7 Algorithms for Parity Games

Hartmut Klauck*

School of Mathematics
Institute for Advanced Study, Princeton

7.1 Introduction

It is the aim of this chapter to review some of the algorithmic approaches to the problem of computing winning strategies (resp. of deciding if a player has a winning strategy from a given vertex) in parity games with finite arenas and other two-player games. Parity games are equivalent via linear time reductions to the problem of modal μ-calculus model checking (see Chapters 10 and 9), and this model checking problem plays a major role in computer-aided verification. Furthermore we will see that the problem is not too hard in a complexity-theoretic sense, while no efficient algorithm for it is known so far. Also parity games are the simplest of a whole chain of two-player games for which no efficient solutions are known, further underlining the importance of looking for an efficient algorithm solving this particular problem.

We will explain why the problem of solving parity games lies in UP $\cap$ co-UP, explore its relations to some other games, and describe the theoretically most efficient algorithm for the problem known so far. Furthermore we describe work on more practically oriented algorithms following the paradigm of strategy improvement, for which a theoretical analysis stays elusive so far.

Recall that in a parity game we are given a (finite) graph with vertices labeled by natural numbers. The vertex set is partitioned into vertices in which Player 0 moves and vertices in which Player 1 moves. In an initialized game we are also given a starting vertex. In a play of the game a token is placed on the starting vertex and is then moved over the graph by Player 0 and Player 1, each making their move if the token is on one of their vertices. For simplicity we assume that the graph is bipartite, so that each move from a Player 1 vertex leads to a Player 0 vertex and vice versa. Each player follows some strategy. If the highest priority of a vertex occurring infinitely often in the play is odd, then Player 1 wins, otherwise Player 0 wins. See Chapter 2 for more details.

Exercise 7.1. Show that one can convert any parity game on a nonbipartite game arena into an equivalent parity game on a bipartite arena in linear time.

It is an important (and deep) result that the players may restrict themselves to memoryless strategies (i.e., define their strategy by picking once and for all a neighbor for each of their vertices thus not considering the path on which they arrive there), see Theorem 6.6 in the previous chapter. This also implies that for each vertex one of the players has a winning strategy, so there are no draws!

* Supported by NSF Grant CCR 9987854.

E. Grädel et al. (Eds.): Automata, Logics, and Infinite Games, LNCS 2500, pp. 107-129, 2002.

If the players use memoryless strategies, a play of the game leads after a finite number of steps to a cycle in the underlying graph.

Rabin [148] first showed a complementation lemma for parity automata working on infinite trees (while providing a decidability result for a certain logic) implicitly also proving the determinacy result for parity games. The application of games to the complementation problem is due to Büchi [21]. Gurevich and Harrington [77] gave an abbreviated proof of Rabin's result, and Emerson and Jutla [55] another simplified proof by showing equivalence to the modal μ-calculus model checking problem (in which complementation is trivial). Their result also implies that in fact a player who has a strategy so that he wins in an initialized game also has a memoryless winning strategy. See also [126, 203] for further work on these problems.

The first question arising is, of course, whether one can decide the winner in a parity game efficiently, i.e., whether one can find the player who wins if the play starts at a given vertex, given that this player plays optimally. We are also interested in finding winning strategies. The aforementioned result that the players can restrict themselves to memoryless strategies immediately implies that the following trivial approach is successful (using exponential time): for a given vertex go through all strategies of Player 1. For each such strategy go through all strategies of Player 0 and check who wins. If there is a strategy of Player 1 that wins against all strategies of Player 0 declare Player 1 the winner, otherwise Player 0 wins. It is the main purpose of this chapter to review some more efficient algorithms solving this problem.

Why are we interested in this problem? There are at least two reasons. One is that the problem is deeply related to several important topics. First of all the problem is equivalent to the problem of modal μ-calculus model checking [55, 56], which in turn is of large importance for computer-aided verification. So better algorithms for the problem lead to better model checkers, making more expressive types of logical statements about finite systems checkable by efficient algorithms. The modal μ-calculus was defined first by Kozen in [100], see Chapter 10. Parity games are also at the heart of an interconnection between languages defined by automata operating on infinite trees and monadic second order logic [183], see Chapter 12.

Another important reason to study the problem is its current complexity-theoretic classification. It is known [92] to lie in UP $\cap$ co-UP (and thus "not too far above P", see [142]), but not known to be in P so far, and it is one of the few natural problems so. Trying to find a polynomial algorithm for the problem is a natural pursuit.

In this chapter we describe the best algorithm for the problem known so far (from [93]), show that the problem is in UP $\cap$ co-UP (following [92]), and discuss other promising approaches to get better algorithms for the problem, mainly the strategy improvement approach first defined in [85], and employed in a completely combinatorial algorithm given in [191].

Further we discuss the connection of the game to other games of infinite duration played on graphs, and see that it is the least difficult to solve of a series of such games all lying in NP$\cap$co-NP. So it is the most natural candidate to attack!

This chapter is structured as follows: in the next section we introduce several games on graphs considered in the chapter and explore some of their properties. In Section 7.3 we deduce a first simple algorithm for the problem based on [204], which is already quite good in both time and space complexity. In Section 7.4 we show why the problem is in UP ∩ co-UP. In Section 7.5 we describe the efficient algorithm due to Jurdziński [93], which yields approximately a quadratically improved runtime compared to the simple algorithm (while maintaining the space complexity). Section 7.6 discusses promising and more practical approaches based on strategy improvement. Section 7.7 collects a few conclusions.

