithm is parameterized with), needs to focus on the second component of game valuations: there are only linearly many different values for the first and third component while there are exponentially many for the second.

Particularly, as there are at most linearly many different cycle nodes that can occur in valuations during a run, there is no real benefit in actually using different cycle nodes. Hence our basic layout of a game exploiting exponential behavior consists of a complex structure leading to one single loop – the only cycle node

that will occur in valuations. In this setting, the strategy iteration algorithm is just improving the paths leading to the cycle node.

More formally: we call a parity game G (in combination with an initial strategy ι) a *sink parity game* iff the following two properties hold:

1. *Sink Existence*: there is a node v^* (called the *sink* of G) with $v^* E v^*$ and $\Omega(v^*) = 1$ reachable from all nodes; also, there is no other node w with $\Omega(w) \leq \Omega(v^*)$.

2. *Sink Seeking*: for each player 0 strategy σ with $\iota \trianglelefteq \sigma$ and each node w, it holds that the cycle component of $\Xi_\sigma(w)$ equals v^*.

Obviously, a sink game is won by player 1. Note that comparing node valuations in a sink game can be reduced to comparing the path components of the respective node valuations, for two reasons. First, the cycle component remains v^*. Second, the path-length component equals the cardinality of the path component, because all nodes except the sink node are more relevant than the cycle node itself. In the case of a sink parity game, we will therefore identify node valuations with their path component.

In order to prove that a parity game is a sink parity game, one simply has to check that the sink existence property holds by looking at the graph, that the game is completely won by player 1, and that the sink is the cycle component of all nodes of the initial strategy.

Lemma 4.13. *Let G be a parity game fulfilling the sink existence property w.r.t. v^*. G is a sink parity game iff G is completely won by player 1 (i.e. $W_1 = V$) and for each node w it holds that the cycle component of $\Xi_\iota(w)$ equals v^*.*

Proof. The "only-if"-part is trivial. For the "if"-part, we need to show that the *sink seeking*-property holds. Let σ be a player 0 strategy with $\iota \trianglelefteq \sigma$, w be an arbitrary node and u be the cycle component of $\Xi_\sigma(w)$. Due to the fact that G is completely won by player 1, u has to be of odd priority. Also, since $\iota \trianglelefteq \sigma$, it holds that $\Omega(u) \leq \Omega(v^*)$ implying $u = v^*$ by the *sink existence*-property. □

In the context of a sink parity game G and a strategy σ, we will sometimes say that a node v *reaches* a node w to denote the fact that w lies on the path $\pi_{v,\sigma,\tau_\sigma}$.

We note that sink parity games are related to *escape payoff games* as defined in Schewe's paper [Sch08]. Escape games essentially allow the players to stop the play at a certain point by moving to a corresponding escape sink node. See Schewe's paper for all the details.

From Sink Parity to Discounted Payoff Games

We now show that the strategy iteration for discounted payoff games behaves exactly the same as the strategy iteration for sink parity games.

Vöge proves in his thesis [Vög00] the following theorem that relates parity game strategy iteration to Puri's algorithm for solving the induced discounted payoff game (in the sense of Chapter 3.4 via an intermediate mean payoff game).

Theorem 4.14 ([Vög00])**.** *Let G be a parity game, H_λ be the induced discounted payoff game and λ be a large enough discount factor. Let σ be a player 0 strategy. For every two nodes v and u the following holds:*

$$v \prec_\sigma^G u \qquad \Rightarrow \qquad v \prec_\sigma^{H_\lambda} u$$

In other words, every improving switch in the original parity game is also an improving switch in the induced discounted payoff game. The reason why this holds true is that by the reduction from parity games to mean payoff games, the priorities are mapped to such extremely large rewards that the largest reward that occurs on a path dominates all lower ones, the largest reward on a cycle dominates all other ones and that the cycle itself dominates all finite paths leading into it.

Theorem 4.14 is almost what we need to show that strategy iteration for discounted payoff games behaves exactly the same on the induced discounted payoff game as the discrete strategy iteration algorithm on the original sink game. Essentially, we need to show the converse which is equivalent to showing

$$\Xi_\sigma^G(v) = \Xi_\sigma^G(u) \qquad \Rightarrow \qquad \Xi_\sigma^{H_\lambda}(v) = \Xi_\sigma^{H_\lambda}(u)$$

However, this statement is not true for every parity game. The reason why a run of the strategy improvement algorithm on general parity games may differ from a run on the induced discounted payoff game is, that the parity game strategy iteration does not care about the priority of all nodes on its path to the dominating cycle node that are less relevant. In the case of sink parity games, the only occurring dominating cycle node has the least priority in the game, and therefore all priorities occurring in paths influence the valuations. Also, the strategy iteration on arbitrary parity games does not consider the priorities of all the nodes on a cycle appearing in a node valuation.

First, we show that optimal player 1 counter strategies in the induced discounted payoff game also eventually reach the sink.

Lemma 4.15. *Let G be a sink parity game with v^* being the sink, H_λ be the induced discounted payoff game, λ be a large enough discount factor and σ be a player 0 strategy s.t. $\iota \trianglelefteq^G \sigma$. Let $v_0 \neq v^*$ be an arbitrary node. Then $\pi_{v_0,\sigma,\tau_\sigma^{H_\lambda}}$ is of the following form:*

$$\pi_{v_0,\sigma,\tau_\sigma^{H_\lambda}} = v_0 v_1 \dots v_{l-1}(v^*)^\omega$$

Second, we show that the value ordering between two different paths leading to the sink again depends solely on the most relevant node in the symmetric difference of the paths.

Lemma 4.16. *Let G be a sink parity game with v^* being the sink, H_λ be the induced discounted payoff game, λ be a large enough discount factor. Let π and ξ be two paths of the form $\pi = u_0 u_1 \dots u_{l-1}(v^*)^\omega$ and $\xi = w_0 w_1 \dots w_{k-1}(v^*)^\omega$ and let $U = \{u_0, \dots, u_{l-1}\}$ and $W = \{w_0, \dots, w_{k-1}\}$. Then $U \prec W$ implies $R_\lambda(\pi) < R_\lambda(\xi)$.*

Third, we derive that the strategy iteration for discounted payoff games behaves exactly the same as the strategy iteration for sink parity games.

Corollary 4.17. *Let G be a sink parity game, H_λ be the induced mean payoff game, λ be a large enough discount factor, and σ be a player 0 strategy s.t. $\iota \trianglelefteq^G \sigma$. For every two nodes v and u the following holds:*

$$v \prec_\sigma^G u \qquad \Longleftrightarrow \qquad v \prec_\sigma^{H_\lambda} u$$

Corollary 4.18. *Discrete strategy iteration on sink parity games behaves exactly the same as policy iteration on induced discounted payoff games.*

Limiting Average Criterion

We apply a slightly different reduction from parity games to payoff games with limiting average criterion. The only difference to the standard reduction is that we assign *zero* cost to the edge looping at the sink node. By this construction, it follows that the *value* of every node equals zero, as all paths eventually end in the sink. The *potential* of the nodes equals the sum over the priority-rewards of the paths, hence, we can show a similar statement as in Lemma 4.16.

Theorem 4.19. *Discrete strategy iteration on sink parity games behaves exactly the same as policy iteration on induced MPGs, DPGs, as well as on the turn-based stochastic extensions.*

4.5 Simplex Algorithm Relations

The most widely used algorithm for solving MDPs is Howard's [How60] *policy iteration* algorithm. The policy iteration algorithm is closely related to the simplex algorithm. It can, however, exploit the special structure of the LPs that correspond to MDPs and perform many pivoting steps simultaneously.

Policy iteration algorithms that perform a single switch at each iteration are, in fact, simplex algorithms. In this chapter, we will formulate the problem of solving Markov decision processes as linear programs and show how the operation of the simplex algorithm on them corresponds to policy iteration on the original MDPs. Turning Markov decision processes into linear programs and their correspondences are well-known, see for instance Puterman [Put94] or Ye [Ye10].

Note that we only know how to cast Markov decision processes as linear programs. Natural attempts for translating classes of infinitary payoff games like turn-based stochastic games into linear programs fail, see, for instance the line of research done by Condon [Con93]. Similarly, the formulation of two and 2.5-player

infinitary payoff games as interior-point problems seems to be very difficult due to singularities and non-smoothness properties of the cost function, see, for instance Petersson and Vorobyov [PV01a].

Fix a Markov decision process $G = (V, V_0, V_R, E, r, p)$ with the discounted reward criterion and discount factor λ. We will explain in the end how to modify the formulations for the limiting average objective. For reasons of simplicity, assume that G is bipartite, i.e. every player 0 node is only connected to randomization nodes and vice versa; additionally assume that every edge controlled by randomization has zero cost.

We consider only Markov decision processes here that satisfy the *(weak) unichain condition*, as this allows a much more succinct formulation of corresponding linear programs, and we will see that our constructions satisfy these conditions.

The *(weak) unichain condition* relates to the notion of sink parity games. We say that a Markov decision process G satisfies the *unichain condition* (see [Put94]) iff the Markov chain obtained from each policy σ has a single irreducible recurrent class.

In other words, a Markov decision process satisfies the unichain condition if there is a single node v^* s.t. for every strategy σ and every node u, we have that v^* can be reached from u conforming to σ with positive probability.

The *weak* unichain condition only demands that the optimal policy has a single irreducible recurrent class. It follows that the optimal policy can be found by the same LPs when being started with an initial basic feasible solution corresponding to a policy with the same single irreducible recurrent class as the optimal policy. Then, by monotonicity, we know that all considered basic feasible solutions will have the same irreducible recurrent class.

Markov Decision Processes as Dual Problems

We start with the *dual* formulation of a Markov decision process, as the interpretation of the resulting linear program is more intuitive than the *primal*. Obviously, we could switch terms and call the following linear program the primal, however, for historical reasons, we stick with literature that usually labels the following LP to be the dual.

The *dual* linear program for *(weak) unichain MDPs* with discounted reward criterion is given by:

$$(D) \qquad \begin{array}{ll} \min & \sum_{u \in V_0} y(u) \\ \text{s.t.} & y(u) \;\geq\; r(u,v) + \lambda \cdot \sum_{w:(v,w)\in E_R} p(v,w)y(w) \quad , \quad (u,v) \in E_0 \end{array}$$

For a given MDP with player 0 controlled edges E_0, called *actions*, let H be the set of constraints defining the linear program. Note that there is a bijective mapping between actions $e \in E_0$ and constraints $h \in H$. More precisely, for $e = (u,v)$, e corresponds to the constraint $y(u) \;\geq\; r(u,v) + \lambda \cdot \sum_{w:(v,w)\in E_R} p(v,w)y(w)$. Thus, we will identify actions with constraints and use the notation interchangingly.

If y^* is an optimal solution of (D), then $y^*(u)$, for every $u \in V_0$, is the value of u under an optimal policy. An optimal policy σ^* can be obtained by letting $\sigma^*(u) = (u,v)$, where $(u,v) \in E_0$ is an edge for which the inequality constraint in (D) is tight, i.e., $y(u) - \lambda \cdot \sum_{w:(v,w)\in E_R} p(v,w)y(w) = r(u,v)$. Such a tight edge is guaranteed to exist.

Theorem 4.20 ([Put94])**.** *A solution to the dual linear program corresponds to an optimal policy and vice versa. Particularly, the dual is feasible and bounded.*

By Theorem 2.16 and Theorem 4.20, we have the following corollary, stating that an optimal policy corresponds to a basis and vice versa:

Corollary 4.21. *Let H be the set of constraints corresponding to the MDP. B is an H-basis iff the actions corresponding to B form an optimal policy.*

The following important fact about bases for the linear program and policies for the corresponding Markov decision process is not hard to see.

Lemma 4.22. *Let σ be a policy and $B \subseteq H$ be the set of constraints corresponding to σ. Let $e \in E_0 \setminus \sigma$ and $h \in H \setminus B$ be the corresponding constraint. Then $\Xi_\sigma \trianglelefteq \Xi_{\sigma[e]}$ iff h is violated by B, in which case* $\textsc{Basis}(B \cup \{h\})$ *corresponds to $\sigma[e]$.*

Proof. Restrict the Markov decision process to subset of actions $\sigma \cup \{e'\}$. Let $B' = \textsc{Basis}(B \cup \{h\})$ and $\sigma' = \sigma[e]$.

If h is violated by B, we have that $v_{B'} > v_B$. By Theorem 4.20 we have that $\Xi_\sigma \trianglelefteq \Xi_{\sigma'}$, as only two policies exist in the MDP.

For the converse, assume that $\Xi_\sigma \trianglelefteq \Xi_{\sigma'}$. Hence, B cannot be a basis for the whole set of constraints. Hence, h must be violated by B. □

In Chapter 4.7, we will give a formulation of the RANDOM-FACET algorithm as a policy iteration pivoting rule for infinitary payoff games, and use Lemma 4.22 to see that any lower bound construction for a Markov decision process immediately transfers to the RANDOM-FACET algorithm for linear programs.

Markov Decision Processes as Primal Problems

Optimal policies for MDPs that satisfy the unichain condition can be found by solving the following *primal* linear program:

$$
(P) \qquad \begin{array}{ll} \max & \sum_{(u,v)\in E_0} r(u,v)x(u,v) \\ \text{s.t.} & \sum_{v\in uE} x(u,v) \;=\; \lambda \cdot \sum_{w\in E_R u, v\in E_0 w} p(w,u)x(v,w)\,,\; u \in V_0 \\ & \sum_{(u,v)\in E_0} x(u,v) \;=\; 1 \\ & x(u,v) \;\geq\; 0 \quad , \quad (u,v) \in E_0 \end{array}
$$

The variable $x(u,v)$, for $(u,v) \in E_0$, stands for the probability (frequency) of using the edge (action) (u,v). The constraints of the linear program are *conservation constraints* that state that the probability of entering a vertex u is equal to the probability of exiting u.

Lemma 4.23. *The basic feasible solutions of (P) correspond directly to policies of the MDP.*

Proof. For each policy σ we can define a feasible setting of primal variables $x(u,v)$, for $(u,v) \in E_0$, such that $x(u,v) > 0$ only if $\sigma(u) = v$.

Conversely, for every bfs $x(u,v)$ we can define a corresponding policy σ. Obviously, every bfs contains $|V_0|$ basic variables. It is easy to see that we cannot have zero variables corresponding to a node $w \in V_0$ in the bsf, hence we must have exactly one variable corresponding to a node in the bsf. □

Theorem 4.24 ([Put94])**.** *A policy corresponding to an optimal bfs of (P) is an optimal policy of the MDP.*

It follows that the improving switches of a given policy coincide exactly with the edges leading to adjacent improved vertices in the primal.

Corollary 4.25. *Let σ be a policy and B a corresponding bsf. Let $e \in E_0 \setminus \sigma$ be an edge corresponding to $x(u,v)$, i.e. $e = (u,v)$. Then $\Xi_\sigma \trianglelefteq \Xi_{\sigma[e]}$ iff there is an improved cost adjacent vertex to B with $x(u,v)$ as basic variable.*

Hence, we conclude the single-switching policy iteration on Markov decision processes is exactly the same as running the simplex on the primal linear program.

Remarks

We end this chapter with the linear programming formulation of the (weakly) unichain Markov decision processes with the limiting average criterion.

This condition implies, in particular, that all vertices have the same value. It is not difficult to check that $\text{VAL}_\sigma(u)$ is indeed the expected reward per turn, when the process starts at u and policy σ is used. The potentials $\text{POT}_\sigma(u)$ represent *biases*. Loosely speaking, the expected reward after N steps, when starting at u and following σ, and when N is sufficiently large, is about $N \cdot \text{VAL}_\sigma(u) + \text{POT}_\sigma(u)$.

The *dual* linear program for *(weak) unichain MDPs* with limiting average criterion is given by:

$$(D) \qquad \begin{array}{ll} \min & z \\ \text{s.t.} & y(u) \;\geq\; r(u,v) - z + \sum_{w:(v,w)\in E_R} p(v,w)y(w) \quad , \quad (u,v) \in E_0 \end{array}$$

If (y^*, z^*) is an optimal solution of (D), then z^* is the common value of all vertices, and $y^*(u)$, for every $u \in V_0$, is the potential of u under an optimal policy.

The linear programs corresponding to the limiting average criterion MDPs constructed in this paragraph are linear programs s.t. *all* pivoting steps performed on these linear programs are *degenerate*. Progress is still being made in each iteration, as some potentials, i.e. dual variables, strictly increase.

Optimal policies for MDPs with the limiting average criterion that satisfy the unichain condition can be found by solving the following *primal* linear program:

$$
(P)\qquad
\begin{array}{ll}
\max & \sum_{(u,v)\in E_0} r(u,v)x(u,v) \\
\text{s.t.} & \sum_{v:(u,v)\in E} x(u,v) = \sum_{v,w:(v,w)\in E_0,(w,u)\in E_R} p(w,u)x(v,w)\,,\ u \in V_0 \\
& \sum_{(u,v)\in E_0} x(u,v) \ = \ 1 \\
& x(u,v) \ \geq \ 0 \quad , \quad (u,v) \in E_0
\end{array}
$$

Due to possible degeneracies, the policy, i.e., basis, corresponding to a given bfs is not necessarily unique. If for some $u \in V_0$ we have $x(u,v) = 0$ for every $(u,v) \in E_0$, then the choice of $\sigma(u)$ is arbitrary.

4.6 Deterministic Rules

There are two major deterministic improvement rules that we handle in this chapter. First, we consider the SWITCH-ALL improvement rule, which was introduced by Howard [How60] for solving problems on Markov decision processes and has been adapted by several other authors for solving nonterminating stochastic games [HK66], simple stochastic games [Con92], discounted and mean payoff games [Pur95, ZP96] as well as parity games [VJ00].

Second, we consider the SWITCH-BEST improvement rule by Schewe [Sch08], which among all possible successor strategies selects one with the best possible improvement.

Both improvement rules are multi-switching methods, and are therefore not applicable for solving linear programming problems. Hence, we only have infinitary payoff games in mind here. Our contribution is to give the first explicit constructions of exponential lower bounds for SWITCH-ALL and SWITCH-BEST on infinitary payoff games.

All technically tedious proofs have been put into Appendix A.2.

4.6.1 Switch All Rule

The SWITCH-ALL or *locally optimizing rule* is generally seen to be the most natural choice for an improvement rule. Consider the case in which all player 0 nodes have at most out-degree two. This implies that a node either has no improving edge or exactly one improving edge, namely the one which is not chosen by the current strategy. In this setting, the SWITCH-ALL improvement rule can be described very concisely:

SWITCH-ALL:	Apply every improving edge simultaneously.

The generalization of this rule in the non-binary case is to apply an improving switch to every improvable node that has the highest valuation of a successor. This does not completely specify a deterministic choice, as it may happen that two successors have the same valuation. Although the name might be a bit confusing in the non-binary setting, we will still call this improvement rule by its common name.

SWITCH-ALL:	Apply the best local improvement in every node simultaneously.

More formally, it holds for every strategy σ, every player 0 node v and every $w \in vE$ that $w \preceq_\sigma \text{SWITCH-ALL}(\sigma)(v)$.

The SWITCH-ALL improvement rule is very general, and applies to all strategy iteration variants for arbitrary infinitary payoff games. It does not apply to linear programming, being a multi-switch improvement rule.

Jurdziński and Vöge [VJ00] were the first to adapt strategy iteration to parity game solving, and proposed the SWITCH-ALL improvement rule as canonical choice. It is very easy to see that the rule can be computed efficiently.

Lemma 4.26 ([VJ00])**.** *The* SWITCH-ALL *rule can be computed in polynomial time.*

The lower bound construction for SWITCH-ALL is a family of sink parity games that implement a binary counter. In order to reduce the overall complexity of the

games, our construction relies on unbounded edge out-degree, yielding a quadratic number of edges in total. We will discuss in the end how the number of edges can be reduced to a linear number and even how to get binary out-degree.

The implementation of the binary counter is based on a structure called *simple cycles* that allows us to encode a single bit state in a given strategy σ. By having n such simple cycles, we can represent every state of an n-bit binary counter. In order to allow strategy improvement the transitions of the binary counter, we need to embed the simple cycles in a more complicated structure called *cycle gadget*, connect the cycle gadgets of the different bits with each other, and with an additional structure called *deceleration lane*.

This chapter is organized as follows. First, we consider the three gadgets that will be used in our lower bound construction, namely *simple cycles*, the *deceleration lane* and *cycle gates*. Then, we present the full construction of our lower bound family and give a high-level description of strategy iteration on these games. Finally, we prove that strategy improvement on the games indeed follows the high-level description.

For the presentation of the gadgets, we assume the context of a sink parity game. The labellings and priorities of the gadgets will match the final priorities of the lower bound family.

Gadgets consist of three kinds of nodes: *input nodes*, *output nodes* and *internal nodes*. Input nodes are nodes that will have incoming edges from outside of the gadget, output nodes will have outgoing edges to the outside of the gadget and internal nodes will not be directly connected to the outside of the gadget.

Simple Cycles

The binary counter will contain a representation of n bits that are realized by n instances of a gadget called a *cycle gate*. The most important part of a cycle gate is the *simple cycle* that we will introduce first. We fix some index i for the simple cycle gadget for the sake of this paragraph in order to have consistent node labellings.

A simple cycle consists of one player 0 controlled internal node d_i that is connected to a set of external nodes D_i in the rest of the graph, and one player 1

controlled input node e_i. The node e_i itself is connected to d_i (therefore the name *simple cycle*) and to one output node $h_i \notin D_i$. We note that all e_i nodes are the *only* player 1 controlled nodes with real choices in the complete lower bound construction.

All priorities of the simple cycle are based on some odd priority p_i. Intuitively, the p_i is considered to be a very small priority compared to the priorities of the other nodes in the external graph that the simple cycle is connected to.

See Figure 4.9 for a simple cycle of index 1 with $p_1 = 3$. The players, priorities and edges are described in Table 4.7.

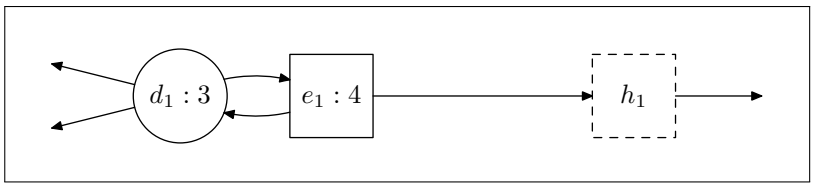


Figure 4.9: Simple Cycle

Node	Player	Priority	Successors
d_i	0	p_i	$\{e_i\} \cup D_i$
e_i	1	$p_i + 1$	$\{d_i,\ h_i\}$
h_i	?	$> p_i + 1$	?
$w \in D_i$	?	$> p_i + 1$	?

Table 4.7: Description of the Simple Cycle

Given a strategy σ, we say that the cycle is *closed* iff $\sigma(d_i) = e_i$ and *open* otherwise. A closed cycle corresponds to a bit which is set while an open cycle corresponds to an unset bit.

The main idea now is to assign priorities to the simple cycle in such a way that the simple cycle is won by player 0, i.e. the most relevant node on the cycle needs to have an even priority. This has important consequences for the behavior of the player 1 controlled node.

First, assume that $\sigma(d_i) = e_i$. The optimal counter-strategy here is $\tau_\sigma(e_i) = h_i$, since otherwise player 0 would win the cycle which is impossible with G being a

sink game. Player 0 is therefore able to *force* player 1 to move out of the cycle; in other words, *setting* a bit corresponds to *forcing* player 1 out of the cycle. In a set bit, the valuation of d_i is essentially the valuation of h_i, i.e. $\Xi_\sigma(d_i) = \Xi_\sigma(h_i) \cup \{d_i, e_i\}$.

Second, assume that $\sigma(d_i) = w$ for some $w \in D_i$, and that $w \prec_\sigma h_i$. It follows that $d_i \prec_\sigma h_i$, hence $\tau_\sigma(e_i) = d_i$. The interesting part is now that $\Xi_\sigma(e_i) = \Xi_\sigma(w) \cup \{d_i, e_i\}$, i.e. e_i is an improving node for d_i (since $\Xi_\sigma(w) \triangle \Xi_\sigma(e_i) = \{d_i, e_i\}$), but updating to e_i would yield a much greater reward than just $\Xi_\sigma(e_i)$ (namely $\Xi_\sigma(h_i) \cup \{e_i\}$ by forcing player 1 to leave the cycle).

Assume now that $w' \in D_i$ with $w \prec_\sigma w'$ but $w' \prec_\sigma h_i$. Obviously, w' and e_i are improving nodes for d_i, but $e_i \prec_\sigma w'$, hence by SWITCH-ALL, player 0 switches to w', although e_i might give a much better valuation. In other words, by moving to d_i, the player 1 node hides the fact that there is a highly profitable node on the other side.

Lemma 4.27. *Let σ be a strategy. The following holds:*

1. *If cycle i is closed, we have $\tau_\sigma(e_i) = h_i$.*
2. *If cycle i is open and $h_i \prec_\sigma \sigma(d_i)$, we have $\tau_\sigma(e_i) = h_i$.*
3. *If cycle i is open and $\sigma(d_i) \prec_\sigma h_i$, we have $\tau_\sigma(e_i) = d_i$.*

Lemma 4.28. *Let σ be a strategy and $w = \max_{\prec_\sigma} D_i$. Let $\sigma' =$ SWITCH-ALL(σ). The following holds:*

1. *If cycle i is closed and $w \prec_\sigma h_i$, we have cycle i σ'-closed ("closed cycle remains closed").*
2. *If cycle i is open, $\sigma(d_i) \neq w$ or $h_i \prec_\sigma w$, we have $\sigma'(d_i) = w$ ("open cycle remains open").*
3. *If cycle i is open, $\sigma(d_i) = w$ and $w \prec_\sigma h_i$, then cycle i is σ'-closed ("open cycle closes").*
4. *If cycle i is closed and $h_i \prec_\sigma w$, we have $\sigma'(d_i) = w$ ("closed cycle opens").*

Open simple cycles have the important property that we can postpone closing them by supplying new nodes $w \in D_i$ in each iteration s.t. $\sigma(d_i) \prec_\sigma w$. We will use this property in the construction of our binary counter. Since we do not want to set all bits at the same time, rather one by one, we need to make sure that unset bits, which are not supposed to be set, remain unset for some time (more precisely, until the respective bit represents the least unset bit), and this will be realized by this property. The device that supplies us with new best-valued external nodes in each iteration is called *deceleration lane* and will be described next.

Deceleration Lane

A *deceleration lane* has several, say m, input nodes and some output nodes, called *roots*. The lower bound construction will only require a deceleration lane with two roots s and r, however, it would be easy to generalize the construction of deceleration lanes to an arbitrary number of roots.

More formally, a deceleration lane consists of m (in our case, m will be $2 \cdot n$) internal nodes $t_1, \ldots, t_m$, one additional internal node c, m input nodes $a_1, \ldots, a_m$ and two output nodes s and r, called *roots* of the deceleration lane.

All priorities of the deceleration lane are based on some odd priority p. We assume that all root nodes have a priority greater than $p + 2m + 1$. See Figure 4.10 for a deceleration lane with $m = 6$ and $p = 15$. The players, priorities and edges are described in Table 4.8.

Node	Player	Priority	Successors
t_1	0	p	$\{s,\ r,\ c\}$
$t_{i>1}$	0	$p + 2i - 2$	$\{s,\ r,\ t_{i-1}\}$
c	0	$p + 2m + 1$	$\{s,\ r\}$
a_i	1	$p + 2i - 1$	$\{t_i\}$
s	?	$> p + 2m + 1$	?
r	?	$> p + 2m + 1$	?

Table 4.8: Description of the Deceleration Lane

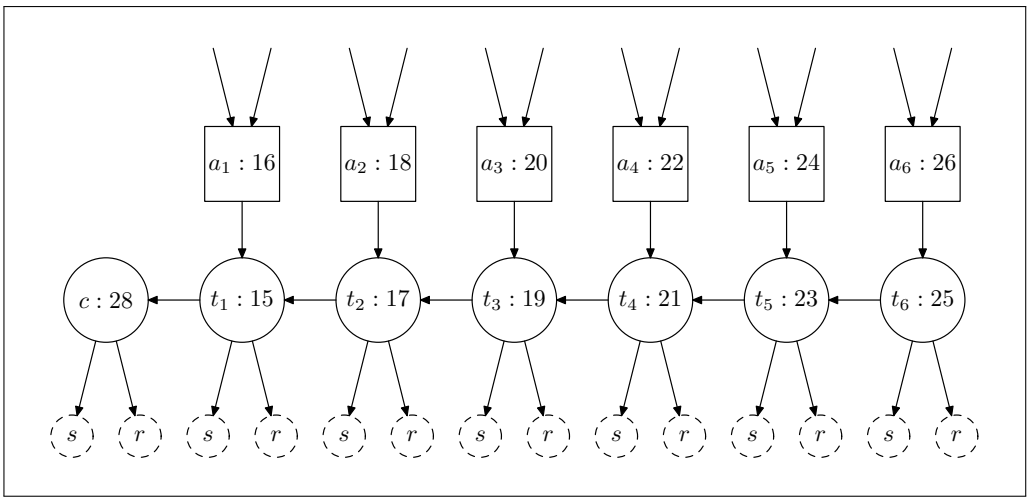


Figure 4.10: A Deceleration Lane (with $m = 6$ and $p = 15$)

A deceleration lane serves the following purpose. Assume that one of the output nodes, say r, has the better valuation compared to the other root node, and assume further that this setting sustains for some iterations.

The input nodes, say $a_1, \ldots, a_m$, now serve as entry points, and all reach the best valued root – r – by some internal nodes. The valuation ordering of all input nodes depends on the iteration: at first, a_1 has a better valuation than all other input nodes. Then, a_2 has a better valuation than all other input nodes and so on.

This process continues until the other output node, say s, has a better valuation than r. Within the next iteration, the internal nodes perform a *resetting* step s.t. all input nodes eventually reach the new root node. One iteration after that, a_1 has the best valuation compared to all other input nodes again.

In other words, by giving one of the roots, say s, a better valuation than another root, say r, it is possible to reset and therefore reuse the lane again. In fact, the lower bound construction will use a deceleration lane with two roots s and r, and will employ s only for resetting, i.e. after some iterations with $r \succ_\sigma s$, there will be one iteration with $s \succ_\sigma r$ and right after that again $r \succ_\sigma s$.

From an abstract point of view, we describe the state of a deceleration lane by which of the two roots is chosen and by how many t_i nodes are already moving down to c. Formally, we say that σ *is in deceleration state* (x, j) (where $x \in \{s, r\}$ and $0 < j \leq m + 1$ a natural number) iff

1. $\sigma(c) = x$,
2. $\sigma(t_1) = c$ if $j > 1$,
3. $\sigma(t_i) = t_{i-1}$ for all $1 < i < j$, and
4. $\sigma(t_i) = x$ for all $j \leq i$.

We say that the deceleration lane is *rooted in* x if σ is in state $(x, *)$, and that the *index is* i if σ is in state $(*, i)$. Whenever a strategy σ is in state (x, i), we define $root(\sigma) = x$ and $ind(\sigma) = i$. In this case, we say that the strategy is *well-behaved*.

The valuation ordering of the deceleration lane can described as follows: (1) if the ordering of the root nodes changes, all input nodes have a worse valuation than the better root, and (2) otherwise the best valued input node is a_{i-1}.

Lemma 4.29. *Let σ be a strategy in deceleration state (x, i). Let $\bar{x}$ denote the other root. Then*

1. *$x \prec_\sigma \bar{x}$ implies $a_j \prec_\sigma \bar{x}$ for all j ("resetting results in unprofitable lane").*
2. *$\bar{x} \prec_\sigma x$ implies $x \prec_\sigma a_i \prec_\sigma \ldots \prec_\sigma a_m \prec_\sigma c \prec_\sigma a_1 \prec_\sigma \ldots \prec_\sigma a_{i-1}$ ("new best-valued node in each iteration").*

The switching behavior of the player 0 controlled nodes can be described as follows: (1) if the ordering of the root node changes, than the whole lane resets, and (2) otherwise the lane assembles further, providing a new best-valued input node.

Lemma 4.30. *Let σ be a strategy that is in deceleration state (x, i). Let $\bar{x}$ denote the other root. Let $\sigma' =$ SWITCH-ALL(σ). Then*

1. *$x \prec_\sigma \bar{x}$ implies that σ' is in state $(\bar{x}, 1)$ ("lane resets").*
2. *$\bar{x} \prec_\sigma x$ implies that σ' is in state $(x, \min(i, m) + 1)$ ("lane assembles one step at a time").*
3. *σ' is well-behaved ("always ending up with well-behaved strategies").*

The main purpose of a deceleration lane is to absorb the update activity of other nodes in such a way that wise (i.e. edges that will result in much better valuations *after* switching and reevaluating) strategy updates are postponed. Consider a node, for instance, that has more than one improving switch; SWITCH-ALL will select the edge with the best valuation to be switched. In order to prevent that one particular improving switch is applied for some iterations, one can connect the node to the input nodes of the deceleration lane.

The particular scenario in which we will use the deceleration lane are *simple cycles* as described in the previous paragraph. We will connect the simple cycles encoding the bits of our counter to the deceleration lane in such a way, that lower cycles have less edges entering the deceleration lane. This construction ensures that lower open cycles (representing unset bits) will close (i.e. set the corresponding bit) before higher open cycles (representing higher unset bits) have their turn to close.

Cycle Gate

The simple cycles will appear in a more complicated gadget, called *cycle gate*. We will have n different cycle gates, and fix some index i for the cycle gate gadget for the sake of this paragraph.

Formally, a cycle gate consists of two internal nodes e_i and h_i, two input nodes f_i and g_i, and two output nodes d_i and k_i. The output node d_i will be connected to a set of other nodes D_i in the game graph, and k_i to some other set K_i as well. The two nodes d_i and e_i form a simple cycle as described earlier.

All priorities of the cycle gate are based on two odd priorities p_i and p'_i. See Figure 4.11 for a cycle gate of index 1 with $p'_1 = 3$ and $p_1 = 33$. The players, priorities and edges are described in Table 4.9.

The main idea behind a cycle gate is to have a pass-through structure *controlled by the simple cycle* that is either very profitable or quite unprofitable. The pass-through structure of the cycle gate has one major input node, named g_i, and one major output node, named k_i. The input node is controlled by player 0 and connected via two paths with the output node; there is a direct edge and a longer path leading through the interior of the cycle gate.

Node	Player	Priority	Successors
d_i	0	p'_i	$\{e_i\} \cup D_i$
e_i	1	$p'_i + 1$	$\{d_i,\ h_i\}$
g_i	0	$p'_i + 3$	$\{f_i,\ k_i\}$
k_i	0	p_i	K_i
f_i	1	$p_i + 2$	$\{e_i\}$
h_i	1	$p_i + 3$	$\{k_i\}$

Table 4.9: Description of the Cycle Gate

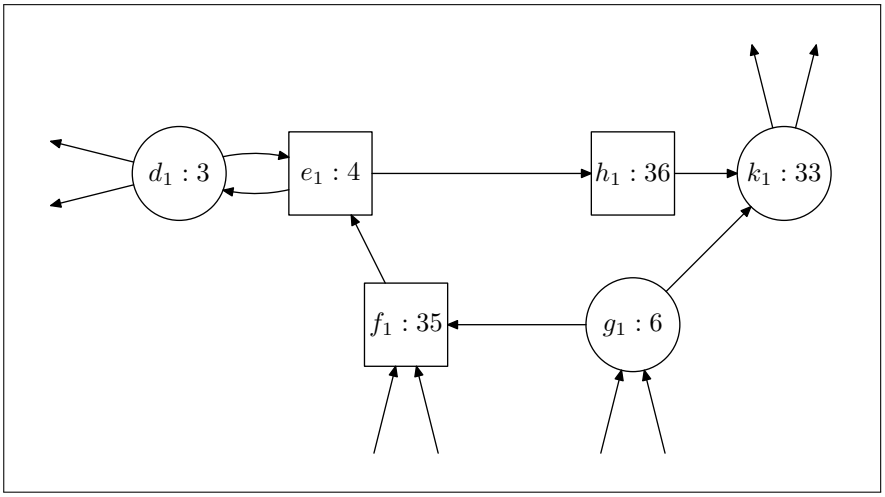


Figure 4.11: A Cycle Gate (index 1 with $p'_1 = 3$ and $p_1 = 33$)

However, the longer path only leads to the output node if the simple cycle, consisting of one player 0 node d_i and one player 1 node e_i, is *closed*. In this case, it is possible and profitable to reach the output node via the internal path; otherwise, this path is not accessible, and hence, the input node has to select the unprofitable direct way to reach the output node.

We will have one additional input node, named f_i, that can only access the path leading through the interior of the cycle gate, for the following purpose. Assume that the simple cycle has just been closed and now the path leading through the interior becomes highly profitable. Hence, the next switching event to happen will be node g_i switching from the direct path to the path through the interior. However, it will be useful to be able to reach the highly profitable path from some parts of the outside

graph one iteration before it is accessible via g_i. For this reason, we include an additional input node f_i that immediately accesses the interior path.

We say that a cycle gate is *closed* resp. *open* iff the interior simple cycle is closed resp. open. Similarly, we say that a cycle gate is *accessed* resp. *skipped* iff the access control node g_i moves through the interior ($\sigma(g_i) = f_i$) resp. directly to k_i.

From an abstract point of view, we describe the state of a cycle gate by a pair $(\beta_i(\sigma), \alpha_i(\sigma)) \in \{0,1\}^2$. The first component describes the state of the simple cycle, and the second component gives the state of the access control node. We write:

1. $\beta_i(\sigma) = 1$ iff the i-th cycle gate is closed, and
2. $\alpha_i(\sigma) = 1$ iff the i-th cycle gate is accessed.

Lemma 4.31. *Let σ be a strategy.*

1. *If gate i is open, we have $f_i \prec_\sigma \sigma(d_i)$.*
2. *If gate i is closed, we have $\sigma(k_i) \prec_\sigma f_i$.*
3. *If gate i is closed and skipped, we have $g_i \prec_\sigma f_i$.*
4. *If gate i is accessed, we have $f_i \prec_\sigma g_i$.*
5. *If gate i is skipped, we have $\sigma(k_i) \prec_\sigma g_i$.*

Lemma 4.32. *Let σ be a strategy and $\sigma' = \text{SWITCH-ALL}(\sigma)$.*

1. *If gate i is σ-closed, then gate i is σ'-accessed ("closed gates will be accessed").*
2. *If gate i is σ-open and $\sigma(d_i) \prec_\sigma h_i$, then gate i is σ'-skipped ("open gates with unprofitable exit nodes will be skipped").*
3. *If gate i is σ-open and $h_i \prec_\sigma \sigma(d_i)$, then gate i is σ'-accessed ("open gates with profitable exit nodes will be accessed").*

The last two items of Lemma 4.32 are based on the uniqueness of priorities in the game, implying that there are no priorities between f_i and h_i.

We will use cycle gates to represent the bit states of a binary counter: unset bits will correspond to cycle gates with the state $(0, 0)$, set bits to the state $(1, 1)$. Setting and resetting bits therefore traverses more than one phase, more precisely, from $(0, 0)$ over $(1, 0)$ to $(1, 1)$, and from the latter again over $(0, 1)$ to $(0, 0)$.

Particularly, it can be observed that the second component of the cycle gate states switches one iteration after the first component in both cases.

Full Construction

In this paragraph, we provide the complete construction of the lower bound family. It essentially consists of a sink x, a deceleration lane of length $2n$ that is connected to the two roots s and r, and n cycle gates. The simple cycles of the cycle gates are connected to the roots and to the deceleration lane such that lower cycle gates have less edges to the deceleration lane. This construction ensures that lower open cycle gates will close before higher open cycle gates.

The output node of a cycle gate is connected to the sink and to the g_*-input nodes of all higher cycle gates. The s root node is connected to all f_*-input nodes, the r root node is connected to all g_*-input nodes.

The games are denoted by $G_n = (V_n, V_{n,0}, V_{n,1}, E_n, \Omega_n)$ and the sets of nodes are $V_n := \{x, s, c, r\} \cup \{t_i, a_i \mid 1 \leq i \leq 2n\} \cup \{d_i, e_i, g_i, k_i, f_i, h_i \mid 1 \leq i \leq n\}$. The players, priorities and edges are described in Table 4.10. The game G_3 is depicted in Figure 4.12.

Fact 4.33. *The game G_n has $10 \cdot n + 4$ nodes, $1.5 \cdot n^2 + 20.5 \cdot n + 5$ edges and $12 \cdot n + 8$ as highest priority. In particular, $|G_n| = \mathcal{O}(n^2)$.*

As an initial strategy we select the following ι. It will correspond to the global counter state in which no bit has been set.

$$\iota(t_1) = c \quad \iota(g_i) = k_i \quad \iota(t_{i>1}, c, d_i) = r \quad \iota(k_i, s, r) = x$$

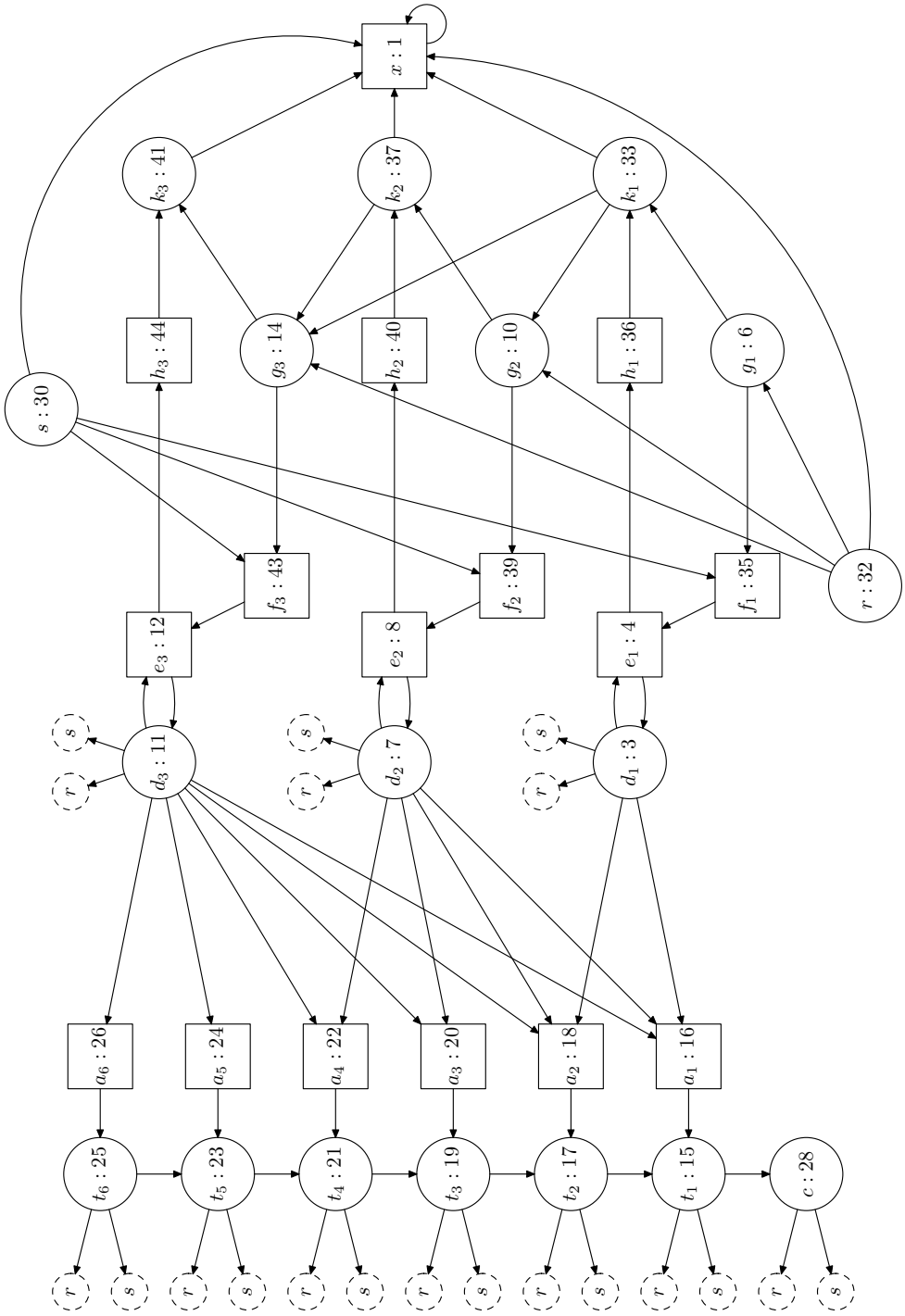


Figure 4.12: SWITCH-ALL Lower Bound Game G_3

Node	Player	Priority	Successors
t_1	0	$4n+3$	$\{s,\ r,\ c\}$
$t_{i>1}$	0	$4n+2i+1$	$\{s,\ r,\ t_{i-1}\}$
a_i	1	$4n+2i+2$	$\{t_i\}$
c	0	$8n+4$	$\{s,\ r\}$
d_i	0	$4i+1$	$\{s,\ e_i,\ r\} \cup \{a_j \mid j < 2i+1\}$
e_i	1	$4i+2$	$\{d_i,\ h_i\}$
g_i	0	$4i+4$	$\{f_i,\ k_i\}$
k_i	0	$8n+4i+7$	$\{x\} \cup \{g_j \mid i < j \leq n\}$
f_i	1	$8n+4i+9$	$\{e_i\}$
h_i	1	$8n+4i+10$	$\{k_i\}$
s	0	$8n+6$	$\{f_j \mid j \leq n\} \cup \{x\}$
r	0	$8n+8$	$\{g_j \mid j \leq n\} \cup \{x\}$
x	1	1	$\{x\}$

Table 4.10: Lower Bound Construction for the Locally Optimizing Policy

Note that ι particularly is well-behaved. Hence, by Lemma 4.30(3) we know that all strategies that will occur in a run of the strategy improvement algorithm will be well-behaved.

We will see later, how the family G_n can be refined in such a way that it only comprises a linear number of edges. We present the games with a quadratic number of edges first as the refined family looks even more confusing and obfuscates the general principle.

Lemma 4.34. *Let $n > 0$.*

1. *The game G_n is completely won by player 1.*
2. *x is the sink of G_n and the cycle component of $\Xi_\iota(w)$ equals x for all w.*

Proof. Let $n > 0$.

1. Note that the only nodes owned by player 1 with an out-degree greater than 1 are e_1,...,e_n. Consider the player 1 strategy τ which selects to move to h_i

from e_i for all i. Now it is the case that $G_n|_\tau$ contains exactly one cycle that is eventually reached no matter what player 0 does, namely the self-cycle at x which is won by player 1.

2. The self-cycle at x obviously is the sink as it can be reached from all other nodes and has the smallest priority 1. Since xEx is the only cycle won by player 1 in $G_n|_\iota$, x must be the cycle component of each ι-node valuation.

□

By Lemma 4.13 it follows that G_n is a sink game, hence it is safe to identify the valuation of a node with its path component from now on.

Lower Bound Description

Here, we describe how the binary counter performs the task of counting by strategy improvement. Our games implement a full binary counter in which every bit is represented by a simple cycle encapsulated in a cycle gate. An unset bit i corresponds to an open simple cycle in cycle gate i, a set bit i corresponds to a closed simple cycle in cycle gate i.

Recall that we represent the bit state of the counter by elements from $\mathcal{B}_n = \{\mathfrak{b} \in \{0,1\}^\infty \mid \forall i > n : \mathfrak{b}_i = 0\}$. For $\mathfrak{b} = (\mathfrak{b}_n, \ldots, \mathfrak{b}_1) \in \mathcal{B}_n$, let $\mathfrak{b}_i$ denote the i-th component in $\mathfrak{b}$ for every $i \leq n$, where $\mathfrak{b}_n$ denotes the most and $\mathfrak{b}_1$ denotes the least significant bit. By $\mathfrak{b} + 1$, we denote the increment of the number represented by $\mathfrak{b}$ by 1. The least resp. greatest bit states are denoted by $\mathbf{0}$ resp. $\mathbf{1}_n$. Given a configuration $\mathfrak{b}$, we access the *i-next set bit* by $\nu_i^n(\mathfrak{b}) = \min(\{n+1\} \cup \{j \geq i \mid \mathfrak{b}_j = 1\})$, and the *$i$-next unset bit* by $\mu_i(\mathfrak{b}) = \min\{j \geq i \mid \mathfrak{b}_j = 0\}$.

From the most abstract point of view, our lower bound construction performs binary counting on $\mathcal{B}_n$. However, the increment of a global bit state requires more than one strategy iteration, more precisely four different phases that will be described next (with one phase of dynamic length).

Every phase is defined w.r.t. a given global counter state $\mathfrak{b} \in \mathcal{B}_n$. Let $\mathfrak{b} \in \mathcal{B}_n$ be a global bit state different from $\mathbf{1}_n$.

An abstract counter performs the increment from $\mathfrak{b}$ to $\mathfrak{b}+1$ by computing $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1][j{<}\mu_1^n(\mathfrak{b}) \mapsto 0]$, i.e. by setting bit $\mu_1^n(\mathfrak{b})$ and by resetting all lower bits $j{<}\mu_1^n(\mathfrak{b})$. In the context of the games, we start in phase 1 corresponding to $\mathfrak{b}$, and then proceed to phase 2 and phase 3 corresponding to $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1]$, from phase 3 to phase 4 corresponding to $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1][j < \mu_1^n(\mathfrak{b}) \mapsto 0]$, and finally from phase 4 to phase 1 again. The transition from phase 2 to phase 3 and from phase 4 to phase 1 handles the correction of the internal structure connecting the cycles with each other.

We now proceed to a more detailed yet informal description of all phases. Given a strategy σ, we denote the *associated simple cycle state* $(\beta_n(\sigma), \ldots, \beta_1(\sigma))$ by b_σ, and the *associated access state* $(\alpha_n(\sigma), \ldots, \alpha_1(\sigma))$ by a_σ.

The first phase, called the *waiting* phase, corresponds to a stable strategy σ in which open cycles are busy waiting to be closed while the deceleration lane is assembling. Cycle gates that correspond to set bits are closed and accessed, while cycle gates of unset bits are open and skipped, i.e. $\mathfrak{b} = b_\sigma = a_\sigma$. The *selector nodes* k_i move to the next higher cycle gate corresponding to a set bit, and both roots are connected to the least set bit $\nu_1^n(\mathfrak{b})$.

The only improving switches in the first phase are edges of open simple cycles and edges of the deceleration lane. The first phase ends, when a simple cycle corresponding to an unset bit has no more edges leading to the deceleration lane that keeps it busy waiting, and closes. Since lower bits have less edges going to the lane, it is clear that this will be the least unset bit $\mu_1^n(\mathfrak{b})$.

The second phase, called the *set* phase, corresponds to a strategy σ in which the least unset bit has just been set, i.e. to the global state $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1] = b_\sigma$. The selector nodes and roots are as in phase 1 and also the access states, i.e. $\mathfrak{b} = a_\sigma$.