Note that throughout the chapter logarithms are always base two, and that space complexity is measured in the logarithmic cost measure (since we will be dealing with problems involving weights). Regarding time complexity we will stick to the measure in which elementary arithmetic operations cost unit time, however. Also note that we will consider max-parity games throughout the chapter, except in Section 7.5, where we will consider min-parity games to save some notation.

7.2 Some More Infinite Games

To describe our algorithms for the solution of parity games and the containment of this problem in UP ∩ co-UP, we will take a detour via some other two-player games, which will be introduced now. Note that most of these games are no games of chance, so no randomness is used in playing these games, just like e.g. in the game of chess. Picking a good strategy in such a game may only be hard because it takes a lot of time to do so! Also all these are games of full information, in which all relevant data are accessible to both players as opposed to e.g. most card games.

For definitions of infinite games, strategies, winning regions and related notions we refer to Chapter 2.

The first kind of games we consider are parity games as defined in Chapter 2. Recall the memoryless determinacy theorem for parity games, Theorem 6.6 in Chapter 6.

Another natural kind of games are *mean payoff games* [49].

Definition 7.1. A **mean payoff game** is a quadruple $(\mathcal{A}, \nu, d, w)$, where $\mathcal{A}$ is an arena without dead ends, ν and d are natural numbers, and $w : E \to \{-d, \dots, d\}$ assigns an integer weight to each edge.

Player 0 wins a play $v_0 v_1 \cdots$, if

$$\liminf_{t \to \infty} \frac{1}{t} \sum_{i=1}^{t} w(v_{i-1}, v_i) \geq \nu.$$

We also refer to the above limit as the *value* that Player 0 wins from Player 1 after the play.

Exercise 7.2. Extend the above definition so that dead ends are allowed. Do this in a way so that both games are equivalent.

[147] and [92] describe polynomial time reductions from parity games to mean payoff games. We will show how to perform such a reduction in the next section. So parity games are not harder to solve than mean payoff games. Again it is known that memoryless strategies suffice for the players of mean payoff games. Surprisingly the proof is much easier than in the case of parity games.

Theorem 7.2 ([49]). *Let $(\mathcal{A}, \nu, d, w)$ be a mean payoff game. Then Player 0 has a winning strategy from a set of vertices iff Player 0 has a memoryless winning strategy from that set.*

More precisely, for each vertex v_0 there is a number $\nu(v_0)$, called the value of v_0, such that

(a) *Player 0 has a memoryless strategy so that for every play $v_0 v_1 \cdots$ in which he follows this strategy*

$$\liminf_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} w(v_{i-1}, v_i) \geq \nu(v_0).$$

(b) *Player 1 has a memoryless strategy so that for every $v_0 v_1 \cdots$ in which she follows this strategy*

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} w(v_{i-1}, v_i) \leq \nu(v_0).$$

The above theorem allows us to speak of an *optimal strategy*, which means a strategy that ensures for all plays (starting at some vertex v) that the corresponding player wins at least the value $\nu(v)$.

To obtain the above result Ehrenfeucht and Mycielski also introduce a finite variant of mean payoff games in which the play stops as soon as a loop is closed and then the payoff of that loop is analyzed. Both games turn out to be basically equivalent [49].

In the next section we will show how to solve mean payoff games in pseudopolynomial time, i.e., in time polynomial in the size of the graph and the unary encoded weights. Together with the reduction from parity games this yields an algorithm for parity games which is already quite good.

The next games we consider are *discounted* payoff games [204]. Here the importance of the weights decreases by a constant factor each time step in a play. So intuitively only a finite beginning of the play is important. Basically this can be viewed as yet another kind of averaging. The game will be important for technical reasons.

Definition 7.3. A **discounted payoff game** is a tuple $(\mathcal{A}, \nu, d, w, \lambda)$ where $\mathcal{A}$ is an arena without dead ends, ν and d are natural numbers, $w : E \to \{-d, \ldots, d\}$ assigns an integer weight to each edge, and $0 < \lambda < 1$ is the discount.

Player 0 wins a play $v_0 v_1 \cdots$, if

$$(1-\lambda) \sum_{i=0}^{\infty} \lambda^i w(v_i, v_{i+1}) \geq \nu.$$

We also refer to the above left hand side as the value that Player 0 wins from Player 1 after the play.

The correction term $(1 - \lambda)$ arises to make sure that the value of a game using only edges of weight a is also a.

Zwick and Paterson prove that for each vertex in a discounted payoff game one of the players has a memoryless winning strategy. We will see the reason for this in Section 7.4. Furthermore we will see in that section that mean payoff games can be reduced in polynomial time to discounted payoff games. Note however that the proofs for the facts that memoryless winning strategies exist become simpler with each game defined so far (and that such a result for a more difficult game does not immediately imply the corresponding result for the easier game).

The most general games we mention are the **simple stochastic games** defined by Condon [40]. In these finite games the vertex set is partitioned into three sets of vertices: vertices in which Player 0 moves, in which Player 1 moves, and random vertices, in which a random successor is chosen, plus two vertices in which 1 is paid by Player 0 to Player 1 resp. 0 is paid by Player 0 to Player 1 (and the game ends). The expected amount paid to Player 1 is the result of the game. Zwick and Paterson [204] show that discounted payoff games can be reduced to simple stochastic games. So these are the most difficult to solve of the games considered here. Moreover they are the only games of chance we consider! Still it is possible to decide in NP∩co-NP whether the payoff of Player 1 exceeds a certain threshold. The reduction from parity games to simple stochastic games that results increases the game arena only by a constant factor. Using an algorithm by Ludwig [117], which solves simple stochastic games with fan-out 2, and the reductions we get the following corollary.

Corollary 7.4. *There is a randomized algorithm which computes the winning regions of a given parity game with m edges in expected time $2^{O(\sqrt{m})}$.*

This is the best algorithm we know if the number of different priorities assigned to vertices is larger than $\sqrt{m}$. The algorithm is notably subexponential, if the graph is sparse. The time bound is understood as the expected value of the running time (over coin tosses of the algorithm) in the worst case (over all inputs).

7.3 A Simple Algorithm

In this section we want to describe a relatively simple algorithm for solving parity games, or rather mean payoff games. The approach can also be adapted to solve discounted payoff games.

Let us consider a parity game $(\mathcal{A}, \Omega)$ where Ω assigns d different priorities to the vertices. Our whole objective is to decrease the dependence of the runtime on d, see Section 6.4 for the first algorithm in this direction presented here. Actually, for very large d our algorithms will not be better than the trivial exponential

time algorithm testing all strategies. Why do we consider this parameter as important? In applications to model checking this parameter gives us the depth of nested fixed points used in expressions we want to check. The weaker the dependence on d is, the more complicated formulae can be checked, e.g. for all constant d our algorithm is polynomial time, which is not so for the trivial algorithm. To see the effect of this compare with Theorem 10.19. Another concern will be space complexity, which we prefer small as well.

In a first step we give the reduction to mean payoff games as in [92]. Afterwards we describe the algorithm of Zwick and Paterson [204] for these games and analyze its performance for the original parity games. The algorithm will be finding fixed points of a certain natural function, a property which also holds (for a less obvious function) for the more complicated algorithm in section 7.5.

Lemma 7.5. *A parity game on n vertices using d different priorities can be reduced in polynomial time to a mean payoff game on n vertices using weights from the set $\{-n^{d-1}, \ldots, n^{d-1}\}$, and using the same game arena.*

Moreover winning strategies of the parity game are winning strategies of the mean payoff game and vice versa.

Proof. Suppose our parity game is $(\mathcal{A}, \Omega)$. W.l.o.g. the priorities are $\{0, \ldots, d-1\}$. The mean payoff game uses the same arena. An edge originating at a vertex v with priority $i = \Omega(v)$ receives the weight $w(v,u) = (-1)^i n^i$. Let $\nu = 0$. Clearly all weights lie in the range $\{-n^{d-1}, \ldots, n^{d-1}\}$. This defines our mean payoff game $(\mathcal{A}, 0, n^d, w)$.

We claim that the value of the mean payoff game exceeds 0 for a pair of memoryless strategies iff the same strategies lead to a play of the game in which the highest priority vertex occurring infinitely often has an even priority.

W.l.o.g. we may assume that the players use memoryless strategies when playing the mean payoff game, see Theorem 7.2. Then a play corresponds to a path ending in a cycle. In the limit defining the value of the play the weights on the initial segment before the cycle contribute zero. So the value of the game is positive iff the sum of the weights on the cycle is positive. The weights are from the set $\{-n^{d-1}, n^{d-2}, -n^{d-3}, \ldots, -n, 1\}$, assuming for concreteness that d is even. Assume the weight w_{max} with the largest absolute value appearing on the cycle is positive. Then the sum of the weights on the cycle is at least $w_{max} - (n-1)w_{max}/n > 0$, since there are at most $n-1$ edges with weights not equal to w_{max} in the cycle. The maximal weight is on an edge originating from the vertex of highest priority, which must be even. Symmetrically if the weight of largest absolute value is negative, the highest priority vertex must be odd.

So the mean payoff game and the parity game behave in the same way for each pair of memoryless strategies, thus they are equivalent, and have the same winning regions, and the same strategies lead to a win. □

Now we show how to solve mean payoff games efficiently if the weights are small.

Theorem 7.6. *Given a mean payoff game $(\mathcal{A}, \nu, d, w)$ where the arena has n vertices and m edges, the winning region for Player 0 can be computed in time $O(n^3md)$ and space $O(n \cdot (\log d + \log n))$.*

Proof. It is our goal to find the values of the vertices efficiently. This immediately gives us the winning region. Let $\nu_k(v)$ denote the following value: the players play the game for k steps starting from vertex v (so they construct a path of length k), then $\nu_k(v)$ denotes the sum of the edge weights traversed if both players play optimally.

We want to compute the values $\nu(v)$ as the limit over k of the $\nu_k(v)$. First let us characterize the latter value.

For every $v \in V$:

$$\nu_k(v) = \begin{cases} \max_{(v,u)\in E}\{w(v,u) + \nu_{k-1}(u)\} & \text{if } v \in V_0, \\ \min_{(v,u)\in E}\{w(v,u) + \nu_{k-1}(u)\} & \text{if } v \in V_1. \end{cases} \tag{7.1}$$

Clearly $\nu_0(v) = 0$ for all $v \in V$. Using this recursion we can easily compute $\nu_k(v)$ for all $v \in V$ in time $O(km)$. Recall that we allow arithmetic operations in unit time. Now we investigate how quickly $\nu_k(v)/k$ approaches $\nu(v)$.

Lemma 7.7. *For all $v \in V$:*

$$\nu_k(v)/k - 2nd/k \leq \nu(v) \leq \nu_k(v)/k + 2nd/k.$$

First let us conclude the theorem from the above lemma. We compute all the values $\nu_k(v)$ for $k = 4n^3d$. This takes time $O(n^3md)$. All we have to store are the $\nu_i(v)$ for the current i and $i-1$. These are numbers of $O(\log(kd))$ bits each, so we need space $O(n(\log d + \log n))$.