The deceleration lane is still assembling, and the improving switches again include edges of open simple cycles and edges of the deceleration lane. Additionally, it is improving for the cycle gate $\mu_1^n(\mathfrak{b})$ to be accessed and for the root s to update to cycle gate $\mu_1^n(\mathfrak{b})$. By performing all these switches, we enter phase three.

The third phase, called the *access* phase, is defined by a renewed correspondence of the cycle gate structure again, i.e. $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1] = b_\sigma = a_\sigma$. The s root is connected to $\mu_1^n(\mathfrak{b})$ while r is still connected to $\nu_1^n(\mathfrak{b})$. This implies that s now has a much better valuation than r.

The cycle gate with the best valuation is now $\mu_1^n(\mathfrak{b})$, hence, there are many improving switches, that eventually lead to cycle gate $\mu_1^n(\mathfrak{b})$. First, there are all nodes of the deceleration lane that have improving switches to s. Second, r has an improving switch to $\mu_1^n(\mathfrak{b})$. Third, lower closed cycles (all lower cycles are closed!) have an improving switch to $\mu_1^n(\mathfrak{b})$ (opening them again). Fourth, all lower selector nodes have an improving switch to $\mu_1^n(\mathfrak{b})$. By performing all these switches, we enter phase 4.

The fourth and last phase, called the *reset* phase, corresponds to a strategy σ that performed the full increment, i.e. $b_\sigma = \mathfrak{b} + 1$. However, the access states are not reset, i.e. $a_\sigma = \mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1]$ and the deceleration lane is moving to root s. By switching the lane back to the initial configuration and the access states to match the simple cycles states, we end up in phase 1 again that corresponds to the incremented global counter state.

Technicalities

In this paragraph, we formalize the phases and prove the claimed transitions correct. For the sake of this paragraph, let σ be a strategy and $\mathfrak{b} \in \mathcal{B}_n$ be a global counter state. All phases will be defined w.r.t. σ and $\mathfrak{b} \in \mathcal{B}_n$. Let $\sigma' = \textsc{Switch-All}(\sigma)$.

To keep everything as simple as possible and to be able to prove all the lemmata without considering special cases, we will assume that $\mathfrak{b}$ is different from $\mathbf{0}$ and that the two highest bits in $\mathfrak{b}$ are zero and remain zero, i.e. we will only use the first $n-2$ bits for counting. Note however, that every bit works as intended in the counter.

Recall that every strategy σ occurring will be well-behaved. In addition to the deceleration lane and the cycle gates, we have two more structures that are controlled by a strategy σ, namely the two roots r and s, and the cycle gate output nodes k_i. We write $\sigma(r) = i$ to denote that $\sigma(r) = g_i$, and $\sigma(r) = n+1$ if $\sigma(r) = x$; we write $\sigma(s) = i$ to denote that $\sigma(s) = f_i$, and $\sigma(s) = n+1$ if $\sigma(s) = x$; we write $\sigma(k_i) = j$ to denote that $\sigma(k_i) = g_j$, and $\sigma(k_i) = n+1$ if $\sigma(k_i) = x$. We also use a more compact notation for the strategy decision of d_i-nodes of open cycles. We write $\sigma(d_i) = j$ if $\sigma(d_i) = a_j$.

Recall that we say that a strategy σ is *rooted* in s or r, if every path in the deceleration lane conforming to σ eventually exits to s resp. r. Likewise, we say

that σ has index i if all nodes of the deceleration lane with smaller index $j < i$ are moving down the lane by σ, and that i is the first index which is directly exiting through the root.

We now proceed to the formal definition of all phases. We say that σ is a $\mathfrak{b}$-phase 1 strategy iff all the following conditions hold:

1. $\mathfrak{b} = b_\sigma = a_\sigma$, i.e. set bits correspond to closed and accessed cycle gates, while unset bits correspond to open and skipped cycle gates,
2. $root(\sigma) = r$, i.e. the strategy is rooted in r,
3. $\sigma(s) = \sigma(r) = \nu_1^n(\mathfrak{b})$, i.e. both roots are connected to the least set bit,
4. $\sigma(k_i) = \nu_{i+1}^n(\mathfrak{b})$, i.e. the selector nodes move to the next set bit,
5. $ind(\sigma) \leq 2\mu_1^n(\mathfrak{b}) + 2$, i.e. the deceleration lane has not passed the least unset bit, and
6. $\sigma(d_j) \neq ind(\sigma) - 1$ for all j with $\mathfrak{b}_j = 0$, i.e. every open cycle node is not connected to the best-valued node of the lane.

Lemma 4.35. *Let σ be a $\mathfrak{b}$-phase 1 strategy with $ind(\sigma) < 2\mu_1^n(\mathfrak{b}) + 2$. Then σ' is a $\mathfrak{b}$-phase 1 strategy with $ind(\sigma') = ind(\sigma) + 1$, and if $ind(\sigma) > 1$, then $\sigma'(d_{\mu_1^n(\mathfrak{b})}) = ind(\sigma) - 1$.*

We say that σ is a $\mathfrak{b}$-phase 2 strategy iff all the following conditions hold:

1. $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1] = b_\sigma$ and $\mathfrak{b} = a_\sigma$, i.e. set bits correspond to closed and accessed (for all set bits except for $\mu_1^n(\mathfrak{b})$) cycle gates, while unset bits correspond to open and skipped cycle gates,
2. $root(\sigma) = r$, i.e. the strategy is rooted in r,
3. $\sigma(s) = \sigma(r) = \nu_1^n(\mathfrak{b})$, i.e. both roots are connected to the former least set bit,
4. $\sigma(k_i) = \nu_{i+1}^n(\mathfrak{b})$, i.e. the selector nodes move to the next set bit,

5. $ind(\sigma) \leq 2\mu_1^n(\mathfrak{b}) + 3$, i.e. the deceleration lane has not passed the next bit, and

6. $\sigma(d_j) \neq ind(\sigma) - 1$ for all $j > \mu_1^n(\mathfrak{b})$ with $\mathfrak{b}_j = 0$, i.e. every higher open cycle node is not connected to the best-valued node of the lane.

Lemma 4.36. *Let σ be a $\mathfrak{b}$-phase 1 strategy with $ind(\sigma) = 2\mu_1^n(\mathfrak{b}) + 2$ and $\sigma(d_{\mu_1^n(\mathfrak{b})}) = ind(\sigma)$. Then σ' is a $\mathfrak{b}$-phase 2 strategy.*

We say that σ is a $\mathfrak{b}$-phase 3 strategy iff all the following conditions hold:

1. $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1] = b_\sigma = a_\sigma$, i.e. set bits correspond to closed and accessed cycle gates, while unset bits correspond to open and skipped cycle gates,

2. $root(\sigma) = r$, i.e. the strategy is rooted in r,

3. $\sigma(s) = \mu_1^n(\mathfrak{b})$ and $\sigma(r) = \nu_1^n(\mathfrak{b})$, i.e. one root is connected to the new set bit and the other one is still connected to the former least set bit,

4. $\sigma(k_i) = \nu_{i+1}^n(\mathfrak{b})$, i.e. the selectors move to the former next set bit,

5. $\sigma(d_j) \neq s$ for all $j > \mu_1^n(\mathfrak{b})$ with $\mathfrak{b}_j = 0$, i.e. every higher open cycle node is not connected to the best-valued root node.

Lemma 4.37. *Let σ be a $\mathfrak{b}$-phase 2 strategy. Then σ' is a $\mathfrak{b}$-phase 3 strategy.*

We say that σ is a $\mathfrak{b}$-phase 4 strategy iff all the following conditions hold:

1. $\mathfrak{b} + 1 = b_\sigma$ and $\mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1] = a_\sigma$, i.e. set bits correspond to closed and accessed cycle gates, while unset bits correspond to open and skipped ($> \mu_1^n(\mathfrak{b})$) resp. accessed ($< \mu_1^n(\mathfrak{b})$) cycles gates,

2. $root(\sigma) = s$, i.e. the strategy is rooted in s,

3. $\sigma(s) = \sigma(r) = \mu_1^n(\mathfrak{b})$, i.e. both roots are connected to the new set bit,

4. $\sigma(k_i) = \nu_{i+1}^n(\mathfrak{b} + 1)$, i.e. the selectors move to the new next set bit,

5. $ind(\sigma) = 0$, i.e. the deceleration lane has reset, and

6. $\sigma(d_j) = s$ for all j with $(\mathfrak{b}+1)_j = 0$, i.e. every open cycle node is connected to the s root.

Lemma 4.38. *Let σ be a $\mathfrak{b}$-phase 3 strategy. Then σ' is a $\mathfrak{b}$-phase 4 strategy.*

Lemma 4.39. *Let σ be a $\mathfrak{b}$-phase 4 strategy and $\mathfrak{b}+1 \neq \mathbf{1}_n$. Then σ' is a $\mathfrak{b}+1$-phase 1 strategy with $ind(\sigma') = 1$.*

Finally, we are ready to prove that our family of games really implements a binary counter. From Lemmata 4.35, 4.36, 4.37, 4.38 and 4.39, we immediately derive the following.

Lemma 4.40. *Let σ be a phase 1 strategy and $b_\sigma \neq \mathbf{1}_n$. There is some $k \geq 4$ s.t. $\sigma' = \text{SWITCH-ALL}^k(\sigma)$ is a phase 1 strategy and $b_{\sigma'} = b_\sigma + 1$.*

Results

Particularly, we conclude that strategy improvement with the SWITCH-ALL rule requires exponentially many iterations on G_n.

Theorem 4.41. *Let $n > 0$. Parity game strategy iteration with* SWITCH-ALL*-rule requires at least 2^n improvement steps on G_n.*

Since G_n is a family of sink parity games, it follows directly by Theorem 4.19 that we have an exponential lower bound for payoff games.

Corollary 4.42. *Payoff game strategy iteration with* SWITCH-ALL*-rule requires exponential time.*

Fearnley [Fea10] was the first to notice that the family of parity games G_n can be translated to Markov decision processes, hence we have the same lower bound here as well.

Theorem 4.43 ([Fea10])**.** *Markov decision process strategy iteration parameterized with the* SWITCH-ALL*-rule requires exponential time.*

One could conjecture that sink parity games form a "degenerate" class of parity games as they are always won by player 1. Remember that the problem of solving parity games is to determine the complete winning sets for both players.

In other words: Is there a family of games on which the strategy improvement algorithm requires exponentially many iterations to find a player 0 strategy that wins at least one node in the game?

The answer to this question is positive. Simply take our lower bound games G_n and remove the edge from e_n to h_n. Remember that the first time player 1 wants to use this edge by best response is, when the binary counter is about to flip bit n, i.e. after it processed 2^{n-1} many counting steps. Eventually, the player 0 strategy is updated s.t. $\sigma(d_n) = e_n$, forcing player 1 by best response to move to h_n. Removing this edge leaves player 1 no choice but to stay in the cycle which is dominated by player 0.

Theorem 4.44. *Parity game strategy iteration with* SWITCH-ALL*-rule requires exponential time to decide the winner of a node.*

Improvement: Linear Number of Edges

Consider the lower bound construction again. It consists of a deceleration lane, cycle gates, two roots and connectives between these structures. All three kinds of structures only have linearly many edges when considered on their own. The quadratic number of edges is solely due to the d_*-nodes of the simple cycles of the cycle gates that are connected to the deceleration lane and due to the k_*-nodes of the cycle gates that are connected to all higher cycle gates.

We focus on the edges connecting the d_*-nodes with the deceleration lane first. Their purpose is twofold: lower cycle gates have less edges to the deceleration lane (so they close first), and as long as an open cycle gate should be prevented from closing, there must be a directly accessible lane input node in every iteration with a better valuation than the currently chosen lane input node.

Instead of connecting d_i to all a_j with $j < 2i + 1$ nodes, it would suffice to connect d_i to two intermediate nodes, say y_i and z_i, that are controlled by player 0

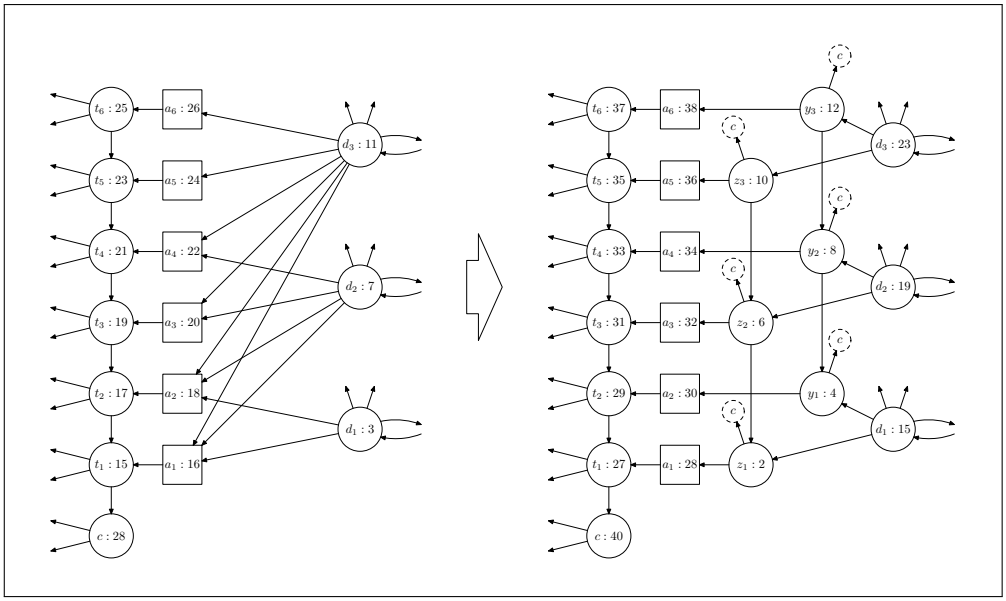


Figure 4.13: Intermediate Layer

with negligible priorities. We connect z_i to all a_j with even $j < 2i + 1$ and y_i to all a_j with odd $j < 2i + 1$. By this construction, we shift the "busy updating"-part alternately to y_i and z_i, and d_i remains updating as well by switching from y_i to z_i and vice versa in every iteration.

Next, we observe that the edges connecting y_i (resp. z_i) to the lane are a proper subset of the edges connecting y_{i+1} (resp. z_{i+1}) to the lane, and hence we adapt our construction in the following way. Instead of connecting y_{i+1} (and similarly z_{i+1}) to all a_j with even $j < 2i + 3$, we simply connect y_{i+1} to a_{2i+1} and to y_i. In order to ensure proper resetting of the two intermediate lanes constituted by y_* and z_* in concordance with the original deceleration, we need to connect every additional node to c. See Figure 4.13 for the construction (note that by introducing new nodes with "negligible priorities", we simply shift all other priorities in the game).

Second, we consider the edges connecting lower cycle gates with higher cycle gates. As the set of edges connecting k_{i+1} with higher g_j is a proper subset of k_i, we can apply a similar construction by attaching an additional lane to cycle gate connections that subsumes shared edges.

Improvement: Binary Out-degree

Every parity game can be linear-time-reduced to an equivalent (in the sense that winning sets and strategies can be easily related to winning sets and strategies in the original game) parity game with an edge out-degree bounded by two. See Figure 4.14 for an example of such a transformation.

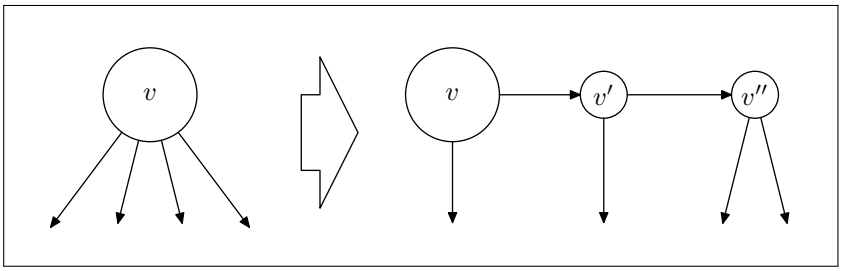


Figure 4.14: Binary Out-degree Transformation

However, not *every* such transformation that can be applied to our construction (for clarity of presentation, we start with our original construction again) yields games on which strategy iteration still requires an exponential number of iterations. We discuss the necessary transformations for every player 0 controlled node in the following, although we omit the exact priorities of additional helper nodes. It suffices to assign arbitrary even priorities to the additional nodes that lie below the priorities of all other nodes of the original game (except for the sink).

First, we consider the two root nodes s and r, that are connected to the sink x and to $f_1,\ldots, f_n$ resp. $g_1,\ldots,g_n$. As r copies the decision (see the transition from the *access* to the *reset* phase) of s, it suffices to describe how the out-degree-two transformation is to be applied to s. We introduce n additional helper nodes $s'_1,\ldots,s'_n$, replace the outgoing edges of s by x and s'_n, connect s'_{i+1} with f_{i+1} and s'_i, and finally s'_1 simply with f_1.

It is still possible to show that s reaches the best valued f_i after one iteration. Assume that s currently reaches some cycle gate i via the ladder that is given by the helper nodes. Let j be the next best-valued cycle gate that just has been set. If $j > i$, it follows that s currently reaches s'_j that moves to s'_{j-1}, but updates within one iteration to f_j. If $j < i$, it must be the case that $j = 1$ (i is the least bit which was

set; j is the least bit which was unset). Moreover, s currently reaches s'_i that moves to f_i. All lower s'_{k+1} with $k+1 < i$ move to s'_k since lower unset cycle gates are more profitable than higher unset cycle gates (unset cycle gates eventually reach one of the roots via the unprofitable f_* nodes). Hence, s'_i updates within one iteration to s'_{i-1}.

Second, there are the output nodes of cycle gates k_1,..., k_n. We apply a very similar ladder-style construction here. For every k_i, we introduce $n-i$ additional helper nodes $k'_{i,j}$ with $i < j \leq n$, replace the outgoing edges of k_i by x and $k'_{i,i+1}$, connect $k'_{i,j}$ with g_j and $k'_{i,j+1}$ (if $j < n$). The argument why this construction suffices runs similarly as for the root nodes.

Third, there are the nodes t_1,...,t_{2n} of the deceleration lane that are connected to three nodes. Again, we introduce an additional helper node t'_i for every t_i, and replace the two edges to r and t_{i-1} resp. c by an edge to t'_i that is connected to r and t_{i-1} resp. c instead. It is not hard to see that this slightly modified deceleration lane still provides the same functionality.

Finally, there are the player 0 controlled nodes d_1,...,d_n of the simple cycles of the cycle gates. Essentially, two transformations are possible here. Both replace d_i by as many helper nodes $d'_{i,x}$ as there are edges from d_i to any other node x but e_i. Then, every $d'_{i,x}$ is connected to the target node x.

The first possible transformation connects every $d'_{i,x}$ with e_i and vice versa, yielding a multicycle with e_i as the center of each cycle. The second possible transformation connects e_i with the first d'_{i,x_1}, d'_{i,x_1} with d'_{i,x_2} etc. and the last d'_{i,x_l} again with e_i, yielding one large cycle. Both replacements behave exactly as the original simple cycle.

The transformation described here results in a quadratic number of nodes since we started with a game with a quadratic number of edges. We note, however, that a similar transformation can be applied to the version of the game with linearly many edges, resulting in a game with binary out-degree of linear size.

4.6.2 Switch Best Rule

The SWITCH-BEST or *globally optimizing rule* computes a globally optimal successor strategy in the sense that the associated valuation is the best under all allowed successor strategies.

SWITCH-BEST: Apply the best possible combination of switches.

More formally, given an *escape* (see [Sch08]) payoff game G, an improvable strategy σ and the improved strategy $\sigma^* = \text{SWITCH-BEST}(\sigma)$, it holds for every non-empty applicable subset of improving switches I that $\Xi_{\sigma[I]} \trianglelefteq \Xi_{\sigma^*}$.

The rule can be interpreted as providing strategy improvement with a *one-step lookahead*; it computes the optimal strategy under all possible strategies that can be reached by a single improvement step.

The interested reader is pointed to Schewe's paper [Sch08] for all the details on how to effectively compute the optimal strategy update. This computation is only defined for *escape payoff games*, which correspond to sink parity games in the parity game world.

Theorem 4.45 ([Sch08])**.** *The* SWITCH-BEST *rule can be computed in polynomial time for deterministic escape payoff games.*

One may be misled to combine the existence of an (artificial) improvement rule SWITCH-LIN, that enforces linearly many iterations in the worst case, with the existence of the improvement rule SWITCH-BEST, that selects the optimal successor strategy in each iteration, in order to propose that SWITCH-BEST should also enforce linearly many iterations in the worst case.

The reason why this proposition is incorrect lies in the intransitivity of optimality of strategy updates. Although $\text{SWITCH-LIN}(\sigma) \trianglelefteq \text{SWITCH-BEST}(\sigma)$ for every strategy σ, this is not necessarily the case for iterated applications, i.e.

$$\text{SWITCH-LIN}(\text{SWITCH-LIN}(\sigma)) \trianglelefteq \text{SWITCH-BEST}(\text{SWITCH-BEST}(\sigma))$$

does not necessarily hold for all strategies σ.

The lower bound construction for the globally optimizing rule again is a family of sink parity games that implement a binary counter by a combination of a (modified) deceleration lane and a chain of (modified) cycle gates.

This chapter is organized as follows. First, we discuss the modifications of the deceleration lane and the cycle gates and why they are required to obtain a lower bound for the globally optimizing rule. Then, we present the full construction along with some remarks to the correctness.

The main difference between the locally optimizing rule and the globally optimizing rule is that the latter takes cross-effects of improving switches into account. It is aware of the impact of any combination of profitable edges, in contrast to the locally optimizing rule that only sees the local valuations, but not the effects.

One example that separates both rules are the simple cycles of the previous chapter: the SWITCH-ALL rule sees that closing a cycle is an improvement, but not that the actual profitability of closing a cycle is much higher than updating to another node of the deceleration lane.

The globally optimizing rule, on the other hand, is well aware of the profitability of closing the cycle in one step. In some sense, the rule has the ability of a *one-step lookahead*. However, our lower bound for the globally optimizing rule is not so different from the original construction – the trick is to hide very profitable choices by structures that cannot be solved by a single strategy iteration. In other words, we simply need to replace the gadgets that can be solved with a one-step lookahead by slightly more complicated variations that cannot be solved within one iteration *and* that maintain this property for as long as it is necessary.

Modified Deceleration Lane

The modified deceleration lane looks almost the same as the original deceleration lane. It has again several, say m, input nodes $a_1, \ldots, a_m$ along with some special input node c. We have two output *roots*, r and s, this time with a slightly different connotation. We call r the *default root* and s the *reset root*.

More formally, a modified deceleration lane consists of m (in our case, m will be $6 \cdot n - 2$) internal nodes $t_1, \ldots, t_m$, m input nodes $a_1, \ldots, a_m$, one additional input node c, the default root output node r and the reset root output node s.

All priorities of the modified deceleration lane are based on some odd priority p. We assume that all root nodes have a priority greater than $p + 2m + 1$. The structural difference between the modified deceleration lane and the original one is that the lane base c only has one outgoing edge leading to the default root r. The players, priorities and edges are described in Table 4.11.

Node	Player	Priority	Successors
t_1	0	p	$\{s,\ r,\ c\}$
$t_{i>1}$	0	$p + 2i - 2$	$\{s,\ r,\ t_{i-1}\}$
c	1	$p + 2m + 1$	$\{r\}$
a_i	1	$p + 2i - 1$	$\{t_i\}$
s	?	$> p + 2m + 1$	?
r	?	$> p + 2m + 1$	?

Table 4.11: Description of the Modified Deceleration Lane

The intuition behind the two roots is the same as before. The default root r serves as an entry point to the cycle gate structure and the reset root s is only used for a short time to reset the whole deceleration lane structure.

We describe the state of a modified deceleration lane again by a tuple specifying which root has been chosen and by how many t_i nodes are already moving down to c. Formally, we say that σ *is in deceleration state* (x, j) (where $x \in \{s, r\}$ and $0 < j \leq m + 1$ a natural number) iff

1. $\sigma(t_1) = c$ if $j > 1$,
2. $\sigma(t_i) = t_{i-1}$ for all $1 < i < j$, and
3. $\sigma(t_i) = x$ for all $j \leq i$.

The modified deceleration lane treats the two roots differently. If the currently best-valued root is the reset root, it is the optimal choice for all t_*- nodes to directly

move to the reset root. In other words, no matter what state the deceleration lane is currently in, if the reset root provides the best valuation, it requires exactly one improvement step to reach the optimal setting.

If the currently best-valued root is the default root, however, it is profitable to reach the root via the lane base c. The globally optimizing rule behaves in this case just like the locally optimizing rule, because the deceleration lane has exactly one improving switch at a time which is also globally profitable.

The following lemma formalizes the intuitive description of the deceleration lane's behavior: a change in the ordering of the root valuations leads to a reset of the deceleration lane, otherwise the lane continues to align its edges to eventually reach the best-valued root node via c.

It is notable that resetting the lane by an external event (i.e. by giving s a better valuation than r) is a bit more difficult than in the case of the locally optimizing rule. Let σ be a strategy and $\sigma' = \text{SWITCH-BEST}(\sigma)$. Assume that the current state of the deceleration lane is (r, i) and now we have that s has a better valuation than r, i.e. $s \succ_\sigma r$. Assume further – which for instance applies to our original lower bound construction – that the next strategy σ' assigns a better valuation to r again, i.e. $r \succ_{\sigma'} s$. Therefore, it would *not* be the globally optimal choice to reset the deceleration lane to s, but instead just to keep the original root r.

In other words, the globally optimizing rule uses its one-step lookahead property to refrain from resetting the lane if the resetting event persists for only one iteration. The solution to fool the one-step lookahead, however, is not too difficult: we just alter our construction in such a way that the resetting root will have a better valuation than the default root for *two* iterations.

Lemma 4.46. *Let σ be a strategy that is in deceleration state (x, i). Let $\bar{x}$ denote the other root. Let $\sigma' = \text{SWITCH-BEST}(\sigma)$.*

1. *$r \succ_\sigma s$, $x = r$ implies that σ' is in state $(r, \min(m, i) + 1)$.*
2. *$\bar{x} \succ_\sigma x$ and $\bar{x} \succ_{\sigma'} x$ implies that σ' is in state $(\bar{x}, 1)$.*

The purpose of the modified deceleration lane is exactly the same as before: we absorb the update activity of cyclic structures that represent the counting bits of the lower bound construction.

Stubborn Cycles

With the locally optimizing rule, we employed simple cycles and hid the fact that the improving edge leading into the simple cycle results in a much better valuation than updating to the next best-valued node of the deceleration lane.

However, simple cycles do not suffice to fool the globally optimizing rule. If it is possible to close the cycle within one iteration, the rule uses its one-step lookahead feature to see that closing the cycle is much more profitable than updating to the deceleration lane.

The solution to this problem is to replace the simple cycle structure by a cycle consisting of more than one player 0 node s.t. it is impossible to close the cycle within one iteration. More precisely, we use a structure consisting again of one player 1 node e and three player 0 nodes d^1, d^2 and d^3, called *stubborn cycle*. We connect all four nodes with each other in such a way that they form a cycle, and connect all player 0 nodes with the deceleration lane. See Figure 4.15 for an example of such a situation.

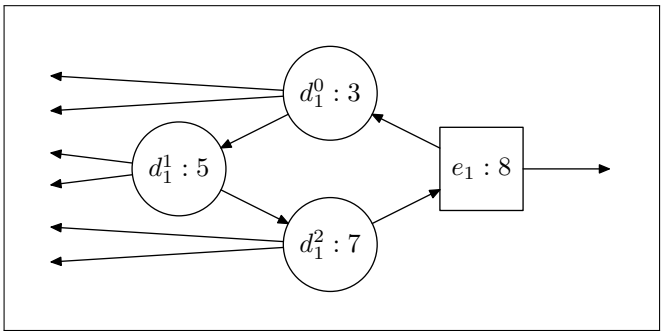


Figure 4.15: A Stubborn Cycle

More precisely, we connect the player 0 nodes in a *round robin* manner to the deceleration lane, for instance d^1 to $a_3, a_6, \ldots$, d^2 to $a_2, a_5, \ldots$, and d^3 to $a_1, a_4, \ldots$. We assume that it is more profitable for player 1 to move into the cyclic structure as long as it is not closed.

Now let σ be a strategy s.t. σ is in state $(r, 6)$ and $\sigma(d^1) = a_3$, $\sigma(d^2) = d^3$ and $\sigma(d^3) = a_4$. There are exactly two improving switches here: d^2 to a_5 (which is the best-valued deceleration node) and d^1 to d^2 (because d^2 currently reaches a_4 via d^3

which has a better valuation than a_3). In fact, the combination of both switches is the optimal choice.

A close observation reveals that the improved strategy has essentially the same structure as the original strategy σ: two nodes leave the stubborn cycle to the deceleration lane and one node moves into the stubborn cycle. By this construction, we can ensure that cycles are not closed within one iteration. In other words, the global rule makes no progress towards closing the cycle (it switches one edge towards the cycle, and one edge away from the cycle, leaving it in the exact same position).

Modified Cycle Gate

We again use a slightly modified version of the cycle gates as a pass-through structure that is either very profitable or quite unprofitable. Essentially, we apply two modifications. First, we replace the simple cycle by a stubborn cycle, for the reasons outlined in the previous paragraph. Second, we put an additional player 0 controlled internal node y_i between the input node g_i and the internal node f_i. It will delay the update of g_i to move to the stubborn cycle after closing the cycle by one iteration. By this, we ensure that the modified deceleration lane will have enough time to reset itself.

Formally, a modified cycle gate consists of three internal nodes e_i, h_i and y_i, two input nodes f_i and g_i, and four output nodes d_i^1, d_i^2, d_i^3 and k_i. The output node d_i^1 (resp. d_i^2 and d_i^3) will be connected to a set of other nodes D_i^1 (resp. D_i^2 and D_i^3) in the game graph, and k_i to some set K_i.

All priorities of the cycle gate are based on two odd priorities p_i and p_i'. See Figure 4.16 for a cycle gate of index 1 with $p_1' = 3$ and $p_1 = 65$. The players, priorities and edges are described in Table 4.12.

From an abstract point of view, we describe the state of a modified cycle gate again by a pair $(\beta_i(\sigma), \alpha_i(\sigma)) \in \{0,1,2,3\} \times \{0,1,2\}$. The first component describes the state of the stubborn cycle, counting the number of edges pointing into

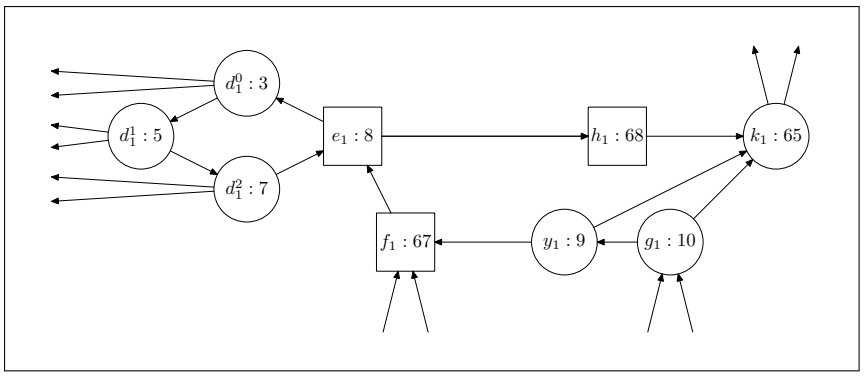


Figure 4.16: A Modified Cycle Gate (index 1 with $p'_1 = 3$ and $p_1 = 65$)

Node	Player	Priority	Successors
d_i^1	0	p'_i	$\{d_i^2\} \cup D_i^1$
d_i^2	0	$p'_i + 2$	$\{d_i^3\} \cup D_i^2$
d_i^3	0	$p'_i + 4$	$\{e_i\} \cup D_i^3$
e_i	1	$p'_i + 5$	$\{d_i^1,\ h_i\}$
y_i	0	$p'_i + 6$	$\{f_i,\ k_i\}$
g_i	0	$p'_i + 7$	$\{y_i,\ k_i\}$
k_i	0	p_i	K_i
f_i	1	$p_i + 2$	$\{e_i\}$
h_i	1	$p_i + 3$	$\{k_i\}$

Table 4.12: Description of the Modified Cycle Gate

the cycle, and the second component gives the state of the two access control nodes. Formally, we have the following:

$$\beta_i(\sigma) = |\{d_i^j \mid \sigma(d_i^j) \notin D_i^j\}| \qquad \alpha_i(\sigma) = \begin{cases} 2 & \text{if } \sigma(g_i) = y_i \\ 0 & \text{if } \sigma(g_i) = \sigma(y_i) = k_i \\ 1 & \text{otherwise} \end{cases}$$

The behavior is formalized in terms of modified cycle gate states as follows. Intuitively, it functions as the original cycle gates: if the cycle is σ-closed and remains closed (one-step lookahead), it is profitable to go through the cycle gate. If the cycle opens by some external event and remains open (one-step lookahead again), it is more profitable to directly move to the output node instead.

Lemma 4.47. *Let σ be a strategy and $\sigma' = \text{SWITCH-BEST}(\sigma)$.*

1. *If $\beta_i(\sigma) = \beta_i(\sigma') = 3$, we have $\alpha_i(\sigma') = \min(\alpha_i(\sigma) + 1, 2)$ ("closed gates will be successively accessed").*
2. *If $\beta_i(\sigma) < 3$, $\beta_i(\sigma') < 3$ and $\sigma(k_i) \succ_{\sigma'} f_i$, we have $\alpha_i(\sigma') = 0$ ("open gates with unprofitable exit nodes will be skipped").*

We use modified cycle gates again to represent the bit states of a binary counter: unset bits will correspond to modified cycle gates with the state $(1, 0)$, set bits to the state $(3, 2)$. Setting and resetting bits therefore traverses more than one phase, more precisely, from $(1, 0)$ over $(2, 0)$, $(3, 0)$ and $(3, 1)$ to $(3, 2)$, and from the latter again over $(1, 2)$ to $(1, 0)$.

Modified Construction

In this paragraph, we provide the complete construction of the lower bound family for SWITCH-BEST. It again consists of a sink x, a modified deceleration lane of length $6n - 3$ that is connected to the two roots s and r, and n modified cycle gates. The stubborn cycles of the cycle gates are connected to the r root, the lane base c and to the deceleration lane. The modified cycle gates are connected to each other in the same manner as in the original lower bound structure for the locally optimizing rule.

The way the stubborn cycles are connected to the deceleration lane is more involved than in the previous lower bound construction. Remember that for all open stubborn cycles, we need to maintain the setting in which two edges point to the deceleration lane while the other points into the cycle. We achieve this task by assigning the three nodes of the respective stubborn cycle to the input nodes of the deceleration lane in a round-robin fashion.

The games are denoted by $H_n = (V_n, V_{n,0}, V_{n,1}, E_n, \Omega_n)$ and the sets of nodes are as follows:

$$V_n := \{x, s, c, r\} \cup \{a_i, t_i \mid 0 < i \leq 6n - 2\} \cup$$
$$\{d_i^1, d_i^2, d_i^3, e_i, f_i, h_i, g_i, y_i, k_i \mid 0 < i \leq n\}$$

The players, priorities and edges are described in Table 4.13. The game H_3 is depicted in Figure 4.17. The edges connecting the cycle gates with the deceleration lane are not included in the figure.

Node	Player	Priority	Successors
t_1	0	$8n+3$	$\{s,\ r,\ c\}$
$t_{i>1}$	0	$8n+2i+1$	$\{s,\ r,\ t_{i-1}\}$
a_i	1	$8n+2i+2$	$\{t_i\}$
c	1	$20n$	$\{r\}$
d_i^1	0	$8i+1$	$\{s,\ c, d_i^2\} \cup \{a_{3j+3} \mid j \leq 2i-2\}$
d_i^2	0	$8i+3$	$\{d_i^3\} \cup \{a_{3j+2} \mid j \leq 2i-2\}$
d_i^3	0	$8i+5$	$\{e_i\} \cup \{a_{3j+1} \mid j \leq 2i-1\}$
e_i	1	$8i+6$	$\{d_i^1,\ h_i\}$
y_i	0	$8i+7$	$\{f_i,\ k_i\}$
g_i	0	$8i+8$	$\{y_i,\ k_i\}$
k_i	0	$20n+4i+3$	$\{x\} \cup \{g_j \mid i < j \leq n\}$
f_i	1	$20n+4i+5$	$\{e_i\}$
h_i	1	$20n+4i+6$	$\{k_i\}$
s	0	$20n+2$	$\{f_j \mid j \leq n\} \cup \{x\}$
r	0	$20n+4$	$\{g_j \mid j \leq n\} \cup \{x\}$
x	1	1	$\{x\}$

Table 4.13: Lower Bound Construction for SWITCH-BEST

Fact 4.48. *The game H_n has $21 \cdot n$ nodes, $3.5 \cdot n^2 + 40.5 \cdot n - 4$ edges and $24 \cdot n + 6$ as highest priority. In particular, $|H_n| = \mathcal{O}(n^2)$.*

As an initial strategy we select the following strategy ι. Again, it corresponds to a global counter setting in which no bit has been set.

$$\iota(t_1) = c \qquad \iota(t_{1<i\leq 3}) = t_{i-1} \qquad \iota(t_{i>3}) = r \qquad \iota(c) = r$$

$$\iota(d_i^1) = d_i^2 \qquad \iota(d_i^2) = a_2 \qquad \iota(d_i^3) = a_1 \qquad \iota(g_i) = k_i$$

$$\iota(y_i) = k_i \qquad \iota(k_i) = x \qquad \iota(s) = x \qquad \iota(r) = x$$

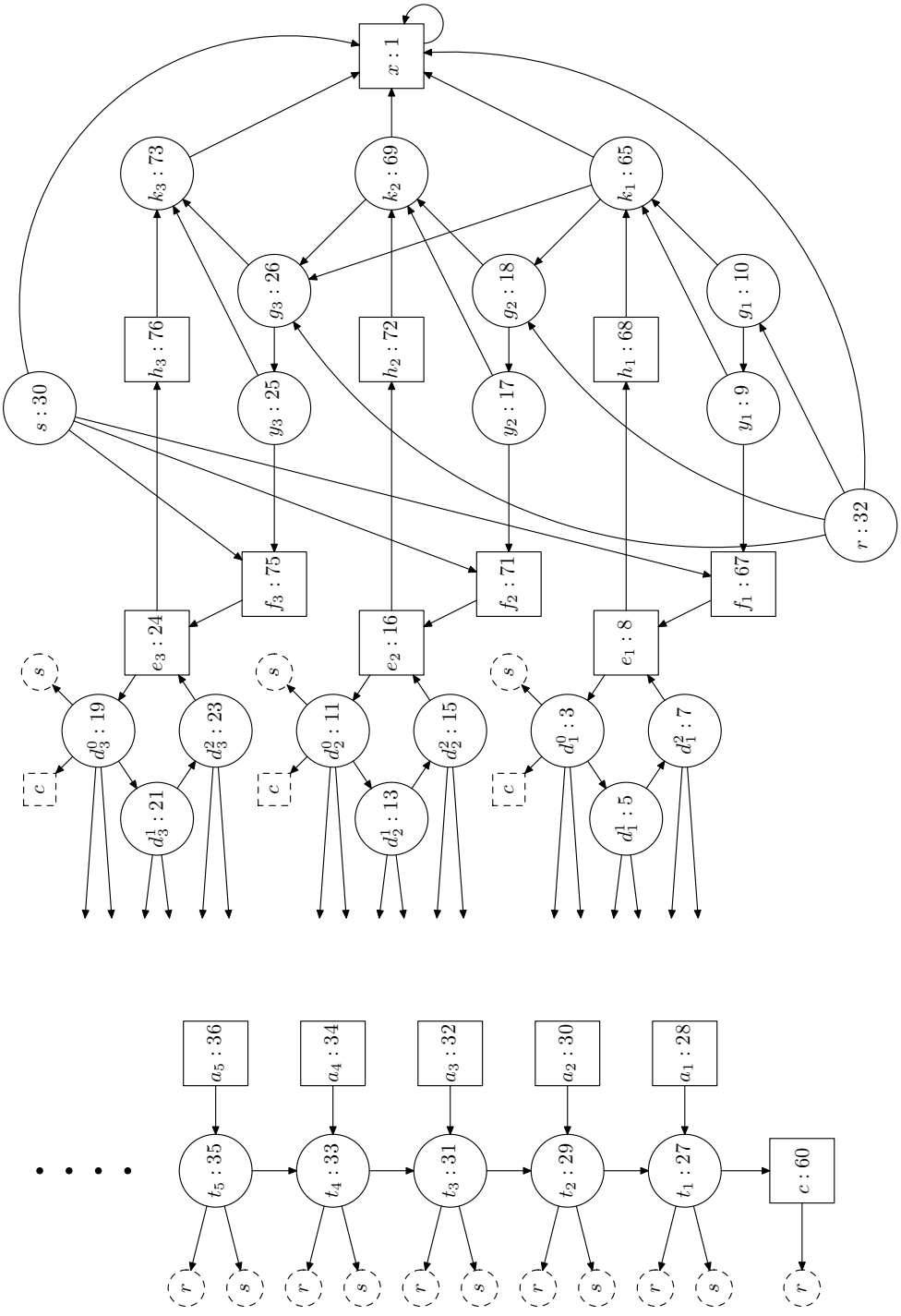


Figure 4.17: SWITCH-BEST Lower Bound Game H_3

It is easy to see that the H_n family again is a family of sink games.

Lemma 4.49. *Let $n > 0$.*

1. *The game H_n is completely won by player 1.*
2. *x is the sink of H_n and the cycle component of $\Xi_\iota(w)$ equals x for all w.*

Again, we note that it is possible to refine the family H_n in such a way that it only comprises a linear number of edges and only out-degree two.

Results

The way to prove the construction corrects runs almost exactly the same as for the locally optimizing rule. Every global counting step is separated into some counting iterations of the deceleration lane with busy updating of the open stubborn cycles of the cycle gates until the least significant open cycle closes. Then, resetting of the lane, reopening of lower cycles and alignment of connecting edges is carried out.

Theorem 4.50. *Let $n > 0$. Parity game strategy iteration with* SWITCH-BEST*-rule requires at least 2^n improvement steps on H_n.*

Since H_n is a family of sink parity games, it follows directly by Theorem 4.19 that we have an exponential lower bound for deterministic payoff games.

Corollary 4.51. *Payoff game strategy iteration with* SWITCH-BEST*-rule requires exponential time.*

4.7 Probabilistic Rules

We consider the randomized pivoting rule RANDOM-EDGE, which among all improving switches chooses one uniformly at random. We also consider RANDOM-FACET, a more complicated randomized pivoting rule suggested by Matoušek, Sharir and Welzl [MSW96]. Our lower bound for the RANDOM-FACET pivoting rule essentially matches the subexponential upper bound of Matoušek *et al.* [MSW96]. Lower

bounds for RANDOM-EDGE and RANDOM-FACET were known before only in *abstract* settings, and not for concrete linear programs or infinitary payoff games.

We show that both RANDOM-EDGE and RANDOM-FACET may lead to an expected subexponential number of iterations on actual linear programs. More specifically, we construct concrete linear programs on which the expected number of iterations performed by RANDOM-EDGE is $2^{\Omega(n^{1/4})}$, where n is the number of variables, and (different) linear programs on which the expected number of iterations performed by RANDOM-FACET is $2^{\Omega(\sqrt{n}/\log^c n)}$, for some fixed $c > 0$.

The linear programs on which RANDOM-EDGE and RANDOM-FACET perform an expected subexponential number of iterations are obtained using the close relation between simplex-type algorithms for solving linear programs and policy iteration for solving infinitary payoff games.

Our lower bound for the RANDOM-EDGE policy iteration for parity games and related two-player games can be extended to arbitrary *randomized multi-switch* improvement rules which select in each iteration step a subset with a certain cardinality of the improving switches arbitrarily at random. RANDOM-EDGE, for instance, always selects subsets with cardinality one, and the deterministic SWITCH-ALL rule always selects the subset with maximal cardinality. Another important randomized multi-switch improvement rule is SWITCH-HALF [MS99], which applies every improving switch with probability $1/2$. The lower bound transfers to all randomized multi-switch improvement rules in that sense.

All technically tedious proofs have been put into Appendix A.3.

4.7.1 Random Facet Rule

The RANDOM-FACET algorithm of Matoušek, Sharir and Welzl [MSW96] is a very simple randomized algorithm for solving LP-type problems. Since parity games and the other classes of games considered here, are LP-type problems [Hal07], the algorithm can be used to solve these games, as was done by Ludwig [Lud95], Petersson and Vorobyov [PV01b], and Björklund *et al.* [BSV03, BV05, BV07].

The algorithm is a *recursive algorithm*, operating on the *dual* of the linear programs induced by Markov decision processes. In the context of infinitary payoff

games, the algorithm can be formulated as follows: The recursion operates on a set of player 0 controlled edges F and on a given initial strategy σ that uses edges included in F. The task of RANDOM-FACET is to compute the optimal strategy w.r.t. the edge set F starting with strategy σ. Obviously, when started with the full edge set and an arbitrary strategy σ, this procedure should return the optimal strategy for the whole game.

RANDOM-FACET:	Compute optimal strategy by recursive exclusion of unused single edges

For concreteness, we describe the operation of the RANDOM-FACET algorithm on parity games. Let $G = (V_0, V_1, E, \Omega)$ be a parity game. Recall that we write $E_0 = E \cap (V_0 \times V)$ to denote the set of player 0 controlled edges, and likewise $E_1 = E \cap (V_1 \times V)$ to denote the set of player 1 controlled edges. Let $F \subseteq E_0$ be a subset of player 0 controlled edges s.t. for every $v \in V_0$ we have a $w \in V$ s.t. $(v, w) \in F$. The game G_F is defined as the subgame of G in which player 0 only has the choices included in F, i.e. $G_F = G|_{E_1 \cup F}$.

Let σ be an initial strategy for player 0. If $\sigma = E_0$, then σ is the only possible strategy for player 0, and is thus also an optimal strategy. Otherwise, the algorithm chooses, uniformly at random, an edge $e \in E_0 \setminus \sigma$ and applies the algorithm recursively on the subgame $G \setminus \{e\} = (V_0, V_1, E \setminus \{e\}, \Omega)$, the game obtained by removing e from G, with the initial strategy σ. The recursive call returns a strategy σ' which is an optimal strategy for player 0 in $G \setminus \{e\}$. If e is not an improving switch for σ', then σ' is also an optimal strategy in G. Otherwise, the algorithm performs the switch $\sigma'[e]$, and recursively calls the algorithm on G, with initial strategy $\sigma'[e]$.

For pseudo-code, see Algorithm 7. By Corollary 4.21 and Lemma 4.22, it is easy to see that this formulation coincides with Algorithm 3 on page 33 for solving linear programs.

It follows from the analysis of [MSW96] that the *expected number of switches* performed by the RANDOM-FACET algorithm on any parity game is $2^{O(\sqrt{n \log n})}$, where $n = |V_0|$ is the number of vertices controlled by player 0 in G.

We also consider a variant RANDOM-FACET* of the RANDOM-FACET algorithm, see Algorithm 8. This variant receives, as a third argument, an *index function*

Algorithm 7 The RANDOM-FACET algorithm

```
1:  procedure RANDOM-FACET(G,σ)
2:      if E_0 = σ then
3:          return σ
4:      else
5:          Choose e ∈ E_0 \ σ uniformly at random
6:          σ' ← RANDOM-FACET(G \ {e}, σ)
7:          if Ξ_σ' ◁ Ξ_σ'[e] then
8:              σ'' ← σ'[e]
9:              return RANDOM-FACET(G, σ'')
10:         else
11:             return σ'
12:         end if
13:     end if
14: end procedure
```

$\texttt{ind} : E_0 \to \mathbb{N}$ that assigns each edge of E_0 a *distinct* natural number. Let $\mathcal{I}(G)$ be the set of all index functions w.r.t. G with range $\{1, 2, \ldots, |E_0|\}$. Instead of choosing a *random* edge e from $E_0 \setminus \sigma$, the algorithm now chooses the edge of $E_0 \setminus \sigma$ with the *smallest index*. We show below that the expected running time of this modified algorithm, when ind is taken to be a *random permutation* of E_0, is exactly equal to the expected running time of the original algorithm. We find it more convenient to work with the modified algorithm. The fact that the ordering of the edges is selected in advance simplifies the analysis. All our results apply, of course, also to the original version.

Let G be a game and let $F \subseteq E_0$. Recall that G_F is the subgame of G in which only the edges of F are available for player 0. Let $\sigma \subseteq F$ be a strategy of player 0 in G_F. We let $\mathbb{E}(G_F, \sigma)$ be the expected number of iterations performed by the call RANDOM-FACET(G_F, σ). Also, we let $\mathbb{E}^*(G_F, \sigma)$ be the expected number of iterations performed by the call RANDOM-FACET$^*(G_F, \sigma, \texttt{ind})$ when ind is taken to be a random permutation of E_0.

We now have the following lemma by the linearity of the expectation; the random choices made by the two recursive calls made by RANDOM-FACET are allowed to be dependent.

Algorithm 8 Variant of the RANDOM-FACET algorithm

```
1:  procedure RANDOM-FACET*(G,σ,ind)
2:      if E_0 = σ then
3:          return σ
4:      else
5:          e ← argmin_{e'∈E_0\σ} ind(e')
6:          σ' ← RANDOM-FACET*(G \ {e}, σ, ind)
7:          if Ξ_{σ'} ◁ Ξ_{σ'[e]} then
8:              σ'' ← σ'[e]
9:              return RANDOM-FACET*(G, σ'', ind)
10:         else
11:             return σ'
12:         end if
13:     end if
14: end procedure
```

Lemma 4.52. *Let G be a game, $F \subseteq E_0$ and $\sigma \subseteq F$ be a player 0 strategy. Then* $\mathbb{E}(G_F, \sigma) = \mathbb{E}^*(G_F, \sigma)$.

We construct explicit parity games such that the expected running time of the RANDOM-FACET algorithm on an n-vertex game is $2^{\tilde{\Omega}(\sqrt{n})}$, where $\tilde{\Omega}(f) = \Omega(f/\log^c n)$, for some constant c. This matches, up to a polylogarithmic factor in the exponent, the upper bound given in [MSW96]. Our games also provide the first *explicit* construction of LP-type problems on which the RANDOM-FACET algorithm is not polynomial.

Then, we show how to transfer our construction to Markov decision processes and finally to concrete linear programs. The improving switches performed by the (abstract) RANDOM-FACET algorithm applied to an MDP correspond directly to the steps performed by the RANDOM-FACET pivoting rule on the corresponding linear program (assuming, of course, that the same random choices are made by both algorithms). The linear programs corresponding to our MDPs supply, therefore, concrete linear programs on which following the RANDOM-FACET pivoting rule leads to an expected subexponential number of iterations.

Randomized Counting

The parity games that we construct, on which the RANDOM-FACET algorithm performs an expected subexponential number of iterations, simulate a *randomized counter*. Such a counter is composed of n bits, all initially set to 0. We say that the i'th bit is *set* if $\mathfrak{b}(i) = 1$ and *resetting* if $\mathfrak{b}(i) = 0$.

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 for all $j \in N$, $\mathfrak{b}(j) = 0$. 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, all bits $j \in N \cap [i-1]$ are reset and the i'th bit is set.

Although it is more natural to increment the counter and then reset, resetting first corresponds to the behavior of the lower bound construction. Finally, a recursive randomized count is performed on $N \cap [i-1]$. A function RANDCOUNT that implements a randomized counter is given in Algorithm 9.

Algorithm 9 Counting with a randomized bit-counter

```
1:  procedure RandCount(N)
2:      if N ≠ ∅ then
3:          Choose i ∈ N uniformly at random
4:          RandCount(N \ {i})
5:          for j ∈ N ∩ [i − 1] do
6:              b(j) ← 0
7:          end for
8:          b(i) ← 1
9:          RandCount(N ∩ [i − 1])
10:     end if
11: end procedure
```

Let $g(n)$ be the expected number of steps (recursive calls) performed by an n-bit randomized counter. It is easy to see that $g(0) = 1$ and that

$$g(n) = 1 + g(n-1) + \frac{1}{n}\sum_{i=0}^{n-1} g(i) \quad , \quad \text{for } n > 0.$$

The asymptotic behavior of $g(n)$ is known quite precisely:

Lemma 4.53 ([FS09], p. 596-597)**.** *It holds that:*

$$g(n) \longrightarrow \frac{e^{2\cdot\sqrt{n}-\frac{1}{2}}}{\sqrt{\pi}\cdot n^{\frac{1}{4}}} \text{ for } n \to \infty.$$

Note that $g(n)$ is, thus, just of the right subexponential form. The challenge, of course, is to construct parity games on which the behavior of the RANDOM-FACET algorithm mimics the behavior of RANDCOUNT. In order to explain the idea for doing so, we examine a simplified version of our construction.

Consider the parity game shown in Figure 4.18, but ignore the shaded rectangle in the background and the fact that some arrows are bold.

Any strategy σ for player 0 encodes a counter state in the following way: $\mathfrak{b}(i) = 1$ iff $(b_i, \mathrm{B}_i), (a_i, \mathrm{A}_i) \in \sigma$. Note that player 1 can always reach T by staying in the left side, and that T has odd priority. Thus, if $\mathfrak{b}(i) = 1$ player 1, who plays a best reply to σ, will use the edge (B_i, c_i), and c_i has a large even priority. Similarly, if $\mathfrak{b}(i) = 1$, player 1 uses the edge $(\mathrm{A}_i, \mathrm{D}_i)$.

The importance of player 1's choice at A_i is that it determines whether lower set bits have access to c_i, which has the large even priority. If $(b_i, \mathrm{B}_i) \in \sigma$ and $(a_i, \mathrm{A}_i) \notin \sigma$, we say that the i'th bit is *resetting*, since it allows the RANDOM-FACET algorithm to remove (a_i, A_i), so that player 1 can block the access to c_i for lower set bits, which initiates the resetting behavior of the counter.

Now, suppose the RANDOM-FACET algorithm removes the edge (b_i, B_i) and solves the game recursively (by counting with the remaining bits). During the recursion we say that the i'th bit is *disabled*. The resulting strategy σ' by player 0 will correspond to the state of the counter where $\mathfrak{b}(j) = 1$, for $j \neq i$, and $(b_i, \mathrm{B}_i), (a_i, \mathrm{A}_i) \notin \sigma$. Using (b_i, B_i) will be an improving switch for player 0, so we get a new strategy $\sigma'' = \sigma'[(b_i, \mathrm{B}_i)]$ for which the i'th bit is resetting.

Next, suppose that (a_i, A_i) is removed and the game solved recursively. Note that this is only possible because the i'th bit is resetting and $(a_i, \mathrm{A}_i) \notin \sigma$. In particular, a corresponding edge could not be picked for a set bit. Since player 1 controls the

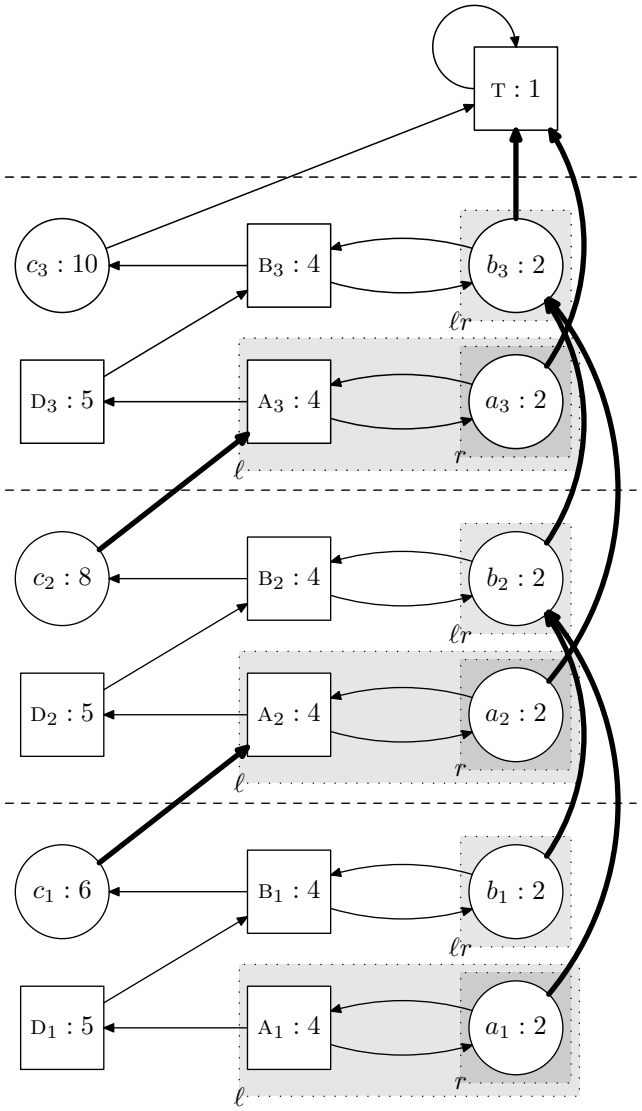


Figure 4.18: Lower bound construction for the RANDOM-FACET algorithm

left half of the graph, and player 0 is unable to use the edge (a_i, A_i), player 1 can avoid the large even priority at c_i by staying to the left until moving from A_i to a_i. Thus, player 0 can only reach c_i from vertices a_j, b_j, with $j < i$, by staying within the right half of the graph. It follows that all counter bits with index $j < i$ are reset. Using the edge (a_i, A_i) will now be an improving switch for player 0, so the strategy is updated such that $\mathfrak{b}(i) = 1$, and we are ready for a second round of counting with the lower bits. During this recursive call the higher set bits are said to be *inactive*.

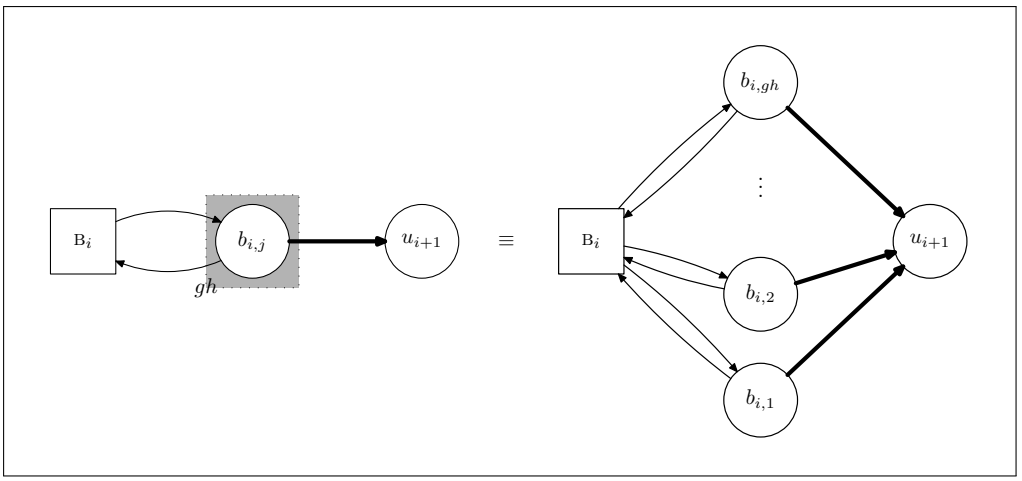


Figure 4.19: Duplication of a subgraph

Note that in order to ensure the correct behavior it was crucial that (b_i, B_i) was picked by the RANDOM-FACET algorithm before (a_i, A_i). It was also crucial that other edges from b_i and a_i, i.e. (b_i, b_{i+1}) and (a_i, b_{i+1}), were not removed. To increase the probability of that happening we make use of *duplication*. A shaded rectangle with ℓ or r in the bottom-left corner indicates that the corresponding subgraph is copied ℓ or r times, respectively; see Figure 4.19. Also, a bold edge indicates that the corresponding edge is copied r times. We show that it suffices for ℓ and r to be logarithmic in n to get a good bound on the probability.

As was the case with RANDOM-FACET, we can obtain a modified version of RANDCOUNT in which all random decisions are made in advance. In the case of RANDCOUNT, this corresponds to choosing a random permutation on $[n]$. Let $\mathcal{S}(n)$ be the set of permutations on $[n]$. A function RANDCOUNT* that implements a modified, randomized counter is given in Algorithm 10; $\varphi \in \mathcal{S}(n)$ is a permutation.

We end this section with a lemma showing that the expected number of steps performed by the original counter, RANDCOUNT, is equal to the expected number of steps performed by the modified counter, RANDCOUNT*, when given a random permutation. More precisely, let $f_n(N, \varphi)$ be the number of steps performed by a call RANDCOUNT$^*(N, \varphi)$, for some permutation $\varphi \in \mathcal{S}(n)$. Let $f_n(N)$ be the expected value of $f_n(N, \varphi)$ when $\varphi \in \mathcal{S}(n)$ is picked uniformly at random.

Lemma 4.54. *Let* $n \in \mathbb{N}$. *Then* $f_n([n]) = g(n)$.

Algorithm 10 Counting with an augmented randomized bit-counter

```
1:  procedure RandCount*(N, φ)
2:      if N ≠ ∅ then
3:          i ← argmin_{j∈N} φ(j)
4:          RandCount*(N \ {i}, φ)
5:          for j ∈ N ∩ [i − 1] do
6:              𝔟(j) ← 0
7:          end for
8:          𝔟(i) ← 1
9:          RandCount*(N ∩ [i − 1], φ)
10:     end if
11: end procedure
```

Full Construction

In this section we define a family of *lower bound games* $G_{n,\ell,r} = (V_0, V_1, E, \Omega)$ that the RANDOM-FACET algorithm requires many iterations to solve. n denotes the number of bits in the simulated randomized bit-counter, and $\ell \geq 1$ and $r \geq 1$ are parameters for the later analysis. We use multi edges for convenience.

A similar graph without multi edges can easily be defined by introducing additional vertices. $G_{n,\ell,r}$ is defined as follows:

$$V_0 := \{a_{i,j,k}\}_{[n]\times[\ell]\times[r]} \cup \{b_{i,j}\}_{[n]\times[\ell r]} \cup \{c_i\}_{[n]}$$

$$V_1 := \{\mathrm{A}_{i,j}\}_{[n]\times[\ell]} \cup \{\mathrm{B}_i\}_{[n]} \cup \{\mathrm{D}_i\}_{[n]} \cup \{\mathrm{T}\}$$

where $\{a_{i,j,k}\}_{[n]\times[\ell]\times[r]}$ is a shorthand for $\{a_{i,j,k} \mid i \in [n], j \in [\ell], k \in [r]\}$, etc. Table 4.14 defines the edge set E and the priority assignment Ω, where $b_{i,*}$ is a shorthand for $\{b_{i,j} \mid j \in [\ell r]\}$, and $(\mathrm{T})_r$ indicates that the edge has multiplicity r.

An example of a lower bound game with three bits, i.e., $n = 3$, is shown in Figure 4.18. A shaded rectangle with label ℓ indicates that the corresponding subgraph has been copied ℓ times. Bold arrows are multi edges with multiplicity r. Note that bold incoming edges for b_i, in fact, only go to the vertex $b_{i,1}$.

The initial strategy σ given as input to the RANDOM-FACET algorithm is described by $\sigma(b_{i,j}) \neq \mathrm{B}_i$ for all $i \in [n]$ and all $j \in [\ell r]$ as well as $\sigma(a_{i,j,k}) \neq \mathrm{A}_{i,j}$ for all $i \in [n]$, all $j \in [\ell]$ and all $k \in [r]$. The choice at vertex c_i is arbitrary.

Node	Successors	Priority	Node	Successors	Priority
$a_{i,j,k}$	$\text{A}_{i,j}, (b_{i+1,1})_r$	2	c_i	$(\text{A}_{i+1,*})_r$	$2i+4$
$a_{n,j,k}$	$\text{A}_{n,j}, (\text{T})_r$	2	c_n	T	$2n+4$
$b_{i,j}$	$\text{B}_i, (b_{i+1,1})_r$	2	$\text{A}_{i,j}$	$\text{D}_i, a_{i,j,*}$	4
$b_{n,j}$	$\text{B}_n, (\text{T})_r$	2	B_i	$c_i, b_{i,*}$	4
T	T	1	D_i	B_i	5

Table 4.14: Edges and Priorities of $G_{n,\ell,r}$

Optimal Strategies

The RANDOM-FACET algorithm operates with a subset of the edges controlled by player 0, $F \subseteq E_0$, such that the corresponding subgame G_F is a parity game. We next introduce notation to concisely describe F. We say that F is *complete* if it contains at least one instance of every multi edge. We define the set of multi edges without multiplicities as:

$$\begin{aligned} M = &\{(a_{i,j,k}, b_{i+1,1}) \mid i \in [n-1], j \in [\ell], k \in [r]\} \cup \\ &\{(a_{n,j,k}, \text{T}) \mid j \in [\ell], k \in [r]\} \cup \{(b_{i,j}, b_{i+1,1}) \mid i \in [n-1], j \in [\ell r]\} \cup \\ &\{(b_{n,j}, \text{T}) \mid j \in [\ell r]\} \cup \{(c_i, \text{A}_{i+1,j}) \mid i \in [n-1], j \in [\ell]\} \end{aligned}$$

Furthermore, for $F \subseteq E_0$ define:

$$b_i(F) = \begin{cases} 1 & \text{if } \forall j \in [\ell r] : (b_{i,j}, B_i) \in F \\ 0 & \text{otherwise} \end{cases}$$

$$a_{i,j}(F) = \begin{cases} 1 & \text{if } \forall k \in [r] : (a_{i,j,k}, A_{i,j}) \in F \\ 0 & \text{otherwise} \end{cases}$$

$$a_i(F) = \begin{cases} 1 & \text{if } \exists j \in [\ell] : a_{i,j}(F) = 1 \\ 0 & \text{otherwise} \end{cases}$$

That is, $b_i(F) = 1$ if and only if F contains every edge leading to B_i, and $a_{i,j}(F) = 1$ if and only if F contains every edge leading to $\text{A}_{i,j}$.

Finally, we define $reset(F) = \max(\{0\} \cup \{i \in [n] \mid b_i(F)=1 \wedge a_i(F)=0\})$ to be the maximum index i for which $b_i(F) = 1$ and $a_i(F) = 0$. Intuitively, all bits with index lower than i will be reset when computing the optimal strategy in the subgame G_F.

Let $F \subseteq E_0$ be a complete set. We say that a strategy $\sigma \subseteq F$ is *well-behaved* if for every $i \in [n]$, all copies $a_{i,j,k}$, for $j \in [\ell]$ and $k \in [r]$, and all copies $b_{i,j}$, for $j \in [\ell r]$, adopt corresponding choices, whenever possible. More formally, for every i, j_1, j_2, k_1, k_2: if $(a_{i,j_1,k_1}, \mathrm{A}_{i,j_1}), (a_{i,j_2,k_2}, \mathrm{A}_{i,j_2}) \in F$, then $\sigma(a_{i,j_1,k_1}) = \mathrm{A}_{i,j_1}$ if and only if $\sigma(a_{i,j_2,k_2}) = \mathrm{A}_{i,j_2}$. Similarly, for every i, j_1, j_2: if $(b_{i,j_1}, \mathrm{B}_i), (b_{i,j_2}, \mathrm{B}_i) \in F$, then $\sigma(b_{i,j_1}) = \mathrm{B}_i$ if and only if $\sigma(b_{i,j_2}) = \mathrm{B}_i$. We show below that for every complete set $F \subseteq E_0$, the optimal strategy of player 0 in G_F is well-behaved.

The essential behavior of a well-behaved policy σ is characterized by two boolean vectors $\alpha(\sigma) = (\alpha_1, \ldots, \alpha_n)$ and $\beta(\sigma) = (\beta_1, \ldots, \beta_n)$ that are defined as follows:

$$\alpha_i(\sigma) = \begin{cases} 1 & \text{if } \forall j \in [\ell]\ \forall k \in [r]: \\ & \quad (a_{i,j,k}, \mathrm{A}_{i,j}) \in F \Rightarrow \sigma(a_{i,j,k}) = \mathrm{A}_{i,j} \\ 0 & \text{if } \forall j \in [\ell]\ \forall k \in [r]: \sigma(a_{i,j,k}) \neq \mathrm{A}_{i,j} \end{cases}$$

$$\beta_i(\sigma) = \begin{cases} 1 & \text{if } \forall j \in [\ell r]: \\ & \quad (b_{i,j}, \mathrm{B}_i) \in F \Rightarrow \sigma(b_{i,j}) = \mathrm{B}_i \\ 0 & \text{if } \forall j \in [\ell r]: \sigma(b_{i,j}) \neq \mathrm{B}_i \end{cases}$$

Similarly, given two boolean vectors $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ we let $\sigma = \sigma(\alpha, \beta)$ be a well-behaved strategy such that $\alpha(\sigma) = \alpha$ and $\beta(\sigma) = \beta$. Note that $\sigma(\alpha, \beta)$ is not uniquely determined, as when $\alpha_i = 0$ or $\beta_i = 0$ we do not specify which copy of a multi-edge is chosen. This choice, however, is irrelevant.

The i'th bit of the randomized bit-counter is interpreted as being *set*, for some complete set F and a well-behaved strategy σ, if $a_i(F) = b_i(F) = 1$ and $\alpha_i(\sigma) = \beta_i(\sigma) = 1$.

Let σ^*_F and τ^*_F be optimal strategies for player 0 and 1, respectively, in the subgame $G_F = (V_0, V_1, F \cup E_1, \Omega)$ defined by edges of F. We show below that

σ_F^* is always well-behaved, and we let $\alpha^*(F) = \alpha(\sigma_F^*)$ and $\beta^*(F) = \beta(\sigma_F^*)$. The following lemma then describes the key parts of optimal strategies in the construction.

Lemma 4.55. *Let $F \subseteq E_0$ be complete. Then σ_F^* is well-behaved and $\beta_i^*(F) = 1$ if and only if $i \geq reset(F)$, and $\alpha_i^*(F) = 1$ if and only if $b_i(F) = 1$ and $i \geq reset(F)$.*

Lower Bound Proof

In this paragraph we consider the expected number of iterations performed by the modified RANDOM-FACET algorithm when applied to our family of lower bound games. We show that for appropriate parameters ℓ and r the number of iterations is, with high probability, at least as large as the expected number of steps performed by the corresponding modified randomized bit-counter.

For brevity, we say that the RANDOM-FACET algorithm takes a set $F \subseteq E_0$ as argument rather than the game G_F. Let $F \subseteq E_0$, σ and `ind` be arguments to the RANDOM-FACET algorithm, and let $e = \operatorname{argmin}_{e' \in F \setminus \sigma} \texttt{ind}(e')$. We distinguish between three types of iterations; *count-iterations*, *reset-iterations* and *irrelevant iterations*. We say that an iteration is a count-iteration if it satisfies the following:

$$reset(F) = 0 \quad \text{and}$$
$$\exists i \in [n] : b_i(F) = 1 \wedge b_i(F \setminus \{e\}) = 0$$

Similarly, we say that an iteration is a reset-iteration if it satisfies:

$$reset(F) = 0 \quad \text{and} \quad reset(F \setminus \{e\}) > 0$$

An iteration that is neither a count-iteration nor a reset-iteration is said to be irrelevant.

In order to correctly simulate a modified randomized bit-counter it must be the case that between any two count-iterations there is a reset-iteration. Furthermore, F must be complete in every count-iteration, as well as in every reset-iteration. To handle these requirements we introduce the notion of a *good* index function. Recall that M is the set of multi edges without multiplicities. We write e_t to refer to the t'th copy of some edge $e \in M$. We say that an index function `ind` is good if it satisfies the following two requirements:

1. $\forall i \in [n]\ \exists j \in [\ell]\ \exists t \in [\ell r]\ \forall k \in [r] :\ \texttt{ind}(b_{i,t}, \mathrm{B}_i) < \texttt{ind}(a_{i,j,k}, \mathrm{A}_{i,j})$
2. $\forall e \in M\ \forall i \in [n]\ \forall j \in [\ell]\ \exists k \in [r]\ \exists t \in [r] :\ \texttt{ind}(a_{i,j,k}, \mathrm{A}_{i,j}) < \texttt{ind}(e_t)$

The first requirement says that for every i, the first edge, according to ind, going to B_i is before the first edge going to $\mathrm{A}_{i,j}$, for some j. The second requirement says that for all i and j, the first edge going to $\mathrm{A}_{i,j}$ is before some copy of every edge from M. Note that when both requirements are combined we also get that for all i, the first edge going to B_i is before some copy of every edge from M.

For $F \subseteq E_0$ and a well-behaved strategy $\sigma \subseteq F$, define for $1 \leq i \leq n$ where $b_i(F) = 1$ and $a_i(F) = 1$:

$$\mathfrak{b}_{F,\sigma}(i) = \begin{cases} 1 & \text{if } \alpha_i(\sigma) = \beta_i(\sigma) = 1, \\ 0 & \text{if } \alpha_i(\sigma) = \beta_i(\sigma) = 0, \\ \bot & \text{otherwise.} \end{cases}$$

The state $\mathfrak{b}_{F,\sigma}(i) = \bot$ is an intermediate state in which the bit is switching from set to unset, or vice versa.

We say that the i'th bit is *disabled* if $b_i(F) = 0$. Furthermore, we say that the i'th bit is *inactive* if $\mathfrak{b}_{F,\sigma}(i) = 1$, and for all $i < j \leq n$, the j'th bit is either disabled or inactive. We also say that the i'th bit is *resetting* if $b_i(F) = 1$, $a_i(F) = 1$, $\beta_i(\sigma) = 1$ but $\alpha_i(\sigma) = 0$. We define the set of active bits $N_{F,\sigma} \subseteq [n]$ such that $i \in N_{F,\sigma}$ if and only if the i'th bit is not disabled, inactive or resetting. Finally, we define the permutation $\phi_{\texttt{ind}}$ of $[n]$ such that for all $i, j \in [n]$:

$$\phi_{\texttt{ind}}(i) < \phi_{\texttt{ind}}(j) \quad \Longleftrightarrow \quad \exists k \in [\ell r]\ \forall t \in [\ell r] : \texttt{ind}(b_{i,k}, \mathrm{B}_i) < \texttt{ind}(b_{j,t}, \mathrm{B}_j)$$

Note that these concepts correspond exactly to the concepts utilized in a randomized bit-counter.

Let $f_{\texttt{ind}}(F, \sigma)$ be the number of iterations (recursive calls) performed by the call RANDOM-FACET$^*(G_F, \sigma, \texttt{ind})$. We denote the expected value of $f_{\texttt{ind}}(F, \sigma)$, when ind is a random good index function picked from the set of permutations of E_0, by $\mathbb{E}^*_{G_{n,\ell,r}}(F, \sigma | \texttt{ind is good})$.

Lemma 4.56. *Let $G_{n,\ell,r}$ be a lower bound game with initial strategy σ for player 0, then*

$$\mathbb{E}^*_{G_{n,\ell,r}}(E_0, \sigma|\texttt{ind} \textit{ is good}) \geq g(n).$$

Lemma 4.57. *Let $G_{n,\ell,r}$ be a lower bound game, and let* `ind` *be chosen uniformly at random from the set of permutations of E_0. Then* `ind` *is good with probability $p_{n,\ell,r}$, where:*

$$p_{n,\ell,r} \geq 1 - n\frac{(\ell!)^2}{(2\ell)!} - n\ell(n(2\ell r + \ell) - \ell)\frac{(r!)^2}{(2r)!}$$

Results

We conclude that policy iteration with the RANDOM-FACET rule requires an exponential number of iterations on the parity games of this chapter.

Theorem 4.58. *The worst-case expected running time of the* RANDOM-FACET *algorithm for n-state parity games is at least $2^{\Omega(\sqrt{n}/\log n)}$.*

Proof. Let $G_{n,\ell,r}$ be a lower bound game, and let σ be the initial strategy. Note that unlike the statement of the lemma n refers to the number of bits in the corresponding randomized bit-counter. From Lemma 4.52, Lemma 4.56 and Lemma 4.53 we have:

$$\begin{aligned}
\mathbb{E}_{G_{n,\ell,r}}(E_0, \sigma) &= \mathbb{E}^*_{G_{n,\ell,r}}(E_0, \sigma) \\
&\geq p_{n,\ell,r} \cdot \mathbb{E}^*_{G_{n,\ell,r}}(E_0, \sigma|\texttt{ind} \text{ is good}) \\
&\geq p_{n,\ell,r} \cdot g(n) \\
&= p_{n,\ell,r} \cdot 2^{\Omega(\sqrt{n})}
\end{aligned}$$

All that remains is, thus, to pick the parameters ℓ and r such that $p_{n,\ell,r}$ is constant. In the following we show that for $\ell = r = 3\log n$ and n sufficiently large, we get $p_{n,\ell,r} \geq \frac{1}{2}$.

It is easy to prove, by induction, that

$$\frac{(k!)^2}{(2k)!} \leq \frac{1}{2^k}.$$

For $\ell = r = 3 \log n$ we then get from Lemma 4.57 that:

$$\begin{aligned} p_{n,\ell,r} &\geq 1 - n\frac{(\ell!)^2}{(2\ell)!} - n\ell(n(2\ell r + \ell) - \ell)\frac{(r!)^2}{(2r)!} \\ &\geq 1 - n\frac{1}{2^\ell} - n\ell(n(2\ell r + \ell) - \ell)\frac{1}{2^r} \\ &= 1 - \frac{1}{n^2} - \frac{\ell(n(2\ell r + \ell) - \ell)}{n^2} \\ &\geq 1 - \frac{1}{n^2} - \frac{81 \log n}{n} \geq \frac{1}{2} \end{aligned}$$

for n sufficiently large.

Since, for $\ell = r = 3 \log n$ and n sufficiently large, the number of vertices of $G_{n,\ell,r}$ is $|V| = n(2\ell r + \ell + 3) + 1 \leq 27n \log^2 n$, the number of bits expressed in terms of the number of vertices is $n = \Omega(|V|/\log^2 |V|)$, and we get:

$$\mathbb{E}_{G_{n,\ell,r}}(E_0, \sigma) = 2^{\Omega(\sqrt{|V|}/\log |V|)}.$$

This concludes the proof.

□

It follows directly by Theorem 4.19 that we have a subexponential lower bound for payoff games.

Corollary 4.59. *Payoff game strategy iteration with* RANDOM-FACET*-rule requires expected subexponential time.*

Markov Decision Processes

We show in this paragraph how the presented parity games can be turned into Markov decision processes to obtain a lower bound in their domain as well. For concreteness, we consider the limiting average criterion here. We show that the RANDOM-FACET algorithm may also require subexponentially many steps to solve MDPs. Due to the connection between MDPs and LPs the same bound then follows for LPs.

The main observation required for the conversion is that, in the parity games of this chapter, the role of the second player, referred to as player 1, is very limited. A

simplified transformation of a vertex A controlled by player 1 is shown in Figure 4.20. Suppose player 1 does not move left unless player 0 moves from both b and b' to A. This behavior of player 1 can be simulated by a randomization vertex that moves left with very low, but positive probability.

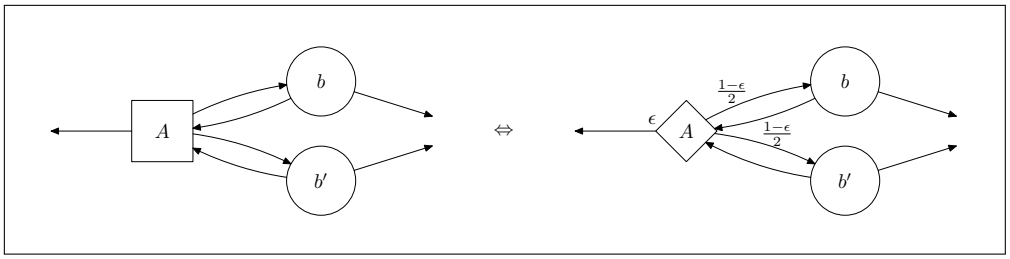


Figure 4.20: Conversion of a vertex controlled by player 1 to a randomization vertex

For integers $n, g, h \geq 1$, we define a family of *lower bound MDPs* with underlying graphs $G_{n,g,h} = (V_0, V_R, E, r, p)$ that the RANDOM-FACET algorithm requires many iterations to solve. n denotes the number of bits in the simulated randomized bit-counter, and g and h are parameters later to be specified in the analysis. We again use multi edges for convenience.

A graphical description of $G_{n,g,h}$ is given in Figure 4.21. Round vertices are controlled by player 0 and at diamond-shaped vertices the choice is made at random according to the probabilities on the outgoing edges. All rewards are described in terms of priorities, and only vertices x_i, y_i and d_i have priorities. Thus, most edges have reward zero.

Formally, $G_{n,g,h}$ is defined as follows.

$$\begin{aligned} V_0 \;&:= \{a_{i,j,k} \mid i \in [n], j \in [g], k \in [h]\} \;\cup\; \{b_{i,j} \mid i \in [n], j \in [gh]\} \cup \\ &\qquad \{d_i, x_i, y_i \mid i \in [n]\} \;\cup\; \{u_i, w_i \mid i \in [n+1]\} \;\cup\; \{t\} \\ V_R \;&:= \{\mathrm{A}_{i,j} \mid i \in [n], j \in [g]\} \;\cup\; \{\mathrm{B}_i \mid i \in [n]\} \end{aligned}$$

With $G_{n,g,h}$, we associate a large number $N \in \mathbb{N}$ and a small number $0 < \varepsilon$. We require N to be at least as large as the number of nodes with priorities, i.e. $N \geq 3n + 1$ and ε^{-1} to be significantly larger than the largest occurring priority

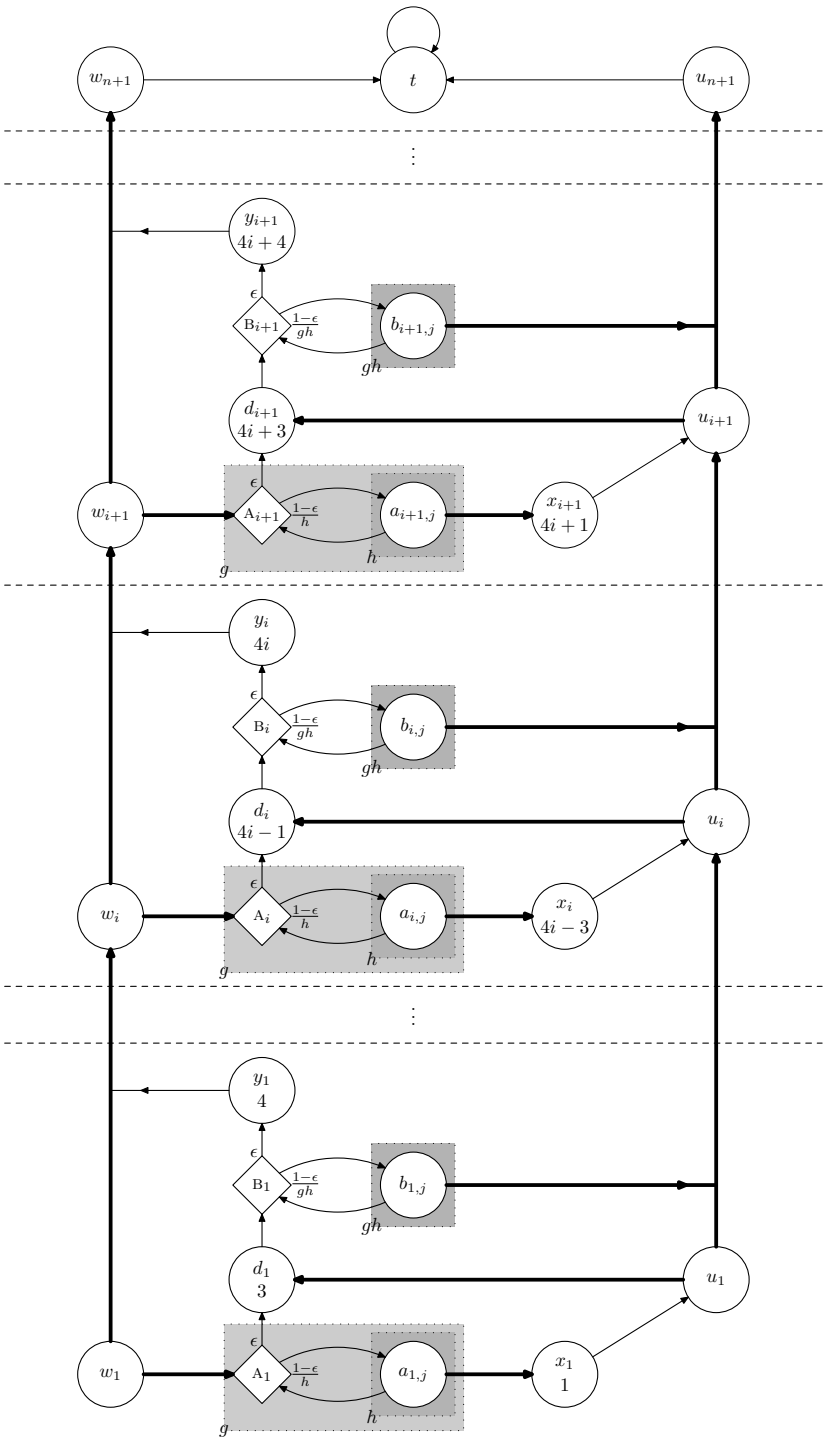


Figure 4.21: Lower bound MDP for the RANDOM-FACET algorithm

induced reward, i.e. $\varepsilon \leq N^{-(4n+8)}$. Node v having priority $\Omega(v)$ means that the cost associated with every outgoing edge of v is $\langle v \rangle = (-N)^{\Omega(v)}$.

Table 4.15 defines the edge set E, the priority assignment function Ω, multiplicities of edges, and the probability assignment function $p : E_R \to [0, 1]$. $b_{i,*}$ is a shorthand for the set $\{b_{i,j} \mid j \in [gh]\}$.

Node V_0	Successors in E_0	Priority Ω	Multiplicity
$a_{i,j,k}$	$\mathrm{A}_{i,j}$	-	1
	x_i		h
$b_{i,j}$	B_i	-	1
	u_{i+1}		h
d_i	B_i	$4i-1$	1
u_i	d_i	-	h
	u_{i+1}		h
u_{n+1}	t	-	1
w_i	$\mathrm{A}_{i,*}$	-	h
	w_{i+1}		h
w_{n+1}	t	-	1
x_i	u_i	$4i-3$	1
y_i	w_{i+1}	$4i$	1
t	t	-	1
Node V_R	**Successors in E_R**	**Priority Ω**	**Probability p**
$\mathrm{A}_{i,j}$	d_i	-	ε
	$a_{i,j,*}$		$\frac{1-\varepsilon}{h}$
B_i	y_i	-	ε
	$b_{i,*}$		$\frac{1-\varepsilon}{gh}$

Table 4.15: Priorities, edges, multiplicities, and transition probabilities of $G_{n,g,h}$

Note that $G_{n,g,h}$ is a unichain. More specifically, we have the following lemma.

Lemma 4.60. *For every strategy σ, the MDP with underlying graph $G_{n,g,h}$ ends in the sink t with probability* 1.

Again, we have the following lemma describing optimal strategies corresponding to complete edge sets.

Lemma 4.61. *Let $F \subseteq E_0$ be complete. Then σ^*_F is well-behaved and $\beta^*_i(F) = 1$ if and only if $i \geq reset(F)$, and $\alpha^*_i(F) = 1$ if and only if $b_i(F) = 1$ and $i \geq reset(F)$.*

The result relies on Lemma 4.57, hence we transfer our main theorem to the MDP and LP world.

Theorem 4.62. *The worst-case expected running time of the* RANDOM-FACET *algorithm for n-state MDPs is at least $2^{\Omega(\sqrt{n}/\log n)}$, even when at most $\mathcal{O}(\log n)$ actions are associated with each state.*

Corollary 4.63. *The worst-case expected running time of* RANDOM-FACET *for LPs of dimension n with $\mathcal{O}(n \log n)$ constraints is at least $2^{\Omega(\sqrt{n}/\log n)}$.*

4.7.2 Random Edge Rule

Perhaps the most natural randomized improvement rule is RANDOM-EDGE, which among all improving switches chooses one uniformly at random. The upper bounds currently known for RANDOM-EDGE are still exponential (see Gärtner and Kaibel [GK07]). For additional results regarding RANDOM-EDGE, see [BDF+95, GHZ98, GTW+03, BP07].

RANDOM-EDGE:	Apply a single improving switch arbitrarily at random.

We show that RANDOM-EDGE might lead to an expected subexponential number of iterations on actual linear programs. More specifically, we construct concrete linear programs on which the expected number of iterations performed by RANDOM-EDGE is $2^{\Omega(n^{1/4})}$, where n is the number of variables.

The lower bound for linear programming again has been obtained by constructing explicit parity games and related MDPs on which we have the same expected number of iterations when solved by policy iteration. For the presentation, we start with Markov decision processes here, and show later, how they can be translated to parity games. For concreteness, we consider the limiting average criterion.

High-level Description

We start with a high-level description of the MDPs on which RANDOM-EDGE performs an expected subexponential number of iterations. The exact details are fairly intricate. In high level terms, our MDPs, and the linear programs corresponding to them, are constructions of 'fault tolerant' *randomized counters*. The challenge in designing such counters is making sure that they count 'correctly' under most sequences of random choices made by the RANDOM-EDGE pivoting rule.

A schematic description of the lower bound MDPs is given in Figure 4.22. The shaded octagons enclosing some of the vertices stand for *cycle gadgets* shown in Figure 4.23. It is useful to assume, at first, that these octagons stand for standard vertices (when we adopt this point of view, we refer to $a_{i,j}$ simply as a_i, and similarly for $b_{i,j}$ and $c_{i,j}$). We shall explain later why they need to be replaced by the cycle gadgets.

The MDP of Figure 4.22 emulates an n-bit counter. It is composed of n identical levels, each corresponding to a single bit of the counter. The 1-st, i-th and $(i+1)$-th levels are shown explicitly in the figure. Levels are separated by dashed lines; n, ℓ_i, h, g, for $1 \leq i \leq n$, are integer parameters for the construction. The MDP includes two *sources* r and s, and one *sink* t. The i-th level contains 7 vertices of V_0, namely, $a_i, b_i, c_i, d_i, u_i, w_i, x_i$, and two randomization vertices A_i and B_i (when the cycle gadgets are used, a_i, b_i and c_i are replaced by collections of vertices $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$). We refer to the vertices a_i, b_i and c_i (and $a_{i,j}, b_{i,j}$ and $c_{i,j}$) as *cycle vertices*, and refer to the corresponding cycles as the A_i-, B_i- and C_i-cycles, respectively. The vertices u_i form the *right lane*, while the vertices w_i form the *left lane*. We use U and W to refer collectively to the vertices of the right and left lanes, respectively.

In each of $a_{i,j}, b_{i,j}, c_{i,j}, u_i, w_i$, player 0, the controller, has two outgoing edges to choose from. Vertices d_i, x_i and y_i have only one outgoing edge, so no decision is made at them (the role of d_i, x_i and y_i will become clear later).

Most edges in Figure 4.22 have an immediate reward of 0 associated with them (such 0 rewards are not shown explicitly in the figure). The only edges that have non-zero rewards associated with them are the edges $a_{i,j} \to x_i$, $b_{i,j} \to s$ and $c_{i,j} \to r$ that have reward $j\epsilon$, where ϵ is a sufficiently small number to be chosen later.

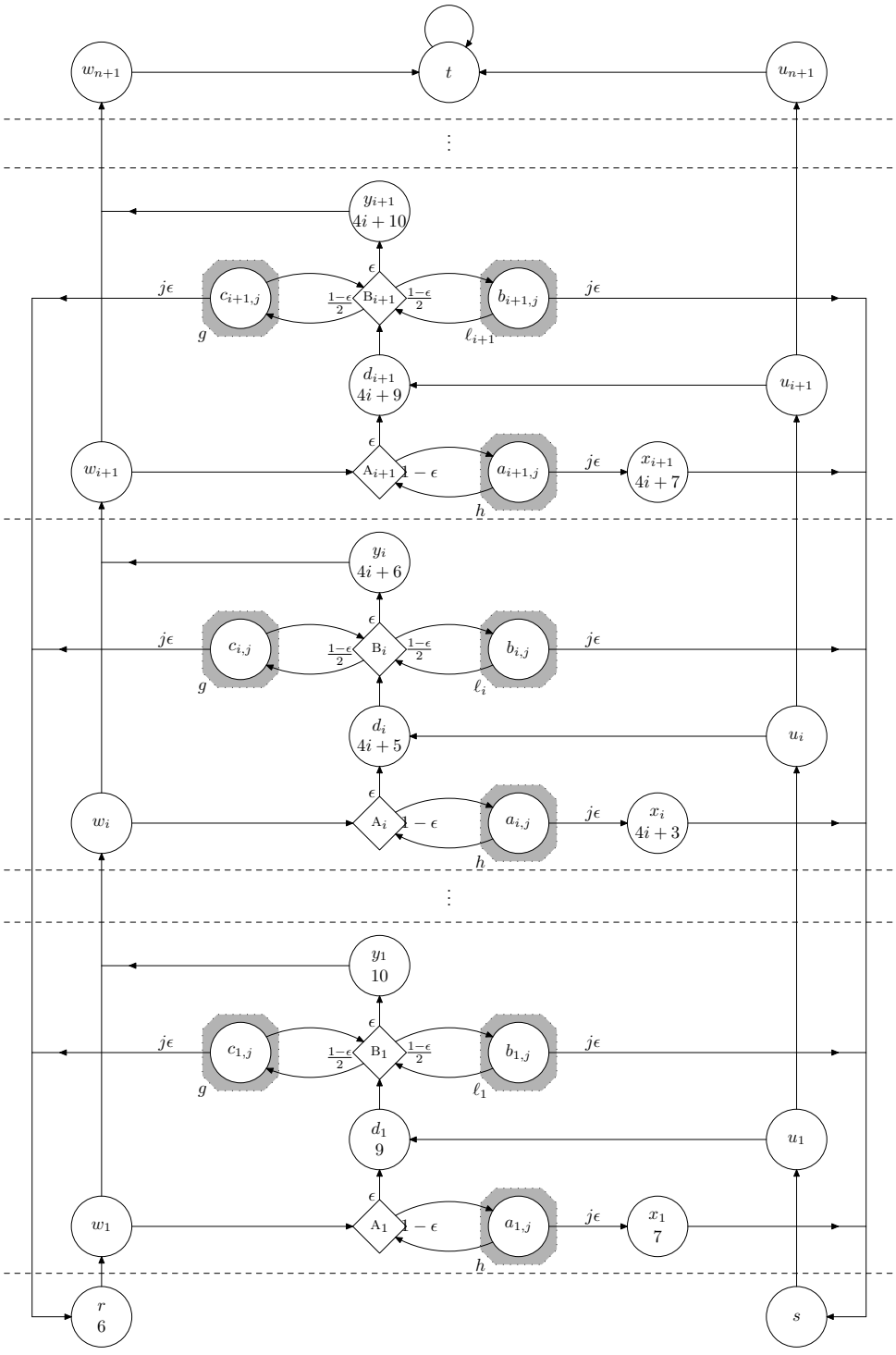


Figure 4.22: Random Edge MDP Construction. The interpretation of the shaded octagons is shown in Figure 4.23

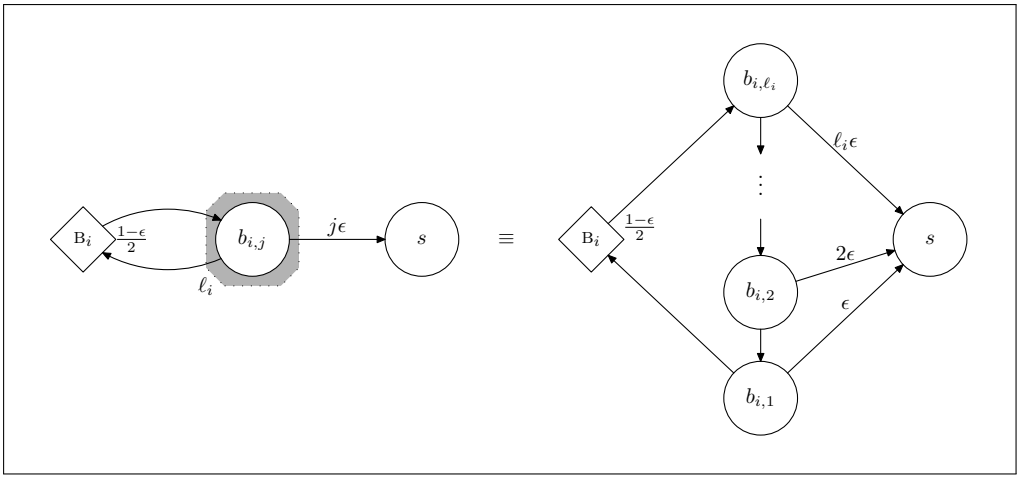


Figure 4.23: A cycle gadget used by the lower bound MDPs for RANDOM-EDGE

In addition to the rewards assigned to some of the edges, some of the vertices are assigned integer *priorities*. If a vertex v has priority $\Omega(v)$ assigned to it, then a reward of $\langle v \rangle = (-N)^{\Omega(v)}$ is *added* to all edges emanating from v, where N is a sufficiently large integer. We use $N = 3n+1$ and $\varepsilon = N^{-(4n+8)}$. Priorities, if present, are listed next to the vertex name (in particular, $\Omega(d_i) = 4i + 5$, $\Omega(x_i) = 4i + 3$, $\Omega(y_i) = 4i + 6$, for $1 \leq i \leq n$, and $\Omega(r) = 6$; all other vertices have no priorities assigned to them). Rewards and priorities are chosen such that priorities are always of higher importance. Note that it is desirable to move through vertices of even priority and to avoid vertices of odd priority, and that vertices of higher numerical priority dominate vertices of lower priority (the idea of using priorities is inspired, of course, by the reduction from parity games to mean payoff games).

Each level has only two randomization vertices. From A_i, the edge $\mathrm{A}_i \to a_i$ (or more specifically $\mathrm{A}_i \to a_{i,\ell_i}$), is chosen with probability $1 - \epsilon$, while the edge $\mathrm{A}_i \to d_i$ is chosen with probability ϵ. Thus, if the A_i-cycle is *closed*, the MDP is guaranteed to eventually move to d_i. From B_i, each of the two edges $\mathrm{B}_i \to b_i$ and $\mathrm{B}_i \to c_i$ are chosen with probability $\frac{1-\epsilon}{2}$, while the edge $\mathrm{B}_i \to y_i$ is chosen with probability ϵ. Again, if both B_i- and C_i-cycles are closed, an eventual transition to y_i is made.

To each state $\mathfrak{b} \in \mathcal{B}_n$ of an n-bit binary counter, we define a corresponding policy $\sigma_{\mathfrak{b}}$ of the MDP. If $\mathfrak{b}_i = 1$, then all three cycles in the i-th level are *closed*, and u_i and w_i point into the level, while if $\mathfrak{b}_i = 0$, then the three cycles are *open*, and u_i and

w_i point to the next level. Our ultimate goal is to show that a run of RANDOM-EDGE, that starts with $\sigma_{0\ldots00}$, visits all 2^n policies $\sigma_{0\ldots00}, \sigma_{0\ldots01}, \sigma_{0\ldots10}, \ldots, \sigma_{1\ldots11}$, with high probability.

Our proof is conceptually divided into two parts. First we investigate the improving switches that can be performed from *well-behaved* policies of the MDP. This allows us to prove that there *exists* a sequence of improving switches that does indeed generate the sequence $\sigma_{0\ldots00}, \sigma_{0\ldots01}, \sigma_{0\ldots10}, \ldots, \sigma_{1\ldots11}$. This is true even if the cycle gadgets of Figure 4.23 are *not* used. A transition from $\sigma_{\mathfrak{b}}$ to $\sigma_{\mathfrak{b}+1}$ involves many improving switches. We partition the path leading from $\sigma_{\mathfrak{b}}$ to $\sigma_{\mathfrak{b}+1}$ into seven sub-paths which we refer to as *phases*. In the following we first give an informal description of the phases, and then describe how the cycle gadgets of Figure 4.23 increase the transition probabilities. Note that some of the mentioned improving switches exist during several phases. We present here the sequences of updates enforced by the gadgets with high probability. A more formal description of the phases is given later.

Let $\mathfrak{b}$ be a state of the bit-counter, and recall that the *least significant unset bit* is denoted by $\mu_1(\mathfrak{b}) := \min(\{i \leq n \mid \mathfrak{b}_i = 0\} \cup \{n{+}1\})$. The phases are as follows:

1. At the beginning of the first phase the policy corresponds to $\sigma_{\mathfrak{b}}$, except that some of the $\textsc{c}_i$-cycles of unset bits are closed. It is improving, however, to open these cycles, since opening the cycle leads through r, which has priority 6, to the lowest set bit. If a $\textsc{c}_i$-cycle is closed it instead moves via the $\textsc{b}_i$-cycle to s to the lowest set bit. Hence, all $\textsc{c}_i$-cycles open during this phase.

2. The initial strategy for the second phase is exactly $\sigma_{\mathfrak{b}}$. It is desirable for all open $\textsc{b}_i$-cycles to close, because this implies moving via the $\textsc{c}_i$-cycles to r. The gadgets indicated by the octagons ensure that only the B-cycle of the least 0-bit $\mu_1(\mathfrak{b})$ gets closed.

3. Since the $\textsc{b}_{\mu_1(\mathfrak{b})}$-cycle is now closed, the $\textsc{c}_{\mu_1(\mathfrak{b})}$-cycle at level $\mu_1(\mathfrak{b})$ also closes as this gives access to $y_{\mu_1(\mathfrak{b})}$, which has a large even priority.