Now we estimate $\nu(v)$ by $\nu'(v) = \nu_k(v)/k$. Clearly

$$\nu'(v) - \frac{1}{2n(n-1)} < \nu'(v) - \frac{2nd}{k} \leq \nu(v) \leq \nu'(v) + \frac{2nd}{k} < \nu'(v) + \frac{1}{2n(n-1)}.$$

Now $\nu(v)$ can be expressed as the sum of weights on a cycle divided by the length of the cycle due to Theorem 7.2, and is thus a rational with denominator at most n. The minimal distance between two such rationals is at least $\frac{1}{n(n-1)}$, so there is exactly one rational number of this type in our interval. It is also easy to find this number. We can go through all denominators l from 1 to n, estimate $\nu(v)$ as $\lceil \nu'(v) \cdot l \rceil / l$ and $\lfloor \nu'(v) \cdot l \rfloor / l$, if one of these numbers is in the interval, we have found the solution. This takes about $O(n)$ steps.

Knowing the vector of values of the game it is easy to compute winning strategies by fixing memoryless strategies that satisfy equation 7.1.

Proof of Lemma 7.7. It is proved in [49] that the values of vertices in a mean payoff game and in its following finite variant are equal: the game is played as the infinite mean payoff game, but when the play forms a cycle the play ends and the mean value of the edges on that cycle is paid to Player 0. Also the optimal such value can be obtained using the same memoryless strategies as in the infinite case.

Let f_0 be a memoryless strategy of Player 0 that achieves the maximal values for all vertices (against optimal strategies of Player 1) in the finite version of the game. Let Player 1 play according to some (not necessarily memoryless) strategy. We show that the value of a k step play starting in v is at least $(k - (n-1)) \cdot \nu(v) - (n-1)d$. Consider any play of length k. The edges of the play are placed consecutively on a stack. Whenever a cycle is formed, the cycle is removed from the stack. Since the edges lying on the stack directly before the removal of the cycle correspond to a play which has just formed its first cycle, the mean value of the edges on the cycle is at least $\nu(v)$, because of the optimality of f_0 against all strategies of Player 1 in the finite version of the game. This process continues, until the play is over and the stack contains no more cycles. In this case there are at most $n-1$ edges on the stack. The weight of each such edge is at least $-d$. Thus the value of the k step play is always at least $(k-(n-1)) \cdot \nu(v) - (n-1)d > k \cdot \nu(v) - 2nd$. So we know there is a memoryless strategy for Player 0, so that he wins at least $k \cdot \nu(v) - 2nd$ in the k step play, no matter what Player 1 does. The other inequality is proved similarly. □

Note that the above proof uses the memoryless determinacy theorem for mean payoff games [49].

Exercise 7.3. (1) Prove that mean payoff games and their finite variants are equal in the above sense. Hint: Use the above idea with the stack.
(2) Use 1. to show that mean payoff games enjoy memoryless determinacy.

Corollary 7.8. *Given a parity game $(\mathcal{A}, \Omega)$ where d different priorities are assigned to vertices, the winning region and strategy of Player 0 can be computed in time $O(n^{d+2}m)$ and space $O(d \cdot n \log n)$.*

So there is a rather efficient solution to the problem if d is small. In section 5 we will see how to further reduce the dependence on d.

7.4 The Problem Is in UP ∩ co-UP

In this section we consider the problem from a complexity-theoretic point of view. First observe that the problem of deciding whether a given vertex belongs to the winning region of Player 0 in a given parity game is in NP: simply guess a memoryless strategy for Player 0. Then remove all edges which are not consistent with the strategy. Then one has to determine whether Player 1 can win if Player 0 uses his strategy, which comes down to testing whether there is no path from the designated vertex to a cycle whose highest priority is odd. This is decidable in deterministic polynomial time.

Exercise 7.4. Show that the following problem can be decided in polynomial time: input is a game arena in which Player 0's strategy is fixed (all vertices of Player 0 have outdegree 1) plus a vertex in the arena. Is there a path from the vertex to a cycle in which the highest priority is odd?

Furthermore since each vertex is either in Player 0's or in Player 1's winning region, the same argument gives an NP algorithm for deciding Player 1's winning region, which is a co-NP algorithm for deciding Player 0's winning region. Thus parity games are solvable in NP∩co-NP. This strongly indicates that the problem is not NP-complete, since otherwise NP would be closed under complement and the polynomial hierarchy would collapse (see e.g. [142]).

Now we review a result by Jurdziński [92] saying that the complexity of the problem is potentially even lower. First we define (for completeness) the complexity class UP (see [142]).

Definition 7.9. A problem is in the class UP, if there is a polynomial time nondeterministic Turing machine, such that for each input that is accepted exactly one computation accepts.

The class UP is believed to be a rather weak subclass of NP.

Our plan to put parity games into UP is as follows: we again use the reduction to mean payoff games. Then we show how to reduce these to discounted payoff games. There is an algorithm due to Zwick and Paterson for solving these games in a very similar fashion to the one described in the previous section. This gives us a set of equations whose unique solution is the vector of values of the game. Furthermore using simple facts from linear algebra we prove that these solutions can be specified with very few bits. Thus we get our unique and short witnesses. Again the argument for co-UP is symmetric.

First we state the following observation from [204], which says that a mean payoff game yields always a discounted payoff game of almost the same value, if the discount factor is chosen large enough. The proof is similar to the proof of Lemma 7.7.