4. Since the $\textsc{a}_{\mu_1(\mathfrak{b})}$-cycle has not yet closed there is (essentially) no access from $\textsc{a}_{\mu_1(\mathfrak{b})}$ to $d_{\mu_1(\mathfrak{b})}$. This implies that lower set bits are unable to reach the domi-

nating even priority at $y_{\mu_1(\mathfrak{b})}$. In particular, the u_i vertices for $i \leq \mu_1(\mathfrak{b})$ are updated to provide access from the source s to $y_{\mu_1(\mathfrak{b})}$.

5. Next, all A_i- and B_i-cycles at levels $i < \mu_1(\mathfrak{b})$ open to reach $y_{\mu_1(\mathfrak{b})}$, and in particular to reach $y_{\mu_1(\mathfrak{b})}$ through a vertex x_i with as low a priority as possible. Note that it is also desirable for B_i-cycles of unset bits at higher levels to open (although they are currently already open). This property is critical for resetting the gadgets.

6. The $\text{A}_{\mu_1(\mathfrak{b})}$-cycle now closes since it is then able to avoid the odd priority at $x_{\mu_1(\mathfrak{b})}$.

7. Finally, since there is now access from $\text{A}_{\mu_1(\mathfrak{b})}$ to $d_{\mu_1(\mathfrak{b})}$ the w_i vertices for $i \leq \mu_1(\mathfrak{b})$ are updated accordingly, and after the phase is over it is again desirable to close lower B_i-cycles. Note also that lower C_i-cycles remained open.

Proving that a long sequence of switches exists is of course not enough. We need to prove that such a long sequence occurs with a sufficiently high probability. To do that we introduce the cycle gadgets of Figure 4.23.

The idea is to make the A_i-, B_i- and C_i-cycles longer such that they are difficult to close. The purpose of the small rewards on the edges is to make sure that only one edge at a time is an improving switch when closing a cycle. Hence, closing a cycle requires a very specific sequence of improving switches. Furthermore, we can use Chernoff bounds to bound the probability of a longer cycle closing before a shorter cycle. By increasing the length of the B_i-cycles for increasing i, we make sure that in phase 2 the B_i-cycle of the lowest unset bit closes first.

On the other hand, when opening a cycle all outgoing edges are simultaneously improving switches. This allows lower bits to reset very fast during phase 5 before the A_i cycle closes in phase 6.

Full Construction

In this paragraph, we formally describe the full construction of our MDPs. For a tuple $\zeta = (n, (\ell_i)_{0 \leq i \leq n}, h, g)$, with $n, \ell_i, h, g > 0$, define an underlying graph

$G_\zeta = (V_0, V_R, E, r, p)$ of an MDP as shown schematically in Figure 4.22. More formally:

$$\begin{aligned} V_0 &:= \{a_{i,j} \mid i \in [n], j \in [h]\} \cup \{b_{i,j} \mid i \in [n], j \in [\ell_i]\} \cup \\ &\quad \{c_{i,j} \mid i \in [n], j \in [g]\} \cup \{d_i, y_i, x_i \mid i \in [n]\} \cup \\ &\quad \{w_i, u_i \mid i \in [n+1]\} \cup \{t, r, s\} \\ V_R &:= \{\mathrm{A}_i, \mathrm{B}_i \mid i \in [n]\} \end{aligned}$$

With G_ζ, we associate a large number $N \in \mathbb{N}$ and a small number $0 < \varepsilon$. We require N to be at least as large as the number of nodes with priorities, i.e. $N \geq 3n+1$ and ε^{-1} to be significantly larger than the largest occurring priority induced reward, i.e. $\varepsilon \leq N^{-(4n+8)}$. Remember that node v having priority $\Omega(v)$ means that the cost associated with every outgoing edge of v is $\langle v \rangle = (-N)^{\Omega(v)}$.

Table 4.16 defines the edge sets, the probabilities, the priorities and the immediate rewards of G_ζ.

Lemma 4.64. *For every strategy σ, the MDP described by G_ζ ends in the sink t with probability* 1.

It is not too hard to see that the absolute potentials of all nodes corresponding to strategies belonging to the phases are bounded by ε^{-1}. More formally we have:

Lemma 4.65. *Let $P = \{r, y_i, x_i, d_i \mid i \leq n\}$ be the set of nodes with priorities. For a subset $S \subseteq P$, let $\sum(S) = \sum_{v=S} \langle v \rangle$. For non-empty subsets $S \subseteq P$, let $v_S \in S$ be the node with the largest priority in S.*

1. *$|\sum(S)| < N^{4n+8}$ and $\varepsilon \cdot |\sum(S)| < 1$ for every subset $S \subseteq P$, and*
2. *$|v_S| < |v_{S'}|$ implies $|\sum(S)| < |\sum(S')|$ for non-empty subsets $S, S' \subseteq P$.*

Lemma 4.66. *Let σ be a strategy belonging to one of the phases specified in Table 4.17. Then $|\mathrm{POT}_\sigma(v)| < N^{4n+8}$ and $\varepsilon \cdot |\mathrm{POT}_\sigma(v)| < 1$ for every node v.*

Node	Successors	Probability
A_i	d_i	ε
	$a_{i,h}$	$1-\varepsilon$
B_i	y_i	ε
	b_{i,ℓ_i}	$\frac{1}{2}\cdot(1-\varepsilon)$
	$c_{i,g}$	$\frac{1}{2}\cdot(1-\varepsilon)$

Node	Successors	Priority
r	w_1	6
x_i	s	$4i+3$
d_i	B_i	$4i+5$
y_i	w_{i+1}	$4i+6$
w_{n+1}	t	-
u_{n+1}	t	-
w_i	w_{i+1}, A_i	-
u_i	u_{i+1}, d_i	-

Node	Successors	Cost
$a_{i,1}$	A_i	0
	x_i	1ε
$a_{i,j+1}$	$a_{i,j}$	0
	x_i	$(j{+}1)\varepsilon$
$b_{i,1}$	B_i	0
	s	1ε
$b_{i,j+1}$	$b_{i,j}$	0
	s	$(j{+}1)\varepsilon$
$c_{i,1}$	B_i	0
	r	1ε
$c_{i,j+1}$	$c_{i,j}$	0
	r	$(j{+}1)\varepsilon$
t	t	-
s	u_1	-

Table 4.16: Random Edge MDP Construction

Next, we will specify and prove an auxiliary lemma that describes the exact behavior of all the cycles appearing in the construction.

The idea behind the cycles is to have a gate that controls the access of other nodes of the graph to the *escape node* of the cycle (d_i resp. y_i) to which the randomized node moves with very low probability.

First, assume that a cycle (or both cycles if there are two) is closed. Although the randomized node circles through the cycles with very high probability (without accumulating any costs), it eventually moves out to the escape node, resulting in the same potential as the potential of the escape node itself.

Second, assume that a cycle is open, i.e. one of the V_0-controlled nodes of the cycle decides to move out of the cycle to some *reset node*. Now, the randomized node selects to move into the cycle with very large probability and therefore leaves the cycle to the reset node with high probability as well. The resulting potential of the randomized node essentially matches the potential of the reset node.

The critical property of cycles is that closing a cycle is a very slow process while opening proceeds at a rapid pace. Closing a cycle takes place when the potential

of the escape node is better than the potential of the reset node. However, in every policy iteration step, there is only one improving edge associated with the cycle, namely the first edge pointing into the cycle which is not included in the current policy. Therefore, closing a cycle can only be performed one edge at a time. Opening a cycle happens in the reverse situation in which the potential of the reset node is better than the potential of the escape node. Here, every node that is currently moving into the cycle has an improving edge to move out of the cycle.

The following lemma formalizes the intuition of the behavior of the cycles. If the escape node has better valuation than the reset nodes, it should be profitable to close the cycle, and otherwise, it should be profitable to open the cycle again. This idea generalizes to the setting in which two cycles are attached to the randomization node. Since both reset nodes necessarily have different potentials, it is always the case that it is profitable to close one of the two cycles (the one with the worse reset node) and while it is closing, the other one is opening. If one of the two cycles is completely closed, the problem is essentially reduced to the case in which only one cycle is attached to the randomization node.

Lemma 4.67. *Let σ be a strategy belonging to one of the phases specified in Table 4.17.*

1. $\text{POT}_\sigma(d_i) < \text{POT}_\sigma(x_i) \Rightarrow A_i$ *opening,*
2. $\text{POT}_\sigma(d_i) > \text{POT}_\sigma(x_i)$, A_i *consecutive, not closed* $\Rightarrow A_i$ *closing,*
3. $\text{POT}_\sigma(s) < \text{POT}_\sigma(r)$, B_i *consecutive, not closed* $\Rightarrow B_i$ *closing,* C_i *opening,*
4. $\text{POT}_\sigma(s) < \text{POT}_\sigma(r) < \text{POT}_\sigma(y_i)$, B_i *closed,* C_i *consecutive, not closed* $\Rightarrow C_i$ *closing, and*
5. $\text{POT}_\sigma(r) < \text{POT}_\sigma(y_i) < \text{POT}_\sigma(s)$, C_i *consecutive, not closed* $\Rightarrow C_i$ *closing,* B_i *opening.*

Counting Phases

In this paragraph, we formally describe the different *phases* that a strategy can be in, as well as the improving switches in each phase. The increment of the binary

counter by one is realized by transitioning through all the phases. We first introduce notation to succinctly describe strategies.

Note that all vertices of V_0 have at most binary out-degree. It will be convenient to describe the decision of a strategy in terms of $\{0,1\}$-values for all vertices of V_0 with binary out-degree. Let σ be a policy and $u \in \{u_i, w_i, a_{i,*}, b_{i,*}, c_{i,*} \mid i \in [n]\}$. We write:

$$\overline{\sigma}(u) = \begin{cases} 1 & \text{if } \sigma(u) = (u,v), \text{ such that } v \notin \{r,s\} \cup \{u_{i+1}, w_{i+1}, x_i \mid i \in [n]\} \\ 0 & \text{otherwise} \end{cases}$$

In other words, $\overline{\sigma}(u) = 1$ iff the node u moves *into* the corresponding level of the construction. We, furthermore, define the *total* number of edges of σ going into the respective cycles as:

$$\alpha_i(\sigma) = \sum_{j \in [h]} \overline{\sigma}(a_{i,j}) \qquad \beta_i(\sigma) = \sum_{j \in [\ell_i]} \overline{\sigma}(b_{i,j}) \qquad \gamma_i(\sigma) = \sum_{j \in [g]} \overline{\sigma}(c_{i,j})$$

We say that a cycle is:

1. *Closed*, if $\alpha_i(\sigma) = h$, $\beta_i(\sigma) = \ell_i$ or $\gamma_i(\sigma) = g$, respectively.
2. *Open*, if it is not closed.
3. *Completely open*, if $\alpha_i(\sigma) = 0$, $\beta_i(\sigma) = 0$ or $\gamma_i(\sigma) = 0$, respectively.
4. *Consecutive*, if the frontmost k vertices move into the cycle, for some k, and all remaining vertices move out of the cycle.

To describe the set of improving edges, we say that a cycle is:

1. *Opening*, if every unused edge moving out of the cycle is an improving switch.
2. *Closing*, if either the cycle is closed and there are no improving switches, or the cycle is consecutive and the only improving switch is $(a_{i,\alpha_i(\sigma)+1}, a_{i,\alpha_i(\sigma)})$, $(b_{i,\beta_i(\sigma)+1}, b_{i,\beta_i(\sigma)})$ or $(c_{i,\gamma_i(\sigma)+1}, c_{i,\gamma_i(\sigma)})$, respectively.

For every $i \in [n]$, we use a succinct notation tuple to provide all necessary information describing the i'th level: `b c a u w`, where `b` describes the B-cycle, `c`

describes the C-cycle, $\mathtt{a}$ describes the A-cycle, $\mathtt{u}$ describes the right lane, and $\mathtt{w}$ describes the left lane.

The first three components, describing the cycles, take one of the following values:

1	cycle is *closed* and *closing*
0	cycle is *completely open* and *opening*
$\uparrow$	cycle is *open*, *consecutive* and *closing*
$\downarrow$	cycle is *opening*

The last two components describe the setting and improving switches of w_i and u_i. For concreteness we give the definitions for w_i, the definitions for u_i are similar.

1	$\overline{\sigma}(w_i) = 1$ and switching is no improvement
0	$\overline{\sigma}(w_i) = 0$ and switching is no improvement
$\searrow$	$\overline{\sigma}(w_i) = 1$ and switching is an improvement
$\nearrow$	$\overline{\sigma}(w_i) = 0$ and switching is an improvement

We write $*$ if we neither care about the current setting nor about any improving switches.

To describe the progress of reassembling the right and left lanes in phases 4 and 7, respectively, we define the index of the lowest level with an incorrect setting as follows:

$$\delta(\sigma, k) = \max\{i{\leq}k \mid i{<}k \iff \overline{\sigma}(u_i){=}1\}$$

$$\eta(\sigma, k) = \max\{i{\leq}k \mid i{<}k \iff \overline{\sigma}(w_i){=}1\}$$

We are now ready to formulate the conditions for strategies that fulfill one of the seven phases along with the improving edges. See Table 4.17 for a complete description (with respect to a given strategy σ and global counter state $\mathfrak{b}$).

Finally, we prove that the improving switches are indeed exactly as specified. The simple but tedious proof uses Lemma 4.66 and Lemma 4.67 to compute the potentials of all important nodes in the game to determine whether a successor of V_0-controlled node is improving or not.

Lemma 4.68. *The improving switches from strategies that belong to the phases are exactly those specified in Table 4.17.*

Phase	$i > \mu_1(\mathfrak{b})$		$i = \mu_1(\mathfrak{b})$	$0 < i < \mu_1(\mathfrak{b})$
	$\mathfrak{b}_i = 1$	$\mathfrak{b}_i = 0$		
1	$1\,1\,1\ \ 1\,1$	$\uparrow\downarrow 0\ \ 0\,0$	$\uparrow\downarrow 0\ \ 0\,0$	$1\,1\,1\ \ 1\,1$
2	$1\,1\,1\ \ 1\,1$	$\uparrow 0\,0\ \ 0\,0$	$\uparrow 0\,0\ \ 0\,0$	$1\,1\,1\ \ 1\,1$
3	$1\,1\,1\ \ 1\,1$	$\uparrow 0\,0\ \ 0\,0$	$1\uparrow 0\ \ 0\,0$	$1\,1\,1\ \ 1\,1$
4	$1\,1\,1\ \ 1\,1$	$\uparrow 0\,0\ \ 0\,0$	$1\,1\uparrow \mathtt{u}_1 0$	$1\,1\,1\ \mathtt{u}_2 1$
5	$1\,1\,1\ \ 1\,1$	$\downarrow\uparrow 0\ \ 0\,0$	$1\,1\uparrow\ 1*$	$\downarrow 1\downarrow\ 0*$
6	$1\,1\,1\ \ 1\,1$	$0\uparrow 0\ \ 0\,0$	$1\,1\uparrow\ 1*$	$0\,1\,0\ \ 0*$
7	$1\,1\,1\ \ 1\,1$	$0\uparrow 0\ \ 0\,0$	$1\,1\,1\ \ 1\mathtt{w}_1$	$\mathfrak{b}\,1\,0\ \ 0\mathtt{w}_2$

Side Conditions

$$\mathtt{u}_1 = \begin{cases} 1 & \text{if } \mu_1(\mathfrak{b}) \neq \delta(\sigma, \mu_1(\mathfrak{b})) \\ \nearrow & \text{otherwise} \end{cases} \qquad \mathtt{w}_1 = \begin{cases} 1 & \text{if } \mu_1(\mathfrak{b}) \neq \eta(\sigma, \mu_1(\mathfrak{b})) \\ \nearrow & \text{otherwise} \end{cases}$$

$$\mathtt{b} = \begin{cases} \uparrow & \text{if } i \geq \eta(\sigma, \mu_1(\mathfrak{b})) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathtt{u}_2 = \begin{cases} \searrow & \text{if } i = \delta(\sigma, \mu_1(\mathfrak{b})) \\ 1 & \text{if } \delta(\sigma, \mu_1(\mathfrak{b})) > i \\ 0 & \text{otherwise} \end{cases} \qquad \mathtt{w}_2 = \begin{cases} \searrow & \text{if } i = \eta(\sigma, \mu_1(\mathfrak{b})) \\ 0 & \text{if } i > \eta(\sigma, \mu_1(\mathfrak{b})) \\ * & \text{otherwise} \end{cases}$$

Table 4.17: Strategies and improving switches of the seven phases

Transition Probabilities

Let $\Sigma_{\mathfrak{b},p}$ be the set of policies that belong to phase p, where $p \in [7]$, with respect to a given setting $\mathfrak{b}$ of the counter. The sets $\Sigma_{\mathfrak{b},p}$ are defined by Table 4.17, where the improving switches from each such policy are also specified. Our goal in this paragraph is to show that if RANDOM-EDGE is run on a policy from $\Sigma_{\mathfrak{b},p}$, then with an extremely high probability a policy from $\Sigma_{\mathfrak{b},p+1}$, or from $\Sigma_{\mathfrak{b}+1,1}$, if $p = 7$, is encountered after polynomially many steps. We show that the probability that this does not hold is $O(\mathrm{e}^{-n})$. The probability that one of the $7 \cdot 2^n$ phases fails is thus $O((2/\mathrm{e})^n)$, i.e., exponentially small.

The vertices of the MDP are partitioned into $3n$ cycles, which we refer to as the A_i-cycle, B_i-cycle and C_i-cycle, for $i \in [n]$, and the two lanes W and U. We use Z as a generic name for each one of these cycles or lanes. Table 4.17 specifies the behavior of each cycle or lane Z during a phase. A cycle Z is in one of the four states $1, 0, \uparrow, \downarrow$, as explained above. Recall that $\uparrow$ means that the cycle is closing, and that $\downarrow$ means that the cycle is opening. A lane Z is either fixed during a stage, or is being realigned.

A phase ends when a specified component Z completely opens, completely closes, or is completely realigned. By looking at Table 4.17 we see, for example, that phase 1 ends when all C_i-cycles, with $\mathfrak{b}_i = 0$, which are opening during the phase, open completely. Note that the B_i-cycles, with $\mathfrak{b}_i = 0$, are closing during the phase, and none of them is allowed to close completely before all the C_i-cycles open completely. Phase 1 *fails* only if one of the B_i-cycles closes completely before all required C_i-cycles open completely.

Similarly, in phase 2, all B_i-cycles with $\mathfrak{b}_i = 0$ are opening. The phase ends successfully if the first such cycle to close completely is the $\mathrm{B}_{\mu_1(\mathfrak{b})}$-cycle. The phase thus fails only if some B_i-cycle, with $i > \mu_1(\mathfrak{b})$ and $\mathfrak{b}_i = 0$ closes completely before the $\mathrm{B}_{\mu_1(\mathfrak{b})}$-cycle.

As a final example, note that phase 4 ends when the right lane U realigns, and that this should happen before the $\mathrm{A}_{\mu_1(\mathfrak{b})}$-cycle and the B_i-cycles, with $i > \mu_1(\mathfrak{b})$ and $\mathfrak{b}_i = 0$, close completely.

We can thus view each phase as being composed of several simultaneous *competitions* between various components, some of which are trying to open while others are trying to close. In each phase, we either like all cycles that are trying to open, to open completely before any other cycle closes completely, as it is the case in phase 1. In other cases, we would like some specified cycle, like the $\mathrm{B}_{\mu_1(\mathfrak{b})}$-cycle in phase 2, or the right lane U in phase 4, to completely close, or realign itself, before any other cycle closes completely.

The competitions carried out during each phase are shown in Table 4.18. We let $\Uparrow(\mathrm{Z}, \mathcal{Z})$ denote a competition in which we want Z to completely close before any other component $\mathrm{Z}' \in \mathcal{Z}$ closes completely. We let $\Downarrow(\mathcal{Z})$ denote the competition in

Trans.	Competition
$1 \to 2$	$\Downarrow(\{\mathrm{B}(i) \mid \mathfrak{b}_i{=}0\})$
$2 \to 3$	$\Uparrow(\mathrm{B}(\mu_1(\mathfrak{b})), \{\mathrm{B}(i) \mid i{>}\mu_1(\mathfrak{b}), \mathfrak{b}_i{=}0\})$
$3 \to 4$	$\Uparrow(\mathrm{C}(\mu_1(\mathfrak{b})), \{\mathrm{B}(i) \mid i{>}\mu_1(\mathfrak{b}), \mathfrak{b}_i{=}0\})$
$4 \to 5$	$\Uparrow(\mathrm{U}, \{\mathrm{A}(\mu_1(\mathfrak{b}))\} \cup \{\mathrm{B}(i) \mid i{>}\mu_1(\mathfrak{b}), \mathfrak{b}_i{=}0\})$
$5 \to 6$	$\Downarrow(\{\mathrm{A}(\mu_1(\mathfrak{b}))\} \cup \{\mathrm{C}(i) \mid i{>}\mu_1(\mathfrak{b}), \mathfrak{b}_i{=}0\})$
$6 \to 7$	$\Uparrow(\mathrm{A}(\mu_1(\mathfrak{b})), \{\mathrm{C}(i) \mid i{>}\mu_1(\mathfrak{b}), \mathfrak{b}_i{=}0\})$
$7 \to 1$	$\Uparrow(\mathrm{W}, \{\mathrm{C}(i) \mid i{>}\mu_1(\mathfrak{b}), \mathfrak{b}_i{=}0\} \cup \{\mathrm{B}(i) \mid i{<}\mu_1(\mathfrak{b})\})$

Trans.	Noise Bounds		
	ξ^B	ξ^C	ξ^A
$1 \to 2$	$\nu(n) + \rho$	0	0
$2 \to 3$	$\nu(n) + \rho + \nu(\ell_{\mu_1(\mathfrak{b})})$	0	0
$3 \to 4$	$\nu(n){+}\rho{+}\nu(\ell_{\mu_1(\mathfrak{b})}){+}\nu(g)$	0	0
$4 \to 5$	$2\nu(n){+}\rho{+}\nu(\ell_{\mu_1(\mathfrak{b})}){+}\nu(g)$	0	$\nu(n)$
$5 \to 6$	0	ρ	$\nu(n){+}\rho$
$6 \to 7$	0	$\rho + \nu(h)$	0
$7 \to 1$	$\nu(n)$	$\rho{+}\nu(h){+}\nu(n)$	0

Table 4.18: Noise bounds and phase transition competitions

which we want all cycles that are currently opening to open completely before any cycle $\mathrm{Z} \in \mathcal{Z}$ closes completely. (Note that in $\Downarrow(\mathcal{Z})$ we do not specify which cycles are opening.)

Competitions between closing cycles and opening cycles are heavily biased towards the opening cycles. This is because a closing cycle has only one improving edge associated with it, while an opening cycle has, in general, many improving switches associated with it. As an improving switch is chosen uniformly at random, an improving switch that belongs to the opening cycle is much more likely to be selected.

In competitions between closing cycles, shorter cycles, or more precisely cycles with less 'missing' edges, clearly have an advantage (both cycles close at the same 'speed'). To make it much more likely that the B_i-cycles that belong to less significant bits close before those corresponding to more significant bits, we use longer B_i-cycles

for the more significant bit positions (the A_i-cycles and C_i-cycles, in contrast, are all of the same length).

We rely on the following two simple probabilistic lemmata.

Lemma 4.69. *Let a be the total length of all the cycles that are currently opening. Then, the probability that a closing cycle acquires at least b new edges before all opening cycles open completely is at most $\frac{a}{2^b}$.*

Lemma 4.70. *The probability that a closing cycle acquires b new edges before a different closing cycle of length a closes completely is at most $\mathrm{e}^{-\frac{1}{2}(b-a)^2/(b+a)}$.*

Let $\nu(a)$ be the value of b for which the probability $\mathrm{e}^{-\frac{1}{2}(b-a)^2/(b+a)}$ of Lemma 4.70 is at most 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 Z is a cycle 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 Z as the *noise level* of Z. 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 bound specified on the right of Table 4.18, at any time during the phase, is exponentially small. Three different noise bounds ξ^B, ξ^C, ξ^A are specified for B_i-cycles, A_i-cycles and C_i-cycles, respectively. A phase ends successfully if the noise level of each cycle never reaches the length of that cycle.

It is not difficult to prove by induction that the probability that the noise level of a cycle exceeds the noise bound given in Table 4.18 is exponentially small. Let us look, for example, at the noise levels of the B_i-cycles. In phase 5, no B_i-cycle is closing, so $\xi^B = 0$ is a (vacuous) upper bound on the noise level of closing B_i-cycles. The same holds for phase 6. Some B_i-cycles are closing in phase 7. All these cycles, however, are completely open at the beginning of phase 7. The competition in phase 7 is with the left lane W. The realignment of a lane may be viewed as the closing of a cycle of length at most n (both lanes and cycles close one edge at a time). Thus, by Lemma 4.70, the probability that the noise level of any of the B_i-counters exceeds $\nu(n)$ is at most e^{-n}. As mentioned $\nu(n) < 5n$. Phase 1 is a cycle opening

competition. By Lemma 4.69, the probability that the noise level of a given B_i-cycle increases by more than $\rho = 2n$ is $O(n^4/2^{2n}) = o(\mathrm{e}^{-n})$. The other noise bound can be verified in a similar manner.

We are now in a position to choose the length of the various cycles. The length h of all the A_i-cycles should satisfy $h > \nu(n) + \rho$. This is satisfied by choosing $h = 8n$. The length g of the C_i-cycles 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}-cycle 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 check that this is satisfied by the choice $\ell_k = 25k^2 n$.

Results

We conclude that policy iteration with the RANDOM-EDGE rule requires an exponential number of iterations on the MDPs of this chapter.

Theorem 4.71. *The expected number of improving switches performed by the* RANDOM-EDGE*-rule on the MDPs constructed in this section, which contain $\mathcal{O}(n^4)$ vertices and edges, is $\Omega(2^n)$.*

Obviously, we can transfer the result to MDPs with the discounted reward criterion (for large enough discount factors).

Corollary 4.72. *The number of improving steps performed by* RANDOM-EDGE *policy iteration with the discounted reward criterion on the MDPs constructed in this section is $\Omega(2^n)$.*

Since Markov decision process policy iteration corresponds immediately to the simplex algorithm for solving related linear programs, we have the following result:

Theorem 4.73. *The number of improving steps performed by* RANDOM-EDGE *simplex algorithm on the linear programs induced by the MDPs constructed in this section is $\Omega(2^n)$.*

Parity Games

We show how the lower bound graphs can be turned into a parity game to provide a lower bound for random edge here as well.

Essentially, the graph is the same. Randomization nodes are replaced by player 1 controlled nodes s.t. the cycles are won by player 0. We assign low unimportant priorities to all nodes that have currently no priority, while giving the nodes on the cycle odd priorities to make sure that moving into the cycle is only profitable by switching one edge at a time.

For a tuple $\zeta = (n, (\ell_i)_{0 \le i \le n}, h, g)$, with $n, \ell_i, h, g > 0$, we define the underlying graph $G_\zeta = (V_0, V_1, E, \Omega)$ of a parity game as shown schematically in Figure 4.24. More formally:

$$\begin{aligned} V_0 \;:=\; & \{a_{i,j} \mid i \in [n], j \in [h]\} \;\cup\; \{b_{i,j} \mid i \in [n], j \in [\ell_i]\} \cup \\ & \{c_{i,j} \mid i \in [n], j \in [g]\} \;\cup\; \{d_i, y_i, x_i \mid i \in [n]\} \cup \\ & \{w_i, u_i \mid i \in [n+1]\} \;\cup\; \{t, r, s\} \\ V_1 \;:=\; & \{\mathrm{A}_i, \mathrm{B}_i \mid i \in [n]\} \end{aligned}$$

Table 4.19 defines the edge sets and the priorities of G_ζ.

Node V	Successors in E	Priority Ω	Node V	Successors in E	Priority Ω
t	t	1	r	w_1	6
w_{n+1}	t	2	s	u_1	2
u_{n+1}	t	2	d_i	B_i	$4i+5$
w_i	w_{i+1}, A_i	2	y_i	w_{i+1}	$4i+6$
u_i	u_{i+1}, d_i	2	B_i	$y_i, b_{i,\ell_i}, c_{i,g}$	4
A_i	$d_i, a_{i,h}$	4	$b_{i,1}$	B_i, s	3
$a_{i,1}$	A_i, x_i	3	$b_{i,j+1}$	$b_{i,j}, s$	3
$a_{i,j+1}$	$a_{i,j}, x_i$	3	$c_{i,1}$	B_i, r	3
x_i	s	$4i+3$	$c_{i,j+1}$	$c_{i,j}, r$	3

Table 4.19: Edges and Priorities of G_ζ

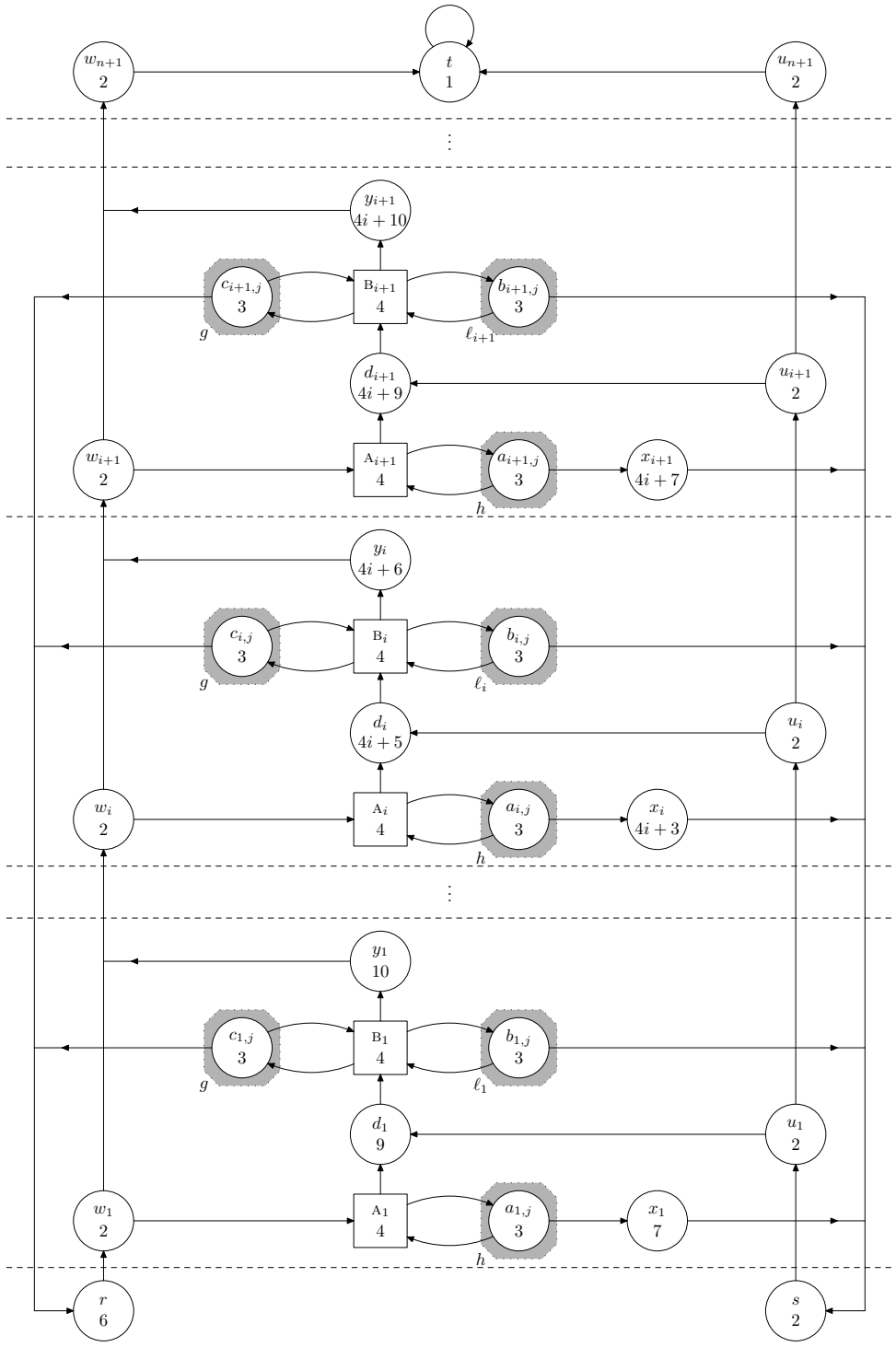


Figure 4.24: Lower bound parity game for RANDOM-EDGE

The first important observation to make is that the parity game is a sink game, which helps us to transfer our result to payoff games. The following lemma corresponds to Lemma 4.64 in the MDP world.

Lemma 4.74. *Starting with an initial strategy σ s.t. $\overline{\sigma}(w_*) = \overline{\sigma}(s_*) = 0$, we have that G_ζ is a sink parity game.*

All other definitions are exactly as before. Particularly, Table 4.17 becomes applicable again. The following lemma has the exact same formulation as Lemma 4.68 in the MDP world.

Lemma 4.75. *The improving switches from strategies in the parity game that belong to the phases are exactly those specified in Table 4.17.*

The reason why this lemma holds is, that the valuations of the parity game nodes are essentially the same as the potentials in the MDP by dropping unimportant $\mathcal{O}(1)$ terms.

All other proofs rely on Table 4.17 and Lemma 4.68, hence we transfer our main theorem to the parity game world.

Theorem 4.76. *Parity game strategy iteration with* RANDOM-EDGE*-rule requires at least 2^n expected improvement steps on G_n.*

Since G_n is a family of sink parity games, it follows directly by Theorem 4.19 that we have a subexponential lower bound for payoff games.

Corollary 4.77. *Payoff game strategy iteration with* RANDOM-EDGE*-rule requires expected subexponential time.*

We mention without proof that the construction gives another subexponential lower bound for Schewe's SWITCH-BEST improvement rule [Sch08].

Remarks

The analysis of the counter as well as the game construction for RANDOM-EDGE can be improved greatly. First, the probabilistic analysis is based on the desire,

that the binary counter operates without *any* faults, meaning that we really want to perform 2^n increment steps. However, counting "good enough", i.e. skipping a small number of increment steps from time to time, would still yield an exponential number of increment steps. Hence, our probabilistic analysis could be relaxed in such a way that the length of the cycles could be reduced. Additional details will appear in subsequent publications.

Second, using a more complicated construction for parity games, we can decrease the lengths of the cycles to be linear in n, resulting in a $2^{\Omega(\sqrt{n})}$ lower bound. The main idea is to have downgoing edges from a node $b_{i,j}$ to nodes $b_{i',j}$ with $i' < i$; which of these downgoing edges is chosen will be controlled by player 1. The difference in the behavior of RANDOM-EDGE on the improved construction is that when all B_i-cycles are competing with each other, higher B_i-cycles actually open all their nodes again that are already subsumed by the least open B_i-cycle. Hence, it is sufficient to have the same length for all B_i-cycles. The improved result is likely to transfer to MDPs and linear programs as well. Additional details will appear in subsequent publications.

4.7.3 Switch Half and all that

Our lower bound for the RANDOM-EDGE policy iteration for parity games and related two-player games can be extended to arbitrary *randomized multi-switch* improvement rules which select in each iteration step an applicable subset with a certain cardinality of the improving switches arbitrarily at random. RANDOM-EDGE, for instance, always selects subsets with cardinality one, and the deterministic SWITCH-ALL rule always selects the subset with maximal cardinality. Another important randomized multi-switch improvement rule is SWITCH-HALF [MS99], which applies every improving switch with probability $1/2$, assuming the binary case.

SWITCH-HALF: Apply every improvement with probability $1/2$.

The lower bound transfers to all randomized multi-switch improvement rules due to the fact that the two kinds of competitions that we have in the analysis are won

with even higher probability, when the cardinality of the number of switches that are to be made at the same time is greater than one.

More generally, we consider a family of improvement rules here that capture the space of *randomized* (meaning that choosing two arbitrary improving switches is equally likely) and *oblivious* (meaning that the improvement rule is not allowed to manage any additional data structure) procedures to select the set of switches to be performed. For simplicity, we assume binary out-degree here.

Let $P = (p_n : \{0, \ldots, n\} \to [0, 1])_{n>0}$ be a family of discrete probability mass functions, i.e. for every $n > 0$ we have $\sum_{i=0}^{n} p_n(i) = 1$. If additionally $p_n(0) < 1$ for every $n > 0$, then we call P *probabilistic positive integer selector*.

Every such $P = (p_n)_{n>0}$ induces an *oblivious randomized improvement rule* $\mathrm{Improve}_P$ as follows. Let G be a game, σ be a strategy, I_σ be the set of improving switches, $n = |I_\sigma| > 0$ and $I \subseteq I_\sigma$ be a non-empty subset of improving switches. Let $i = |I| > 0$. Then $\mathrm{Improve}_P(G, \sigma) = \sigma[I]$ with probability

$$\frac{1}{\binom{n}{i}} \cdot \frac{p_n(i)}{1 - p_n(0)}$$

See Algorithm 11 for an algorithmic presentation of this rule.

Algorithm 11 Oblivious Randomized Improvement Rule

```
i ← 0
while i = 0 do
    Choose i ≤ |I_σ| at random according to p_{|I_σ|}
end while
Choose I ⊆ I_σ with |I| = i uniformly at random
return σ[I]
```

The oblivious randomized improvement rule captures many interesting rules from the literature, it particularly incorporates the following ones:

1. RANDOM-EDGE by $p_n(1) = 1$ for all $n > 0$.

2. SWITCH-HALF by $p_n(k) = \binom{n}{k} \cdot 2^{-n}$ for all $n > 0$.

3. SWITCH-ALL by $p_n(n) = 1$ for all $n > 0$.

4. SWITCH $0 < q \leq 1$ by $p_n(k) = \binom{n}{k} \cdot q^k \cdot (1-q)^{n-k}$ for all $n > 0$.

Theorem 4.78. *Policy iteration with oblivious randomized improvement rules for infinitary payoff games requires expected subexponential time. Particularly, the* SWITCH-HALF*-rule requires expected subexponential time.*

4.8 Memorizing Rules

There is one famous history-based improvement rule that we consider in this chapter.

Known as the LEAST-ENTERED rule, Zadeh's pivoting method [Zad80] belongs to the family of memorizing improvement rules, which among all improving switches chooses one which has been applied least often.

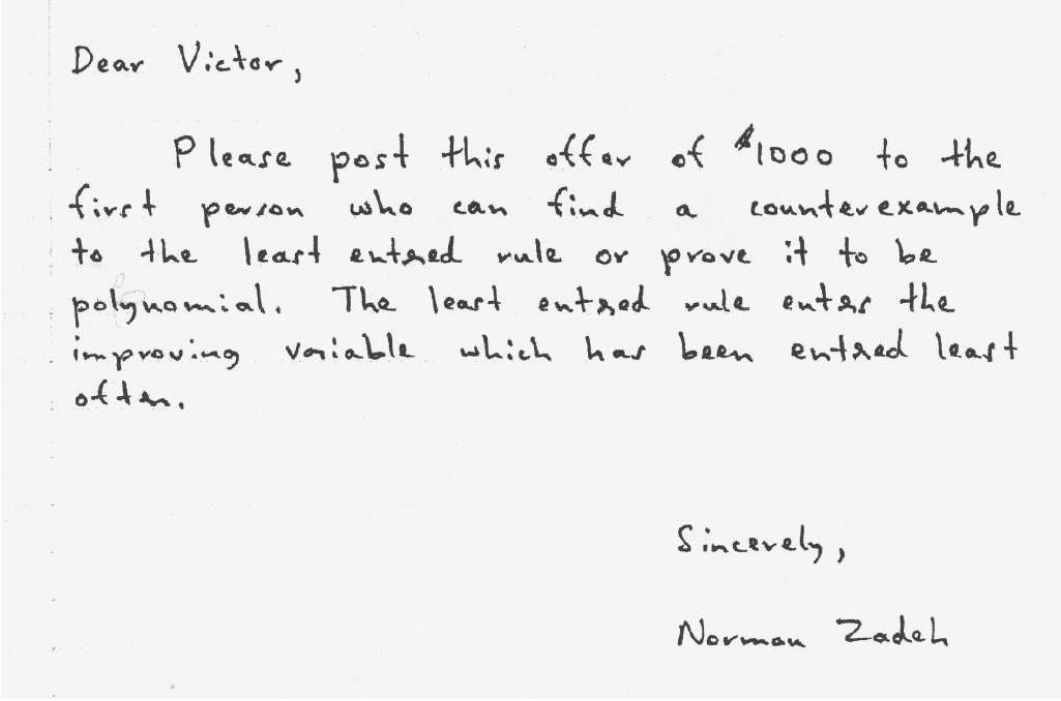
Zadeh's pivoting rule is formulated for linear programming solving, and has entered the folklore of convex optimization. The pivoting rule was proposed around 1980 [Zad80], and Zadeh offered a little prize of $1000 to anyone who can show that the rule admits polynomially many iterations or to prove that there is a family of linear programs on which the pivoting rule requires subexponentially many iterations to find the optimum. Zadeh formulated his offer in a letter to Victor Klee, see Figure 4.25 (from [Zie04]).

Our contribution is to give the first explicit construction of a subexponential lower bound for LEAST-ENTERED in the context of limiting average Markov decision processes. We transfer the result to discounted reward criterion MDPs, the simplex algorithm for solving linear programs, to parity game strategy iteration, and to policy iteration for the other classes of infinitary payoff games.

All technically tedious proofs have been put into Appendix A.4.

4.8.1 Zadeh's Pivoting Rule

Zadeh's LEAST-ENTERED pivoting rule is a *deterministic*, *memorizing* improvement rule which among all improving pivoting steps from the current basic feasible


Dear Victor,

Please post this offer of $1000 to the first person who can find a counterexample to the least entered rule or prove it to be polynomial. The least entered rule enters the improving variable which has been entered least often.

Sincerely,

Norman Zadeh

Figure 4.25: Zadeh's Rule

solution (or *vertex*) chooses one which has been entered least often. It was originally described in terms of solving linear programs [Zad80]. When applied to the primal linear program of an MDP, it is equivalent to the variant of the policy iteration algorithm, in which the improving switch is chosen among all improving switches to be one, which has been chosen least often. This is the foundation of our lower bound for the LEAST-ENTERED rule.

LEAST-ENTERED:	Apply a switch that has been switched least often.

We describe Zadeh's pivoting rule now formally in the context of MDPs. As a memorization structure, we introduce an *occurrence record*, which is a map $\phi : E_0 \to \mathbb{N}$ that specifies for every player 0 edge of the given MDP how often it has been used. Among all improving switches in the set I_σ for a given policy σ, we need to choose an edge $e \in I_\sigma$ that has been selected least often. We denote the set of *least occurred improving switches* by $I_\sigma^\phi = \{e \in I_\sigma \mid \phi(e) \leq \phi(e') \text{ for all } e' \in I_\sigma\}$.

See Algorithm 12 for a pseudo-code specification of the LEAST-ENTERED pivoting rule for solving MDPs.

Algorithm 12 Zadeh's Improvement Algorithm

```
1: procedure LEAST-ENTERED(G,σ)
2:     φ(e) ← 0 for every e ∈ E_0
3:     while I_σ ≠ ∅ do
4:         e ← select edge from I_σ^φ
5:         φ(e) ← φ(e) + 1
6:         σ ← σ[e]
7:     end while
8: end procedure
```

In the original specification of Zadeh's algorithm [Zad80], there is no clear objective how to *break ties* whenever $|I_\sigma^\phi| > 1$. In fact, we know that the asymptotic behavior of Zadeh's improvement rule highly depends on the method that is used to break ties, at least in the world of MDPs, PGs and policy iteration for games in general. We have the following corollary which is easy to verify (the idea is that there is at least one improving switch towards the optimal policy in each step) by Lemma 4.2.

Corollary 4.79. *Let G be an MDP with n nodes and σ_0 be a policy. There is a sequence policies $\sigma_0, \sigma_1, \ldots, \sigma_N$ and a sequence of* different *switches $e_1, e_2, \ldots, e_N$ with $N \leq n$ s.t. σ_{N-1} is optimal, $\sigma_{i+1} = \sigma_i[e_{i+1}]$ and e_{i+1} is an σ_i-improving switch.*

Since all switches are different in the sequence, it follows immediately that there is always a way to break ties that results in a linear number of pivoting steps to solve an MDP with Zadeh's improvement rule. However, there is no obvious method of breaking ties. The question whether Zadeh's pivoting rule solves MDPs (and LPs) in polynomial time should therefore be phrased *independently* of the heuristic of breaking ties. In other words, we as "lower bound designers" are the ones that choose a particular tie breaking rule.

Formally, we write $(\sigma, \phi) \rightsquigarrow (\sigma', \phi')$ iff there is an edge $e \in I_\sigma^\phi$ s.t. $\sigma' = \sigma[e]$ and $\phi' = \phi[e \mapsto \phi(e) + 1]$. Let $\rightsquigarrow^+$ denote the transitive closure of $\rightsquigarrow$. The question, whether Zadeh's improvement rule admits a polynomial number of

iterations independently of the method of breaking ties is therefore equivalent to the question, whether the length of every sequence $(\sigma_0, \phi_0) \rightsquigarrow^+ \ldots \rightsquigarrow^+ (\sigma_N, \phi_N)$ can be polynomially bounded in the size of the game.

We will not specify the tie-breaking rule used for our lower bound explicitly, due to the fact that the rule itself is not a natural one. Instead, our proof just relies on the $\rightsquigarrow$-relation, witnessing in every improvement step that we only select an improving switch that has been applied least often.

Fair Counting

In high level terms, our PGs, MDPs, and the linear programs corresponding to them, are constructions of 'pairwise alternating' binary counters. Consider a normal binary counter: less significant bits are switched more often than higher bits, when counting from 0 to $2^n - 1$. Zadeh's rule would not go through all steps from 0 to $2^n - 1$ on such a counter, because higher bits will be switched before they are supposed to be switched, as the switching times that are associated with higher bits will catch up with the switching times associated with lower bits. Zadeh's rule, in a sense, requires a "fair" counter that operates correctly when all bits are switched equally often.

Our solution to this problem is to represent each bit i in the original counter by *two* bits i' and i'' s.t. only one of those two is actively working as representative for i. After switching the representative for i – say i' – from 0 to 1 and back to 0, we change the roles of i' and i'' s.t. i'' becomes the active representative for i. The inactive i' can now, while i'' switches from 0 to 1 and back to 0, catch up with the rest of the counter in terms of switching fairness: while i' is inactive, we switch i' from 0 to 1 back and forth (without effecting the rest of the counter as i' is the inactive representative) until the number of switching times catches up with the number of switching times of the rest of the counter again.

Another viable approach could be to implement more sophisticated binary counters like Gray codes (see e.g. [BS96]). However, the construction of an MDP or PG that models the behavior of a Gray code-based counter seems to be a very difficult task.

High-level Description

We start with a high-level description of the MDPs on which LEAST-ENTERED performs an expected subexponential number of iterations. A schematic description of the lower bound MDPs is given in Figure 4.26.

The MDP of Figure 4.26 emulates an n-bit counter. It is composed of n identical levels, each corresponding to a single bit of the counter. The i-th level ($i = 1 \dots n$) is shown explicitly in the figure. Levels are separated by dashed lines. The MDP includes one *source* s and one *sink* t.

All edges in Figure 4.26 have an immediate reward of 0 associated with them (such 0 rewards are not shown explicitly in the figure) unless stated otherwise as follows: Some of the vertices are assigned integer *priorities*. If a vertex v has priority $\Omega(v)$ assigned to it, then a reward of $\langle v \rangle = (-N)^{\Omega(v)}$ is *added* to all edges emanating from v, where N is a sufficiently large integer. We use $N \geq 7n + 1$ and $\varepsilon \leq N^{-(2n+11)}$. Priorities, if present, are listed next to the vertex name. Note that it is profitable for the controller, to move through vertices of even priority and to avoid vertices of odd priority, and that vertices of higher numerical priority dominate vertices of lower priority (the idea of using priorities is inspired, of course, by the reduction from parity games to mean payoff games).

Each level i contains two (i.e. $j = 0, 1$) instances of a gadget that consists of a randomization vertex A_i^j and two (i.e. $l = 0, 1$) attached cycles with player 0 controlled nodes $b_{i,l}^j$. Therefore, we will call these gadgets from now on *bicycle gadgets*, and refer to the instance with j=0 resp. j=1 as to *bicycle* 0 resp. *bicycle* 1.