Lemma 7.10. *Let $(\mathcal{A}, \nu, d, w)$ be a mean payoff game with an arena on n vertices, played beginning in vertex v. Then rounding the value of the discounted payoff game $(\mathcal{A}, \nu, d, w, \lambda)$ with $\lambda \geq 1 - 1/(4n^3/d)$ to the nearest rational with denominator smaller than n yields the value of the mean payoff game.*

The following characterization of the values of vertices in a discounted payoff game will be useful [204].

Lemma 7.11. *The value vector $\bar{\nu} = (\nu(v_1), \ldots, \nu(v_n))$ containing the values of vertices in a discounted payoff game equals the unique solution of the following system of n equations*

$$x_v = \begin{cases} \max_{(v,u)\in E}\{(1-\lambda)\cdot w(v,u) + \lambda x_u\} & \text{if } v \in V_0, \\ \min_{(v,u)\in E}\{(1-\lambda)\cdot w(v,u) + \lambda x_u\} & \text{if } v \in V_1. \end{cases} \tag{7.2}$$

Proof. Let F be a function that maps a vector $\bar{x}$ to the vector $\bar{y}$ such that

$$y_v = \begin{cases} \max_{(v,u)\in E}\{(1-\lambda)\cdot w(v,u) + \lambda x_u\} & \text{if } v \in V_0, \\ \min_{(v,u)\in E}\{(1-\lambda)\cdot w(v,u) + \lambda x_u\} & \text{if } v \in V_1. \end{cases}$$

Then we are interested in vectors $\bar{x}$ with $\bar{x} = F(\bar{x})$, the fixed points of F. Let $||\bar{y}||_\infty$ denote the maximum norm, then

$$\forall \bar{y}, \bar{z} : ||F(\bar{y}) - F(\bar{z})||_\infty \leq \lambda ||\bar{y} - \bar{z}||_\infty.$$

Since $0 < \lambda < 1$ we have that F is a contracting function (with respect to the maximum norm). Thus the limit $\bar{x} = \lim_{n\to\infty} F^n(0)$ exists and is the unique solution to $\bar{x} = F(\bar{x})$.

Now Player 0 can use the following strategy, provided he knows the vector $\bar{x} = F(\bar{x})$: at vertex v choose the neighboring vertex u that maximizes $(1 - \lambda)w(v, u) + \lambda x_u$. Then Player 0 wins at least x_v in a play starting at v. On the other hand Player 1 may fix a strategy analogously so that her loss is also at most x_v. Thus the solution of $F(\bar{x}) = \bar{x}$ is the vector of values of the game. □

Obviously this lemma leads to a UP algorithm for the solution of discounted payoff games, if the vector of values can be described by short numbers. Then we can just guess these numbers and verify that the equations are satisfied. What is a short number? The number must be representable using a polynomial number of bits in the size of the game. The size of the game is the length of a description of the game, including edge weights and λ.

But first let us note that the strategies obtained from the system of equations are indeed memoryless. The proof of Lemma 7.11 does not presuppose such a memoryless determinacy result.

Corollary 7.12. *Let $(\mathcal{A}, \nu, d, w, \lambda)$ be a discounted payoff game. Then Player 0 [Player 1] has a winning strategy from a set of vertices iff Player 0 [Player 1] has a memoryless winning strategy from that set.*

Lemma 7.13. *The solution of the equations 7.2 can be written with polynomially many bits.*

Proof. Let N be the size of the binary representation of the discounted payoff game. Let $\bar{\nu}$ be the unique solution of the equations. Then this vector can be written

$$\bar{\nu} = (1 - \lambda) \cdot \bar{w} + \lambda \cdot Q \cdot \bar{\nu},$$

where $\bar{w}$ is a suitable vector containing weights $w(v, u)$, and Q is a 0,1-matrix containing only a single one per row. Note that in order to write down this system of equations one has to know the winning strategy.

Assume that $\lambda = a/b$ is a rational included in the game representation, with integers a, b satisfying $\log a, \log b < N$. Let $A = b \cdot I - a \cdot Q$ for the identity matrix I, then A is an integer matrix with at most two nonzero integer entries per row.

The above equation can then be rewritten

$$A \cdot \bar{\nu} = (b - a) \cdot \bar{w}.$$

Due to Cramer's rule the solution of this system can be written as the vector containing $\det A_v / \det A$ on position v where A_v is obtained from A by replacing column v with $(a - b) \cdot \bar{w}$.

The entries of A and A_v are bounded in absolute value by 2^N. This implies that the determinants of the matrices A, A_v are at most $2^{O(N \cdot |V|)}$. But then the solution of the system of equation can be written by using a polynomial number of bits in the length N. □

So we get the following.

Corollary 7.14. *Deciding whether a vertex is in the winning region of Player 0 is possible in UP ∩ co-UP for parity games, mean payoff games, and discounted payoff games.*

Exercise 7.5. Formally describe how a nondeterministic Turing machine can solve the decision problem associated to parity games unambiguously in polynomial time.

Exercise 7.6. Devise an algorithm for discounted payoff games similar to the algorithm described in Theorem 7.6.

7.5 A Better Algorithm

Now we describe the best algorithm for the solution of parity games known so far, again due to Jurdziński [93]. The time complexity of the algorithm is $O(d \cdot m \cdot \frac{n}{\lfloor d/2 \rfloor}^{\lfloor d/2 \rfloor})$ for min-parity games with n vertices, m edges, and $d \geq 2$ different priorities. An algorithm with comparable time complexity has been given by Seidl in [161]. But as opposed to previous algorithms Jurdziński's algorithm uses only space polynomially depending on d, namely $O(dn \log n)$, when achieving this time bound (note that we use the logarithmic measure for space complexity).

The algorithm is fairly simple to describe and analyze after several technical concepts have been explained.

First note that we will apply comparisons in the following to tuples of natural numbers, referring to their lexicographical ordering. Furthermore we will use symbols like $<_i$, referring to the lexicographical ordering when restricted to the first i components of a tuple (ignoring the other components). So e.g. $(2, 4, 3) < (2, 4, 5)$, but $(2, 4, 3) =_2 (2, 4, 5)$. Denote $[i] = \{0, \ldots, i-1\}$.

For a technical reason in this section Player 0 wins, if the *lowest* priority occurring infinitely often is even, i.e., we are considering min-parity games. The max-parity game can obviously be reduced to this variant and vice versa. Also we exclude dead ends from the game graph, see Exercise 2.8.

Exercise 7.7. How can we reduce min-parity to max-parity games?

Fix a memoryless strategy of one player. This can be regarded as throwing out all edges which are not consistent with this strategy. The remaining game graph will be called a **solitaire game**, since the game is now played by one player only. Obviously it suffices for this player to find a path leading to a cycle in which the lowest vertex priority makes him win the game! So call a cycle **even**, if the lowest priority of a vertex in the cycle is even, and otherwise **odd**.

Furthermore call a memoryless strategy f_0 of Player 0 **closed** on a set of vertices W, if every play starting in W and consistent with f_0 stays in W, i.e., if for all $v \in W \cap V_0 : f_0(v) \in W$ and for all $v \in W \cap V_1$ and all $u \in vE : u \in W$.

Now we see a simple condition that makes a player win:

Lemma 7.15. *Let f_0 be a memoryless strategy of Player 0 which is closed on a set W. Then f_0 is a winning strategy from all vertices in W iff all simple cycles in the restriction of the solitaire game of f_0 to the vertices in W are even.*

Proof. From each vertex either Player 1 or Player 0 has a winning strategy. If Player 1 has a winning strategy, then this can be assumed to be memoryless. So assume Player 0 plays according to f_0 and consider the resulting solitaire game. Then Player 1 can win from a vertex v iff she can fix an edge for each vertex so that the resulting path from v ends in a simple cycle which is odd. If no such cycle exists, Player 1 cannot win (and Player 0 wins). If such a cycle exists, then Player 1 wins iff she can find a path to that cycle. This happens at least for all vertices on that cycle, so there are vertices where f_0 is not winning. □

The key notion in the algorithm will be a *parity progress measure*. These are labelings of the vertices of graphs with tuples of natural numbers having certain properties. First we consider such labelings for solitaire games.

Definition 7.16. Let $(\mathcal{A}, \Omega)$ be a solitaire game with vertex priorities $\Omega(v) \leq d$. A function $\rho : V_0 \cup V_1 \to \mathbb{N}^{d+1}$ is a **parity progress measure** for the solitaire game, if for all edges (v, w):

(a) $\rho(v) \geq_{\Omega(v)} \rho(w)$ if $\Omega(v)$ is even.
(b) $\rho(v) >_{\Omega(v)} \rho(w)$ if $\Omega(v)$ is odd.

The intuition behind the above definition is best explained through the following lemma.

Lemma 7.17. *If there is a parity progress measure for a solitaire game $\mathcal{G} = (\mathcal{A}, \Omega)$, then all simple cycles in $\mathcal{G}$ are even.*

In particular in this case Player 0's strategy used to derive $\mathcal{G}$ is winning.

Proof. Let ρ be a parity progress measure for a solitaire game $\mathcal{G}$. Suppose there is an odd cycle $v_1, \ldots, v_l$ in $\mathcal{G}$, let $i = \Omega(v_1)$ be the lowest priority on the cycle, which is odd. Then according to the definition of a parity progress measure $\rho(v_1) >_i \rho(v_2) \geq_i \cdots \geq_i \rho(v_l) \geq_i \rho(v_1)$, which is a contradiction. □

So parity progress measures are witnesses for winning strategies. It is true that the above condition can also be reversed, i.e., if Player 0 wins from all vertices, then there is a parity progress measure. But an important feature will be that we can show the reverse condition while considering only a suitably bounded number of parity progress measures. We will then be able to replace the search for a winning strategy by the search for a parity progress measure from a relatively small set.

To define this "small" set let $\mathcal{G} = (\mathcal{A}, \Omega)$ be a solitaire game and Ω be a function mapping vertices to $\{0, \ldots, d\}$, and let V_i denote the set of vertices having priority i. By definition there are $d+1$ such sets. Instead of using $\mathbb{N}^{d+1}$ as the range of values of our parity progress measure we will use a set $M_{\mathcal{G}}$ defined by

$$M_{\mathcal{G}} := [1] \times [|V_1| + 1] \times [1] \times [|V_3| + 1] \times [1] \times \cdots \times [1] \times [|V_d| + 1],$$

assuming for simplicity that d is odd.

Lemma 7.18. *If all simple cycles in a solitaire game $\mathcal{G} = (\mathcal{A}, \Omega)$ are even, then there is a parity progress measure $\rho\colon V \to M_{\mathcal{G}}$.*

Proof. We define the parity progress measure explicitly from the solitaire game $\mathcal{G}$ (as opposed to the inductive proof given in [93]). Let $a_i(v)$ be the maximal number of vertices with priority i occurring on any path in $\mathcal{G}$ starting in v and containing no vertex with priority smaller than i. This value is infinite, if infinitely many vertices with priority i occur on some path with no smaller priority occurring on that path. If v has priority smaller than i or there is no path featuring a vertex with priority i but no smaller priority, then $a_i(v) = 0$.