From A_i^j (with $j = 0, 1$), the edge $\mathrm{A}_i^j \to b_{i,l}^j$ (with $l = 0, 1$), is chosen with probability $\frac{1-\varepsilon}{2}$, while the edge $\mathrm{A}_i^j \to d_i^j$ is chosen with probability ε. Thus, if both $\sigma(b_{i,0}^j) = \mathrm{A}_i^j$ and $\sigma(b_{i,1}^j) = \mathrm{A}_i^j$, the MDP is guaranteed to eventually move from A_i^j to d_i^j (this is similar to the use of randomization nodes by Fearnley [Fea10]). We say that a bicycle gadget is

- *closed* iff both $\sigma(b_{i,0}^j) = \mathrm{A}_i^j$ and $\sigma(b_{i,1}^j) = \mathrm{A}_i^j$,
- *open* iff $\sigma(b_{i,0}^j) \neq \mathrm{A}_i^j$ or $\sigma(b_{i,1}^j) \neq \mathrm{A}_i^j$, and

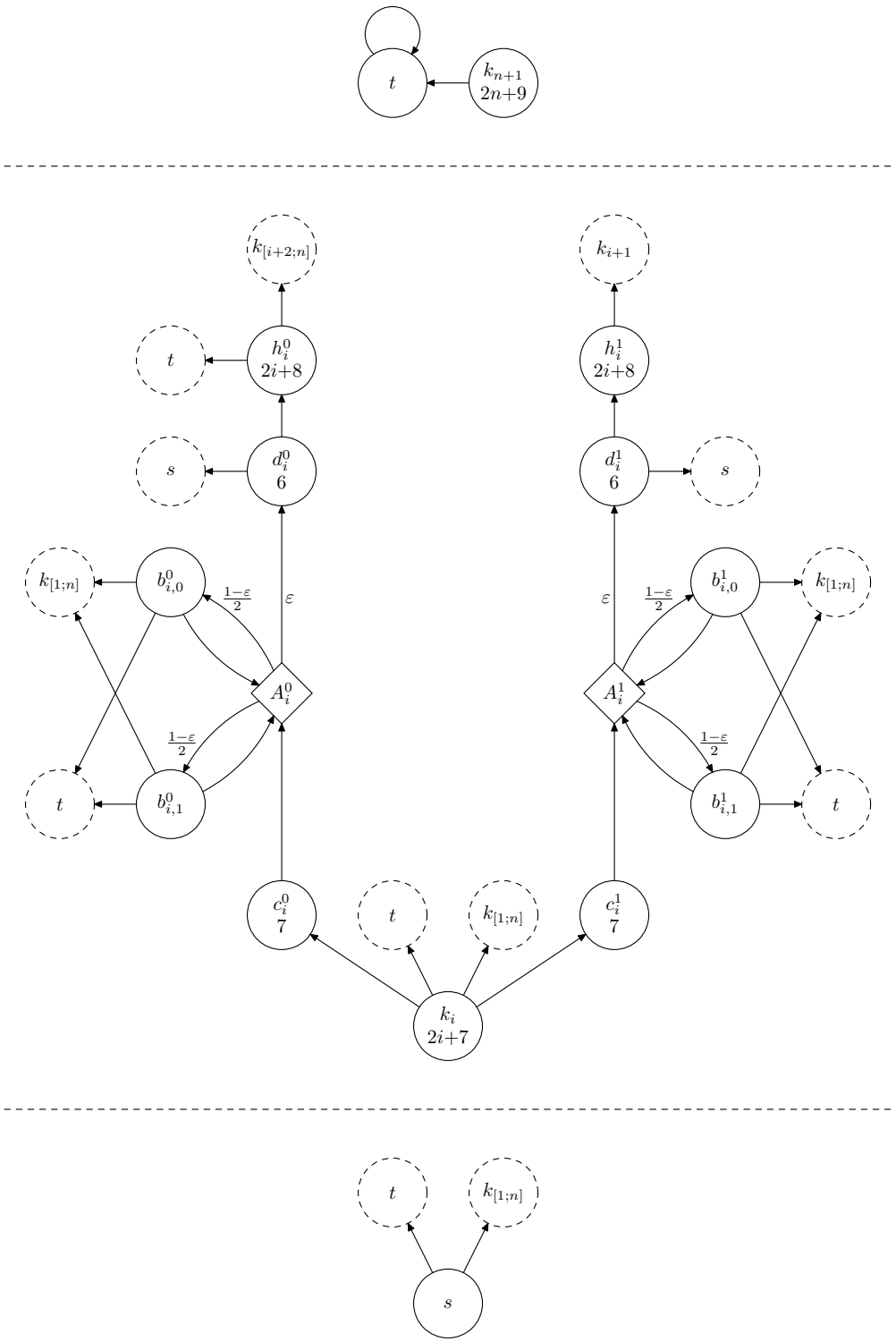


Figure 4.26: Least Entered MDP Construction

- *completely open* iff $\sigma(b^j_{i,0}) \neq \mathrm{A}^j_i$ and $\sigma(b^j_{i,1}) \neq \mathrm{A}^j_i$.

Recall our notation to succinctly describe binary counters. It will be convenient for us to consider counter configurations with an *infinite* tape, where unused bits are zero. The set of n-bit configurations is formally defined as $\mathcal{B}_n = \{\mathfrak{b} \in \{0,1\}^\infty \mid \forall i > n : \mathfrak{b}_i = 0\}$.

We start with index one, i.e. $\mathfrak{b} \in \mathcal{B}_n$ is essentially a tuple $(\mathfrak{b}_n, \ldots, \mathfrak{b}_1)$, with $\mathfrak{b}_1$ being the least and $\mathfrak{b}_n$ being the most significant bit. By $\mathbf{0}$, we denote the configuration in which all bits are zero, and by $\mathbf{1}_n$, we denote the configuration in which the first n bits are one. We write $\mathcal{B} = \bigcup_{n>0} \mathcal{B}_n$ to denote the set of all counter configurations.

Given a configuration $\mathfrak{b}$, we access the *i-next set bit* by $\nu^n_i(\mathfrak{b}) = \min(\{n+1\} \cup \{j \geq i \mid \mathfrak{b}_j = 1\})$, and the *$i$-next unset bit* by $\mu_i(\mathfrak{b}) = \min\{j \geq i \mid \mathfrak{b}_j = 0\}$.

The i-th level of the MDP corresponds to the i-th bit. A set bit is represented by a *closed* bicycle gadget. Every level has two bicycle gadgets, but only one of them is *actively* representing the i-th bit.

Whether bicycle 0 or bicycle 1 is active in level i depends on the setting of the $i{+}1$-th bit. If it is set, i.e. $\mathfrak{b}_{i+1} = 1$, then bicycle 1 is active in the i-th level; otherwise, if $\mathfrak{b}_{i+1} = 0$, we have that bicycle 0 is active in the i-th level.

Our proof is conceptually divided into two parts. First we investigate the improving switches that can be performed from certain policies of the MDP. This allows us to prove the *existence* of a sequence of improving switches that indeed generates the sequence of policies $\sigma_{0\ldots00}, \sigma_{0\ldots01}, \sigma_{0\ldots10}, \ldots, \sigma_{1\ldots11}$. A transition from $\sigma_\mathfrak{b}$ to $\sigma_{\mathfrak{b}+1}$ involves many intermediate improvement steps. We partition the path leading from $\sigma_\mathfrak{b}$ to $\sigma_{\mathfrak{b}+1}$ into six sub-paths which we refer to as *phases*. In the following, we first give an informal description of the phases. The second part of our proof will be to show that the way we want to apply the improving switches is compliant with the associated occurrence records.

Before starting to describe what happens in the different phases, we describe the "ideal" configuration of a policy, which belongs to phase 1: (1) all active bicycles corresponding to set bits are closed, (2) all other bicycles are completely open,

moving to the least set bit, (3) all entry points k_i move to the active bicycle if bit i is set and to the least set bit otherwise, (4) the source s moves to the least set bit, (5) all upper selection nodes h_i^0 move to the next *accessible* set bit (i.e. to the next set bit with index $\geq i{+}2$), and (6) the selection nodes d_i^j move higher up iff the immediately accessed bit is the next set bit (i.e. d_i^0 moves higher up iff $\mathfrak{b}_{i+1} = 0$ and d_i^1 moves higher up iff $\mathfrak{b}_{i+1} = 1$).

Note that the two upper selection nodes h_i^0 and h_i^1 cannot select the same entry points. The left node, h_i^0, can select from the entry points k_{i+2} up to k_n, while the right node, h_i^1, can only move to k_{i+1}. The intuition behind this is that bit $i{+}1$ is set every second time bit i is flipped, resulting in the alternating activation of the two bit representatives for i.

Now, we are ready to informally describe all phases.

1. At the beginning of the first phase, we only have open bicycles that are competing with each other to close. Inactive bicycles may have to catch up with active bicycles, and hence, are allowed to switch both player 0 edges inward, and therefore close the gadget. All active open bicycles move exactly one edge inward in this phase.

   So far, no active open bicycles have been closed. The last switch that is performed in this phase is to move the remaining edge of the active bicycle associated with the least unset bit inward, and therefore close the gadget.

2. In this phase, we need to make the recently set bit i accessible by the rest of the MDP, which will be via the k_i node. We switch here from k_i to c_i^j, where j denotes the active representative in this level.

   Note that k_i now has the highest potential among all other k_*. Note that generally, k_l has a higher potential than k_z for a set bit l and an unset bit z, and that k_l has a higher potential than k_z for two set bits l and z iff $l < z$.

3. In the third phase, we perform the major part of the *resetting* process. By resetting, we mean to unset lower bits again, which corresponds to reopening the respective bicycles.

Also, we want to update all other inactive or active but not set bicycles again to move to the entry point k_i. In other words, we need to update the lower entry points k_z with $z < i$ to move to k_i, and the bicycle nodes $b^j_{z,l}$ to move to k_i.

We apply these switches by first switching the entry node k_z for some $z < i$, and then the respective bicycle nodes $b^j_{z,l}$.

4. In the fourth phase, we update the upper selection nodes h^0_z for all $z < i - 1$ of the bits that have been reset. All nodes h^0_z should move to k_i.

5. In the fifth phase, we update the source node to finally move to the entry point corresponding to the recently set bit i.

6. In the last phase, we only have to update the selection nodes d^j_z for all $z < i$ of the bits that have been reset. We finally end up in a phase 1 policy again with the counter increased by one.

Full Construction

In this paragraph, we formally describe the full construction of our MDPs. We define an underlying graph $G_n = (V_0, V_R, E, r, p)$ of an MDP as shown schematically in Figure 4.26 (we use the notation $k_{[i;j]}$ to indicate that player 0 in fact has edges to every node k_l with $i \leq l \leq j$) as follows:

$$\begin{aligned} V_0 &:= \{b^0_{i,0}, b^1_{i,0}, b^0_{i,1}, b^1_{i,1}, d^0_i, d^1_i, h^0_i, h^1_i, c^0_i, c^1_i \mid i \in [n]\} \cup \\ &\quad \{k_i \mid i \in [n+1]\} \cup \{t, s\} \\ V_R &:= \{\mathrm{A}^0_i, \mathrm{A}^1_i \mid i \in [n]\} \end{aligned}$$

With G_n, we associate a large number $N \in \mathbb{N}$ and a small number $0 < \varepsilon$. We require N to be at least as large as the number of nodes with priorities, i.e. $N \geq 7n+1$ and ε^{-1} to be significantly larger than the largest occurring priority induced reward, i.e. $\varepsilon \leq N^{-(2n+11)}$. Remember that node v having priority $\Omega(v)$ means that the cost associated with every outgoing edge of v is $\langle v \rangle = (-N)^{\Omega(v)}$.

Table 4.20 defines the edge sets, the probabilities, the priorities and the immediate rewards of G_n (note that h_i^0 has the successors t, k_{i+2}, ... k_n; particularly, h_n^0 has only t as successor).

Node	Successors	Probability
A_i^j	d_i^j	ε
	$b_{i,0}^j$	$\frac{1}{2} \cdot (1-\varepsilon)$
	$b_{i,1}^j$	$\frac{1}{2} \cdot (1-\varepsilon)$

Node	Successors	Priority
t	t	-
s	$t, k_{[1;n]}$	-

Node	Successors	Priority
k_{n+1}	t	$2n{+}9$
k_i	$c_i^0, c_i^1, t, k_{[1;n]}$	$2i{+}7$
h_i^0	$t, k_{[i+2;n]}$	$2i{+}8$
h_i^1	k_{i+1}	$2i{+}8$
c_i^j	A_i^j	7
d_i^j	h_i^j, s	6
$b_{i,*}^j$	$t, \mathrm{A}_i^j, k_{[1;n]}$	-

Table 4.20: Least Entered MDP Construction

As designated *initial policy* σ^*, we use $\sigma^*(d_i^j) = h_i^j$, and $\sigma^*(\_) = t$ for all other player 0 nodes with non-singular out-degree. It is not hard to see that, starting with this initial policy, the MDP satisfies the weak unichain condition.

Lemma 4.80. *The Markov chains obtained by the initial and the optimal policy reach the sink t almost surely (i.e. the sink t is the single irreducible recurrent class).*

It is not too hard to see that the absolute potentials of all nodes corresponding to policies belonging to the phases are bounded by ε^{-1}. More formally we have:

Lemma 4.81. *Let $P = \{k_*, h_*^*, c_*^*, d_*^*\}$ be the set of nodes with priorities. For a subset $S \subseteq P$, let $\sum(S) = \sum_{v \in S} \langle v \rangle$. For non-empty subsets $S \subseteq P$, let $v_S \in S$ be the node with the largest priority in S.*

1. *$|\sum(S)| < N^{2n+11}$ and $\varepsilon \cdot |\sum(S)| < 1$ for every subset $S \subseteq P$, and*
2. *$|v_S| < |v_{S'}|$ implies $|\sum(S)| < |\sum(S')|$ for non-empty subsets $S, S' \subseteq P$.*

Lower Bound Proof

In this paragraph, we formally describe the different *phases* that a policy can be in, as well as the improving switches in each phase. The increment of the binary counter

by one is realized by transitioning through all the phases. Finally, we describe the corresponding occurrence records that appear in a run of the policy iteration on the MDPs.

We first introduce notation to succinctly describe policies. It will be convenient to describe the decision of a policy σ in terms of integers rather than concrete target vertices. Let σ be a policy. We define a function $\overline{\sigma}(v)$ as follows.

$\sigma(v)$	t	k_i	h^*_*	s	A^*_*	c^j_i
$\overline{\sigma}(v)$	$n+1$	i	1	0	0	$-j$

Additionally, we write $\overline{\sigma}(\mathrm{A}^j_i) = 1$ if $\sigma(b^j_{i,0}) = \mathrm{A}^j_i$ and $\sigma(b^j_{i,1}) = \mathrm{A}^j_i$, and $\overline{\sigma}(\mathrm{A}^j_i) = 0$ otherwise.

We are now ready to formulate the conditions for policies that fulfill one of the six phases along with the improving edges. See Table 4.21 for a complete description (with respect to a bit configuration $\mathfrak{b}$). We say that a strategy σ is a phase p strategy with configuration $\mathfrak{b}$ iff every node is mapped by σ to a choice included in the respective cell of the table. Cells that contain more than one choice indicate that strategies of the respective phase are allowed to match any of the choices.

Table 4.22 specifies the sets of improving switches by providing for each phase p a subset L^p_σ and a superset U^p_σ s.t. $L^p_\sigma \subseteq I_\sigma \subseteq U^p_\sigma$. The intuition behind this method of giving the improving switches is that we will only *use* switches from L^p_σ while making sure that *no* other switches from U^p_σ are applied.

The following lemma tells us that all occurring potentials in the policy iteration are *small* compared to N^{2n+11}. Particularly, ε-times potentials are almost negligible.

Lemma 4.82. *Let σ be a policy belonging to one of the phases specified in Table 4.21. Then $|\mathrm{POT}_\sigma(v)| < N^{2n+11}$ and $\varepsilon \cdot |\mathrm{POT}_\sigma(v)| < 1$ for every node v.*

We finally arrive at the following main lemma describing the improving switches.

Lemma 4.83. *The improving switches from policies that belong to the phases in Table 4.21 are bounded by those specified in Table 4.22, i.e. $L^p_\sigma \subseteq I_\sigma \subseteq U^p_\sigma$ for a phase p policy σ.*

Phase	1	2	3
$\overline{\sigma}(s)$	r	r	r
$\overline{\sigma}(d_i^0)$	$1-\mathfrak{b}_{i+1}$	$1-\mathfrak{b}_{i+1}$	$1-\mathfrak{b}_{i+1}$
$\overline{\sigma}(d_i^1)$	$\mathfrak{b}_{i+1}$	$\mathfrak{b}_{i+1}$	$\mathfrak{b}_{i+1}$
$\overline{\sigma}(h_i^0)$	$\nu_{i+2}^n(\mathfrak{b})$	$\nu_{i+2}^n(\mathfrak{b})$	$\nu_{i+2}^n(\mathfrak{b})$
$\overline{\sigma}(b_{*,*}^*)$	0, r	0, r	0, r, r'
$\overline{\sigma}(\mathrm{A}_i^{\mathfrak{b}_{i+1}})$	$\mathfrak{b}_i$	$*$	$*$
$\overline{\sigma}(\mathrm{A}_i^{\mathfrak{b}'_{i+1}})$	$*$	$\mathfrak{b}'_i$	$\mathfrak{b}'_i$

Phase	4	5	6
$\overline{\sigma}(s)$	r	r	r'
$\overline{\sigma}(d_i^0)$	$1-\mathfrak{b}_{i+1}$	$1-\mathfrak{b}_{i+1}$	$1-\mathfrak{b}_{i+1}$, $1-\mathfrak{b}'_{i+1}$
$\overline{\sigma}(d_i^1)$	$\mathfrak{b}_{i+1}$	$\mathfrak{b}_{i+1}$	$\mathfrak{b}_{i+1}$, $\mathfrak{b}'_{i+1}$
$\overline{\sigma}(h_i^0)$	$\nu_{i+2}^n(\mathfrak{b})$, $\nu_{i+2}^n(\mathfrak{b}')$	$\nu_{i+2}^n(\mathfrak{b}')$	$\nu_{i+2}^n(\mathfrak{b}')$
$\overline{\sigma}(b_{*,*}^*)$	0, $\nu_1^n(\mathfrak{b})$	0, $\nu_1^n(\mathfrak{b})$	0, $\nu_1^n(\mathfrak{b})$
$\overline{\sigma}(\mathrm{A}_i^{\mathfrak{b}_{i+1}})$	$*$	$*$	$*$
$\overline{\sigma}(\mathrm{A}_i^{\mathfrak{b}'_{i+1}})$	$\mathfrak{b}'_i$	$\mathfrak{b}'_i$	$\mathfrak{b}'_i$

Phase	1–2	4–6
$\overline{\sigma}(k_i)$	$\begin{cases} r & \text{if } \mathfrak{b}_i = 0 \\ -\mathfrak{b}_{i+1} & \text{if } \mathfrak{b}_i = 1 \end{cases}$	$\begin{cases} r' & \text{if } \mathfrak{b}'_i = 0 \\ -\mathfrak{b}'_{i+1} & \text{if } \mathfrak{b}'_i = 1 \end{cases}$

Phase	3
$\overline{\sigma}(k_i)$	$\begin{cases} r, r' & \text{if } \mathfrak{b}'_i = 0 \text{ and } \mathfrak{b}_i = 0 \\ -\mathfrak{b}_{i+1}, r' & \text{if } \mathfrak{b}'_i = 0 \text{ and } \mathfrak{b}_i = 1 \\ -\mathfrak{b}'_{i+1} & \text{if } \mathfrak{b}'_i = 1 \end{cases}$

Phase 3	Side Conditions: for every i and every j we have
(a)	$\left(\mathfrak{b}'_i = 0 \text{ and } (\exists j, l. \overline{\sigma}(b_{i,l}^j) = r')\right)$ implies $\overline{\sigma}(k_i) = r'$
(b)	$\left(\mathfrak{b}'_i = 0, \mathfrak{b}'_j = 0, \overline{\sigma}(k_i) = r' \text{ and } \overline{\sigma}(k_j) \neq r'\right)$ implies $i > j$

Table 4.21: Policy Phases (where $\mathfrak{b}' = \mathfrak{b} + 1$, $r = \nu_1^n(\mathfrak{b})$ and $r' = \nu_1^n(\mathfrak{b}')$)

Phase p	Improving Switches Subset L^p_σ
1	$\{(b^j_{i,l}, \mathrm{A}^j_i) \mid \sigma(b^j_{i,l}) \neq \mathrm{A}^j_i\}$
2	$\{(k_{r'}, c^{\mathfrak{b}', r'+1}_{r'})\}$
3	$\{(k_i, k_{r'}) \mid \overline{\sigma}(k_i) \neq r' \wedge \mathfrak{b}'_i = 0\} \cup$ $\{(b^j_{i,l}, k_{r'}) \mid \overline{\sigma}(b^j_{i,l}) \neq r' \wedge \mathfrak{b}'_i = 0\} \cup$ $\{(b^j_{i,l}, k_{r'}) \mid \overline{\sigma}(b^j_{i,l}) \neq r' \wedge \mathfrak{b}'_{i+1} \neq j\}$
4	$\{(h^0_i, k_{\nu^n_{i+2}(\mathfrak{b}')}) \mid \overline{\sigma}(h^0_i) \neq \nu^n_{i+2}(\mathfrak{b}')\}$
5	$\{(s, k_{r'})\}$
6	$\{(d^0_i, x) \mid \sigma(d^0_i) \neq x \wedge \overline{\sigma}(d^0_i) \neq \mathfrak{b}'_{i+1}\} \cup$ $\{(d^1_i, x) \mid \sigma(d^1_i) \neq x \wedge \overline{\sigma}(d^1_i) = \mathfrak{b}'_{i+1}\}$

Phase p	Improving Switches Superset U^p_σ
1	L^1_σ
2	$L^1_\sigma \cup L^2_\sigma$
3	$U^4_\sigma \cup \{(k_i, k_z) \mid \overline{\sigma}(k_i) \notin \{z, r'\}, z \leq r' \wedge \mathfrak{b}'_i = 0\} \cup$ $\{(b^j_{i,l}, k_z) \mid \overline{\sigma}(b^j_{i,l}) \notin \{z, r'\}, z \leq r' \wedge \mathfrak{b}'_i = 0\} \cup$ $\{(b^j_{i,l}, k_z) \mid \overline{\sigma}(b^j_{i,l}) \notin \{z, r'\}, z \leq r' \wedge \mathfrak{b}'_{i+1} \neq j'\}$
4	$U^5_\sigma \cup \{(h^0_i, k_l) \mid l \leq \nu^n_{i+2}(\mathfrak{b}')\}$
5	$U^6_\sigma \cup \{(s, k_i) \mid \overline{\sigma}(s) \neq i \wedge i < r'\} \cup$ $\{(d^j_i, x) \mid \sigma(d^j_i) \neq x \wedge i < r'\}$
6	$L^1_\sigma \cup L^6_\sigma$

Table 4.22: Improving Switches (where $\mathfrak{b}' = \mathfrak{b} + 1$ and $r' = \nu^n_1(\mathfrak{b}')$)

Note that phase 1 policies do not say anything about the particular configuration of inactive or open bicycles. To specify that all bicycles are either closed or completely opened, we say that a phase 1 policy σ is an *initial phase 1* policy if $\overline{\sigma}(b^j_{i,l}) = 0$ iff $\mathfrak{b}_i = 1$ and $\mathfrak{b}_{i+1} = j$.

Next, we specify the occurrence records w.r.t. $\mathfrak{b} \in \mathcal{B}_n$ that we want to have for an initial phase 1 policy σ . As described earlier, the entries of the occurrence records essentially depend on the number of bit flips of a certain index that have happened while counting up to $\mathfrak{b}$.

More precisely, we need to be able to count the number of occurred bit settings that match a certain *scheme*, which is a description of how a certain bit configuration

should look like. Formally, a *scheme* is a set $S \subseteq (\mathbb{N} \setminus \{0\}) \times \{0,1\}$. Let $\mathfrak{b} \in \mathcal{B}$. We write $S \models \mathfrak{b}$ iff $\mathfrak{b}_i = q$ for all $(i,q) \in S$. We can now define the set of bit configurations leading to $\mathfrak{b}$ that *match* the scheme. Formally, we define the *match set* as $M(\mathfrak{b}, S) = \{\mathfrak{b}' \leq \mathfrak{b} \mid S \models \mathfrak{b}'\}$.

We are most interested in schemes that correspond to flipping the i-th bit to one, i.e. schemes that demand for every bit $j < i$ to be zero. We define the *flip set* w.r.t. an index i and an additional scheme S by $F(\mathfrak{b}, i, S) = M(\mathfrak{b}, S \cup \{(i,1)\} \cup \{(j,0) \mid 0 < j < i\})$. We drop the parameter S if $S = \emptyset$.

We use the flip set to specify two numbers. First, we define the number of *bit flips* as the cardinality of the flip set by $f(\mathfrak{b}, i, S) = |F(\mathfrak{b}, i, S)|$. Second, we compute the *maximal flip number* representation in the flip set by $g(\mathfrak{b}, i, S) = \max(\{0\} \cup \{|\mathfrak{b}'| \mid \mathfrak{b}' \in F(\mathfrak{b}, i, S)\})$.

Table 4.23 specifies the occurrence record of an initial phase 1 policy. The technical conditions for the cycle components essentially say that (1) both cycle edges attached to A_i^j differ at most by one, that (2) the addition of both edges belonging to an active unset cycle equal $|\mathfrak{b}|$, that (3) the addition of both edges belonging to an active set cycle equal the maximal flip number when the respective bit was set, and that (4) recently opened inactive cycles are in the process of catching up with $|\mathfrak{b}|$ again.

We are now ready to specify our main lemma describing the transitioning from an initial phase 1 policy corresponding to $\mathfrak{b}$ to a successor initial phase 1 policy corresponding to $\mathfrak{b}'$, complying with the respective occurrence records.

Lemma 4.84. *Let σ be an initial phase 1 policy with configuration $\mathfrak{b} < \mathbf{1}_n$. There is an initial phase 1 policy σ' with configuration $\mathfrak{b}' = \mathfrak{b} + 1$ s.t. $(\sigma, \phi^{\mathfrak{b}}) \rightsquigarrow^+ (\sigma', \phi^{\mathfrak{b}'})$.*

It follows immediately that the MDPs provided here indeed simulate a binary counter.

Results

We conclude that policy iteration with the LEAST-ENTERED rule requires exponentially many iterations on G_n.

Edge e	$(*, t)$	(s, k_r)	(h^0_*, k_r)
$\phi^{\mathfrak{b}}(e)$	0	$f(\mathfrak{b}, r)$	$f(\mathfrak{b}, r)$
Edge e	$(b^j_{i,*}, k_r)$		
$\phi^{\mathfrak{b}}(e)$	$f(\mathfrak{b}, r, \{(i,0)\}) + f(\mathfrak{b}, r, \{(i,1),(i{+}1,1{-}j)\})$		
Edge e	(k_i, k_r)	(k_i, c^j_i)	
$\phi^{\mathfrak{b}}(e)$	$f(\mathfrak{b}, r, \{(i,0)\})$	$f(\mathfrak{b}, i, \{(i{+}1, j)\})$	
Edge e	(d^j_i, s)	(d^j_i, h^j_i)	
$\phi^{\mathfrak{b}}(e)$	$f(\mathfrak{b}, i{+}1) - j \cdot \mathfrak{b}_{i+1}$	$f(\mathfrak{b}, i{+}1) - (1{-}j) \cdot \mathfrak{b}_{i+1}$	
Complicated Conditions			
$\lvert\phi^{\mathfrak{b}}(b^j_{i,0}, \mathrm{A}^j_i) - \phi^{\mathfrak{b}}(b^j_{i,1}, \mathrm{A}^j_i)\rvert \leq 1$			
$\phi^{\mathfrak{b}}(b^j_{i,0}, \mathrm{A}^j_i) + \phi^{\mathfrak{b}}(b^j_{i,1}, \mathrm{A}^j_i) = \begin{cases} g^* + 1 & \text{if } \mathfrak{b}_i = 1 \text{ and } \mathfrak{b}_{i+1} = j \\ g^* + 1 + 2 \cdot z & \text{if } \mathfrak{b}_{i+1} \neq j \\ \lvert\mathfrak{b}\rvert & \text{otherwise} \end{cases}$, where $g^* = g(\mathfrak{b}, i, \{(i{+}1, j)\})$ and $z := \lvert\mathfrak{b}\rvert - g^* - 2^{i-1} < \frac{1}{2}(\lvert\mathfrak{b}\rvert - 1 - g^*)$			

Table 4.23: Occurrence Records

Theorem 4.85. *The number of improving steps performed by* LEAST-ENTERED *policy iteration with limiting average criterion on the MDPs constructed in this section, which contain $\mathcal{O}(n^2)$ vertices and edges, is $\Omega(2^n)$.*

Obviously, we can transfer the result to MDPs with the discounted reward criterion (for large enough discount factors).

Corollary 4.86. *The number of improving steps performed by* LEAST-ENTERED *policy iteration with the discounted reward criterion on the MDPs constructed in this section is $\Omega(2^n)$.*

Since Markov decision process policy iteration corresponds immediately to the simplex algorithm for solving related linear programs, we have the following result.

Theorem 4.87. *The number of improving steps performed by* LEAST-ENTERED *simplex algorithm on the linear programs induced by the MDPs constructed in this section is $\Omega(2^n)$.*

Parity Games

We show how the lower bound graphs can be turned into a parity game to provide a lower bound for other classes of infinitary payoff games as well.

Essentially, the graph is exactly the same. Randomization nodes are replaced by player 1 controlled nodes s.t. the cycles are won by player 0. We assign low unimportant priorities to all nodes that have currently no priority.

We define the underlying graph $G_n = (V_0, V_1, E, \Omega)$ of a parity game as shown schematically in Figure 4.27. More formally:

$$\begin{aligned} V_0 &:= \{b^0_{i,0}, b^1_{i,0}, b^0_{i,1}, b^1_{i,1}, d^0_i, d^1_i, h^0_i, h^1_i, c^0_i, c^1_i \mid i \in [n]\} \cup \\ &\quad \{k_i \mid i \in [n+1]\} \cup \{t, s\} \\ V_1 &:= \{\mathrm{A}^0_i, \mathrm{A}^1_i \mid i \in [n]\} \end{aligned}$$

Table 4.24 defines the edge sets and the priorities of G_n.

Node	Successors	Priority	Node	Successors	Priority
d^j_i	h^j_i, s	6	k_{n+1}	t	$2n{+}9$
A^j_i	$d^j_i, b^j_{i,0}, b^j_{i,1}$	4	k_i	$c^0_i, c^1_i, t, k_{[1;n]}$	$2i{+}7$
$b^j_{i,*}$	$t, \mathrm{A}^j_i, k_{[1;n]}$	3	h^0_i	$t, k_{[i+2;n]}$	$2i{+}8$
t	t	3	h^1_i	k_{i+1}	$2i{+}8$
s	$t, k_{[1;n]}$	3	c^j_i	A^j_i	7

Table 4.24: Least Entered PG Construction

The first important observation to make is that the parity game is a sink game, which helps us to transfer our result to payoff games. The following lemma corresponds to Lemma 4.80 in the MDP world.

Lemma 4.88. *Starting with the designated initial policy, we have that G_n is a sink parity game.*

All other definitions are exactly as before. Particularly, Table 4.21 and Table 4.22 become applicable again. The following lemma has the exact same formulation as Lemma 4.83 in the MDP world.

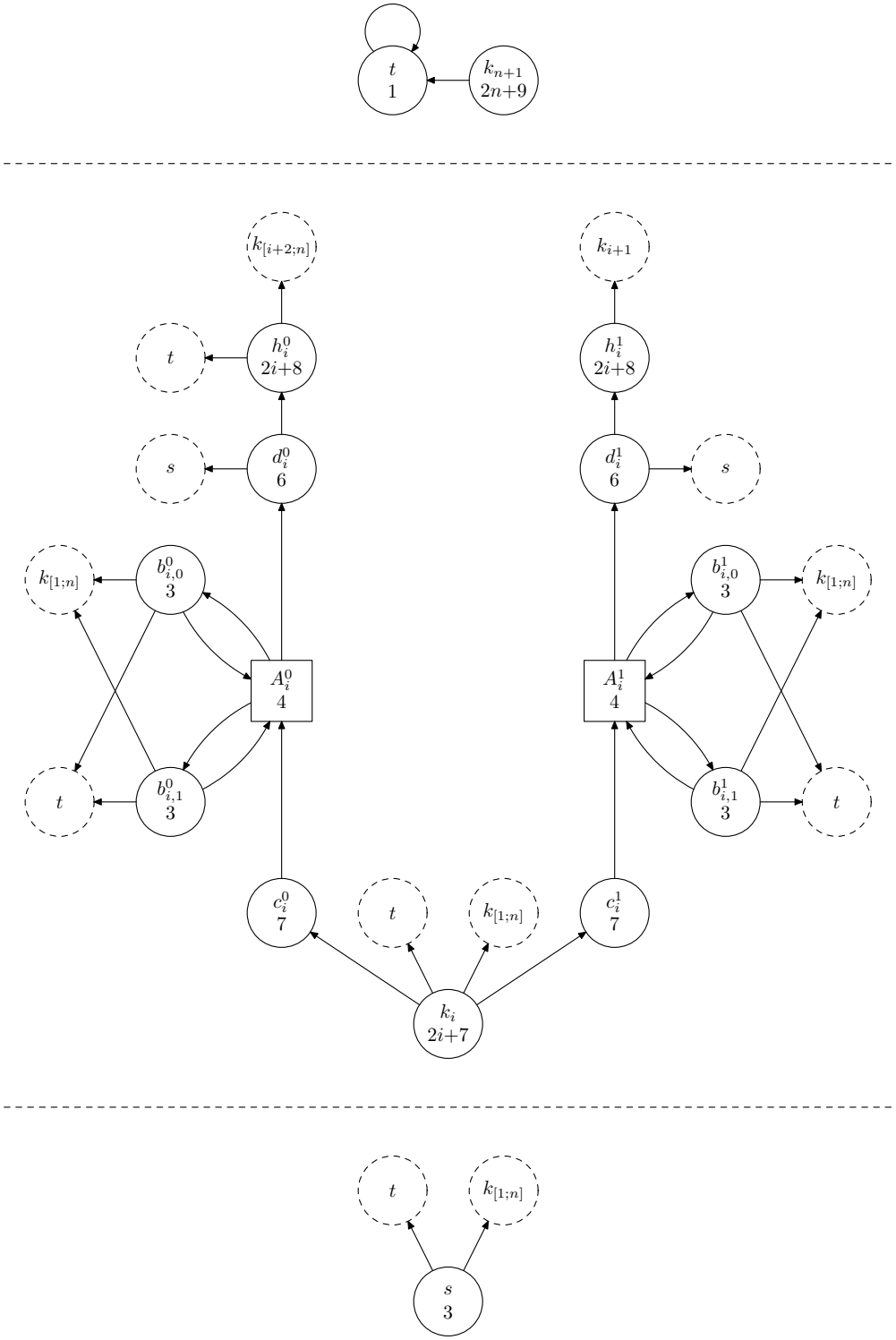


Figure 4.27: Least Entered Parity Game Construction

Lemma 4.89. *The improving switches from policies that belong to the phases in Table 4.21 are bounded by those specified in Table 4.22, i.e.* $L_\sigma^p \subseteq I_\sigma \subseteq U_\sigma^p$ *for a phase* p *policy* σ*.*

The reason why this lemma holds is that the valuations of the parity game nodes are essentially the same as the potentials in the MDP by dropping unimportant $\mathcal{O}(1)$ terms.

All other proofs rely on Table 4.21, Table 4.22 and Lemma 4.83, hence we transfer our main theorem to the parity game world.

Theorem 4.90. *Parity game strategy iteration with* LEAST-ENTERED*-rule requires at least* 2^n *improvement steps on* G_n*.*

Since G_n is a family of sink parity games, it follows directly by Theorem 4.19 that we have a subexponential lower bound for payoff games.

Corollary 4.91. *Payoff game strategy iteration with* LEAST-ENTERED*-rule requires exponential time.*

The tie-breaking rule that we employed to prove the lower bound was non-explicit and definitely not a natural one. It would be interesting to see, whether it is easily possible to transform the MDPs presented here, in order to obtain an exponential lower bound for Zadeh's rule with a natural tie-breaking rule.

5

Lower Bounds for Other Methods

We present a rigorous treatment of the remaining two other methods for solving parity games, for which no lower bounds have been known, namely the *recursive algorithm* and the *model checking algorithm*.

Our Contribution

We show that the recursive algorithm due to Zielonka [Zie98] and the model checking algorithm by Stevens and Stirling [SS98] for solving parity games require exponential time in the worst case by giving (different) explicit constructions of parity games on which they perform badly.

5.1 Recursive Algorithm

The most natural approach to solve parity games is a recursive decomposition. Indeed, there is the so-called *recursive algorithm* that is induced by the constructive proof of memoryless determinacy by Zielonka [Zie98].

It decomposes the game at hand to smaller ones recursively by simultaneous induction on the number of priorities and the number of nodes in the game. In the base case, if the game is empty, the empty winning sets can be directly obtained. In the other cases winning sets and winning strategies can be assembled out of winning sets and strategies for smaller subgames and an attractor strategy for one of the players reaching the set of nodes with maximal priority in the game.

The Algorithm

Recall that the i-attractor of a set $U \subseteq V$ is the least set W s.t. $U \subseteq W$ and whenever $v \in V_i$ and $vE \cap W \neq \emptyset$, or $v \in V_{1-i}$ and $vE \subseteq W$, then $v \in W$. Hence, the i-attractor of U contains all nodes from which player i can move "towards" U and player $1-i$ must move "towards" U. The i-attractor of U is denoted by $Attr_i(G, U)$.

Given an arbitrary attractor set A, we define the game $G \setminus A$ as the game restricted to the nodes $V \setminus A$, formally:

$$G \setminus A := (V \setminus A, V_0 \setminus A, V_1 \setminus A, E \setminus (A \times V \cup V \times A), \Omega \mid_{V \setminus A})$$

Note again that A being an attractor ensures the required totality of $G \setminus A$.

The algorithm is based on the observation that higher priorities in a parity game dominate all lower priorities, no matter how many there are. Let p be the highest priority occurring in the game G, let U be a non-empty set of nodes with priority p and let i be the parity of p. Now remove the i-attractor A of U and consider the so obtained subgame G'.

If player i wins the whole game G', then i also wins the whole game G: whenever player $1-i$ decides to visit A, player i's winning strategy would be to reach U. Then every play that visits A infinitely often has p as the highest priority occurring infinitely often, or otherwise it stays eventually in G' and hence is won by i.

Otherwise, if player i does not win G' completely, i.e. player $1-i$ wins a non-empty subset W'_{1-i}, we know player $1-i$ also wins on W'_{1-i} w.r.t. G, because player i cannot force player $1-i$ to leave W'_{1-i}. Hence, we compute the $1-i$-attractor B of W'_{1-i} w.r.t. G, remove it as safe winning region for $1-i$ from the game and recursively solve the subgame $G \setminus B$.

The algorithm therefore can be specified as follows. In the original version of the algorithm, the non-empty subset U of nodes with priority p is the whole set of nodes with priority p. The freedom of choosing U can be seen as a rule for the recursive algorithm. Nevertheless, there is no indication of any benefits for practical solving as well as for the analysis of the lower bound by choosing a proper subset here. See Algorithm 13 for a pseudo-code specification.

Algorithm 13 Recursive Algorithm

procedure SOLVE(G)
  if $V_G = \emptyset$ **then**
    $(W_0, \sigma_0) \leftarrow (\emptyset, \bot)$
    $(W_1, \sigma_1) \leftarrow (\emptyset, \bot)$
    return (W_0, σ_0), (W_1, σ_1)
  else
    $p \leftarrow \max\{\Omega_G(v) \mid v \in V_G\}$
    $i \leftarrow p \mod 2$
    $U \leftarrow$ non-empty subset of $\{v \in V_G \mid \Omega_G(v) = p\}$
    $\tau \leftarrow$ arbitrary strategy for player i on U
    $(A, \tau') \leftarrow Attr_i(G, U)$
    $(W'_0, \sigma'_0), (W'_1, \sigma'_1) \leftarrow$ SOLVE($G \setminus A$)
    if $W'_{1-i} = \emptyset$ **then**
      $(W_i, \sigma_i) \leftarrow (V_i, \sigma'_i \cup \tau \cup \tau')$
      $(W_{1-i}, \sigma_{1-i}) \leftarrow (\emptyset, \bot)$
      return (W_0, σ_0), (W_1, σ_1)
    else
      $(B, \varrho) \leftarrow Attr_{1-i}(G, W'_{1-i})$
      $(W''_0, \sigma''_0), (W''_1, \sigma''_1) \leftarrow$ SOLVE($G \setminus B$)
      $(W_i, \sigma_i) \leftarrow (W''_i, \sigma''_i)$
      $(W_{1-i}, \sigma_{1-i}) \leftarrow (W''_{1-i} \cup B, \sigma''_{1-i} \cup \varrho \cup \sigma'_{1-i})$
      return (W_0, σ_0), (W_1, σ_1)
    end if
  end if
end procedure

It is not hard to see that this algorithm is sound. Note that the correctness also implies that there are always positional winning strategies for parity games.

Theorem 5.1 ([Zie98])**.** *Let G be a parity game.* SOLVE*(G) terminates and returns the winning sets with positional winning strategies for both players.*

Proof. Let $G = (V, V_0, V_1, E, \Omega)$. We prove the claim by an induction on the number of nodes $|V|$. If G is empty, the algorithm obviously terminates and returns the correct winning sets and strategies.

For the the induction step, let $|V| > 0$. Let $p = \max\{\Omega(v) \mid v \in V\}$, $i = p \mod 2$, U be a non-empty subset of $\{v \in V \mid \Omega(v) = p\}$, τ be an arbitrary strategy for player i on U and $(A, \tau') = Attr_i(G, U)$.

Consider the game $G' = G \setminus A$. Obviously, $|V_{G'}| < |V|$. By induction hypothesis, it follows that SOLVE(G') terminates and that it returns winning sets W'_0, W'_1 as well as positional winning strategies σ'_0, σ'_1 for G'.

If now $W'_{1-i} = \emptyset$, let $\sigma_i = \sigma'_i \cup \tau \cup \tau'$. We claim that player i indeed wins on V_i following strategy σ_i. Let π be a σ_i-conforming play in G; we distinguish whether π eventually stays in W'_i. If that is the case, it is obviously won by player i, because σ'_i is a winning strategy for i on W'_i by assumption. Otherwise, if π visits A infinitely often, then player i enforces infinitely many visits to U as well by the attractor strategy τ'. Hence, the highest priority occurring infinitely often is p. Therefore, SOLVE(G) terminates and returns the winning sets with positional winning strategies for both players.

Otherwise, if $W'_{1-i} \neq \emptyset$, let $(B, \varrho) = Attr_{1-i}(G, W'_{1-i})$. Consider the game $G'' = G \setminus B$. Obviously, $|V_{G''}| < |V|$. By induction hypothesis, it follows that SOLVE(G'') terminates and that it returns winning sets W''_0, W''_1 as well as positional winning strategies σ''_0, σ''_1 for G''.

First, we argue that σ''_i is still a winning strategy on W''_i in the game G for player i. Let π be a σ''_i-conforming play in G starting from a node in W''_i. It is impossible for player $1-i$ to escape from W''_i, as escaping directly to W''_{1-i} is a contradiction to W''_i being the i-winning set in G'' and escaping directly to B is impossible with B being an $1-i$-attractor.

Second, we argue that $\sigma_{1-i} = \sigma''_{1-i} \cup \varrho \cup \sigma'_{1-i}$ is a winning strategy on $W''_{1-i} \cup B$ for player $1-i$. Let π be a σ_{1-i}-conforming play in G; we distinguish whether π eventually stays in W''_{1-i}. If that is the case, it is obviously won by player $1-i$, because σ''_{1-i} is a winning strategy for $1-i$ on W''_{1-i} by assumption. Otherwise, if π visits B infinitely often, player $1-i$ enforces infinitely many visits to W'_{1-i} as well by the attractor strategy τ'; π even stays in W'_{1-i}, because W'_{1-i} is the $1-i$-winning set w.r.t. G' and player i cannot enforce a visit to A with A being an i-attractor in G. Hence, W'_{1-i} is won by player $1-i$ by assumption. Therefore, SOLVE(G) terminates and returns the winning sets with positional winning strategies for both players. $\square$

Exponential Upper Bound

For the analysis of the runtime complexity, let $\texttt{Rec}(G)$ denote the total number of SOLVE-calls that are executed in order to solve a given parity game G. We fix the U-selection rule now that chooses the whole set of nodes with priority p.

A non-trivial upper bound can be easily derived: due to the fact that the number of different priorities is strictly reduced in the first recursive call and the number of nodes is strictly reduced in both recursive calls, the following recurrence $f(n, p)$, where n is an upper bound on the number of nodes and p is an upper bound on the number of different priorities, obviously describes an upper bound on the number of iterations that are required to solve a game with at most p different priorities and at most n nodes.

$$f(0, 0) = 1$$

$$f(n + 1, p + 1) \leq f(n, p) + f(n, p + 1)$$

Since $f(n, p) = n^p$ satisfies the recurrence, this yields the upper bound n^p. Similarly, it would be possible to derive that for arbitrary selection policies 2^n would be an upper bound on the number of iterations.

Theorem 5.2. *Let G be a parity game. Then $\texttt{Rec}(G) \in \mathcal{O}(|V_G|^{ind(G)})$ w.r.t. the whole-set rule and $\texttt{Rec}(G) \in \mathcal{O}(2^{|V_G|})$ for arbitrary policies.*

Exponential Lower Bound

We provide a concrete exponential lower bound on the number of iterations required by the recursive algorithm to solve parity games. The construction will yield the lower bound independently of the actual rule, because every "important" node in our family of games will have a different priority.

The games will be denoted by $G_n = (V_n, V_{n,0}, V_{n,1}, E_n, \Omega_n)$ and are of linear size. The sets of nodes are

$$V_n := \{a_1, \ldots, a_n, b_1, \ldots, b_n, c_0, \ldots, c_{n-1}, d_0, \ldots, d_{n-1}, e_0, \ldots, e_{n-1}\}$$

The players, priorities and edges are described in Table 5.1. The game G_3 is depicted in Figure 5.1; recall that nodes owned by player 0 are drawn as circles and nodes owned by player 1 are drawn as rectangles.

Node	Player	Priority	Successors
a_i	$1-(i \mod 2)$	$1-(i \mod 2)$	$\{b_i, d_{i-1}\}$
b_i	$i \mod 2$	$1-(i \mod 2)$	$\{a_i\} \cup (\{c_i\} \cap V_n)$
c_i	$1-(i \mod 2)$	$3i+5$	$\{b_{i+1}, d_i\}$
d_i	$i \mod 2$	$3i+4$	$\{e_i\} \cup (\{d_{i-1}, d_{i+1}\} \cap V_n)$
e_i	$1-(i \mod 2)$	$3i+3$	$\{b_{i+1}, d_i\}$

Table 5.1: The Recursive Lower Bound Game G_n

Fact 5.3. *The game G_n has $5 \cdot n$ nodes, $11 \cdot n - 3$ edges and $3 \cdot n + 2$ as the highest priority. In particular, $|G_n| = \mathcal{O}(n)$.*

Basically, solving the game G_n, requires G_{n-1} to be solved within the first recursive descent and G_{n-2} to be solved within the second recursive descent. Therefore, the number of recursion steps can be described by the *Fibonacci sequence*.

The Fibonacci sequence is a function $\mathit{fib} : \mathbb{N} \to \mathbb{N}$ which is recursively defined as follows:

$$\begin{aligned} \mathit{fib}(0) &= 0 \\ \mathit{fib}(1) &= 1 \\ \mathit{fib}(n+2) &= \mathit{fib}(n+1) + \mathit{fib}(n) \quad \text{for all } n \end{aligned}$$

It is a well-known fact that $\mathit{fib} \in \Omega\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$, and since $\frac{1+\sqrt{5}}{2} > 1$, this particularly implies that the Fibonacci sequence has exponential asymptotic behavior.

Lemma 5.4. *The game G_n is completely won by player $1-(n \mod 2)$.*

Proof. By induction on n. It is easy to see that G_1 is won by player 0 and G_2 is won by player 1. Let now $n > 2$ and $i = 1-(n \mod 2)$. We know by induction hypothesis that G_{n-2} is won by i.

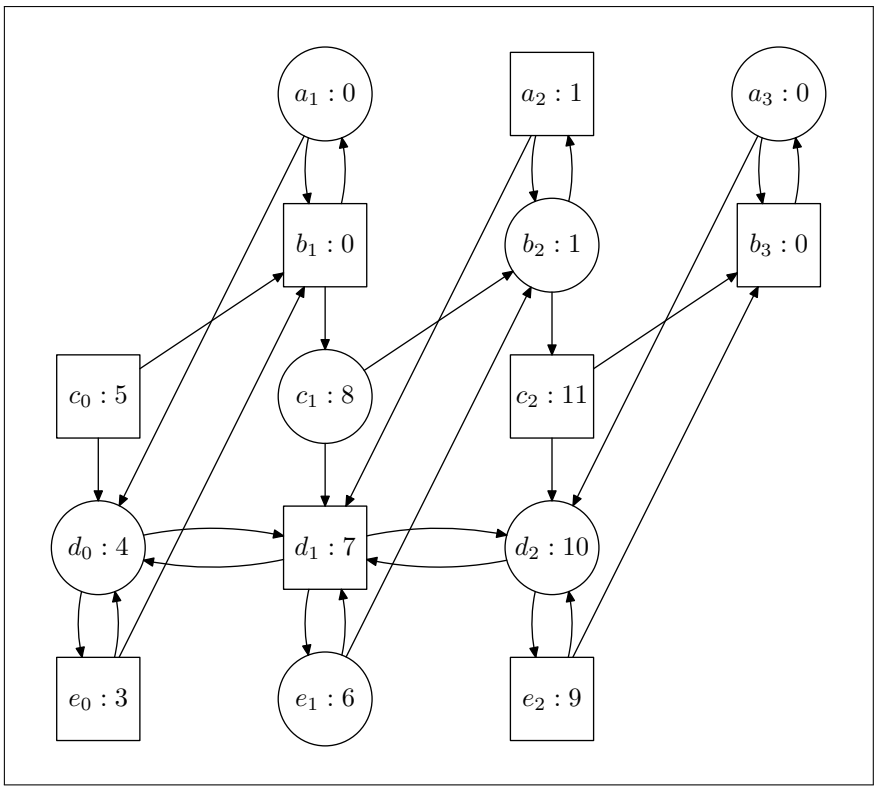


Figure 5.1: The Recursive Lower Bound Game G_3

Now attach the following strategy to the winning strategy for i on G_{n-2}: $\sigma(a_n) = b_n$, $\sigma(b_{n-1}) = c_{n-1}$, $\sigma(d_{n-1}) = e_{n-1}$ and $\sigma(c_{n-2}) = \sigma(e_{n-2}) = b_{n-1}$.

It is easy to see that a_n and b_n are won by player i. Hence, b_{n-1}, c_{n-1}, d_{n-1}, e_{n-1}, c_{n-2} and e_{n-2} are won by player i.

Therefore $G_{n-2} \subseteq G_n$ is still won by player i, as moving to c_{n-2} from nodes in G_{n-2} results in a win of player i. Hence, also a_{n-1} and d_{n-2} are won by player i. □

We will now show that solving G_n requires at least $fib(n)$ many iterations, which directly implies that the recursive algorithm requires exponentially many iterations on the family $(G_i)_{i>0}$.

Theorem 5.5. *For all $n > 0$, it holds that* $\texttt{Rec}(G_n) \geq fib(n)$.

Proof. By induction on n. For $n = 1, 2$ this is certainly true. For $n > 2$, we have to show that the solving computation w.r.t. G_n finally requires G_{n-1} and G_{n-2} to be solved in independent subcomputations:

The highest priority in G_n is $p = 3n+2$, solely due to $U = \{c_{n-1}\}$, and its parity is $i := n \mod 2$. The i-attractor of U is $A = U$, because the only node leading into c_{n-1} — namely b_{n-1} — is owned by $1-i$ and has more than one edge.

Let $G'_n = G_n \setminus A$. We will now show that sub-solving G'_n requires G_{n-1} to be solved.

- The highest priority in G'_n is $p' = 3n + 1$, solely due to $U' = \{d_{n-1}\}$, and its parity is $i' = 1 - (n \mod 2)$. The i'-attractor of U' clearly is $A' = \{a_n, b_n, d_{n-1}, e_{n-1}\}$, because the only node leading into A' — namely d_{n-2} — is owned by $1 - i'$ and has an edge not leading into A'.

  Now note that $G'_n \setminus A' = G_{n-1}$ which is to be computed next within this subcomputation.

Due to Lemma 5.4, G_{n-1} is completely won by player i, and A' is obviously won by player $1-i$; due to the fact that the only edge connecting G_{n-1} and A' in the game G_n is the edge from d_{n-2} to d_{n-1}, it is safe to conclude that solving G'_n indeed returns a partition into winning sets $W'_i = G_{n-1}$ and $W'_{1-i} = A'$.