We then set $\rho(v) = (0, a_1(v), 0, a_3(v), 0, \ldots, 0, a_d(v))$ and claim that this is a parity progress measure with the desired property.

First assume that some $a_i(v)$ is not finite for some odd i. Then there is an infinite path starting at v such that the path contains no vertex with lower priority than i, but infinitely many vertices with priority i. Thus the path must contain some vertex with priority i twice, and we can construct a cycle with least priority i, a contradiction to the assumption of the lemma.

Now we show that we have actually defined a mapping $\rho\colon V \to M_{\mathcal{G}}$. Assume that $a_i(v)$ is larger than the number of vertices with priority i. Due to the definition of $a_i(v)$ there is a path originating in v such that $a_i(v)$ vertices with priority i show up before a vertex with priority smaller than i. If $a_i(v)$ is larger than the number of vertices with priority i, such a vertex occurs twice. Consequently there is a cycle containing as least priority i, again a contradiction.

It remains to show that we defined a parity progress measure. Let (v, w) be any edge and i any odd number. If $i = \Omega(v)$, then $a_i(v) = a_i(w) + 1$. For all smaller odd i we get $a_i(v) \geq a_i(w)$, because the edge (v, w) extended by a path starting in w that contains k vertices with priority i but no smaller priority, yields a path starting in v that contains k vertices with priority i but no smaller priority. Thus for all v with odd priority $\rho(v) >_{\Omega(v)} \rho(w)$ and for all v with even priority $\rho(v) \geq_{\Omega(v)} \rho(w)$. □

The construction allows a nice interpretation of the constructed parity progress measure. The tuple assigned to a vertex contains for all odd priorities the maximal number of times this priority can be seen if Player 1 moves over the graph, until a vertex with smaller priority is seen. Note that this interpretation is not applicable to all parity progress measures.

Exercise 7.8. Find a parity game and a parity progress measure for which the above intuition is not true.

What have we achieved by now? Given a strategy of one player we can construct the solitaire game. Then a parity progress measure for such a graph exists if and only if Player 0 has a winning strategy from all vertices. Also parity progress measures from a relatively small set suffice for this. Our current formulation does not allow to deal with graphs in which both winning regions are nonempty. Secondly we have to extend our notion of a progress measure to deal with game arenas, i.e., to graphs in which Player 1 has more than one option to choose a strategy.

Now consider again the construction of the parity progress measure given in the proof of the above lemma. If we drop the condition that all simple cycles are even, then some of the values $a_i(v)$ are infinite. Clearly, if $a_i(v) = \infty$, then there is a path from v that sees infinitely many odd i and no smaller priorities, so Player 1 might just walk that path and win. If, on the other hand, there is no i with $a_i(v) = \infty$, then Player 1 cannot win from v, because all paths starting in v eventually reach an even priority occurring infinitely often. Note that we excluded dead ends from game arenas in this section. We have a clear distinction of the winning regions in a solitaire game.

So we introduce one more symbol into $M_{\mathcal{G}}$. Let $M_{\mathcal{G}}^{\top}$ denote $M_{\mathcal{G}} \cup \{\top\}$ where $\top$ is larger than all elements of $M_{\mathcal{G}}$ in the order $>_i$ for all i. If we identify all $\rho(v)$ containing the value ∞ at least once with $\top$, we get an **extended parity progress measure** for solitaire games where the vertices with label $\top$ constitute the winning region of Player 1.

To extend the notion of a progress measure to game arenas, we simply demand that for each vertex in which Player 0 moves, there is at least one neighbor satisfying a progress relation.

Definition 7.19. Let $\mathrm{prog}(\rho, v, w)$ denote the least $m \in M_{\mathcal{G}}^{\top}$ such that $m \geq_{\Omega(v)} \rho(w)$, and, if $\Omega(v)$ is odd, then $m >_{\Omega(v)} \rho(w)$ or $m = \rho(w) = \top$.

A function $\rho : V \to M_{\mathcal{G}}^{\top}$ is a **game progress measure**, if for all $v \in V$ the following two conditions hold:

(a) if $v \in V_0$ then $\rho \geq_{\Omega(v)} \mathrm{prog}(\rho, v, w)$ for some edge (v, w).
(b) if $v \in V_1$ then $\rho \geq_{\Omega(v)} \mathrm{prog}(\rho, v, w)$ for all edges (v, w).

Furthermore let $||\rho|| = \{v \in V : \rho(v) \neq \top\}$.

Let us explain the intuition behind the above definition. A parity progress measure captures the existence of a winning strategy for Player 0 from all vertices in a solitaire game. The key feature of a parity progress measure is that it decreases on edges originating from vertices with odd parity and does not increase on other edges (with respect to some order depending on the priorities of vertices).

In a game arena (as opposed to a solitaire game) the strategy of Player 0 is not fixed, i.e., usually vertices belonging to both players have outdegree larger

than one. Also there are usually nonempty winning regions for Player 0 and for Player 1.

A game progress measure is defined with respect to Player 0. For each vertex the above "decreasing" property must hold for *some* edge, if the vertex belongs to Player 0, and for *all* edges, if the vertex belongs to Player 1. So we demand the *existence* of an edge with the "decreasing" property for the multiple edges originating in vertices belonging to Player 0. Furthermore we have introduced the $\top$ element to deal with vertices in the possibly nonempty winning region of Player 1. Note that in case we have assigned the top element to a vertex we cannot demand that an edge leading to that vertex decreases the progress measure. That is the reason for introducing the complications in the prog-notation.