Since W'_{1-i} is not empty, one has to compute the $1-i$-attractor of W'_{1-i} w.r.t. G_n, which is $B = A' \cup \{b_{n-1}, c_{n-1}, c_{n-2}, e_{n-2}\}$, because all nodes leading into B — namely a_{n-1}, b_{n-2} and d_{n-2} — are owned by player i and have edges not leading into B.

Let $G''_n = G_n \setminus B$. We will finally show that sub-solving G''_n requires G_{n-2} to be solved.

- The highest priority in G''_n is $p'' = 3n - 2$, solely due to $U'' = \{d_{n-2}\}$, and its parity is $i'' = n \mod 2$. The i''-attractor of U'' is $A'' = \{a_{n-1}, d_{n-2}\}$, because the only other node leading into A'' — namely d_{n-3} — is owned by $1 - i''$ and has an edge not leading into A''.

  Now note that $G''_n \setminus A'' = G_{n-2}$ which is to be computed next within this subcomputation.

□

Remarks

Although our construction establishes an exponential lower bound on the runtime complexity of the recursive algorithm, the practicability of this algorithm is generally underestimated. In fact, it is one of the best in practice [FL09]. It seems to be very difficult to adapt Zielonka's algorithm to other game classes like mean payoff games. The algorithm relies on the strong property that a higher priority dominates all lower ones, no matter how many there are, and this property is obviously wrong in general when transferred to real payoff games.

One of the best – in terms of a known upper bound on the worst-case runtime – deterministic parity game algorithms is the so-called *dominion decomposition algorithm* by Jurdziński, Paterson and Zwick [JPZ06]. It is based on the observation that the recursive algorithm performs particularly bad when there are small dominions in the given game. The dominion decomposition algorithm therefore tries to find dominions up to some size l by exhaustive search and if it finds one, it removes it, along with its attractor, from the game. Otherwise, the original recursive algorithm is invoked while performing the dominion decomposition scheme in the Recursive Calls.

The parameter l is specified in terms of the size of the given game G s.t. the resulting upper bound is as low as possible: for arbitrary parity games the authors prove (with $l = \lceil \sqrt{|V_G|} \rceil$) that the upper bound is $|V_G|^{\mathcal{O}(\sqrt{|V_G|})}$. Currently, this algorithm is one of the two deterministic procedures that solve parity games in subexponential time. The other one is Schewe's so-called *big step algorithm* [Sch07], which is based on a similar idea.

5.2 Model Checking Algorithm

Solving parity games can be divided into global and local solving: while global solvers determine for each position in the game which player can win starting from there, local solvers are additionally given one single position in the game for which it should be determined who wins starting from there.

Clearly, the local and global problem are interreducible, the global one solves the local one for free, and the global one is solved by calling the local one linearly many times. But neither of these indirect methods is particularly clever. Thus, there are algorithms for the global, and other algorithms for the local problem. Solving parity games locally particularly applies to the model checking problem of the modal μ-calculus, as one is only interested in determining who wins the position that corresponds to the initial model checking tuple, containing the root formula and the initial state of the transition system.

There are many algorithms that solve parity games globally, but surprisingly there is only one algorithm – at least to our knowledge – that is a genuine local solver, namely the one by Stevens and Stirling [SS98], to which we will refer to as the *model checking algorithm*. In fact, it is directly defined as a model checking problem in [SS98]; since μ-calculus model-checking and solving parity games are interreducible, we will study the model checking algorithm directly as a local parity game solving algorithm here.

It basically explores a game depth-first and whenever it reaches a cycle, it stops, storing the node starting the cycle along with a cycle progress measure as an *assumption* for the cycle-winning player. Then, the exploration is backtracked in the sense that if the losing player could have made other moves, they are again explored depth-first. If this leads to a cycle-win for the other player, the whole process starts again, now with respect to the other player. Whenever the backtracking finally leads to the starting node of a cycle, the node is registered as a *decision* for the player which basically can be seen as being a preliminary winning node for the respective player. Additionally, if there are assumptions of the other player for the respective node, these assumptions are dropped, and all depending assumptions and decisions are invalidated.

The Algorithm

The model checking algorithm by Stevens and Stirling essentially explores the game, starting in the initial node, depth-first until it encounters a repeat. It relies on interrelated data structures, called *decisions* and *assumptions*, that have to be

organized in a dependency ordering. Instead of managing a whole dependency graph, the authors pursue a simplified approach relying on time-stamping.

As a drawback, this simplified approach may lead to a removal of more decisions than necessary in a run of the algorithm; on the other hand, it seems to be faster in practice [SS98] and remains sound and complete. We note that our lower bound construction does not rely on the mechanism of dependency since it never happens that decisions have to be invalidated.

In order to compare the profitability of certain plays in the explored game, the algorithm introduces a structure called *index*. Essentially, the index of a play π denotes for every occurring priority p how often it occurs in π without seeing any greater priorities afterwards.

Let G be a parity game. A *G-index* is a map $i : \mathtt{ran}(\Omega_G) \to \mathbb{N}$; let π be a finite play. The *π-associated index* i_π is defined by

$$i_\pi : p \mapsto |\{j < |\pi| \mid \Omega(\pi(j)) = p \text{ and } \Omega(\pi(k)) \leq p \text{ for every } k > j\}|$$

Let $\mathbf{0}$ denote the index that maps every priority to 0. The index of a play can be calculated inductively by applying the following *priority addition function* $i \oplus p$, which takes an index i and a priority p and is defined as follows

$$(i \oplus p)(q) := \begin{cases} i(q) & \text{if } q > p \\ i(q) + 1 & \text{if } q = p \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that $i_\pi = (\ldots(\mathbf{0} \oplus \Omega(\pi(0))) \oplus \ldots) \oplus \Omega(\pi(|\pi| - 1))$.

Next, we define a total ordering w.r.t. a given player $u \in \{0, 1\}$ on indices that intuitively measures the usefulness of indices w.r.t. player u. For two indices i and j let $i >_u j$ hold iff there is some $p \in \mathtt{ran}(\Omega_G)$ s.t.

- $i(p) \neq j(p)$,
- $i(h) = j(h)$ for all $h > p$ and

- $i(p) > j(p)$ iff $p \equiv_2 u$

This p will be called the *most significant difference between i and j*. In other words, $i >_u j$ is a lexicographic ordering on indices based on the u-reward ordering on the components of the indices. In order to denote that $i >_u j$ holds with p being the most significant difference, we also write $i >_u^p j$.

By considering the lexicographic ordering on indices induced by the relevance ordering on components, it is not too hard to see that the following holds:

Corollary 5.6. *Let G be a parity game, π and π' be plays in G with $|\pi| < |\pi'|$ and $\pi(k) = \pi'(k)$ for all $k < |\pi|$. Then $i_\pi \neq i_{\pi'}$, hence either $i_\pi >_0 i_{\pi'}$ or $i_\pi >_1 i_{\pi'}$.*

Again, it is easy to see that the most significant differences between i and j, and j and k can be used to obtain the most significant difference between i and k.

Corollary 5.7. *Let G be a parity game, i, j and k be G-indices, $p \in \{0, 1\}$ be a player and w and q be priorities.*

1. *If $i <_p^w j$, $i <_p^q k$ and $q < w$ it follows that $k <_p^w j$.*
2. *If $j <_p^w i$, $k <_p^q i$ and $q < w$ it follows that $j <_p^w k$.*
3. *If $i <_p^w j$ and $j <_p^q k$ it follows that $i <_p^{\max(w,q)} k$.*

The intuition behind an index is similar to the idea of the discrete valuations in the strategy iteration: giving an abstract description of an associated play disregarding the ordering. While valuations in strategy iteration essentially include all relevant nodes in the play without regarding the ordering, an index in contrast is a window to the immediate past of a play, in the sense that smaller priorities are hidden before the last occurrence of a higher priority.

Exploring the game, the algorithm maintains a playlist storing for each node in the play the index associated with the node, which edges originating from the node remained unvisited, the time at which it was added to the playlist and if it was used as an *assumption* for one or both of the two players.

Formally, a *playlist entry* is a tuple (v, i, t, b, a) with $v \in V_G$, i a G-index, $t \subseteq vE_G$, $b \in \mathbb{N}$ and $a \subseteq \{0, 1\}$. A *playlist* is a map l with a domain $\{0, \ldots, k-1\}$ for some $k \in \mathbb{N}$ that maps each $i < k$ to a playlist entry. The length of a playlist l is denoted by $|l| := k$ and the empty playlist is denoted by $[]$. To access the i-th entry of the playlist, we write l_i.

Let l be a playlist and e be a playlist entry. Adding e to the top of l is denoted by $e :: l$ and formally defined as follows: $e :: l$ is the playlist with the domain $\{0, \ldots, |l|\}$ that maps every $i < |l|$ to l_i and $i = |l|$ to e.

When comparing two playlists l and l', we are often not interested in whether they are differing in the assumption component of one playlist entry. Hence, we define $l \equiv l'$ to hold iff $|l| = |l'|$ and for every $k < |l|$, we have that $l_k = (v, i, t, b, \_)$ implies $l'_k = (v, i, t, b, \_)$ (an occurrence of _ in a tuple is to be seen as an unbound existentially quantified variable).

A *decision* for a player p at a node v is a triple (i, t, u) where i is a G-index, $t \in \mathbb{N}$ is a time-stamp and $u \in V_G \cup \{\bot\}$ s.t. $u = \bot$ if $v \notin V_p$ and $u \in vE_G$ if $v \in V_p$. Intuitively, the decision (i, t, u) at v for p tells us that if a play reaches v with an index j that is not p-worse than i, it can be won by p if all assumptions before t actually hold true. Additionally, if v is a p-choice point, u denotes the corresponding strategy decision that should be played.

During a run of the algorithm, it happens that decisions are removed, depending on their time-stamp. For instance, if a node v was used as an assumption for p at a time t and it turns out later on that v will now be used as a decision for $1-p$, all decisions for p that have been made after t are to be removed. Hence, the algorithm maintains a set of decisions for each player and each node.

Formally, a *decision map* for player p is a map d with domain V_p that maps each node $v \in V_p$ to a set of decisions for p at v. The decision map assigning to each node the empty set is denoted by $\mathbf{e}$. A decision map d for player p induces a p-strategy σ_d which is defined on each node v with $d(v) \neq \emptyset$ as follows: $\sigma_d(v) = u$ iff there is a time t s.t. $(\_, t, u) \in d(v)$ and for all $(\_, s, \_) \in d(v)$ it holds that $s \leq t$.

Whenever the algorithm hits a repeat while exploring the game, the player p winning the cycle is determined, and the starting node of the cycle in the playlist is

marked to be used as an assumption for p. Then, the playlist is backtracked in order to determine whether player $1-p$ could have made better choices. Additionally, if exploring the game reaches a node at which a decision d for either one of the players p is *applicable*, meaning that the current index is p-greater than the decision index, the backtracking process is also invoked. EXPLORE accepts six parameters: the game G, the current node v, the current index i, the current playlist l, the time-counter c and the tuple of decision stacks d for both players. See Algorithm 14 for a pseudo-code specification. Note that the statement $i >_p j$ used in the algorithm holds for either one of the two players due to Corollary 5.6.

Algorithm 14 model checking algorithm: Explore Routine

```
1:  procedure EXPLORE(G, v, i, l, c, d)
2:      if (∃p ∈ {0,1}. ∃(j,_,_) ∈ d_p(v) : i ≥_p j) then
3:          return BACKTRACK(G, v, p, l, c + 1, d)
4:      else if (∃i < |l| : l_i = (v, j, t, b, a)) then
5:          p ← {0,1} s.t. i >_p j
6:          return BACKTRACK(G, v, p, l[i ↦ (v, j, t, b, a ∪ {p})], c + 1, d)
7:      else
8:          w ← SELECT(G, v, vE)
9:          return EXPLORE(G, w, i ⊕ Ω(w), (v, i, vE \ {w}, c, ∅) :: l, c + 1, d)
10:     end if
11: end procedure
```

The algorithm leaves the choice of selecting the next successor, that is to be visited, open: for a game G, the function SELECT chooses for every node v and for every non-empty successor subset $\emptyset \neq t \subseteq vE_G$, a node $\text{SELECT}(G, w, t) \in t$.

Backtracking passes through the playlist in reverse order until it finds a choice point of the other player that still contains unexplored edges. While backtracking, all nodes that are removed from the top of the playlist are added to the decision set of the player for whom the playlist is backtracked. Whenever adding a decision for a player p at v, the algorithm checks whether v was used as an assumption for $1-p$ and if so, all depending $1-p$ decisions are removed.

If the backtracking processes finally encounters a choice point of the other player with unexplored choices, the exploration process continues at that node. Otherwise, the algorithm finishes. BACKTRACK accepts six parameters: the game G, the current

node v, the player p for which backtracking is to be performed, the playlist l, the time-counter c and the tuple of decision stacks d for both players. See Algorithm 15 for a pseudo-code specification.

Algorithm 15 model checking algorithm: Backtrack Routine

procedure BACKTRACK(G, v, p, l, c, (d_0, d_1))
  if ($l = []$) **then**
    return (p, σ_{d_p})
  else
    $(w, i, t, b, a) :: m \leftarrow l$
    if ($w \in V_p$) **or** ($t = \emptyset$) **then**
      $u \leftarrow \begin{cases} \bot & \text{if } t = \emptyset \\ v & \text{otherwise} \end{cases}$
      $d'_p \leftarrow d_p[w \mapsto d_p(w) \cup \{(i, c, u)\}]$
      $d'_{1-p} \leftarrow \begin{cases} y \mapsto \{(j, s, z) \in d_{1-p}(y) \mid j < b\} & \text{if } (1-p) \in a \\ d_{1-p} & \text{otherwise} \end{cases}$
      return BACKTRACK(G, w, p, m, c, (d'_0, d'_1))
    else
      $u \leftarrow$ SELECT(G, w, t)
      $l' \leftarrow (w, i, t \setminus \{u\}, b, a) :: m$
      return EXPLORE(G, u, $i \oplus \Omega(u)$, l', c, (d_0, d_1))
    end if
  end if
end procedure

The model checking algorithm is then realized by the DECIDE routine that takes a game G and node v for which the winner is to be decided along with a winning strategy. It just invokes the EXPLORE-routine by initializing all required parameters with the default values. See Algorithm 16 for a pseudo-code specification.

Algorithm 16 model checking algorithm: Decide Routine

procedure DECIDE(G, v)
  return EXPLORE(G, v, $\mathbf{0} \oplus \Omega(v)$, [], 1, $(\mathbf{e}, \mathbf{e})$)
end procedure

It is easy to see that the model checking algorithm always terminates. Correctness essentially follows from the observation that the algorithm eventually explores a

winning strategy for one of the two players and every backtracking operation results in indices that are won by the respective player.

Theorem 5.8 ([SS98])**.** *Let G be a parity game and v be a node in G. Calling* DECIDE(G, v) *terminates and returns a tuple (p, σ) s.t. $v \in W_p$ and σ is a p-winning strategy starting in v.*

Exponential Upper Bound

For the analysis of the runtime complexity, let $\texttt{MC}(G, v)$ denote the total number of EXPLORE-calls that are executed in order to solve a given parity game G and initial node v.

A trivial upper bound can be easily derived: due to the fact the algorithm explores every path ending in a cycle at most once, it follows that the depth of the search tree is bounded by the number of nodes and the out-degree of every point in the search tree is bounded by the out-degree of respective node in the game. Hence, a trivial upper bound on the runtime complexity is $\mathcal{O}(n^n)$ assuming that n is the number of nodes in the game.

Theorem 5.9. *Let G be a parity game and $v \in G$. Then $\texttt{MC}(G, v) \in 2^{\mathcal{O}(n \cdot \log n)}$.*

Exponential Lower Bound

We present a concrete family of parity games on which the (expected) runtime of the model checking algorithm is exponential. Obviously, the analysis of the lower bound depends to some extent on the selection policy SELECT. We follow the most natural selection policy here that selects a successor node uniformly at random.

$$\text{SELECT}^R(G, v, R) \equiv u \in R \text{ with probability } \frac{1}{|R|}$$

Employing this selection policy, we prove that the (expected) number of steps is at least exponential on a certain family of games. Nevertheless, for other reasonable policies, it is possible to show that a lower bound on the expected number of steps is also exponential in a similar way.

The games will be denoted by $G_n = (V_n, V_{n,0}, V_{n,1}, E_n, \Omega_n)$ and are of linear size. All nodes are owned by player 1.

$$V_n := \{a_0, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n\}$$

The priorities and edges are described in Table 5.2. The game G_2 is depicted in Figure 5.2.

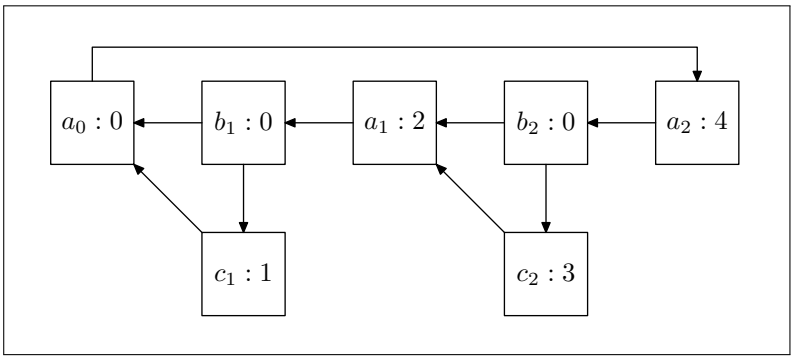


Figure 5.2: The Model Checking Lower Bound Game G_3

Node	Priority	Successors
a_0	0	$\{a_n\}$
$a_{i>0}$	$2 \cdot i$	$\{b_i\}$
b_i	0	$\{c_i, a_{i-1}\}$
c_i	$2 \cdot i - 1$	$\{a_{i-1}\}$

Table 5.2: The Model Checking Lower Bound Game G_n

Fact 5.10. *The game G_n has $3 \cdot n + 1$ nodes, $4 \cdot n + 1$ edges and $2 \cdot n$ as highest priority. In particular, $|G_n| = \mathcal{O}(n)$.*

Obviously, G_n is completely won by player 0, because every cycle eventually goes through a_n, which has the highest priority in the game and is even. The exponential behavior on these games is enforced as follows: assume for the time being that the selection policy always select a_{i-1} first and then c_i.

Exploring the game starting in a_n, assume that the process is currently at b_i choosing to advance to a_{i-1}. Eventually, a_n will be reached again and the play will be backtracked w.r.t. player 0. The backtracking process finally moves back to b_i, advancing to the unexplored c_i subsequently.

Now note that c_i has an odd priority that is greater than all other priorities occurring afterwards, i.e. $\Omega(c_i) > \Omega(q_j)$ with $j < i$ and $q_j \in \{a_j, b_j, c_j\}$. Hence, there is no applicable decision that has been added during the backtracking process. Therefore, advancing from c_i again to a_{i-1} recursively, starts the whole process again.

Otherwise, if the selection policy chooses c_i first instead of a_{i-1}, there will be an applicable decision afterwards, avoiding the second recursive descent.

Again, we analyze the runtime complexity of the model checking algorithm in terms of $\texttt{MC}(G, v)$, i.e. the time-counter that is maintained by the EXPLORE-routine. The following function f_n will be shown to capture the progression of it accurately. Let $n \in \mathbb{N}$ and $i < n$.

$$f_n : i \mapsto \begin{cases} 1 & \text{if } i = 0 \\ f_n(i-1) + 4 & \text{if } i > 0 \text{ and } \textsc{Select}(G_n, b_i, b_i E) = c_i \\ 2 \cdot f_n(i-1) + 4 & \text{otherwise} \end{cases}$$

Lemma 5.11. *Let* $n \in \mathbb{N}$. *Then* $\texttt{MC}(G_n, a_n) = f_n(n) + 1$.

See Chapter B.1 for the proof of Lemma 5.11.

Theorem 5.12. *Deciding* (G_n, a_n) *via* $\textsc{Decide}(G_n, a_n)$ *with* $\textsc{Select}^R$ *requires an expected number of* $9 \cdot 1.5^n - 7$ EXPLORE*-steps. Particularly, a lower bound on the expected worst-case runtime of the parity game model checking algorithm is* $1.5^{\Omega(n)}$.

Proof. By Lemma 5.11, it requires $f_n(n) + 1$ EXPLORE-steps to decide a_n. Since $\textsc{Select}^R$ uniformly selects edges, the expected number of steps $\bar{f}_n$ can be written as follows (with $\bar{f}_n(0) = 1$):

$$\bar{f}_n(i+1) = \frac{1}{2} \cdot (1 \cdot \bar{f}_n(i) + 4) + \frac{1}{2} \cdot (2 \cdot \bar{f}_n(i) + 4) = \frac{3}{2} \cdot \bar{f}_n(i) + 4$$

By induction on $i < n$, it directly follows that $\bar{f}_n(i) = 9 \cdot 1.5^i - 8$, hence particularly $\bar{f}_n(n) + 1 = 9 \cdot 1.5^n - 7$. □

Remarks

Although our construction establishes an exponential lower bound on the runtime complexity of the model checking algorithm, it has been the only truly local algorithm for solving parity games for a long time. Its performance in practice is a bit mixed: for problem instances with small dominions like model checking or satisfiability checking, the algorithm generally works quite well. However, its performance is pretty poor in comparison to the other algorithms when solving parity games globally [FL09].

We have recently shown that parity games can also be solved by a local variant of strategy improvement [FL10a].

6

All is well that ends well

We considered the policy iteration technique for solving infinitary payoff games and the simplex algorithm for solving linear programs in this thesis. We have shown that the correspondences between the different classes of games and linear programming can be used to transfer lower bound constructions from one domain to the other.

Particularly, we have shown that essentially all natural pivoting and improvement rules, for which until now no non-trivial bounds have been known in a concrete setting, are actually exponential or at least subexponential in the worst case, for both the simplex algorithm and the policy iteration method.

We gave exponential lower bounds for the deterministic SWITCH-ALL and SWITCH-BEST rules for solving infinitary payoff games, subexponential lower bounds for the randomized RANDOM-EDGE and RANDOM-FACET rules for solving games and linear programs, a subexponential lower bound for the randomized SWITCH-HALF rule for solving games, and finally a subexponential lower bound for Zadeh's LEAST-ENTERED rule for solving games and linear programs. All these problems have been open for several decades.

For the sake of completeness, we have shown lower bounds for two other important algorithms for solving parity games. We have proven that the recursive algorithm as well as the model checking algorithm require exponential time in the worst case.

Future Work

The most important question that obviously remains is whether parity games (or any of the other infinitary payoff two-player games) are solvable in polynomial

time. Policy iteration still seems to be the most promising candidate for giving rise to a polynomial-time procedure. However, it might be necessary to investigate non-standard improvement rules that go far beyond of what we use today.

Since parity games are the simplest class in this hierarchy of games, we think that this class should be the easiest to find a polynomial-time algorithm for. In fact, it is not too hard to show that many pivoting rules, including SWITCH-ALL, solve parity games in polynomial time, if player 1 does not appear in structures similar to the player 0 dominated cycles that we use in this thesis. We think that a rigorous analysis of these structures could help to design a pivoting rule that solves parity games efficiently.

As for the computational complexity of infinitary two-player games, it would be interesting to see whether there are deeper connections between solving the games and other NP-problems, that are neither known to be in P, nor known to be NP-complete, like the graph isomorphism problem or the factorization problem of natural numbers.

In the domain of linear programming, the most important open problems are, perhaps, whether linear programs can be solved in strongly polynomial time, whether any of the many variants of the polynomial Hirsch conjecture holds, and whether there is a polynomial-time admitting pivoting rule.

We think that it would be promising to analyze whether it is possible to extend Markov decision processes slightly in such a way that we can still reduce them to linear programming, but without being able to show that their diameter is small. This would allow us to construct a counter example to the Hirsch conjecture in the domain of games, which has been proven to be a very helpful abstraction when constructing concrete linear programs.

A

Proofs of Chapter 4

A.1 Proofs of Chapter 4.4

Lemma 4.15. *Let G be a sink parity game with v^* being the sink, H_λ be the induced discounted payoff game, λ be a large enough discount factor and σ be a player 0 strategy s.t. $\iota \trianglelefteq^G \sigma$. Let $v_0 \neq v^*$ be an arbitrary node. Then $\pi_{v_0,\sigma,\tau_\sigma^{H_\lambda}}$ is of the following form:*

$$\pi_{v_0,\sigma,\tau_\sigma^{H_\lambda}} = v_0 v_1 \dots v_{l-1}(v^*)^\omega$$

Proof. Consider the games $G' := G|_\sigma$ and $H' := H|_\sigma$ and note that $\tau_\sigma^G = \tau_\sigma^{G'}$ as well as $\tau_\sigma^{H_\lambda} = \tau_\sigma^{H'_\lambda}$. Note that G' is won by player 1 following $\tau_\sigma^{G'}$ since G is a sink parity game.

By Theorems 3.24 and 3.25 it follows that $\tau_\sigma^{H'_\lambda}$ must also be a player 1 winning strategy for the whole game G'. Therefore, it follows that every play $\pi_{v_0,\sigma,\tau_\sigma^{H'_\lambda}}$ eventually ends in a cycle with a dominating cycle node w^* of odd priority, hence $\Omega(w^*) \geq \Omega(v^*)$.

If $\Omega(w^*) > \Omega(v^*)$, it follows that there is a w^*-dominated cycle reachable in G' starting from v_0. But since $\Xi_\sigma(v_0)^G = (v^*, \_, \_)$, this cannot be the case. Hence $\Omega(w^*) = \Omega(v^*)$, implying that $w^* = v^*$. □

Lemma 4.16. *Let G be a sink parity game with v^* being the sink, H_λ be the induced discounted payoff game, λ be a large enough discount factor. Let π and ξ be two paths of the form $\pi = u_0 u_1 \dots u_{l-1}(v^*)^\omega$ and $\xi = w_0 w_1 \dots w_{k-1}(v^*)^\omega$ and let $U = \{u_0, \dots, u_{l-1}\}$ and $W = \{w_0, \dots, w_{k-1}\}$. Then $U \prec W$ implies $R_\lambda(\pi) < R_\lambda(\xi)$.*

Proof. W.l.o.g. assume that the priority assignment function is injective. Let $V = \{v_0, \ldots, v_{n-1}\}$ s.t. $p_{n-1} > p_{n-2} > \ldots > p_0$ with $p_i = \Omega(v_i)$ and $v_0 = v^*$, and let λ be the discount factor of H. W.l.o.g. assume that $n > 2$ since otherwise both paths are necessarily the same. Let $a : \{1, \ldots, n-1\} \to \{0, \ldots, n-2, \bot\}$ be a map s.t.

$$a(i) = \begin{cases} j & \text{if } u_j = v_i \\ \bot & \text{if there is no } j \text{ s.t. } u_j = v_i \end{cases}$$

and let $b : \{1, \ldots, n-1\} \to \{0, \ldots, n-2, \bot\}$ be defined accordingly for w_j. Set $\lambda^\bot := 0$. Note that the following holds:

$$R_\lambda(\pi) = \sum_{i=1}^{n-1} \lambda^{a(i)} \cdot (-n)^{p_i} + \frac{n \cdot \lambda^l}{\lambda - 1} \qquad R_\lambda(\xi) = \sum_{i=1}^{n-1} \lambda^{b(i)} \cdot (-n)^{p_i} + \frac{n \cdot \lambda^k}{\lambda - 1}$$

Let $m = \max\{i \mid (a(i) = \bot \text{ and } b(j) \neq \bot) \text{ or } (a(i) \neq \bot \text{ and } b(j) = \bot)\}$ and note that m indeed is well-defined. Set

$$\delta := R_\lambda(\xi) - R_\lambda(\pi) = \delta_1 + \delta_2 + \delta_3 + \delta_4$$

where

$$\delta_1 := \sum_{i=m+1}^{n-1} (\lambda^{b(i)} - \lambda^{a(i)}) \cdot (-n)^{p_i} \qquad \delta_2 := (\lambda^{b(m)} - \lambda^{a(m)}) \cdot (-n)^{p_m}$$

$$\delta_3 := \sum_{i=1}^{m-1} (\lambda^{b(i)} - \lambda^{a(i)}) \cdot (-n)^{p_i} \qquad \delta_4 := \frac{n \cdot (\lambda^k - \lambda^l)}{\lambda - 1}$$

Regarding δ_1, let $m < i < n$ and consider that $|\lambda^{b(i)} - \lambda^{a(i)}| \leq |1 - \lambda^{n-2}|$. The following holds:

$$|\lambda^{b(i)} - \lambda^{a(i)}| \cdot n^{p_i} \leq |1 - \lambda^{n-2}| \cdot n^{p_i} = (\sum_{j=0}^{n-3} \lambda^j) \cdot (1 - \lambda) \cdot n^{p_i}$$

$$\leq n \cdot (1 - \lambda) \cdot n^{p_i} = \frac{n^{p_i+1}}{4 \cdot n^3 \cdot n^{p_{n-1}}} \leq n^{-2}$$

We conclude that $|\delta_1| \leq \frac{n-1-m}{n^2} \leq 1$.

Regarding δ_2, note that $b(m) \neq \perp$ implies that p_m is even and $b(m) = \perp$ implies that p_m is odd. Let $c = b(m)$ iff $b(m) \neq \perp$ and $c = a(m)$ otherwise. Hence the following holds:

$$\begin{aligned}\delta_2 &= \lambda^c \cdot n^{p_m} \geq \lambda^{n-1} \cdot n^{p_m} = (\lambda^{n-1} - 1) \cdot n^{p_m} + n^{p_m} \\ &= (\sum_{j=0}^{n-2} \lambda^j) \cdot (\lambda - 1) \cdot n^{p_m} + n^{p_m} \geq (1-n) \cdot (1-\lambda) \cdot n^{p_m} + n^{p_m} \\ &= \frac{1-n}{4 \cdot n^{p_{n-1}+3}} \cdot n^{p_m} + n^{p_m} \geq \frac{3}{4} \cdot n^{p_m}\end{aligned}$$

Regarding δ_3, let $0 < i < m$ and consider that $|\lambda^{b(i)} - \lambda^{a(i)}| \leq 1$. The following holds:

$$|\delta_3| \leq \sum_{i=1}^{m-1} |\lambda^{b(i)} - \lambda^{a(i)}| \cdot n^{p_i} \leq \sum_{i=1}^{m-1} n^{p_i}$$

Now we need to distinguish on whether $p_m = 2$. If so, note that $m = 1$, $b(m) \neq \perp$ and $k = l + 1$. Hence, regarding δ_4, the following holds:

$$|\delta_4| = \frac{n \cdot |\lambda^{l+1} - \lambda^l|}{|\lambda - 1|} = n \cdot \lambda \leq n$$

Therefore we conclude (remember that $n > 2$)

$$\delta \geq \delta_2 - |\delta_1| - |\delta_3| - |\delta_4| \geq \frac{3}{4} \cdot n^2 - 1 - 0 - n > 0$$

Otherwise, if $p_m > 2$, it holds that $|\lambda^k - \lambda^l| \leq |1 - \lambda^{n-1}| \leq (n-1) \cdot (1-\lambda)$ and hence $|\delta_4| \leq n^2 - n$. Additionally, consider δ_3 again.

$$|\delta_3| \leq \sum_{i=1}^{m-1} n^{p_i} \leq \sum_{i=2}^{p_m-1} n^i = \sum_{i=0}^{p_m-1} n^i - 1 - n = \frac{n^{p_m} - 1}{n-1} - 1 - n$$

We conclude the following (remember again that $n > 2$):

$$\begin{aligned}
\delta &\geq \delta_2 - |\delta_1| - |\delta_3| - |\delta_4| \\
&\geq \frac{3}{4} \cdot n^{p_m} - 1 - \frac{n^{p_m} - 1}{n-1} + 1 + n - n^2 + n \\
&= \frac{3}{4} \cdot n^{p_m} - \frac{n^{p_m} - 1}{n-1} - (n-1)^2 + 1 \\
&\geq \frac{3}{4} \cdot n^{p_m} - \frac{n^{p_m}}{2} - (n-1)^2 + 1 \\
&= \frac{1}{4} \cdot n^{p_m} - (n-1)^2 + 1 > 0
\end{aligned}$$

□

A.2 Proofs of Chapter 4.6

Lemma 4.35. *Let σ be a $\mathfrak{b}$-phase 1 strategy with $ind(\sigma) < 2\mu_1^n(\mathfrak{b}) + 2$. Then σ' is a $\mathfrak{b}$-phase 1 strategy with $ind(\sigma') = ind(\sigma) + 1$, and if $ind(\sigma) > 1$, then $\sigma'(d_{\mu_1^n(\mathfrak{b})}) = ind(\sigma) - 1$.*

Proof. Let σ be a $\mathfrak{b}$-phase 1 strategy, $\Xi := \Xi_\sigma$ and $ind(\sigma) < 2\mu_1^n(\mathfrak{b}) + 2$.

We first compute the valuations for all those nodes directly that do not involve any complicated strategy decision of player 1. Obviously, $\Xi(x) = \emptyset$. By Lemma 4.27(1) we know that for all set bits i (i.e. $\mathfrak{b}_i = 1$) we have the following.

$$\Xi(e_i) = \{e_i\} \cup \Xi(h_i) \quad \Xi(d_i) = \{e_i, d_i\} \cup \Xi(h_i) \quad \Xi(f_i) = \{e_i, f_i\} \cup \Xi(h_i)$$

Using these equations, we are able to compute many other valuations that do not involve any complicated strategy decision of player 1. Let $U_j = \{g_j, f_j, e_j, h_j, k_j\}$. The following holds (by $\mathtt{CF}_p(A)$ we denote the *characteristic function* that returns A if p holds and $\emptyset$ otherwise):

$$\Xi(k_i) = \{k_i\} \cup \bigcup\{U_j \mid j{>}i, \mathfrak{b}_j{=}1\} \qquad \Xi(h_i) = \{h_i, k_i\} \cup \bigcup\{U_j \mid j{>}i, \mathfrak{b}_j{=}1\}$$

$$\Xi(g_i) = \{g_i, k_i\} \cup \bigcup\{U_j \mid j \geq i, \mathfrak{b}_j = 1\} \quad \Xi(r) = \{r\} \cup \bigcup\{U_j \mid \mathfrak{b}_j = 1\}$$

$$\Xi(s) = \{s\} \cup \mathtt{CF}_{b \neq \mathtt{0}_n}(\bigcup\{U_j \mid \mathfrak{b}_j = 1\} \setminus \{g_{\nu_1^n(\mathfrak{b})}\})$$

$$\Xi(c) = \{c, r\} \cup \bigcup\{U_j \mid \mathfrak{b}_j = 1\}$$

$$\Xi(t_i) = \{t_i\} \cup \Xi(r) \cup \mathtt{CF}_{i<ind(\sigma)}(\{t_j \mid j < i\} \cup \{c\})$$

$$\Xi(a_i) = \{a_i\} \cup \Xi(t_i)$$

It is easy to see that we have the following orderings on the nodes specified above:

$$s \prec_\sigma r \prec_\sigma a_* \prec_\sigma h_* \tag{a}$$

By Lemma 4.27(2), it follows from (a) that $\tau_\sigma(e_i) = d_i$ for all unset bits i (i.e. $\mathfrak{b}_i = 0$), hence we are able to compute the valuations of the remaining nodes.

$$\Xi(e_i) = \{e_i\} \cup \Xi(d_i) \qquad\qquad \Xi(f_i) = \{e_i, f_i\} \cup \Xi(d_i)$$

It is easy to see that for every i with $\mathfrak{b}_i = 0$ and every j with $\mathfrak{b}_j = 1$ s.t. there is no $i < i' < j$ with $\mathfrak{b}_{i'} = 1$, the following holds:

$$f_i \prec_\sigma f_j \qquad g_i \prec_\sigma g_j \tag{b}$$

Also, for $i > j$ with $\mathfrak{b}_i = 1$ and $\mathfrak{b}_j = 1$ we have

$$f_i \prec_\sigma f_j \qquad g_i \prec_\sigma g_j \tag{c}$$

By (a) and Lemma 4.29(2) we obtain that the following holds:

$$a_{ind(\sigma)} \prec_\sigma \ldots \prec_\sigma a_{2n} \prec_\sigma a_1 \prec_\sigma \ldots \prec_\sigma a_{ind(\sigma)-1} \tag{d}$$

We are now ready to prove that σ' is of the desired form.

(1) By Lemma 4.28(1) and (a) we derive that closed cycles remain closed. By Lemma 4.32(1) we derive that closed cycles remain accessed. By (a) and Lemma 4.32(2) we derive that open cycles remain skipped.

By phase 1 condition (5), phase 1 condition (6), (d), it follows that for every j with $\mathfrak{b}_j = 0$, there is an improving node a_* for d_j. By Lemma 4.28(2), we conclude that open cycles remain open.

(2) By (a) and Lemma 4.30(2).

(3) By (b) and (c).

(4) By (b) and (c).

(5) By Lemma 4.30(2).

(6) By Lemma 4.30(2) and Lemma 4.29(2).

By Lemma 4.30(2) it follows that $ind(\sigma') = ind(\sigma) + 1$.

If $ind(\sigma) > 1$, then we have by (a) and (d) that $\sigma'(d_{\mu_1^n(\mathfrak{b})}) = ind(\sigma) - 1)$. □

Lemma 4.36. *Let σ be a $\mathfrak{b}$-phase 1 strategy with $ind(\sigma) = 2\mu_1^n(\mathfrak{b}) + 2$ and $\sigma(d_{\mu_1^n(\mathfrak{b})}) = ind(\sigma)$. Then σ' is a $\mathfrak{b}$-phase 2 strategy.*

Proof. This can be shown essentially the same way as Lemma 4.35. The only difference now is that $d_{\mu_1^n(\mathfrak{b})}$ has no more improving switches to the deceleration lane and hence, by Lemma 4.28(3), we learn that the $\mu_1^n(\mathfrak{b})$-cycle has to close. □

Lemma 4.37. *Let σ be a $\mathfrak{b}$-phase 2 strategy. Then σ' is a $\mathfrak{b}$-phase 3 strategy.*

Proof. Again, this can be shown essentially as the previous Lemmata 4.35 and 4.36. The main difference is that now $f_i \prec_\sigma f_{\mu_1^n(\mathfrak{b})}$ for all $i \neq \mu_1^n(\mathfrak{b})$ which is why $\sigma'(s) = \mu_1^n(\mathfrak{b})$, and that by Lemma 4.32(1) we have that the $\mu_1^n(\mathfrak{b})$-th gate is σ'-accessed. □

Lemma 4.38. *Let σ be a $\mathfrak{b}$-phase 3 strategy. Then σ' is a $\mathfrak{b}$-phase 4 strategy.*

Proof. Let σ be a $\mathfrak{b}$-phase 3 strategy, $\Xi := \Xi_\sigma$ and $\mathfrak{b}' := \mathfrak{b}[\mu_1^n(\mathfrak{b}) \mapsto 1]$.

We first compute the valuations for all those nodes directly that do not involve any complicated strategy decision of player 1. Obviously, $\Xi(x) = \emptyset$. By Lemma 4.27(1) we know that for all set bits i (i.e. $\mathfrak{b}'_i = 1$) we have the following.

$$\Xi(e_i) = \{e_i\} \cup \Xi(h_i) \quad \Xi(d_i) = \{e_i, d_i\} \cup \Xi(h_i) \quad \Xi(f_i) = \{e_i, f_i\} \cup \Xi(h_i)$$

Using these equations, we are able to compute many other valuations that do not involve any complicated strategy decision of player 1. Let $U_j = \{g_j, f_j, e_j, h_j, k_j\}$.

$$\begin{aligned}
\Xi(k_i) &= \{k_i\} \cup \bigcup \{U_j \mid j{>}i, \mathfrak{b}_j{=}1\} \\
\Xi(h_i) &= \{h_i, k_i\} \cup \bigcup \{U_j \mid j{>}i, \mathfrak{b}_j{=}1\} \\
\Xi(g_i) &= \{g_i, k_i\} \cup \bigcup \{U_j \mid j \geq i, \mathfrak{b}_j{=}1\} \cup \mathtt{CF}_{i=\mu_1^n(\mathfrak{b})} U_i \\
\Xi(r) &= \{r\} \cup \bigcup \{U_j \mid \mathfrak{b}_j{=}1\} \\
\Xi(s) &= \{s\} \cup \bigcup \{U_j \mid j{\geq}\mu_1^n(\mathfrak{b}), \mathfrak{b}'_j{=}1\} \setminus \{g_{\mu_1^n(\mathfrak{b})}\} \\
\Xi(c) &= \{c, r\} \cup \bigcup \{U_j \mid \mathfrak{b}_j = 1\} \\
\Xi(t_i) &= \{t_i\} \cup \Xi(r) \cup \mathtt{CF}_{i<ind(\sigma)}(\{t_j \mid j < i\} \cup \{c\}) \\
\Xi(a_i) &= \{a_i\} \cup \Xi(t_i)
\end{aligned}$$

We have the following orderings on the nodes specified above:

$$r \prec_\sigma a_* \prec_\sigma h_{*<\mu_1^n(\mathfrak{b})} \prec_\sigma s \prec_\sigma h_{*\geq\mu_1^n(\mathfrak{b})} \tag{a}$$

Note that the last inequality $s \prec_\sigma h_{i\geq\mu_1^n(\mathfrak{b})}$ holds for the following reason: If i corresponds to a set bit, then the path from s eventually reaches the node h_i, but the highest priority on the way to h_i is f_i, which is odd. If i on the other hand corresponds to an unset bit, then path from s to the sink shares the common postfix with h_i, which starts with the node $\sigma(k_i)$. Comparing the two differing prefixes shows that the most significant difference is h_i itself, which is even.

By Lemma 4.27(3), it follows from (a) that $\tau_\sigma(e_i) = d_i$ for all unset bits i (i.e. $\mathfrak{b}'_i = 0$), hence we are able to compute the valuations of the remaining nodes.

$$\Xi(e_i) = \{e_i\} \cup \Xi(d_i) \qquad\qquad \Xi(f_i) = \{e_i, f_i\} \cup \Xi(d_i)$$

It is easy to see that for every i with $(\mathfrak{b}+1)_i = 0$ and every j with $(\mathfrak{b}+1)_j = 1$ s.t. there is no $i < i' < j$ with $(\mathfrak{b}+1)_{i'} = 1$, the following holds:

$$f_i \prec_\sigma f_j \qquad g_i \prec_\sigma g_j \tag{b}$$

Also, for $i > j$ with $(\mathfrak{b}+1)_i = 1$ and $(\mathfrak{b}+1)_j = 1$ we have

$$f_i \prec_\sigma f_j \qquad g_i \prec_\sigma g_j \tag{c}$$

We are now ready to prove that σ' is of the desired form.

(1) By Lemma 4.28(1) and (a) we derive that closed cycles with index $i \geq \mu_1^n(\mathfrak{b})$ remain closed. By Lemma 4.28(4) and (a) we derive that closed cycles with index $i < \mu_1^n(\mathfrak{b})$ open. By Lemma 4.32(1) we derive that closed cycles remain accessed. By (a) and Lemma 4.32(2) we derive that open cycles remain skipped.

By phase 3 condition (5) and (a), it follows that for every j with $\mathfrak{b}_j = 0$, there is the improving node s for d_j. By Lemma 4.28(2), we conclude that open cycles remain open.

(2) By (a) and Lemma 4.30(1).

(3) By (b) and (c).

(4) By (b) and (c).

(5) By Lemma 4.30(1).

(6) By (a) and Lemma 4.28(2).

□

Lemma 4.39. *Let σ be a $\mathfrak{b}$-phase 4 strategy and $\mathfrak{b}+1 \neq \mathbf{1}_n$. Then σ' is a $\mathfrak{b}+1$-phase 1 strategy with $ind(\sigma') = 0$.*

Proof. Let σ be a $\mathfrak{b}$-phase 4 strategy, $\Xi := \Xi_\sigma$ and $\mathfrak{b}' = \mathfrak{b} + 1$.

We first compute the valuations for all those nodes directly that do not involve any complicated strategy decision of player 1. Obviously, $\Xi(x) = \emptyset$. By Lemma 4.27(1) we know that for all set bits i (i.e. $\mathfrak{b}'_i = 1$) we have the following.

$$\Xi(e_i) = \{e_i\} \cup \Xi(h_i) \quad \Xi(d_i) = \{e_i, d_i\} \cup \Xi(h_i) \quad \Xi(f_i) = \{e_i, f_i\} \cup \Xi(h_i)$$

Using these equations, we are able to compute many other valuations that do not involve any complicated strategy decision of player 1. Let $U_j = \{g_j, f_j, e_j, h_j, k_j\}$. The following holds:

$$\Xi(k_i) = \{k_i\} \cup \bigcup\{U_j \mid j{>}i, \mathfrak{b}'_j{=}1\} \quad \Xi(h_i) = \{h_i, k_i\} \cup \bigcup\{U_j \mid j{>}i, \mathfrak{b}'_j{=}1\}$$

$$\Xi(r) = \{r\} \cup \bigcup\{U_j \mid \mathfrak{b}'_j = 1\} \qquad \Xi(s) = \{s\} \cup (\bigcup\{U_j \mid \mathfrak{b}'_j = 1\} \setminus \{g_{\mu_1^n(\mathfrak{b})}\})$$

$$\Xi(c) = \{c, s\} \cup \bigcup\{U_j \mid \mathfrak{b}'_j = 1\} \qquad \Xi(t_i) = \{t_i\} \cup \Xi(s)$$

$$\Xi(a_i) = \{a_i, t_i\} \cup \Xi(s)$$

Additionally for all $i \geq \mu_1^n(\mathfrak{b})$, we have:

$$\Xi(g_i) = \{g_i, k_i\} \cup \bigcup\{U_j \mid j \geq i, \mathfrak{b}'_j = 1\}$$

It is easy to see that we have the following orderings on the nodes specified above.

$$s \prec_\sigma a_* \prec_\sigma r \prec_\sigma h_* \tag{a}$$

By Lemma 4.27(2), it follows from (a) that $\tau_\sigma(e_i) = d_i$ for all unset bits i (i.e. $\mathfrak{b}'_i = 0$), hence we are able to compute the valuations of the remaining nodes.

$$\Xi(d_i) = \{d_i\} \cup \Xi(s) \quad \Xi(e_i) = \{e_i, d_i\} \cup \Xi(s) \quad \Xi(f_i) = \{f_i, e_i, d_i\} \cup \Xi(s)$$

Additionally for all $i < \mu_1^n(\mathfrak{b})$, we have:

$$\Xi(g_i) = \{g_i, f_i, e_i, d_i\} \cup \Xi(s)$$

This completes the valuation of Ξ for all nodes.

It is easy to see that for every i with $\mathfrak{b}'_i = 0$ and every j with $\mathfrak{b}'_j = 1$ s.t. there is no $i < i' < j$ with $\mathfrak{b}'_{i'} = 1$, the following holds:

$$f_i \prec_\sigma f_j \qquad g_i \prec_\sigma g_j \tag{b}$$

Similarly, for $i > j$ with $\mathfrak{b}'_i = 1$ and $\mathfrak{b}'_j = 1$ we have

$$f_i \prec_\sigma f_j \qquad g_i \prec_\sigma g_j \tag{c}$$

We are now ready to prove that σ' is of the desired form.

(1) By Lemma 4.28(1) and (a) we derive that closed cycles remain closed. By Lemma 4.32(1) we derive that closed cycles remain accessed. By (a) and Lemma 4.32(2) we derive that open cycles remain or will be skipped.

By Lemma 4.28(2) and (a), we conclude that open cycles remain open.

(2) By (a) and Lemma 4.30(2).

(3) By (b) and (c).

(4) By (b) and (c).

(5) By Lemma 4.30(1) it follows that $ind(\sigma') = 1$.

(6) By (a) it follows that $\sigma'(d_i) = r$ for every i with $\mathfrak{b}'_i = 0$.

□

A.3 Proofs of Chapter 4.7

Lemma 4.52. *Let G be a game, $F \subseteq E_0$ and $\sigma \subseteq F$ be a player 0 strategy. Then $\mathbb{E}(G_F, \sigma) = \mathbb{E}^*(G_F, \sigma)$.*

Proof. By lexicographic induction on $(|F|, \Xi_\sigma)$. Let $H = G_F$. For $|F| = |\sigma|$, we clearly have $\mathbb{E}(H, \sigma) = \mathbb{E}^*(H, \sigma) = 1$. For the induction step, let $|F| > |\sigma|$. It is easy to see that

$$\mathbb{E}(H, \sigma) = \frac{1}{|F \setminus \sigma|} \sum_{e \in F \setminus \sigma} \mathbb{E}(H, e, \sigma)$$

$$\mathbb{E}(H, e, \sigma) = \mathbb{E}(H \setminus \{e\}, \sigma) + \mathbb{1}_{e \in \sigma_H} \cdot \mathbb{E}(H, \sigma_{H \setminus \{e\}}[e])$$

where $\mathbb{1}_{\texttt{pred}} \in \{0, 1\}$ denotes the *indicator function*, i.e., $\mathbb{1}_{\texttt{pred}} = 1$ iff `pred` holds.

For an index function i, let $e_i = \operatorname{argmin}_{e' \in F \setminus \sigma} i(e')$ and $H_i = H \setminus \{e_i\}$. Similarly we have the following:

$$\mathbb{E}^*(H, \sigma) = \frac{1}{|\mathcal{I}(H)|} \sum_{i \in \mathcal{I}(H)} \mathbb{E}^*(H, \sigma, i)$$

$$\mathbb{E}^*(H, \sigma, i) = \mathbb{E}^*(H_i, \sigma, i) + \mathbb{1}_{e_i \in \sigma_H} \cdot \mathbb{E}^*(H, \sigma_{H_i}[e_i], i)$$

Finally the following holds:

$$\begin{aligned}
\mathbb{E}^*(H, \sigma) =& \frac{1}{|\mathcal{I}(H)|} \sum_{i \in \mathcal{I}(H)} \mathbb{E}^*(H_i, \sigma, i) + \frac{1}{|\mathcal{I}(H)|} \sum_{i \in \mathcal{I}(H)} \mathbb{1}_{e_i \in \sigma_H} \cdot \mathbb{E}^*(H, \sigma_{H_i}[e_i], i) \\
=& \sum_{e \in F \setminus \sigma} \frac{1}{|F \setminus \sigma|} \sum_{i \in \mathcal{I}(H \setminus \{e\})} \frac{\mathbb{E}^*(H \setminus \{e\}, \sigma, i)}{|\mathcal{I}(H \setminus \{e\})|} + \\
& \sum_{e \in F \setminus \sigma} \frac{\mathbb{1}_{e \in \sigma_H}}{|F \setminus \sigma|} \sum_{i \in \mathcal{I}(H)} \frac{\mathbb{E}^*(H, \sigma_{H \setminus \{e\}}[e], i)}{|\mathcal{I}(H)|} \\
\overset{IH}{=}& \sum_{e \in F \setminus \sigma} \frac{1}{|F \setminus \sigma|} \mathbb{E}(H \setminus \{e\}, \sigma) + \sum_{e \in F \setminus \sigma} \frac{\mathbb{1}_{e \in \sigma_H}}{|F \setminus \sigma|} \mathbb{E}(H, \sigma_{H \setminus \{e\}}[e]) \\
=& \mathbb{E}(H, \sigma)
\end{aligned}$$

□

Lemma 4.54. *Let $n \in \mathbb{N}$. Then $f_n([n]) = g(n)$.*

Proof. It is not hard to see that $f_n(\emptyset, \varphi) = 1$ and $f_n(N, \varphi) = 1 + f_n(N \setminus \{i\}, \varphi) + f_n(N \cap [i], \varphi)$ for $N \neq \emptyset$ and $i = \operatorname{argmin}_{j \in N} \varphi(j)$. We also have

$$f_n(N) = \frac{1}{|\mathcal{S}(n)|} \sum_{\varphi \in \mathcal{S}(n)} f_n(N, \varphi)$$

We are now ready to show $f_n([n]) = g(n)$ by induction on n. The claim obviously holds true for $n = 0$. Let now $n > 0$ and $i = \varphi^{-1}(1)$. Then:

$$\begin{aligned}
f_n([n]) =& \frac{1}{|\mathcal{S}(n)|} \sum_{\varphi \in \mathcal{S}(n)} 1 + f_n([n] \setminus \{i\}, \varphi) + f_n([i], \varphi) \\
=& 1 + \frac{1}{|\mathcal{S}(n-1)|} \sum_{\varphi \in \mathcal{S}(n-1)} f_{n-1}([n-1], \varphi) + \\
& \frac{1}{|\mathcal{S}(n)|} \sum_{\varphi \in \mathcal{S}(n)} f_i([i], \varphi|_{\varphi([i])}) \\
=& 1 + g(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} \frac{1}{|\mathcal{S}(i)|} \sum_{\varphi \in \mathcal{S}(i)} f_i([i], \varphi) \\
=& 1 + g(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} g(i) = g(n)
\end{aligned}$$

□

Lemma A.1. *Let* $A_F^k(v) = \{v \in \Xi_{\sigma_F^*}(v) \mid \Omega(v) \geq k\}$. *Let* $F \subseteq E_0$ *be complete, then:*

$$\beta_i^*(F) = \begin{cases} 1 & \text{if } A_F^5(c_i) \succ A_F^5(b_{i+1,1}) \\ 0 & \text{if } A_F^5(c_i) \prec A_F^5(b_{i+1,1}) \end{cases}$$

$$A_F^5(\mathrm{B}_i) = \begin{cases} A_F^5(c_i) & \text{if } \beta_i^*(F) = 0 \text{ or } b_i(F) = 1 \\ A_F^5(b_{i+1,1}) & \text{otherwise} \end{cases}$$

$$A_F^5(b_{i,j}) = \begin{cases} A_F^5(c_i) & \text{if } \beta_i^*(F) = 1 \text{ and } b_i(F) = 1 \\ A_F^5(b_{i+1,1}) & \text{otherwise} \end{cases}$$

and similarly:

$$\alpha_i^*(F) = \begin{cases} 1 & \textit{if } A_F^5(\mathrm{D}_i) \succ A_F^5(b_{i+1,1}) \\ 0 & \textit{if } A_F^5(\mathrm{D}_i) \prec A_F^5(b_{i+1,1}) \end{cases}$$

$$A_F^5(\mathrm{A}_{i,j}) = \begin{cases} A_F^5(\mathrm{D}_i) & \textit{if } \alpha_i^*(F) = 0 \textit{ or } a_{i,j}(F) = 1 \\ A_F^5(b_{i+1,1}) & \textit{otherwise} \end{cases}$$

$$A_F^5(a_{i,j,k}) = \begin{cases} A_F^5(\mathrm{D}_i) & \textit{if } \alpha_i^*(F) = 1 \textit{ and } a_{i,j}(F) = 1 \\ A_F^5(b_{i+1,1}) & \textit{otherwise} \end{cases}$$

Proof. We only provide the proof for the first three relations. The proof of the other three relations is analogous.

We consider three cases. In each case we present the optimal choices and verify that they are, indeed, optimal. First, if $A_F^5(c_i) \succ A_F^5(b_{i+1,1})$ and $b_i(F) = 1$, then $\tau_F^*(\mathrm{B}_i) = c_i$ and $\beta_i^*(F) = 1$. To verify this we observe that:

$$A_F(\mathrm{B}_i) = \{\mathrm{B}_i\} \cup A_F(c_i)$$

$$A_F(b_{i,j}) = \{b_{i,j}, \mathrm{B}_i\} \cup A_F(c_i) \quad \text{for all } j \in [\ell r]$$

Hence, $A_F(c_i) \prec A_F(b_{i,j})$ for all $j \in [\ell r]$, and $A_F^5(b_{i+1,1}) \prec A_F^5(\mathrm{B}_i)$. It follows that no player can gain by changing strategy.

Next, consider the case where $A_F^5(c_i) \succ A_F^5(b_{i+1,1})$ and $\beta_i^*(F) = 0$. There exists a $j \in [\ell r]$ such that $(b_{i,j}, \mathrm{B}_i) \notin F$. Then $\tau_F^*(\mathrm{B}_i) = b_{i,j}$, and $\beta_i^*(F) = 1$. To verify this we observe that:

$$A_F(\mathrm{B}_i) = \{\mathrm{B}_i, b_{i,j}\} \cup A_F(b_{i+1,1})$$

$$A_F(b_{i,j}) = \begin{cases} \{b_{i,j}\} \cup A_F(\mathrm{B}_i) & \text{if } (b_{i,j}, \mathrm{B}_i) \in F \\ \{b_{i,j}\} \cup A_F(b_{i+1,1}) & \text{otherwise} \end{cases}$$

It again follows that no player can gain by changing strategy.

If $A^5_F(c_i) \prec A^5_F(b_{i+1,1})$, then $\tau^*_F(B_i) = c_i$ and $\beta^*_i(F) = 0$. It follows that:

$$A^5_F(\mathrm{B}_i) = A^5_F(c_i)$$
$$A^5_F(b_{i,j}) = A^5_F(b_{i+1,1}) \quad \text{for all } j \in [\ell r]$$

Hence, no player can gain by changing strategy.

□

We prove Lemma 4.55 by constructing σ^*_F and τ^*_F by backwards induction from T. The former lemma allows us to handle cycles while doing so.

Lemma 4.55. *Let $F \subseteq E_0$ be complete. Then σ^*_F is well-behaved and $\beta^*_i(F) = 1$ if and only if $i \geq reset(F)$, and $\alpha^*_i(F) = 1$ if and only if $b_i(F) = 1$ and $i \geq reset(F)$.*

Proof. We first consider the case where $i \geq reset(F)$. To prove the lemma we simply go through the vertices using induction, observing at each vertex the optimal choice and the obtained valuation. More precisely, we use backward induction on i, with induction hypothesis $A^4_F(\mathrm{A}_{i+1,j}) \preceq A^4_F(b_{i+1,1})$, for all $j \in [\ell]$, with equality for some j. For the base case, T takes the role as both b_{n+1} and $\mathrm{A}_{n+1,j}$, for all $j \in [\ell]$, and the statement is clearly correct. We then observe the following:

1. Induction hypothesis:

$$\forall j \in [\ell] : \; A^5_F(\mathrm{A}_{i+1,j}) \preceq A^5_F(b_{i+1})$$
$$\exists j' \in [\ell] : \; A^5_F(\mathrm{A}_{i+1,j'}) = A^5_F(b_{i+1})$$

2. c_i moves to $\mathrm{A}_{i+1,j'}$, where j' is defined in 1:

$$\sigma^*_F(c_i) = \mathrm{A}_{i+1,j'}$$
$$A^6_F(c_i) = \{c_i\} \cup A^6_F(b_{i+1,1})$$

3. $\beta^*_i(F) = 1$, by Lemma A.1.

4. If $b_i(F) = 1$, then:

   (a) By 2 and Lemma A.1:

$$A^6_F(\mathrm{B}_i) = \{c_i\} \cup A^6_F(b_{i+1,1})$$
$$A^6_F(b_{i,1}) = \{c_i\} \cup A^6_F(b_{i+1,1})$$

   (b) $A^6_F(\mathrm{D}_i) = \{c_i\} \cup A^6_F(b_{i+1,1})$.

   (c) $\alpha^*_i(F) = 1$, by Lemma A.1.

   (d) If $a_i(F) = 1$, then by 4a, 4b and Lemma A.1:

$$\forall j \in [\ell] : \; A^6_F(\mathrm{A}_{i,j}) \preceq A^6_F(b_{i,1})$$
$$\exists j' \in [\ell] : \; A^6_F(\mathrm{A}_{i+1,j'}) = A^6_F(b_{i,1})$$

   (e) If $a_i(F) = 0$, then by Lemma A.1:

$$\forall j \in [\ell] : \; A^6_F(\mathrm{A}_{i,j}) = A^6_F(b_{i+1,1})$$

5. If $b_i(F) = 0$, then:

   (a) By Lemma A.1:

$$A^5_F(\mathrm{B}_i) = A^5_F(b_{i+1,1})$$
$$A^5_F(b_{i,1}) = A^5_F(b_{i+1,1})$$

   (b) $A^5_F(\mathrm{D}_i) = \{\mathrm{D}_i\} \cup A^5_F(b_{i+1,1})$.

   (c) $\alpha^*_i(F) = 0$, by Lemma A.1.

   (d) $\forall j \in [\ell] : \; A^6_F(\mathrm{A}_{i,j}) = A^6_F(b_{i+1,1})$.

Note that the statement of the lemma follows from 3, 4c and 5c. For $i > reset(F)$, the induction step follows from 4d and 5d.

For $i = reset(F)$ we have $A^6_F(\mathrm{A}_{i,j}) = A^6_F(b_{i+1,1})$ and $A^6_F(b_{i,1}) = \{c_i\} \cup A^6_F(b_{i+1,1})$, from 4e and 4a, respectively. We use this as the basis for continuing using

backward induction on i, for $i < reset(F)$. In the following let $k = 2 \cdot reset(F) + 4$. We use the induction hypothesis $A^k_F(b_{i+1,1}) = \{c_{reset(F)}\} \cup A^k_F(\mathrm{A}_{i+1,j})$, for all j. We observe:

1. Induction hypothesis:

$$\forall j \in [\ell] : \; A^k_F(b_{i+1,1}) = \{c_{reset(F)}\} \cup A^k_F(\mathrm{A}_{i+1,j})$$

2. $A^k_F(c_i) = A^k_F(\mathrm{A}_{i+1,j'})$, for some j'.

3. By Lemma A.1:

$$\begin{aligned} &\beta^*_i(F) = 0 \\ &A^k_F(\mathrm{B}_i) = A^k_F(\mathrm{A}_{i+1,j'}) \\ &A^k_F(b_{i,1}) = \{c_{reset(F)}\} \cup A^k_F(\mathrm{A}_{i+1,j'}) \end{aligned}$$

4. $A^k_F(\mathrm{D}_i) = A^k_F(\mathrm{A}_{i+1,j'})$.

5. By Lemma A.1:

$$\begin{aligned} &\alpha^*_i(F) = 0 \\ &\forall j \in [\ell] : \; A^k_F(\mathrm{A}_{i,j}) = A^k_F(\mathrm{A}_{i+1,j'}) \end{aligned}$$

Note that the statement of the lemma, as well as the induction step, follows from 3 and 5.

□

Lemma 4.56. *Let $G_{n,\ell,r}$ be a lower bound game with initial strategy σ for player 0, then*

$$\mathbb{E}^*_{G_{n,\ell,r}}(E_0, \sigma | \texttt{ind} \textit{ is good}) \geq g(n).$$

Proof. Using forward induction, we first show that $N_{F,\sigma}$ and $\mathfrak{b}_{F,\sigma}(i)$, for $1 \leq i \leq n$, are updated in the same way as in a modified randomized bit-counter with n bits,

and we provide bounds for $f_{\texttt{ind}}(F, \sigma)$. We later use backward induction to combine these observations and complete the proof.

Forward induction: For the first step we use the following induction hypothesis:

- If the call RANDOM-FACET$^*(G_F, \sigma, \texttt{ind})$ performs either a count-iteration or a reset-iteration then F is complete.
- If there is no resetting bit then, for all $i \in N_{F,\sigma}$, $\mathfrak{b}_{F,\sigma}(i) = 0$.
- If the i'th bit is resetting then it is the only resetting bit, and for all $i \in N_{F,\sigma}$, $\mathfrak{b}_{F,\sigma}(i) = 1$.

This is clearly true for the first iteration, since E_0 is complete and for the initial input we have $N_{E_0,\sigma} = [n]$, and for all $1 \leq i \leq n$, $\mathfrak{b}_{E_0,\sigma}(i) = 0$.

Let F and σ be given, and let $e = \operatorname{argmin}_{e' \in F \setminus \sigma} \texttt{ind}(e')$. Any iteration is either a count-iteration, a reset-iteration or irrelevant. We consider the three cases separately.

Case 1: Assume that the iteration is a count-iteration. Then $e = (b_{i,j}, \mathtt{B}_i)$ for some i and j, and in particular $i \in N_{F,\sigma}$, since $e \notin \sigma$. By the same argument $\mathfrak{b}_{F,\sigma}(i) = 0$, and, hence, there can be no resetting bit.

Since $i \in N_{F,\sigma}$, we have $i = \operatorname{argmin}_{j \in N_{F,\sigma}} \phi_{\texttt{ind}}(j)$. During the first recursive call the i'th bit is disabled, and we get:

$$N_{F \setminus \{e\},\sigma} = N_{F,\sigma} \setminus \{i\}.$$

Let $\sigma' = \sigma^*_{F \setminus \{e\}}$. Since $F \setminus \{e\}$ is still complete, we can apply Lemma 4.55 to $F \setminus \{e\}$ as well as F, and it follows that σ' is not optimal for F. Since $reset(F \setminus \{e\}) = 0$ and $b_i(F \setminus \{e\}) = 0$ we get for $\sigma'' = \sigma'[e]$:

$$\forall j \in N_{F,\sigma} : \sigma''(b_j) = 1$$

$$\forall j \in N_{F,\sigma} \setminus \{i\} : \sigma''(a_j) = 1$$

$$\sigma''(a_i) = 0$$

Hence, for the second recursive call the i'th bit is resetting, and $N_{F,\sigma''} = N_{F,\sigma} \cap [i-1]$, and the induction step is complete.

Also, note that:

$$f_{\texttt{ind}}(F, \sigma) = 1 + f_{\texttt{ind}}(F \setminus \{e\}, \sigma) + f_{\texttt{ind}}(F, \sigma'').$$

Case 2: Assume that the iteration is a reset-iteration. Then $e = (a_{i,j,k}, \mathrm{A}_{i,j})$ for some i, j and k, and the i'th bit can neither be disabled nor inactive. If there is a resetting bit then $i \notin N_{F,\sigma}$, since $e \notin \sigma$ and $\mathfrak{b}_{F,\sigma}(i') = 1$ for all $i' \in N_{F,\sigma}$. Hence, if there is a resetting bit then i must be resetting. On the other hand, if there is no resetting bit, then $i \notin N_{F,\sigma}$ since the index function `ind` is good, and the first requirement for a good index function would imply that we instead have $e = (b_{i,j'}, \mathrm{B}_i)$, for some j'. Thus, the i'th bit must be resetting, and we have $reset(F \setminus \{e\}) = i$.

Let $\sigma' = \sigma^*_{F\setminus\{e\}}$. $F \setminus \{e\}$ remains complete, and we apply Lemma 4.55 to $F \setminus \{e\}$ as well as F, and see that σ' is not optimal for F. For $\sigma'' = \sigma'[e]$ we then get:

$$\forall j \in N_{F,\sigma} : \mathfrak{b}_{F,\sigma''}(j) = 0$$

$$\mathfrak{b}_{F,\sigma''}(i) = 1$$

Thus, $N_{F,\sigma''} = N_{F,\sigma}$ for the second recursive call, there is no resetting bit, and the induction step follows.

By ignoring contributions to the number of iterations from the first recursive call, as well as from the reset-iteration itself, we get $f_{\texttt{ind}}(F, \sigma) \geq f_{\texttt{ind}}(F, \sigma'')$. Also note that we do not need to argue that the induction hypothesis holds during the first recursive call, since in the end we are only interested in the resulting optimal strategy.

Case 3: Assume that the iteration is irrelevant. Then $N_{F\setminus\{e\},\sigma} = N_{F,\sigma}$, $\mathfrak{b}_{F\setminus\{e\},\sigma}(i) = \mathfrak{b}_{F,\sigma}(i)$, for all i, and $reset(F) = reset(F \setminus \{e\})$. Note also that either $F \setminus \{e\}$ is complete, or there are no following count-iterations or reset-iterations. This follows from the assumption that `ind` is a good index function, and the second requirement

for a good index function. If $F \setminus \{e\}$ is not complete, then $N_{F,\sigma} = \emptyset$ since the choice of e would otherwise be different. For the first recursive call σ remains the same, and the induction step follows.

We ignore contributions to the number of iterations from the second recursive call, and since we know the optimal strategy σ_F^* from Lemma 4.55, we do not need to argue that the invariants are satisfied during the second recursive call. Thus, $f_{\texttt{ind}}(F, \sigma) \geq f_{\texttt{ind}}(F \setminus \{e\}, \sigma)$.

Backward induction: Let $N \subseteq [n]$, and let ϕ be a permutation of $[n]$. Recall that the function f_n is defined as $f_n(\emptyset, \phi) = 1$, and for $N \neq \emptyset$:

$$f_n(N, \phi) = f_n(N \setminus \{i\}, \phi) + f_n(N \cap [i-1], \phi),$$

where $i = \operatorname{argmin}_{j \in N} \phi(j)$. Furthermore, $f_n(N)$ is the expected value of $f_n(N, \phi)$ when ϕ is picked uniformly at random, and from Lemma 4.54 we have $f_n([n]) = g(n)$.

Let $F \subseteq E_0$, σ and $\texttt{ind}$ be arguments for the RANDOM-FACET algorithm for some iteration, and let $e = \operatorname{argmin}_{e' \in F \setminus \sigma} \texttt{ind}(e')$ and $i = \operatorname{argmin}_{j \in N_{F,\sigma}} \phi_{\texttt{ind}}(j)$. We next show that:

$$f_{\texttt{ind}}(F, \sigma) \geq f_n(N_{F,\sigma}, \phi_{\texttt{ind}}),$$

if the iteration does not appear during the first recursive call of a reset-iteration or the second recursive call of an irrelevant iteration, i.e., if the previous induction hypothesis and case analysis is valid. The inequality is proved by backward induction.

For the basis consider the case where $F = \sigma$. Since there is no resetting edge we have $N_{F,\sigma} = \emptyset$, and we get $f_{\texttt{ind}}(F, \sigma) = f_n(N_{F,\sigma}, \phi_{\texttt{ind}}) = 1$. For the induction step we observe that:

- If the iteration is a count-iteration, then:

$$\begin{aligned}
&f_{\texttt{ind}}(F,\sigma) = \\
&1 + f_{\texttt{ind}}(F \setminus \{e\},\sigma) + f_{\texttt{ind}}(F,\sigma'') \geq \\
&1 + f_n(N_{F\setminus\{e\},\sigma},\phi_{\texttt{ind}}) + f_n(N_{F,\sigma''},\phi_{\texttt{ind}}) = \\
&1 + f_n(N_{F,\sigma} \setminus \{i\},\phi_{\texttt{ind}}) + f_n(N_{F,\sigma} \cap [i-1],\phi_{\texttt{ind}}) = \\
&f_n(N_{F,\sigma},\phi_{\texttt{ind}}).
\end{aligned}$$

- If the iteration is a reset-iteration, then:

$$f_{\texttt{ind}}(F,\sigma) \;\geq\; f_{\texttt{ind}}(F,\sigma'') \;\geq\; f_n(N_{F,\sigma''},\phi_{\texttt{ind}}) \;=\; f_n(N_{F,\sigma},\phi_{\texttt{ind}}).$$

- If the iteration is irrelevant, then:

$$f_{\texttt{ind}}(F,\sigma) \;\geq\; f_{\texttt{ind}}(F \setminus \{e\},\sigma) \;\geq\; f_n(N_{F\setminus\{e\},\sigma},\phi_{\texttt{ind}}) \;=\; f_n(N_{F,\sigma},\phi_{\texttt{ind}}).$$

Conclusion: Note that $\phi_{\texttt{ind}}$ is a random permutation of $[n]$ when $\texttt{ind}$ is picked uniformly at random from the set of good index functions that are permutations of E_0. Thus, since $f_{\texttt{ind}}(F,\sigma) \geq f_n(N_{F,\sigma},\phi_{\texttt{ind}})$ we get:

$$\mathbb{E}^*_{G_{n,\ell,r}}(F,\sigma|\texttt{ind is good}) \geq f_n(N_{F,\sigma})$$

and in particular, for the initial input:

$$\mathbb{E}^*_{G_{n,\ell,r}}(E_0,\sigma|\texttt{ind is good}) \quad\geq\quad f_n(N_{E_0,\sigma}) \quad=\quad f_n([n]) \quad=\quad g(n),$$

which concludes the proof.

□

Lemma 4.57. *Let $G_{n,\ell,r}$ be a lower bound game, and let* ind *be chosen uniformly at random from the set of permutations of E_0. Then* ind *is good with probability $p_{n,\ell,r}$, where:*

$$p_{n,\ell,r} \geq 1 - n\frac{(\ell!)^2}{(2\ell)!} - n\ell(n(2\ell r + \ell) - \ell)\frac{(r!)^2}{(2r)!}$$

Proof. The main idea of the proof is to use the following observation. Let $S = \{b_1, \ldots, b_\ell, a_1, \ldots, a_\ell\}$ and ϕ be a uniformly random permutation of S. Then:

$$Pr\big[\forall i \in [\ell]\ \forall j \in [\ell] : \phi(b_i) > \phi(a_j)\big] = \frac{(\ell!)^2}{(2\ell)!}$$

Recall that the first requirement for a good index function ind was:

1. $\forall i \in [n]\ \exists j \in [\ell]\ \exists t \in [\ell r]\ \forall k \in [r]$:

$$\texttt{ind}(b_{i,t}, \mathrm{B}_i) < \texttt{ind}(a_{i,j,k}, \mathrm{A}_{i,j})$$

We first consider the probability that a random index function ind does not satisfy the first requirement for being good.

Let $1 \leq i \leq n$ be fixed. We identify each element of $S^i = \{b^i_1, \ldots, b^i_\ell, a^i_1, \ldots, a^i_\ell\}$ with a set of r edges, such that for all $j \in [\ell]$:

$$b^i_j := \{(b_{i,r(j-1)+k}, \mathrm{B}_i) \mid k \in [r]\}$$
$$a^i_j := \{(a_{i,j,k}, \mathrm{A}_{i,j}) \mid k \in [r]\}$$

Furthermore, we define the permutation $\phi^i_{\texttt{ind}}$ of S^i such that for two distinct elements $x, y \in S^i$:

$$\phi^i_{\texttt{ind}}(x) < \phi^i_{\texttt{ind}}(y) \quad \Longleftrightarrow \quad \exists e \in x\ \forall e' \in y : \texttt{ind}(e) < \texttt{ind}(e')$$

Note that when ind is a random permutation of E_0, then $\phi^i_{\texttt{ind}}$ is a random permutation of S^i. Now ind satisfies the first requirement for being good if and only if:

$$\forall i \in [n]\ \exists j \in [\ell]\ \exists t \in [\ell] : \phi^i_{\texttt{ind}}(b^i_t) < \phi^i_{\texttt{ind}}(a^i_j)$$

It follows that the probability that the first requirement is not satisfied is at most $n\frac{(\ell!)^2}{(2\ell)!}$.

Next, we consider the probability of $\texttt{ind}$ not satisfying the second requirement. Recall that the second requirement was:

2. $\forall e \in M\ \forall i \in [n]\ \forall j \in [\ell]\ \exists k \in [r]\ \exists t \in [r]$:

$$\texttt{ind}(a_{i,j,k}, \mathrm{A}_{i,j}) < \texttt{ind}(e_t)$$

In fact, this case is simpler than the first. Define:

$$S^{e,i,j} = \{e_1, \ldots, e_r, (a_{i,j,1}, \mathrm{A}_{i,j}), \ldots, (a_{i,j,r}, \mathrm{A}_{i,j})\}$$

and define $\phi^{e,i,j}_{\texttt{ind}}$ such that for two distinct elements $x, y \in S^{e,i,j}$:

$$\phi^{e,i,j}_{\texttt{ind}}(x) < \phi^{e,i,j}_{\texttt{ind}}(y) \iff \texttt{ind}(x) < \texttt{ind}(y)$$

Again we see that when $\texttt{ind}$ is a random permutation of E_0, then $\phi^{e,i,j}_{\texttt{ind}}$ is a random permutation of $S^{e,i,j}$. Since $|M| = n(2\ell r + \ell) - \ell$, it follows that the probability that the second requirement is not satisfied is at most $n\ell(n(2\ell r + \ell) - \ell)\frac{(r!)^2}{(2r)!}$, and the statement of the lemma follows.

□

Lemma 4.60. *For every strategy σ, the MDP with underlying graph $G_{n,g,h}$ ends in the sink t with probability* 1.

Proof. Let σ be a strategy. We write $v \rightsquigarrow v'$ to denote that the MDP conforming with σ starting in v reaches v' with positive probability. Note that $v \rightsquigarrow v'$ and $v' \rightsquigarrow v''$ implies $v \rightsquigarrow v''$.

We need to show that $v \rightsquigarrow t$ for every node v. Obviously, $t, w_{n+1}, u_{n+1} \rightsquigarrow t$.

First, it is easy to see by backwards induction on $i \leq n$ that $\mathrm{A}_i \rightsquigarrow d_i \rightsquigarrow \mathrm{B}_i \rightsquigarrow y_i \rightsquigarrow w_{i+1} \rightsquigarrow t$, and hence also that $x_i \rightsquigarrow u_i \rightsquigarrow t$. We then also have $w_i, a_i, b_i \rightsquigarrow t$.

Since all vertices reach t with positive probability, and t is an absorbing state, the statement of the lemma follows. □

Lemma 4.61. *Let $F \subseteq E_0$ be complete. Then σ^*_F is well-behaved and $\beta^*_i(F) = 1$ if and only if $i \geq reset(F)$, and $\alpha^*_i(F) = 1$ if and only if $b_i(F) = 1$ and $i \geq reset(F)$.*

Proof. First, recall that from Lemma 4.60 we know that all vertices have the same value, namely $\text{VAL}_{\sigma_F}(v) = \text{VAL}_{\sigma_F}(t) = 0$, for all $v \in V$. Hence, we will focus only on potentials.

Note that except for cycling among vertices in the sets $\{\text{B}_i, b_{i,j} \mid j \in [gh]\}$ and $\{\text{A}_{i,j}, a_{i,j,k} \mid k \in [h]\}$, it is not possible to visit a vertex other than t twice. The idea of the proof is to describe σ^*_F using induction starting from t. To handle cycling we make use of the following two simple observations.

1. $\beta^*_i(F) = 1$ if and only if $\text{POT}_{\sigma^*_F}(y_i) > \text{POT}_{\sigma^*_F}(u_{i+1})$.
2. $\alpha^*_i(F) = 1$ if and only if $\text{POT}_{\sigma^*_F}(d_i) > \text{POT}_{\sigma^*_F}(x_i)$.

To verify the two observations note that edges involved in cycling have cost zero, and, hence, it is irrelevant how many times for instance B_i is visited. Furthermore, it is optimal to increase the chance of ending at y_i if and only if $\text{POT}_{\sigma^*_F}(y_i) > \text{POT}_{\sigma^*_F}(u_{i+1})$.

We split the proof into four cases:

(i) $i > reset(F)$ and $b_i(F) = 1$.

(ii) $i > reset(F)$ and $b_i(F) = 0$.

(iii) $i = reset(F)$.

(iv) $i < reset(F)$

Cases (i), (ii) and (iii) are shown jointly by backward induction on i. For the induction hypothesis we assume that $\text{POT}_{\sigma^*_F}(w_{i+1}) = \text{POT}_{\sigma^*_F}(u_{i+1})$. This clearly holds true for $i = n$. It follows that

$$\text{POT}_{\sigma^*_F}(y_i) = \langle y_i \rangle + \text{POT}_{\sigma^*_F}(w_{i+1}) > \text{POT}_{\sigma^*_F}(u_{i+1}),$$

and, hence, by observation 1., $\beta_i^*(F) = 1$.

Case (i). Assume that $i > reset(F)$ and $b_i(F) = 1$. Then

$$\begin{aligned}\textsc{pot}_{\sigma_F^*}(d_i) &= \langle d_i \rangle + \textsc{pot}_{\sigma_F^*}(\textsc{b}_i) \\ &= \langle d_i \rangle + \textsc{pot}_{\sigma_F^*}(y_i) \\ &= \langle d_i \rangle + \langle y_i \rangle + \textsc{pot}_{\sigma_F^*}(w_{i+1}) \\ &> \textsc{pot}_{\sigma_F^*}(u_{i+1})\end{aligned}$$

Hence, the optimal choice at u_i is $\sigma_F^*(u_i) = d_i$, and furthermore

$$\textsc{pot}_{\sigma_F^*}(x_i) = \langle x_i \rangle + \textsc{pot}_{\sigma_F^*}(u_i) = \langle x_i \rangle + \textsc{pot}_{\sigma_F^*}(d_i) < \textsc{pot}_{\sigma_F^*}(d_i).$$

By observation 2., it then follows that $\alpha_i^*(F) = 1$.

Since $i > reset(F)$ we have $a_i(F) = 1$, and we get that there exists a $j \in [g]$ such that:

$$\textsc{pot}_{\sigma_F^*}(\textsc{a}_{i,j}) = \textsc{pot}_{\sigma_F^*}(d_i) = \langle d_i \rangle + \langle y_i \rangle + \textsc{pot}_{\sigma_F^*}(w_{i+1}) > \textsc{pot}_{\sigma_F^*}(w_{i+1}).$$

Hence, the optimal choice at w_i is $\sigma_F^*(w_i) = \textsc{a}_{i,j}$, and furthermore:

$$\textsc{pot}_{\sigma_F^*}(w_i) = \textsc{pot}_{\sigma_F^*}(d_i) = \textsc{pot}_{\sigma_F^*}(u_i)$$

which completes the induction step.

Case (ii). Assume that $i > reset(F)$ and $b_i(F) = 0$. Then

$$\textsc{pot}_{\sigma_F^*}(d_i) = \langle d_i \rangle + \textsc{pot}_{\sigma_F^*}(\textsc{b}_i) = \langle d_i \rangle + \textsc{pot}_{\sigma_F^*}(u_{i+1}) + \mathcal{O}(1) < \textsc{pot}_{\sigma_F^*}(u_{i+1})$$

Hence, the optimal choice at u_i is $\sigma_F^*(u_i) = u_{i+1}$, and

$$\textsc{pot}_{\sigma_F^*}(x_i) = \langle x_i \rangle + \textsc{pot}_{\sigma_F^*}(u_i) = \langle x_i \rangle + \textsc{pot}_{\sigma_F^*}(u_{i+1}) > \textsc{pot}_{\sigma_F^*}(d_i).$$

By observation 2., it then follows that $\alpha_i^*(F) = 0$, and, furthermore, for all $j \in [g]$

$$\text{POT}_{\sigma_F^*}(\text{A}_{i,j}) < \text{POT}_{\sigma_F^*}(x_i) < \text{POT}_{\sigma_F^*}(u_{i+1}) = \text{POT}_{\sigma_F^*}(w_{i+1}).$$

Hence, the optimal choice at w_i is $\sigma_F^*(w_i) = w_{i+1}$, and $\text{POT}_{\sigma_F^*}(w_i) = \text{POT}_{\sigma_F^*}(w_{i+1})$, which completes the induction step.

Case (iii). Assume that $i = reset(F)$. Then $b_i(F) = 1$ and $a_i(F) = 0$. The choices and potentials at vertices y_i, B_i, $b_{i,j}$, d_i, u_i, and x_i are exactly the same as in case (i), and in particular $\alpha_i^*(F) = 1$. Since $a_i(F) = 0$ we, however, get that for all $j \in [g]$:

$$\begin{aligned}\text{POT}_{\sigma_F^*}(\text{A}_{i,j}) &= \text{POT}_{\sigma_F^*}(x_i) + \mathcal{O}(1) \\ &= \langle x_i \rangle + \langle d_i \rangle + \langle y_i \rangle + \text{POT}_{\sigma_F^*}(w_{i+1}) + \mathcal{O}(1) \\ &> \text{POT}_{\sigma_F^*}(w_{i+1})\end{aligned}$$

So, the optimal choice at w_i is $\sigma_F^*(w_i) = \text{A}_{i,j}$, where $j = \text{argmax}_{j' \in [g]} \text{POT}_{\sigma_F^*}(\text{A}_{i,j'})$, and it follows that:

$$\text{POT}_{\sigma_F^*}(w_i) = \langle x_i \rangle + \text{POT}_{\sigma_F^*}(u_i) + \mathcal{O}(1). \tag{A.1}$$

Case (iv). Assume that $i < reset(F)$. Assume, furthermore, by induction that:

$$\text{POT}_{\sigma_F^*}(w_{i+1}) = \langle x_{i+1} \rangle + \text{POT}_{\sigma_F^*}(u_{i+1}) + \mathcal{O}(1).$$

The base case follows from equation (A.1) of case (iii).

We then have

$$\text{POT}_{\sigma_F^*}(y_i) = \langle y_i \rangle + \text{POT}_{\sigma_F^*}(w_{i+1}) < \text{POT}_{\sigma_F^*}(u_{i+1}),$$

and, hence, by observation 1., $\beta_i^*(F) = 0$. Furthermore,

$$\text{POT}_{\sigma_F^*}(d_i) = \langle d_i \rangle + \text{POT}_{\sigma_F^*}(\text{B}_i) = \langle d_i \rangle + \text{POT}_{\sigma_F^*}(u_{i+1}) + \mathcal{O}(1) < \text{POT}_{\sigma_F^*}(u_{i+1})$$

Thus, the optimal choice at u_i is $\sigma^*_F(u_i) = u_{i+1}$, and

$$\text{POT}_{\sigma^*_F}(x_i) = \langle x_i \rangle + \text{POT}_{\sigma^*_F}(u_i) = \langle x_i \rangle + \text{POT}_{\sigma^*_F}(u_{i+1}) > \text{POT}_{\sigma^*_F}(d_i).$$

By observation 2., it then follows that $\alpha^*_i(F) = 0$, and, furthermore, for all $j \in [g]$:

$$\text{POT}_{\sigma^*_F}(\text{A}_{i,j}) = \text{POT}_{\sigma^*_F}(x_i) + \mathcal{O}(1) = \langle x_i \rangle + \text{POT}_{\sigma^*_F}(u_i) + \mathcal{O}(1) =$$

$$\langle x_i \rangle + \text{POT}_{\sigma^*_F}(u_{i+1}) + \mathcal{O}(1) > \text{POT}_{\sigma^*_F}(w_{i+1}).$$

Finally, we then get that the optimal choice at w_i is $\sigma^*_F(w_i) = \text{A}_{i,j}$, for some arbitrary $j \in [g]$, and it follows that:

$$\text{POT}_{\sigma^*_F}(w_i) = \langle x_i \rangle + \text{POT}_{\sigma^*_F}(u_i) + \mathcal{O}(1).$$

This completes the induction step and concludes the proof.

□

Lemma 4.64. *For every strategy σ, the MDP described by G_ζ ends in the sink t with probability* 1.

Proof. Let σ be a strategy. We write $v \rightsquigarrow v'$ to denote that the MDP conforming with σ starting in v reaches v' with positive probability. Note that $v \rightsquigarrow v'$ and $v' \rightsquigarrow v''$ implies $v \rightsquigarrow v''$.

We need to show that $v \rightsquigarrow t$ for every node v. Obviously, $t, w_{n+1}, u_{n+1} \rightsquigarrow t$.

First, it is easy to see by backwards induction on $i \leq n$ that $\text{A}_i \rightsquigarrow d_i \rightsquigarrow \text{B}_i \rightsquigarrow y_i \rightsquigarrow w_{i+1} \rightsquigarrow t$, and hence also that $w_i, u_i \rightsquigarrow t$.

Second, it follows immediately that $r, s \rightsquigarrow t$, and hence also $x_i \rightsquigarrow t$. Finally, $a_i, b_i, c_i \rightsquigarrow t$.

Since all vertices reach t with positive probability, and t is an absorbing state, the statement of the lemma follows. □

Lemma 4.66. *Let σ be a strategy belonging to one of the phases specified in Table 4.17. Then $|\text{POT}_\sigma(v)| < N^{4n+8}$ and $\varepsilon \cdot |\text{POT}_\sigma(v)| < 1$ for every node v.*

Proof. Let $\mathfrak{b}$ be a global bit state, $k = \mu_1(\mathfrak{b})$ and σ be a strategy belonging to one of the phases with global bit state $\mathfrak{b}$. Let $\mathfrak{b}' = \mathfrak{b} + 1$. Let $\delta = \delta(\sigma, k)$, $\eta = \eta(\sigma, k)$, $S_i = \sum_{j \geq i,\, \mathfrak{b}_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$, and $T_i = \sum_{j \geq i,\, \mathfrak{b}'_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$.

It suffices to show that $|\textsc{pot}_\sigma(v)| < N^{4n+8}$ for every node v. Obviously, $\textsc{pot}_\sigma(t) = 0$.

It is not too hard to see that the following holds:

$$\textsc{pot}_\sigma(s) \in [S_1; T_1] \qquad \textsc{pot}_\sigma(r) \in [S_1; T_1] + \langle r \rangle$$
$$\textsc{pot}_\sigma(w_i) \in [S_i; T_i] \qquad \textsc{pot}_\sigma(u_i) \in [S_i; T_i]$$
$$\textsc{pot}_\sigma(x_i) \in [S_i; T_i] + \langle x_i \rangle \qquad \textsc{pot}_\sigma(y_i) \in [S_{i+1}; T_{i+1}] + \langle y_i \rangle$$

We derive for all the other nodes that the following holds:

$$\textsc{pot}_\sigma(\mathrm{B}_i) \in [S_1; T_{i+1} + \langle y_i \rangle]$$
$$\textsc{pot}_\sigma(d_i) \in [S_1; T_{i+1} + \langle y_i \rangle] + \langle d_i \rangle$$
$$\textsc{pot}_\sigma(b_{i,*}) \in [S_1; T_{i+1} + \langle y_i \rangle]$$
$$\textsc{pot}_\sigma(c_{i,*}) \in [S_1; T_{i+1} + \langle y_i \rangle]$$
$$\textsc{pot}_\sigma(\mathrm{A}_i) \in [S_1 + \langle x_i \rangle; T_{i+1} + \langle y_i \rangle + \langle d_i \rangle]$$
$$\textsc{pot}_\sigma(a_{i,*}) \in [S_1 + \langle x_i \rangle; T_{i+1} + \langle y_i \rangle + \langle d_i \rangle]$$

By Lemma 4.65, we have $|\textsc{pot}_\sigma(v)| < N^{4n+8}$ for every node v. □

Lemma 4.67. *Let σ be a strategy belonging to one of the phases specified in Table 4.17.*

1. $\textsc{pot}_\sigma(d_i) < \textsc{pot}_\sigma(x_i) \Rightarrow A_i$ *opening,*
2. $\textsc{pot}_\sigma(d_i) > \textsc{pot}_\sigma(x_i)$, A_i *consecutive, not closed* $\Rightarrow A_i$ *closing,*
3. $\textsc{pot}_\sigma(s) < \textsc{pot}_\sigma(r)$, B_i *consecutive, not closed* $\Rightarrow B_i$ *closing,* C_i *opening,*
4. $\textsc{pot}_\sigma(s) < \textsc{pot}_\sigma(r) < \textsc{pot}_\sigma(y_i)$, B_i *closed,* C_i *consecutive, not closed* $\Rightarrow C_i$ *closing, and*

5. $\text{POT}_\sigma(r) < \text{POT}_\sigma(y_i) < \text{POT}_\sigma(s)$, C_i *consecutive, not closed* $\Rightarrow$ C_i *closing,* B_i *opening.*

Proof. Let σ be a strategy belonging to one of the phases specified in Table 4.17.

1. Let $\text{POT}_\sigma(d_i) < \text{POT}_\sigma(x_i)$. We need to show that A_i is opening. We consider two cases.

   If A_i is closed, let $a_{i,j}$ be an arbitrary node on the cycle. It is easy to see that $\text{POT}_\sigma(()\text{A}_i) = \text{POT}_\sigma(d_i)$ and $\text{POT}_\sigma(()a_{i,j}) = \text{POT}_\sigma(d_i)$. It follows that $(a_{i,j}, x_i)$ is an improving edge for every j.

   If A_i is not closed, let $a_{i,j}$ be an arbitrary node on the cycle. Again, we consider two cases here.

   If $j > \overline{\gamma}_i(\sigma) + 1$ it follows that $a_{i,j}$ cannot reach the node A_i via the current strategy or via switching itself. Let $l < j$ be the largest l s.t. $\sigma(a_{i,l}) = 0$. It follows that $\text{POT}_\sigma(a_{i,j-1}) = \text{POT}_\sigma(x_i) + \varepsilon l$. Computing the difference of both choices x_i and $a_{i,j-1}$ shows that switching out of the cycle is profitable.

   $$\text{POT}_\sigma(x_i) + \varepsilon j - \text{POT}_\sigma(a_{i,j-1}) = \varepsilon(j - l) > 0$$

   If $j \leq \overline{\gamma}_i(\sigma) + 1$ it follows that $a_{i,j}$ can reach the node A_i via the current strategy or via switching itself. Assume w.l.o.g. that $j > 1$ (case $j = 1$ almost the same). Let l be the largest l s.t. $\sigma(a_{i,l}) = 0$. It follows that $\text{POT}_\sigma(a_{i,j-1}) = (1-\varepsilon)\cdot(\text{POT}_\sigma(x_i)+\varepsilon l)+\varepsilon\text{POT}_\sigma(d_i)$. Computing the difference of both choices x_i and $a_{i,j-1}$ shows that switching out of the cycle is profitable.

   $$\text{POT}_\sigma(x_i) + \varepsilon j - \text{POT}_\sigma(a_{i,j-1}) = \varepsilon(j - (1-\varepsilon)l + \text{POT}_\sigma(x_i) - \text{POT}_\sigma(d_i)) > 0$$

   since $\text{POT}_\sigma(x_i) - \text{POT}_\sigma(d_i) > h$.

2. Let $\text{POT}_\sigma(d_i) > \text{POT}_\sigma(x_i)$, A_i consecutive, not closed. We need to show that A_i is closing.

   Let $l = \overline{\gamma}_i(\sigma)$. It is not hard to see that the following holds for all $j \leq l$:

   $$\text{POT}_\sigma(a_{i,j}) = (1-\varepsilon)\cdot(\text{POT}_\sigma(x_i) + \varepsilon h) + \varepsilon\text{POT}_\sigma(d_i)$$

First, we compute the difference of both choices of $a_{i,j}$ for $j \leq l+1$ to show that switching into the cycle is profitable. Assume w.l.o.g. that $j > 1$ (case $j = 1$ almost the same).

$$\text{POT}_\sigma(x_i) + \varepsilon j - \text{POT}_\sigma(a_{i,j-1}) = \varepsilon(j - (1-\varepsilon)h + \text{POT}_\sigma(x_i) - \text{POT}_\sigma(d_i)) < 0$$

since $\text{POT}_\sigma(x_i) - \text{POT}_\sigma(d_i) < h$.

Second, let $j > l+1$. As before, it is easy to see that moving out of the cycle is profitable for node $a_{i,j}$.

The other statements can be shown the same way. □

Lemma 4.68. *The improving switches from strategies that belong to the phases are exactly those specified in Table 4.17.*

Proof. Let $\mathfrak{b}$ be a global bit state, $k = \mu_1(\mathfrak{b})$ and σ be a strategy belonging to one of the phases with global bit state $\mathfrak{b}$. Let $\mathfrak{b}' = \mathfrak{b} + 1$. Let $\delta = \delta(\sigma, k)$, $\eta = \eta(\sigma, k)$, $S_i = \sum_{j \geq i,\, \mathfrak{b}_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$, $S_i^l = \sum_{l \geq j \geq i,\, \mathfrak{b}_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$, $T_i = \sum_{j \geq i,\, \mathfrak{b}'_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$, and $T_i^l = \sum_{l \geq j \geq i,\, \mathfrak{b}'_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$.

First, we apply Lemma 4.65 and compute the potentials of all important nodes, see Table A.1 for all the potentials. Second, we compute the differences between the potentials of two successors of a node to determine which edges are improving switches, see Table A.2 for all the potential differences. Third, we derive from Table A.2 that the improving switches w.r.t. w_i and u_i are exactly those specified in Table 4.17. Fourth, we apply Lemma 4.67 to derive from Table A.2 that the improving switches w.r.t. $b_{i,j}$, $c_{i,j}$, and $a_{i,j}$ are exactly those specified in Table 4.17. □

Phase	1–4	5–7
$\text{POT}_\sigma(s)$	S_1	T_1

Phase	1–4	5–7
$\text{POT}_\sigma(r)$	$S_1{+}\langle r\rangle$	$[S_1; T_1{+}\langle x_1\rangle]{+}\langle r\rangle{+}\mathcal{O}(1)$

Phase	1–4	5–7
$\text{POT}_\sigma(x_i)$	$S_1{+}\langle x_i\rangle$	$T_1{+}\langle x_i\rangle$

Phase	1–3	4		5–7
		$i \leq \delta$	$i > \delta$	
$\text{POT}_\sigma(u_i)$	S_i		T_i	

Phase	1–4	5–6	7
		$i > k$	$i > \eta$
$\text{POT}_\sigma(w_i)$	S_i		T_i

Phase	1–4	5–6	
		$i \geq k$	$i < k$
$\text{POT}_\sigma(y_i)$	$S_{i+1}{+}\langle y_i\rangle$		$[S_{i+1}; T_1{+}\langle x_{i+1}\rangle]{+}\langle y_i\rangle{+}\mathcal{O}(1)$

Phase	7			
	$i \geq \eta$	$i{+}1 = \eta = k$	$i{+}1 = \eta < k$	$i{+}1 < \eta$
$\text{POT}_\sigma(y_i)$	$T_{i+1}{+}\langle y_i\rangle$	$S_{i+1}{+}\langle y_i\rangle{+}\mathcal{O}(1)$	$T_1{+}\langle x_{i+1}\rangle{+}\langle y_i\rangle{+}\mathcal{O}(1)$	$T_1{+}[\langle x_\eta\rangle; \langle x_{i+1}\rangle]{+}\langle y_i\rangle{+}\mathcal{O}(1)$

Phase	1–4		5-6		7	
	$\mathfrak{b}_i{=}0$	$\mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}0$	$i{<}k \vee i{>}k, \mathfrak{b}_i{=}0$	$i{=}k \vee i{>}k, \mathfrak{b}_i{=}1$
$\text{POT}_\sigma(\text{A}_i)$	$S_1{+}\langle x_i\rangle{+}\mathcal{O}(1)$	S_i	T_i	$T_1{+}\langle x_i\rangle{+}\mathcal{O}(1)$		T_i

Phase	1–2	
	$\mathfrak{b}_i{=}0$	$\mathfrak{b}_i{=}1$
$\text{POT}_\sigma(d_i)$	$S_1{+}\langle d_i\rangle{+}\frac{1}{2}\langle r\rangle{+}\mathcal{O}(1)$	S_i

Phase	3		
	$\mathfrak{b}_i{=}1$	$i{=}k$	$i{>}k, \mathfrak{b}_i{=}0$
$\text{POT}_\sigma(d_i)$	S_i	$S_1{+}\langle d_i\rangle{+}\langle r\rangle{+}\mathcal{O}(1)$	$S_1{+}\langle d_i\rangle{+}\frac{1}{2}\langle r\rangle{+}\mathcal{O}(1)$

Phase	4		
	$\mathfrak{b}_i{=}1$	$i{=}k$	$i{>}k, \mathfrak{b}_i{=}0$
$\text{POT}_\sigma(d_i)$	S_i	T_k	$S_1{+}\frac{1}{2}\langle r\rangle{+}\langle d_i\rangle{+}\mathcal{O}(1)$

Phase	5–7		
	$i{=}k \vee i{>}k, \mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}0$	$i{<}k$
$\text{POT}_\sigma(d_i)$	T_i	$T_1{+}\frac{1}{2}\langle r\rangle{+}\langle d_i\rangle{+}[S_1^k{-}\langle y_k\rangle{-}\langle d_k\rangle; \langle x_1\rangle]{+}\mathcal{O}(1)$	$T_1{+}\langle d_i\rangle{+}\mathcal{O}(1)$

Table A.1: Potentials

Phase	1–4	5–7
$\text{POT}_\sigma(r)-\text{POT}_\sigma(s)$	$\langle r\rangle > 0$	$[S_1^k-\langle y_k\rangle-\langle d_k\rangle;\langle x_1\rangle]+\langle r\rangle+\mathcal{O}(1) < 0$

Phase	1–4
$\text{POT}_\sigma(y_i)-\text{POT}_\sigma(r)$	$\langle y_i\rangle-\langle r\rangle-S_1^i > 0$

Phase	1–4	5–6	
		$i \geq k$	$i < k$
$\text{POT}_\sigma(y_i)-\text{POT}_\sigma(s)$	$\langle y_i\rangle-S_1^i > 0$	$\langle y_i\rangle-T_1^i > 0$	$[S_{i+1}^k-\langle y_k\rangle-\langle d_k\rangle;\langle x_{i+1}\rangle]+\langle y_i\rangle+\mathcal{O}(1) < 0$

Phase	7			
	$i \geq \eta$	$i{+}1 = \eta = k$	$i{+}1 = \eta < k$	$i{+}1 < \eta$
$\text{POT}_\sigma(y_i)-\text{POT}_\sigma(s)$	$\langle y_i\rangle-T_1^i > 0$	$\langle y_i\rangle-\langle y_k\rangle-\langle d_k\rangle+\mathcal{O}(1) < 0$	$\langle x_{i+1}\rangle+\langle y_i\rangle+\mathcal{O}(1) < 0$	$[\langle x_\eta\rangle;\langle x_{i+1}\rangle]+\langle y_i\rangle+\mathcal{O}(1) < 0$

Phase	1–4		5–7		7	
	$\mathfrak{b}_i{=}0$	$\mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}0$	$k{\geq}i{+}1{>}\eta$	$i{=}k$
$\text{POT}_\sigma(\text{A}_i)-\text{POT}_\sigma(w_{i+1})$	$S_1^i+\langle x_i\rangle+\mathcal{O}(1) < 0$	$\langle y_i\rangle+\langle d_i\rangle > 0$		$T_1^i+\langle x_i\rangle+\mathcal{O}(1) < 0$		$\langle y_k\rangle+\langle d_k\rangle > 0$

Phase	1–2		3		
	$\mathfrak{b}_i{=}0$	$\mathfrak{b}_i{=}1$	$\mathfrak{b}_i{=}1$	$i{=}k$	$i{>}k, \mathfrak{b}_i{=}0$
$\text{POT}_\sigma(d_i)-\text{POT}_\sigma(x_i)$	$\langle d_i\rangle-\langle x_i\rangle+\frac{1}{2}\langle r\rangle+\mathcal{O}(1) < 0$	$-S_1^{i-1}-\langle x_i\rangle > 0$		$\langle d_i\rangle-\langle x_i\rangle+\langle r\rangle+\mathcal{O}(1) < 0$	$\langle d_i\rangle-\langle x_i\rangle+\frac{1}{2}\langle r\rangle+\mathcal{O}(1) < 0$

Phase	4		
	$\mathfrak{b}_i{=}1$	$i{=}k$	$i{>}k, \mathfrak{b}_i{=}0$
$\text{POT}_\sigma(d_i)-\text{POT}_\sigma(x_i)$	$-S_1^{i-1}-\langle x_i\rangle > 0$	$\langle y_k\rangle+\langle d_k\rangle-\langle x_k\rangle > 0$	$\langle d_i\rangle-\langle x_i\rangle+\frac{1}{2}\langle r\rangle+\mathcal{O}(1) < 0$

Phase	5–7		
	$i{=}k \vee i{>}k, \mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}0$	$i{<}k$
$\text{POT}_\sigma(d_i)-\text{POT}_\sigma(x_i)$	$-T_1^{i-1}-\langle x_i\rangle > 0$	$\frac{1}{2}\langle r\rangle+\langle d_i\rangle-\langle x_i\rangle+[S_1^k-\langle y_k\rangle-\langle d_k\rangle;\langle x_1\rangle]+\mathcal{O}(1) < 0$	$\langle d_i\rangle-\langle x_i\rangle+\mathcal{O}(1) < 0$

Phase	1–2		3		
	$\mathfrak{b}_i{=}0$	$\mathfrak{b}_i{=}1$	$\mathfrak{b}_i{=}1$	$i{=}k$	$i{>}k, \mathfrak{b}_i{=}0$
$\text{POT}_\sigma(d_i)-\text{POT}_\sigma(u_{i+1})$	$S_1^i+\langle d_i\rangle+\frac{1}{2}\langle r\rangle+\mathcal{O}(1) < 0$	$\langle y_i\rangle+\langle d_i\rangle > 0$		$S_1^i+\langle d_i\rangle+\langle r\rangle+\mathcal{O}(1) < 0$	$S_1^i+\langle d_i\rangle+\frac{1}{2}\langle r\rangle+\mathcal{O}(1) < 0$

Phase	4		
	$i{=}k \vee i < \delta \vee i{>}k, \mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}0$	$k{>}i{\geq}\delta$
$\text{POT}_\sigma(d_i)-\text{POT}_\sigma(u_{i+1})$	$\langle y_i\rangle+\langle d_i\rangle > 0$	$S_1^i+\langle d_i\rangle+\frac{1}{2}\langle r\rangle+\mathcal{O}(1) < 0$	$S_i^k-\langle y_k\rangle-\langle d_k\rangle < 0$

Phase	5–7		
	$i{=}k \vee i{>}k, \mathfrak{b}_i{=}1$	$i{>}k, \mathfrak{b}_i{=}0$	$i{<}k$
$\text{POT}_\sigma(d_i)-\text{POT}_\sigma(u_{i+1})$	$\langle y_i\rangle+\langle d_i\rangle > 0$	$T_1^i+\frac{1}{2}\langle r\rangle+\langle d_i\rangle+[S_1^k-\langle y_k\rangle-\langle d_k\rangle;\langle x_1\rangle]+\mathcal{O}(1) < 0$	$T_1^i+\langle d_i\rangle+\mathcal{O}(1) < 0$

Table A.2: Potential Differences

Lemma 4.69. *Let a be the total length of all the cycles that are currently opening. Then, the probability that a closing cycle acquires at least b new edges before all opening cycles 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:

$$\begin{aligned} p(a,b) &= \frac{a}{a+1}p(a-1,b) + \frac{1}{a+1}p(a,b-1) \\ &\leq \frac{a}{a+1}\frac{a-1}{2^b} + \frac{1}{a+1}\frac{a}{2^{b-1}} \\ &= \frac{a(a-1)+2a}{(a+1)2^b} \\ &= \frac{a}{2^b} \end{aligned}$$

□

Lemma 4.70. *The probability that a closing cycle acquires b new edges before a different closing cycle of length a closes completely is at most $\mathrm{e}^{-\frac{1}{2}(b-a)^2/(b+a)}$.*

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 [MU05]), 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. □

A.4 Proofs of Chapter 4.8

Lemma 4.82. *Let σ be a policy belonging to one of the phases specified in Table 4.21. Then $|\text{POT}_\sigma(v)| < N^{2n+11}$ and $\varepsilon \cdot |\text{POT}_\sigma(v)| < 1$ for every node v.*

Proof. Let σ be a policy belonging to one of the phases with configuration $\mathfrak{b}$. Let $\mathfrak{b}' = \mathfrak{b} + 1$. Let $S_i = \sum_{j \geq i,\, \mathfrak{b}_j = 1} \big(\langle k_j \rangle + \langle c_j^0 \rangle + \langle d_j^0 \rangle + \langle h_j^0 \rangle\big)$ and similarly $T_i = \sum_{j \geq i,\, \mathfrak{b}'_j = 1} \big(\langle k_j \rangle + \langle c_j^0 \rangle + \langle d_j^0 \rangle + \langle h_j^0 \rangle\big)$.

It suffices to show that $|\text{POT}_\sigma(v)| < N^{2n+11}$ for every node v. Obviously, $\text{POT}_\sigma(t) = 0$.

It is not too hard to see that the following holds:

$$\text{POT}_\sigma(s) \in [S_1; T_1] \qquad\qquad \text{POT}_\sigma(k_i) \in [\langle k_i \rangle + S_1; T_i]$$

We derive for all the other nodes that the following holds:

$$\begin{aligned}
\text{POT}_\sigma(h_i^j) &\in [\langle h_i^j\rangle + \langle k_{i+1}\rangle + S_1; \langle h_i^j\rangle + T_{i+1}] \\
\text{POT}_\sigma(d_i^j) &\in [\langle d_i^j\rangle + S_1; \langle d_i^j\rangle + \langle h_i^j\rangle + T_{i+1}] \\
\text{POT}_\sigma(\mathrm{A}_i^j) &\in [S_1; \langle d_i^j\rangle + \langle h_i^j\rangle + T_{i+1}] \\
\text{POT}_\sigma(b_{i,l}^j) &\in [S_1; \langle d_i^j\rangle + \langle h_i^j\rangle + T_{i+1}] \\
\text{POT}_\sigma(c_i^j) &\in [\langle c_i^j\rangle + S_1; \langle c_i^j\rangle + \langle d_i^j\rangle + \langle h_i^j\rangle + T_{i+1}]
\end{aligned}$$

By Lemma 4.81, we have $|\text{POT}_\sigma(v)| < N^{2n+11}$ for every node v. □

Next, we will specify and prove an auxiliary lemma that describes the exact behavior of all the bicycles appearing in the construction.

The idea behind the bicycles is to have a gate that controls the access of other nodes of the graph to the *escape node* of the bicycle (d_i^j) to which the randomized node moves with very low probability.

First, assume that both cycles attached to a node A_i^j are moving inward. Although the randomized node circles through the cycles with very high probability (without accumulating any costs), it eventually moves out to the escape node, resulting in the same potential as the potential of the escape node itself.

Second, assume that the bicycle is open, i.e. one of the V_0-controlled nodes of the bicycle decides to move out of the gadget to some *reset node*. Now, the randomized node selects to move into the cycle with very large probability and therefore leaves the cycle to the reset node with high probability as well. The resulting potential of the randomized node essentially matches the potential of the reset node.

The following lemma formalizes the intuition of the behavior of the bicycles. If the escape node has better valuation than the reset nodes, it should be profitable to close the bicycle, and otherwise, it should be profitable to open the bicycle again.

Lemma A.2. *Let σ be a policy belonging to one of the phases specified in Table 4.21. Let $U = \{t, k_*\}$ and $u \in U$.*

1. $\overline{\sigma}(b^j_{i,l}) = 0$ *and* $\overline{\sigma}(b^j_{i,1-l}) = 0 \Rightarrow \text{POT}_\sigma(u) > \text{POT}_\sigma(d^j_i)$ *iff* $(b^j_{i,l}, u) \in I_\sigma$,
2. $\overline{\sigma}(b^j_{i,l}) \neq 0$, $\overline{\sigma}(b^j_{i,1-l}) \neq 0$ *and* $\overline{\sigma}(b^j_{i,l}) \neq \overline{\sigma}(b^j_{i,1-l}) \Rightarrow \text{POT}_\sigma(\sigma(b^j_{i,1-l})) > \text{POT}_\sigma(\sigma(b^j_{i,l}))$ *iff* $(b^j_{i,l}, \text{A}^j_i) \in I_\sigma$,
3. $\overline{\sigma}(b^j_{i,l}) \neq 0$ *and* $\overline{\sigma}(b^j_{i,1-l}) = \overline{\sigma}(b^j_{i,l}) \Rightarrow \text{POT}_\sigma(d^j_i) > \text{POT}_\sigma(\sigma(b^j_{i,l}))$ *iff* $(b^j_{i,l}, \text{A}^j_i) \in I_\sigma$,
4. $\overline{\sigma}(b^j_{i,l}) \neq 0$ *and* $\overline{\sigma}(b^j_{i,1-l}) = 0 \Rightarrow \text{POT}_\sigma(d^j_i) > \text{POT}_\sigma(\sigma(b^j_{i,l}))$ *iff* $(b^j_{i,l}, \text{A}^j_i) \in I_\sigma$,
5. $\overline{\sigma}(b^j_{i,l}) = 0$, $\overline{\sigma}(b^j_{i,1-l}) \neq 0$ *and* $\text{POT}_\sigma(d^j_i) > \text{POT}_\sigma(\sigma(b^j_{i,1-l})) \Rightarrow \text{POT}_\sigma(u) > \text{POT}_\sigma(\sigma(b^j_{i,1-l}))$ *iff* $(b^j_{i,l}, u) \in I_\sigma$, *and*
6. $\overline{\sigma}(b^j_{i,l}) = 0$, $\overline{\sigma}(b^j_{i,1-l}) \neq 0$ *and* $\text{POT}_\sigma(d^j_i) < \text{POT}_\sigma(\sigma(b^j_{i,1-l})) \Rightarrow \text{POT}_\sigma(u) \geq \text{POT}_\sigma(\sigma(b^j_{i,1-l}))$ *iff* $(b^j_{i,l}, u) \in I_\sigma$.

Proof. Let σ be a policy belonging to one of the phases specified in Table 4.21.

1. It follows that $\text{POT}_\sigma(\text{A}^j_i) = \text{POT}_\sigma(d^j_i)$.
2. It follows that $\text{POT}_\sigma(\text{A}^j_i) = \frac{1}{2}\text{POT}_\sigma(\sigma(b^j_{i,l})) + \frac{1}{2}\text{POT}_\sigma(\sigma(b^j_{i,1-l})) + \mathcal{O}(1)$.
3. It follows that $\text{POT}_\sigma(\text{A}^j_i) = (1-\varepsilon)\text{POT}_\sigma(\sigma(b^j_{i,l})) + \varepsilon\text{POT}_\sigma(d^j_i)$.
4. It follows that $\text{POT}_\sigma(\text{A}^j_i) = \frac{1-\varepsilon}{1+\varepsilon}\text{POT}_\sigma(\sigma(b^j_{i,l})) + \frac{2\varepsilon}{1+\varepsilon}\text{POT}_\sigma(d^j_i)$.
5. This can be shown the same way.
6. This can be shown the same way.

□

Finally, we prove that the improving switches are indeed exactly as specified. The simple but tedious proof uses Lemma 4.82 and Lemma A.2 to compute the potentials of all important nodes in the game to determine whether a successor of V_0-controlled node is improving or not.

Lemma 4.83. *The improving switches from policies that belong to the phases in Table 4.21 are bounded by those specified in Table 4.22, i.e.* $L^p_\sigma \subseteq I_\sigma \subseteq U^p_\sigma$ *for a phase* p *policy* σ.

Proof. Let σ be a policy belonging to one of the phases with configuration $\mathfrak{b}$. We assume that σ is a phase 1 policy. The improving switches for the other phases can be shown the same way.

Let $S_i^l = \sum_{l \geq j \geq i,\, \mathfrak{b}_j = 1} \left(\langle k_j \rangle + \langle c_j^0 \rangle + \langle d_j^0 \rangle + \langle h_j^0 \rangle \right)$ and $S_i = S_i^n$.

First, we apply Lemma 4.81 and compute the potentials of all nodes.

Node	t	s	c_i^j
Potential	0	S_1	$\langle c_i^j \rangle + \text{POT}_\sigma(\text{A}_i^j)$

Node	h_i^0	h_i^1
Potential	$\langle h_i^0 \rangle + S_{i+2}$	$\langle h_i^1 \rangle + \text{POT}_\sigma(k_{i+1})$

Node	k_i		d_i^j	
	$\mathfrak{b}_i{=}1$	$\mathfrak{b}_i{=}0$	$\mathfrak{b}_{i+1}{=}j$	$\mathfrak{b}_{i+1}{\neq}j$
Potential	S_i	$\langle k_i \rangle + S_1$	$\langle d_i^j \rangle + \text{POT}_\sigma(h_i^j)$	$\langle d_i^j \rangle + S_1$

Node	A_i^j		$b_{i,l}^j$	
	$\overline{\sigma}(\text{A}_i^j){=}1$	$\overline{\sigma}(\text{A}_i^j){\neq}1$	$\overline{\sigma}(\text{A}_i^j){=}1$	$\overline{\sigma}(\text{A}_i^j){\neq}1$
Potential	$\text{POT}_\sigma(d_i^j)$	$S_1 + \mathcal{O}(1)$	$\text{POT}_\sigma(d_i^j)$	$S_1 + \mathcal{O}(1)$

Second, we observe the following ordering on the potentials of all "entry points" in the game graph.

1. $\mathfrak{b}_i = 1$ implies $\text{POT}_\sigma(k_i) > \text{POT}_\sigma(t)$,
2. $\mathfrak{b}_i = 1$ and $\mathfrak{b}_j = 0$ implies $\text{POT}_\sigma(k_i) > \text{POT}_\sigma(k_j)$,
3. $\mathfrak{b}_i = 1$, $\mathfrak{b}_j = 1$ and $i < j$ implies $\text{POT}_\sigma(k_i) > \text{POT}_\sigma(k_j)$.

Third, we derive that there are no improving switches for s and h_i^0. Fourth, we compute the differences between the potentials of the successors of d_i^j to see that there are no improving switches for these nodes.

Diff.	$\text{POT}_\sigma(h_i^0) - \text{POT}_\sigma(s)$		$\text{POT}_\sigma(h_i^1) - \text{POT}_\sigma(s)$	
	$\mathfrak{b}_{i+1}{=}1$	$\mathfrak{b}_{i+1}{=}0$	$\mathfrak{b}_{i+1}{=}1$	$\mathfrak{b}_{i+1}{=}0$
Value	$\langle h_i^0 \rangle - S_1^{i+1} < 0$	$\langle h_i^0 \rangle - S_1^i > 0$	$\langle h_i^1 \rangle - S_1^i > 0$	$\langle h_i^1 \rangle + \langle k_{i+1} \rangle < 0$

Fifth, we show that there are no improving switches for the entry points k_i by computing the potential differences between S_1 and c_i^j if $\mathfrak{b}_i = 0$ and additionally between c_i^j and c_i^{1-j} if $\mathfrak{b}_i = 1$.

Difference	$\text{POT}_\sigma(c_i^j) - \text{POT}_\sigma(c_i^{1-j})$	
	$\mathfrak{b}_i = 1, \mathfrak{b}_{i+1} = j$	
	$\overline{\sigma}(\text{A}_i^{1-j})=1$	$\overline{\sigma}(\text{A}_i^{1-j})=0$
Value	$\langle h_i^0 \rangle - S_1^i > 0$	$\langle h_i^0 \rangle + \langle d_i^j \rangle - S_1^i + \mathcal{O}(1) > 0$

Difference	$\text{POT}_\sigma(c_i^j) - S_1$		
	$\mathfrak{b}_i = 1, \mathfrak{b}_{i+1} = j$	$\mathfrak{b}_i = 0$	
		$\overline{\sigma}(\text{A}_i^j)=1$	$\overline{\sigma}(\text{A}_i^j)=0$
Value	$\langle h_i^j \rangle + \langle d_i^j \rangle - S_1^i > 0$	$\langle c_i^j \rangle + \langle d_i^j \rangle < 0$	$\langle c_i^j \rangle + \mathcal{O}(1) < 0$

Finally, we consider all $b_{i,l}^j$ nodes and show that the set of improving switches is indeed $\{(b_{i,l}^j, \text{A}_i^j) \mid \sigma(b_{i,l}^j) \neq \text{A}_i^j\}$. Therefore, we compute the potential difference between d_i^j and S_1, and apply Lemma A.2.

Difference	$\text{POT}_\sigma(d_i^j) - S_1$	
	$\mathfrak{b}_{i+1} = j$	$\mathfrak{b}_{i+1} \neq j$
Value	$\langle d_i^j \rangle + \langle h_i^j \rangle - S_1^i > 0$	$\langle d_i^j \rangle > 0$

□

Lemma 4.84. *Let σ be an initial phase 1 policy with configuration $\mathfrak{b} < \mathbf{1}_n$. There is an initial phase 1 policy σ' with configuration $\mathfrak{b}' = \mathfrak{b} + 1$ s.t. $(\sigma, \phi^{\mathfrak{b}}) \rightsquigarrow^+ (\sigma', \phi^{\mathfrak{b}'})$.*

Proof. Let σ_1 be an initial phase 1 policy with configuration $\mathfrak{b} < \mathbf{1}_n$. Let $\mathfrak{b}' = \mathfrak{b} + 1$ and $r = \mu_1(\mathfrak{b})$. Let $\phi_1 = \phi^{\mathfrak{b}}$.

The idea of this lemma is to undergo all six phases of Table 4.17 while performing improving switches towards the desired subsequent occurrence record.

More formally: we construct additional $(\sigma_2, \phi_2), \ldots, (\sigma_7, \phi_7)$ s.t.

- $(\sigma_p, \phi_p) \rightsquigarrow^+ (\sigma_{p+1}, \phi_{p+1})$,

- σ_p is in phase p with configuration $\mathfrak{b}$ if $p < 7$, and
- $\phi_7 = \phi^{\mathfrak{b}'}$ and σ_7 is an initial phase 1 policy with configuration $\mathfrak{b}'$

The construction is now as follows. We implicitly apply Lemma 4.83 when referring to the improving switches of a phase.

1. The only improving switches in this phase are from $b^j_{i,l}$ to A^j_i. This will be the only phase in which we will be making any switches of this kind.

   The first observation to make is that $g(\mathfrak{b}, i, \{(i{+}1, j)\}) = g(\mathfrak{b}', i, \{(i{+}1, j)\})$ if $i \neq r$.

   First, there are bicycles s.t. $\mathfrak{b}_i = 1$ and $\mathfrak{b}_{i+1} = j$, hence they are already closed, hence we cannot increase their respective occurrence records. In other words, we need to show that $\phi_1(b^j_{i,l}, \mathrm{A}^j_i) = \phi_7(b^j_{i,l}, \mathrm{A}^j_i)$.

   If $\mathfrak{b}'_i = 1$, i.e. $i > r$, it follows by $g(\mathfrak{b}, i, \{(i{+}1, j)\}) = g(\mathfrak{b}', i, \{(i{+}1, j)\})$ that $\phi_1(b^j_{i,l}, \mathrm{A}^j_i) = \phi_7(b^j_{i,l}, \mathrm{A}^j_i)$.

   Otherwise, if $\mathfrak{b}'_i = 0$, i.e. $i < r$, it follows that we have $\phi_7(b^j_{i,l}, \mathrm{A}^j_i) = g(\mathfrak{b}', i, \{(i{+}1, j)\}) + 1 + 2 \cdot (|\mathfrak{b}'| - g(\mathfrak{b}', i, \{(i{+}1, j)\}) - 2^{i-1})$. In other words, we need to show that $|\mathfrak{b}'| - g(\mathfrak{b}', i, \{(i{+}1, j)\}) = 2^{i-1}$. And this is true, because it required 2^{i-1} counting steps to count with all the lower bits.

   Second, there are bicycles s.t. $\mathfrak{b}_{i+1} \neq j$ and $\phi_1(b^j_{i,0}, \mathrm{A}^j_i) + \phi_1(b^j_{i,1}, \mathrm{A}^j_i) < |\mathfrak{b}|$. We will see that, $i \neq r$. Hence, we know that $g(\mathfrak{b}, i, \{(i{+}1, j)\}) = g(\mathfrak{b}', i, \{(i{+}1, j)\})$. In this case, we have $\phi_1(b^j_{i,l}, \mathrm{A}^j_i){+}2 = \phi_7(b^j_{i,l}, \mathrm{A}^j_i)$. Hence, by flipping both edges of these bicycles, we can make sure that we comply to the objective occurrence record.

   Third, there are bicycles s.t. $\mathfrak{b}_i = 0$ or $\mathfrak{b}_{i+1} \neq j$ that have $\phi_1(b^j_{i,0}, \mathrm{A}^j_i) + \phi_1(b^j_{i,1}, \mathrm{A}^j_i) = |\mathfrak{b}|$. Obviously, r belongs to this class of bicycles. It is easy to see that $\phi_1(b^j_{i,l}, \mathrm{A}^j_i) + 1 = \phi_7(b^j_{i,l}, \mathrm{A}^j_i)$ for $i \neq r$ and $\phi_1(b^j_{i,l}, \mathrm{A}^j_i) + 2 = \phi_7(b^j_{i,l}, \mathrm{A}^j_i)$ for $i = r$. Hence, by switching one edge for all $i \neq r$ and both edges for $i = r$, we can make sure that we comply to the objective occurrence record.

The order in which all switches are to be performed is therefore as follows. We close both edges of all second class bicycles and one edge of every third class bicycle. Finally, we close the second edge of bicycle r.

We now have, for all *open* bicycles, that $\phi_2(b^j_{i,0}, \mathrm{A}^j_i) + \phi_2(b^j_{i,1}, \mathrm{A}^j_i) = |\mathfrak{b}'|$.

2. The only improving switches in this phase are still from $b^j_{i,l}$ to A^j_i, and from k_r to $c_r^{\mathfrak{b}'_{r+1}}$.

   Is easy to see that $2f(\mathfrak{b}', r, \{(r{+}1, \mathfrak{b}'_{r+1})\}) \leq |\mathfrak{b}'|$, hence we can ensure to make that switch without closing any additional bicycles.

   Also note that $\phi_2(k_r, c_r^{\mathfrak{b}'_{r+1}}) + 1 = \phi_7(k_r, c_r^{\mathfrak{b}'_{r+1}})$, and for all other edges $(i, j) \neq (r, \mathfrak{b}'_{r+1})$ of this kind we have $\phi_2(k_i, c^j_i) = \phi_7(k_i, c^j_i)$.

3. In this phase, there are many improving switches. In order to fulfill all side conditions for phase 3, we need to perform all switches from higher indices to smaller indices, and k_i to k_r before $b^j_{i,l}$ with $\mathfrak{b}'_{i+1} \neq j$ or $\mathfrak{b}'_i = 0$ to k_r.

   The reader can easily check from that we can perform the switches in the desired ordering.

4.-6. These can be shown similarly.

□

B

Proofs of Chapter 5

B.1 Proofs of Chapter 5.2

We define three predicates $\Psi_n(j,i,l,d)$, $\Phi_n(j,i,d)$ and $\Delta_n(j,i,d)$ that will be used as pre- and postconditions in the induction. Let $j \leq n$, i be an G_n-index, l be a G_n-playlist and $d = (d_0, d_1)$ be G_n-decisions.

- $\Psi_n(j,i,l,d)$ is defined to hold iff all of the following conditions hold:
  - (a) $l_k = (q, \_, \_, \_, \_)$ implies that there is an $h > j$ s.t. $(q \in \{a_h, b_h, c_h\})$ for every $k < |l|$
  - (b) If $j < n$: $l_0 = (a_n, \mathbf{0} \oplus \Omega(a_n), 0, \emptyset, \_)$
  - (c) $i(\Omega(a_n)) = 1$
  - (d) $d_1(q) = \emptyset$ for all $q \in V_n$
- $\Phi_n(j,i,d)$ is defined to hold iff the following condition holds:
  - (a) $(k, \_, \_) \in d_0(q)$ implies that there is a w s.t. $i <_0^w k$ with $\Omega(w) > 2j$ for all $q \in V_n$ and all G_n-indices k
- $\Delta_n(j,i,d)$ is defined to hold iff all of the following conditions hold:
  - (a) $(k, \_, \_) \in d_0(q)$ with $i >_0 k$ implies that there is a w s.t. $i >_0^w k$ with $\Omega(w) < 2j$ for all $q \in V_n$ and all G_n-indices k
  - (b) $(i, \_, \_) \in d_0(a_j)$

We analyze the runtime complexity of the model checking algorithm in terms of $\texttt{MC}(G, v)$, i.e. the time-counter that is maintained by the EXPLORE-routine.

Recall that we want to prove that the following function f_n captures the progression of it accurately. Let $n \in \mathbb{N}$ and $i < n$.

$$f_n : i \mapsto \begin{cases} 1 & \text{if } i = 0 \\ f_n(i-1) + 4 & \text{if } i > 0 \text{ and } \textsc{Select}(G_n, b_i, b_i E) = c_i \\ 2 \cdot f_n(i-1) + 4 & \text{otherwise} \end{cases}$$

Lemma B.1. *Let $j \leq n$, i be an G_n-index, l be a G_n-playlist, $d = (d_0, d_1)$ be G_n-decisions and $c \in \mathbb{N}$ s.t. $\Psi_n(j, i, l, d)$ and $\Phi_n(j, i, d)$ hold. Then, calling* $\textsc{Explore}(G_n, a_j, i, l, c, d)$ *leads to* $\textsc{Backtrack}(G_n, a_j, 0, l', c', d')$*, where $c' = c + f_n(j) + 1$ and $l \equiv l'$ s.t. $\Psi_n(j, i, l', d')$ and $\Delta_n(j, i, d')$ hold.*

Proof. By induction on j.

For $j = 0$, let $\Psi_n(0, i, l, d)$ and $\Phi_n(0, i, d)$ hold; $\textsc{Explore}(G_n, a_0, i, l, c, d)$ directly invokes $\textsc{Explore}(G_n, a_n, i_1, l_1, c + 1, d)$ where $i_1 = i \oplus \Omega(a_n)$ and $l_1 = (a_0, i, \emptyset, c, \emptyset) :: l$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a)) nor a repeat in the playlist (Ψ_n (a)).

Running $\textsc{Explore}(G_n, a_n, i_1, l_1, c + 1, d)$ encounters a repeat, as $(a_n, \mathbf{0} \oplus \Omega(a_n), 1, \emptyset, \_)$ is in l_1 by Ψ_n (b). Due to the fact that $i(\Omega(a_n)) = 1$ by Ψ_n (c) and $i_1(\Omega(a_n)) = 2$, it directly follows that the repeat is profitable for player 0.

Hence, $\textsc{Backtrack}(G_n, a_n, 0, l_2, c + 2, d)$ is called, where $l_2 \equiv l_1$. Since l_2 is not empty and the top entry has no other edges to visit, $\textsc{Backtrack}(G_n, a_0, 0, l', c + 1, d')$ is invoked, where $l' \equiv l$, $d'_1 = d_1$ and $d'_0 = d_0[a_0 \mapsto d_0(a_0) \cup \{(i, c, \bot)\}]$. Note that $\Psi_n(j, i, l', d')$ as well as $\Delta_n(j, i, d')$ hold.

For $j \rightsquigarrow j + 1$, let $\Psi_n(j + 1, i, l, d)$ and $\Phi_n(j + 1, i, d)$ hold; $\textsc{Explore}(G_n, a_{j+1}, i, l, c, d)$ directly invokes $\textsc{Explore}(G_n, b_{j+1}, i_1, l_1, c + 1, d)$ where $i_1 = i \oplus \Omega(b_{j+1})$ and $l_1 = (a_{j+1}, i, \emptyset, c, \emptyset) :: l$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a)) nor a repeat in the playlist (Ψ_n (a)).

Let $w = \textsc{Select}(G_n, b_{j+1}, b_{j+1} E)$. We will now distinguish on whether $w = c_{j+1}$ or $w = a_j$.

- Case $w = a_j$: Calling EXPLORE$(G_n, b_{j+1}, i_1, l_1, c+1, d)$ directly invokes EXPLORE$(G_n, a_j, i_2, l_2, c+2, d)$ where $i_2 = i_1 \oplus \Omega(a_j)$ and $l_2 = (b_{j+1}, i_1, \{c_{j+1}\}, c+1, \emptyset) :: l_1$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a) by Corollary 5.7) nor a repeat in the playlist (Ψ_n (a)).

  Now note that $\Psi_n(j, i_2, l_2, d)$ as well as $\Phi_n(j, i_2, d)$ hold: $\Psi_n(j, \ldots)$ (a) holds by construction of l_2 and $\Psi_n(j+1, \ldots)$ (a); if $j+1 < n$, $\Psi_n(j, \ldots)$ (b) holds due to $\Psi_n(j+1, \ldots)$ (b), otherwise by construction of l_2; $\Psi_n(j, \ldots)$ (c) and (d) obviously hold by construction and $\Psi_n(j+1, \ldots)$ (c) and (d). Lastly, $\Phi_n(j, i_2, d)$ holds due to $\Phi_n(j+1, i, d)$ and Corollary 5.7.

  By induction hypothesis, eventually BACKTRACK$(G_n, a_j, 0, l_3, c + f_n(j) + 3, d^i)$ gets called, where $l_3 \equiv l_2$ and $\Psi_n(j, i_2, l_3, d^i)$ and $\Delta_n(j, i_2, d^i)$ hold. Since c_{j+1} remained unexplored, EXPLORE$(G_n, c_{j+1}, i_4, l_4, c + f_n(j) + 3, d^i)$ is invoked, where $i_4 = i_1 \oplus \Omega(c_{j+1})$ and $l_4 \equiv (b_{j+1}, i_1, \emptyset, c+1, \emptyset) :: l_1$.

  Consider that $\Phi_n(j+1, i_4, d^i)$ holds: let $(k, \_, \_) \in d_0^i(q)$ for an arbitrary q. By construction, $i_4 <_0^{c_{j+1}} i_2$. Hence, if $i_2 \leq_0 k$, it follows by Corollary 5.7 that $\Phi_n(j+1, i_4, d^i)$ holds. Otherwise, we apply $\Delta_n(j, i_2, d^i)$ and conclude that $i_2 >_0^w k$ with $\Omega(w) < 2j$, thus by Corollary 5.7 it follows that $\Phi_n(j+1, i_4, d^i)$ holds.

  Subsequently, EXPLORE$(G_n, a_j, i_5, l_5, c + f_n(j) + 4, d^i)$ gets called, where $i_5 = i_4 \oplus \Omega(a_j)$ and $l_5 \equiv (c_{j+1}, i_4, \emptyset, c + f_n(j) + 1, d^i) :: l_4$, as there is neither an applicable decision (Ψ_n (d) and $\Phi_n(j+1, i_4, d^i)$ (a)) nor a repeat in the playlist (Ψ_n (a)).

  Note that $\Psi_n(j, i_5, l_5, d^i)$ as well as $\Phi_n(j, i_5, d^i)$ hold: $\Psi_n(j, \ldots)$ (a) holds by construction of l_5 and $\Psi_n(j+1, \ldots)$ (a); if $j+1 < n$, $\Psi_n(j, \ldots)$ (b) holds due to $\Psi_n(j+1, \ldots)$ (b), otherwise by construction of l_5; $\Psi_n(j, \ldots)$ (c) and (d) obviously hold by construction and $\Psi_n(j+1, \ldots)$ (c) and (d). Lastly, $\Phi_n(j, i_5, d^i)$ holds due to $\Phi_n(j+1, i_4, d^i)$ and Corollary 5.7.

  By induction hypothesis, eventually BACKTRACK$(G_n, a_j, 0, l_6, c', d^{ii})$ gets called, where $l_6 \equiv l_5$, $c' = c+2f_n(j)+5 = c+f_n(j+1)+1$ and $\Psi_n(j, i_5, l_6, d^{ii})$ and $\Delta_n(j, i_5, d^{ii})$ hold. Since l_6 is not empty and the top entry has no other

transitions to visit, BACKTRACK$(G_n, c_{j+1}, 0, l_7, c', d^{iii})$ is invoked, where $l_7 \equiv l_4$ and $d^{iii} = d^{ii}[c_{j+1} \mapsto d^{ii}(c_{j+1}) \cup \{(i_4, c + f_n(j) + 1, \bot)\}]$.

Again since l_7 is not empty and the top entry has no other transitions to visit, BACKTRACK$(G_n, b_{j+1}, 0, l_8, c', d^{iv})$ is invoked, where $l_8 \equiv l_1$ and $d^{iv} = d^{iii}[b_{j+1} \mapsto d^{iii}(b_{j+1}) \cup \{(i_1, c + 1, \bot)\}]$.

Lastly, BACKTRACK$(G_n, a_{j+1}, 0, l_9, c', d^v)$ is called for the same reasons, where $l_9 \equiv l$ and $d^v = d^{iv}[a_{j+1} \mapsto d^{iv}(a_{j+1}) \cup \{(i, c, \bot)\}]$.

It remains to show that $\Psi_n(j + 1, i, l_9, d^v)$ and $\Delta_n(j + 1, i, d^v)$ hold: Since $d^v_2 = d_2$ and $l_9 \equiv l$, it directly follows that $\Psi_n(j + 1, i, l_9, d^v)$ by assuming $\Psi_n(j+1, i, l, d)$. $\Delta_n(j+1, i, d^v)$ (b) obviously holds as $\{(i, c, \bot)\} \in d^v(a_{j+1})$; for $\Delta_n(j+1, i, d^v)$ (a) let $q \in V_n$ be arbitrary s.t. there is $(k, \_, \_) \in d^v_0(q)$ with $i >_0 k$. If $(k, \_, \_) \in d^{ii}_0(q)$ it follows by induction hypothesis that $i_5 >^w_0 k$ with $\Omega(w) < 2j$; since $i >^{c_{j+1}}_0 i_5$ and $\Omega(c_{j+1}) = 2(j + 1) - 1$, Corollary 5.7 implies that $i >^{c_{j+1}}_0 k$ with $\Omega(c_{j+1}) < 2(j+1)$. Otherwise, if $(k, \_, \_) \notin d^{ii}_0(q)$, then $k \in \{i, i_1, i_4\}$; only $i_4 <_0 i$, again with c_{j+1}: $i_4 <^{c_{j+1}}_0 i$.

- Case $w = c_{j+1}$: Calling EXPLORE$(G_n, b_{j+1}, i_1, l_1, c + 1, d)$ invokes EXPLORE(G_n,c_j,i_2,l_2,$c+2$,d) where $i_2 = i_1 \oplus \Omega(c_{j+1})$ and $l_2 = (b_{j+1}, i_1, \{a_j\}, c+1, \emptyset) :: l_1$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a) by Corollary 5.7) nor a repeat in the playlist (Ψ_n (a)).

  Subsequently, EXPLORE$(G_n, a_j, i_3, l_3, c + 3, d^i)$ gets called, where $i_3 = i_2 \oplus \Omega(a_j)$ and $l_3 \equiv (c_{j+1}, i_2, \emptyset, c + 2, d) :: l_2$, as there is neither an applicable decision (Ψ_n (d) and Φ_n (a) by Corollary 5.7) nor a repeat in the playlist (Ψ_n (a)).

  Now note that $\Psi_n(j, i_3, l_3, d)$ as well as $\Phi_n(j, i_3, d)$ hold: $\Psi_n(j, \ldots)$ (a) holds by construction of l_3 and $\Psi_n(j + 1, \ldots)$ (a); if $j + 1 < n$, $\Psi_n(j, \ldots)$ (b) holds due to $\Psi_n(j + 1, \ldots)$ (b), otherwise by construction of l_3; $\Psi_n(j, \ldots)$ (c) and (d) obviously hold by construction and $\Psi_n(j + 1, \ldots)$ (c) and (d). Lastly, $\Phi_n(j, i_3, d)$ holds due to $\Phi_n(j + 1, i, d)$ and Corollary 5.7.

  By induction hypothesis, eventually BACKTRACK$(G_n, a_j, 0, l_4, c + f_n(j) + 4, d^i)$ gets called, where $l_4 \equiv l_3$ and $\Psi_n(j, i_3, l_4, d^i)$ and $\Delta_n(j, i_3, d^i)$ hold.

Since l_4 is not empty and the top entry has no other transitions to visit, BACKTRACK(G_n,c_{j+1},0,l_5,$c + f_n(j) + 4$,d^{ii}) is invoked, where $l_5 \equiv l_2$ and $d^{ii} = d^i[c_{j+1} \mapsto d^i(c_{j+1}) \cup \{(i_2, c + 2, \bot)\}]$.

Since $b_{j+1}Ea_j$ remained unexplored, EXPLORE$(G_n, a_j, i_6, l_6, c+f_n(j)+4, d^{ii})$ is invoked, where $i_6 = i_1 \oplus \Omega(a_j)$ and $l_6 \equiv (b_{j+1}, i_1, \emptyset, c + 1, \emptyset) :: l_1$. As $\Delta_n(j, i_3, d^i)$ (b) holds by induction hypothesis, $(i_3, \_, \_) \in d_0^{ii}(a_j)$ and $i_6 >_0 i_3$, a decision is applicable, hence BACKTRACK$(G_n, a_j, 0, l_6, c', d^i i)$ is invoked where $c' = c + f_n(j) + 5 = c + f_n(j + 1) + 1$.

Because l_6 is not empty and the top entry has no other transitions to visit, BACKTRACK$(G_n, b_{j+1}, 0, l_7, c + f_n(j) + 5, d^{iii})$ is invoked, where $l_7 \equiv l_1$ and $d^{iii} = d^{ii}[b_{j+1} \mapsto d^{ii}(b_{j+1}) \cup \{(i_1, c + 1, \bot)\}]$.

Lastly, BACKTRACK$(G_n, a_{j+1}, 0, l_8, c', d^{iv})$ is called for the same reasons, where $l_8 \equiv l$ and $d^{iv} = d^{iii}[a_{j+1} \mapsto d^{iii}(a_{j+1}) \cup \{(i, c, \bot)\}]$.

It remains to show that $\Psi_n(j + 1, i, l_8, d^{iv})$ and $\Delta_n(j + 1, i, d^{iv})$ hold: Since $d_2^{iv} = d_2$ and $l_8 \equiv l$, it directly follows that $\Psi_n(j + 1, i, l_8, d^{iv})$ by assuming $\Psi_n(j + 1, i, l, d)$. $\Delta_n(j + 1, i, d^{iv})$ (b) obviously holds as $\{(i, c, \bot)\} \in d^{iv}(a_{j+1})$; for $\Delta_n(j+1, i, d^{iv})$ (a) let $q \in V_n$ be arbitrary s.t. there is $(k, \_, \_) \in d_0^{iv}(q)$ with $i >_0 k$. If $(k, \_, \_) \in d_0^i(q)$ it follows by induction hypothesis that $i_3 >_0^w k$ with $\Omega(w) < 2j$; since $i >_0^{c_{j+1}} i_3$ and $\Omega(c_{j+1}) = 2(j + 1) - 1$, Corollary 5.7 implies that $i >_0^{c_{j+1}} k$ with $\Omega(c_{j+1}) < 2(j + 1)$. Otherwise, if $(k, \_, \_) \notin d_0^i(q)$, then $k \in \{i, i_1\}$; none of them is $<_0$-less than i.

□

Lemma 5.11. *Let* $n \in \mathbb{N}$. *Then* $\mathtt{MC}(G_n, a_n) = f_n(n) + 1$.

Proof. Calling DECIDE(G_n, a_n) directly invokes EXPLORE(G_n, a_n, $\mathbf{0} \oplus \Omega(a_n)$, $[]$, 0, $(\mathbf{e}, \mathbf{e})$). Note that the given arguments trivially satisfy Ψ_n as well as Φ_n, hence Lemma B.1 implies that eventually BACKTRACK$(G_n, a_n, 0, [], f_n(n) + 1, d')$ with some decisions d' is called. By definition of BACKTRACK, it follows that the algorithm directly terminates, hence it requires $f_n(n) + 1$ EXPLORE-steps in total. □

Bibliography

[AC78] 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.

[ADD00] R. B. Ash and C. A. Doléans-Dade. *Probability and Measure Theory*. Academic Press, 2000.

[AM09] D. Andersson and P. B. Miltersen. The complexity of solving stochastic games on graphs. In *ISAAC '09: Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 112–121, Berlin, Heidelberg, 2009. Springer.

[Ame94] N. Amenta. Helly-type theorems and generalized linear programming. *Discrete & Computational Geometry*, 12:241–261, 1994.

[AZ96] N. Amenta and G. M. Ziegler. Deformed products and maximal shadows of polytopes. In *Advances in Discrete and Computational Geometry*, pages 57–90, Providence, 1996. American Mathematical Society. Contemporary Mathematics 223.

[BDF+95] A.Z. Broder, M.E. Dyer, A.M. Frieze, P. Raghavan, and E. Upfal. The worst-case running time of the random simplex algorithm is exponential in the height. *Information Processing Letters*, 56(2):79–81, 1995.

[Bel57] R. E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[Ber01] D. P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, second edition, 2001.

[BM08] A. Beckmann and F. Moller. On the complexity of parity games. In *Proceedings of the BCS Conference Visions of Computer Science*, 2008.

[BP07] J. Balogh and R. Pemantle. The Klee-Minty random edge chain moves with linear speed. *Random Structures & Algorithms*, 30(4):464–483, 2007.

[BS96] G. S. Bhat and C. D. Savage. Balanced gray codes. *Electronic Journal of Combinatorics*, 3:2–5, 1996.

[BSV03] H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, STACS'03*, volume 2607 of *LNCS*, pages 663–674. Springer, 2003.

[BV05] H. Björklund and S. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.

[BV07] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

[CGG+05] A. Costan, S. Gaubert, E. Goubault, M. Martel, and S. Putot. A policy iteration algorithm for computing fixed points in static analysis of programs. In *CAV*, pages 462–475, 2005.

[CH08] K. Chatterjee and T. A. Henzinger. Value iteration. In O. Grumberg and H. Veith, editors, *25 Years of Model Checking*, pages 107–138. Springer, Berlin, Heidelberg, 2008.

[Chv83] V. Chvátal. *Linear programming*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York, 1983.

[Con92] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[Con93] A. Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in*

Discrete Mathematics and Theoretical Computer Science, pages 51–73. American Mathematical Society, 1993.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[Der70] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, Inc., Orlando, FL, USA, 1970.

[EJ91] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proceedings of the 32nd Symposium on Foundations of Computer Science*, pages 368–377, San Juan, 1991. IEEE.

[EJS93] E. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *Proceedings of the 5th Conference on CAV, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[EM79] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

[Fea10] J. Fearnley. Exponential lower bounds for policy iteration. *CoRR*, abs/1003.3418, 2010.

[FL09] O. Friedmann and M. Lange. Solving parity games in practice. In *Proc. 7th Int. Symp. on Automated Technology for Verification and Analysis, ATVA'09*, volume 5799 of *LNCS*, pages 182–196, 2009.

[FL10a] O. Friedmann and M. Lange. Local strategy improvement for parity game solving. In *First International Symposium on Games, Automata, Logics and Formal Verification, GandALF'10*, volume 25 of *EPTCS*, pages 118–131, 2010.

[FL10b] O. Friedmann and M. Lange. A solver for modal fixpoint logics. *Electronic Notes Theoretical Computer Science*, 262:99–111, May 2010.

[FLL10] O. Friedmann, M. Latte, and M. Lange. A decision procedure for CTL* based on tableaux and automata. In *IJCAR*, pages 331–345, 2010.

[FS09] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[Gär95] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24:1018–1035, 1995.

[Gär02] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002.

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC'82*, pages 60–65. ACM, ACM Press, 1982.

[GHZ98] B. Gärtner, M. Henk, and G. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, 18(3):349–372, 1998.

[GK07] B. Gärtner and V. Kaibel. Two new bounds for the random-edge simplex-algorithm. *Discrete Mathematics*, 21(1):178–190, 2007.

[GKK88] V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28:85–91, 1988.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[GMR08] B. Gärtner, W. D. Morris, and L. Rüst. Unique sink orientations of grids. *Algorithmica*, 51:200–235, April 2008.

[GS79] D. Goldfarb and W.Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277 – 285, 1979.

[GS06] B. Gärtner and I. Schurr. Linear programming and unique sink orientations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 749–757, 2006.

[GS07] T. Gawlitza and H. Seidl. Precise relational invariants through strategy iteration. In *CSL*, pages 23–40, 2007.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, LNCS. Springer, 2002.

[GTW+03] B. Gärtner, F. Tschirschnitz, E. Welzl, J. Solymosi, and P. Valtr. One line and n points. *Random Structures & Algorithms*, 23(4):453–471, 2003.

[Hal07] 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.

[HK66] A. J. Hofmann and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[How60] R. Howard. *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.

[Jer73] R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973.

[JPY88] D. Johnson, C. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[JPZ06] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, SODA'06*, pages 117–123. ACM, 2006.

[Jur98] M. Jurdziński. Deciding the winner in parity games is in $\mathtt{UP} \cap \mathtt{coUP}$. *Information Processing Letters*, 68(3):119–124, 1998.

[Jur00] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm (extended abstract). In *Proceedings of the 24th STOC*, pages 475–482, 1992.

[Kal97] G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997.

[Kar78] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, New York, NY, USA, 1984. ACM.

[Kha79] L. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[KK92] G. Kalai and D. Kleitman. Quasi-polynomial bounds for the diameter of graphs and polyhedra. *Bulletin of the American Mathematical Society*, 26:315–316, 1992.

[KL93] A. V. Karzanov and V. N. Lebedev. Cyclical games with prohibitions. *Mathematical Programming*, 60:277–293, 1993.

[KM72] V. Klee and G. L. Minty. How good is the simplex algorithm? *Inequalities*, III:159–179, 1972.

[KW08] D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, ICALP'08*, volume 5125 of *LNCS*, pages 724–735. Springer, 2008.

[Lad75] R. E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[Lud95] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117(1):151–155, 1995.

[Mar75] D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

[Mat94] J. Matoušek. Lower bounds for a subexponential optimization algorithm. *Random Structures and Algorithms*, 5(4):591–608, 1994.

[MG07] J. Matoušek and B. Gärtner. *Understanding and using linear programming*. Springer, 2007.

[MS99] Y. Mansour and S. P. Singh. On the complexity of policy iteration. In *Proceedings of the 15th UAI*, pages 401–408, 1999.

[MS06] J. Matoušek and T. Szabó. RANDOM EDGE can be exponential on abstract cubes. *Advances in Mathematics*, 204(1):262–277, 2006.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4-5):498–516, 1996.

[MTZ10] O. Madani, M. Thorup, and U. Zwick. Discounted deterministic markov decision processes and discounted all-pairs shortest paths. *ACM Transactions on Algorithms*, 6(2):1–25, 2010.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and computing, Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming*. SIAM, 1994.

[OPS04] J. Orlin, A. Punnen, and A. Schulz. Approximate local search in combinatorial optimization. In *SIAM Journal on Computing*, pages 587–596, 2004.

[Pit06] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proceedings of the 21st Symposium on Logic in Computer Science, LICS'06*, pages 255–264. IEEE Computer Society, 2006.

[Pur95] A. Puri. *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, University of California, Berkeley, 1995.

[Put94] M. L. Puterman. *Markov decision processes*. Wiley, 1994.

[PV01a] V. Petersson and S. Vorobyov. Parity games: Interior-point approach. Technical Report 2001-008, Department of Information Technology, Uppsala University, May 2001.

[PV01b] V. Petersson and S. Vorobyov. A randomized subexponential algorithm for parity games. *Nordic Journal of Computing*, 8(3):324–345, 2001.

[PY88] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 229–234, New York, NY, USA, 1988. ACM.

[San10] F. Santos. A counterexample to the hirsch conjecture. *CoRR*, abs/1006.2814v1, 2010.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[Sch07] S. Schewe. Solving parity games in big steps. In *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'07*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007.

[Sch08] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proceedings of the 17th Annual Conference on Computer Science Logic, CSL'08*, volume 5213 of *LNCS*, pages 369–384. Springer, 2008.

[Sha53] L. S. Shapley. Stochastic games. *Proceedings of National Academy of Sciences USA*, 39:1095–1100, 1953.

[Sri08] G. Srinivasan. *Operations Research: Principles And Applications*. Phi Learning, 2008.

[SS98] P. Stevens and C. Stirling. Practical model-checking using games. In B. Steffen, editor, *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[SS05] I. Schurr and T. Szabó. Jumping doesn't help in abstract cubes. In *IPCO*, pages 225–235, 2005.

[Sti95] C. Stirling. Local model checking games. In *Proceedings of the 6th Conference on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[SW92] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, pages 569–579, London, UK, 1992. Springer.

[SW01] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *Proceedings of the 42th FOCS*, pages 547–555, 2001.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[VAW03] A. Vincent, A. Arnold, and I. Walukiewicz. Games for synthesis of controllers with partial observations. *Theoretical Computer Science*, 303(1):7–34, 2003.

[VJ00] J. Vöge and M. Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *Proceedings of the 12th International Conference on Computer Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.

[Vög00] J. Vöge. *Strategiesynthese für Paritätsspiele auf endlichen Graphen*. PhD thesis, University of Aachen, 2000.

[Ye97] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Wiley-Interscience, 1997.

[Ye05] Y. Ye. A new complexity result on solving the Markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005.

[Ye10] Y. Ye. The simplex method is strongly polynomial for the Markov decision problem with a fixed discount rate. Available at http://www.stanford.edu/~yyye/simplexmdp1.pdf, 2010.

[Zad80] N. Zadeh. What is the worst case behaviour of the simplex algorithm? Technical report, Department of Operations Research, Stanford, 1980.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1–2):135–183, 1998.

[Zie04] G. M. Ziegler. Typical and extremal linear programs. In M. Grotschel and M. W. Padberg, editors, *The Sharpest Cut (MPS-Siam Series on Optimization)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.

[ZP96] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

Index