If we restrict a game graph with a game progress measure ρ to the vertices in $||\rho||$, we get a solitaire game with a parity progress measure. Assume that this parity progress measure equals the one constructed in the proof of Lemma 7.18. In this case we get the following interpretation of the game progress measure: the component $\rho_i(v)$ for some odd i and some $v \notin ||\rho||$ contains the number of times Player 1 may force Player 0 to see priority i before some smaller priority occurs, if Player 0 tries to minimize that value and Player 1 tries to maximize it. Unfortunately this intuition does not hold true for all possible parity progress measures as noted before, see Exercise 7.8

It is easy to find a game progress measure by assigning $\top$ to all vertices. This measure does not tell us much. But it will turn out that we can try to maximize the size of $||\rho||$ and find the winning region of Player 0.

First we define a strategy from the measure ρ. Let $f_0^\rho \colon V_0 \to V$ be a strategy for Player 0 defined by taking for each vertex v a successor w which minimizes $\rho(w)$.

Lemma 7.20. *If ρ is a game progress measure, then f_0^ρ is a winning strategy for Player 0 from all vertices in $||\rho||$.*

Proof. Restrict the game arena to the vertices in $||\rho||$. If we now fix the strategy f_0^ρ we get that ρ is a parity progress measure on the resulting solitaire game. This implies that all simple cycles in the solitaire game are even (using Lemma 7.17) and the strategy wins from all vertices in $||\rho||$, if f_0^ρ is closed on $||\rho||$ due to Lemma 7.15. But this is true, since the strategy would violate the conditions of its game progress measure if it would use an edge leading from $||\rho||$ to a vertex labeled $\top$ in the solitaire game. □

So we are after game progress measures with large $||\rho||$.

Lemma 7.21. *For each parity game there is a game progress measure ρ such that $||\rho||$ is the winning region of Player 0.*

Proof. Since each vertex is either in the winning region of Player 0 or of Player 1 we can assume that a winning strategy for Player 0 never leaves his winning set, otherwise Player 1 could win after such a step. Fixing a memoryless winning strategy with this winning region and restricting the vertices to the winning

region yields a solitaire game $\mathcal{G}$ containing no simple even cycle. Thus due to Lemma 7.18 there is a parity progress measure ρ with values in $M_{\mathcal{G}}$. If we now set $\rho(v) = \top$ for all vertices outside of $\mathcal{G}$ we get a game progress measure as demanded. □

We are now almost done. Given a game, we have to find a game progress measure that has a maximal number of vertices which do not have value $\top$. But it is actually not really clear how to compute game progress measures at all, except trivial ones.

So we take the following approach. We consider the set of *all* functions $V \to M_{\mathcal{G}}^{\top}$. Our goal is to find one such function which is a game progress measure, and in particular one with a maximal winning region. First we define an ordering on these functions. Let ρ, σ be two such functions, then $\rho \sqsubseteq \sigma$, if for all $v \in V$ we have $\rho(v) \leq \sigma(v)$. If also $\rho \neq \sigma$, then we write $\rho \sqsubset \sigma$. With this ordering we have a complete lattice structure on our set of functions. We will define certain monotone operators in this lattice. The game progress measure we are looking for is the least common fixed point of these operators.

We start from a function mapping all vertices to the all zero vector and apply the set of operators that "push the function" towards a game progress measure. Eventually this process will actually stop at a fixed point of the operators.

The applied operators work on one vertex label only, and in the worst case during a run of the algorithm the label of such a vertex may take on all its possible values. But then the number of such steps is no more than n times the number of all labels, which is $n \cdot |M_{\mathcal{G}}^{\top}|$.

Let us define the operators now.

Definition 7.22. The operator $\mathrm{Lift}(\rho, v)$ is defined for $v \in V$ and $\rho : V \to M_{\mathcal{G}}^{\top}$ as follows:

$$\mathrm{Lift}(\rho, v)(u) := \begin{cases} \rho(u) & \text{if } u \neq v, \\ \max\{\rho(v), \min_{(v,w)\in E} \mathrm{prog}(\rho, v, w)\} & \text{if } u = v \in V_0, \\ \max\{\rho(v), \max_{(v,w)\in E} \mathrm{prog}(\rho, v, w)\} & \text{if } u = v \in V_1. \end{cases}$$

The following lemmas are obvious.

Lemma 7.23. *For all $v \in V$ the operator Lift$(\cdot, v)$ is monotone with respect to the ordering $\sqsubseteq$.*

Lemma 7.24. *A function $\rho\colon V \to M_{\mathcal{G}}^{\top}$ is a game progress measure iff it is a simultaneous fixed point of all Lift$(\cdot, v)$ operators, i.e., iff Lift$(\rho, v) \sqsubseteq \rho$ for all $v \in V$.*

Exercise 7.9. Prove the lemmas.

Now we have a correspondence between fixed points and game progress measures. We are interested in a game progress measure inducing the winning region. To find such a measure we will be computing the *least* simultaneous fixed point of all the operators. Due to a theorem of Tarski [175] and Knaster such a least fixed point exists and can be computed in the following way (see also Chapter 20 in the appendix):

We start with the function μ assigning 0 to every vertex. Then as as long as $\mu \sqsubset \text{Lift}(\mu, v)$ for some v, apply the lift operator $\mu := \text{Lift}(\mu, v)$.

When the algorithm terminates, it has found the least simultaneous fixed point of all lift operators. This is a game progress measure, and as we have seen it is easy to derive a strategy for Player 0 from it.

Theorem 7.25. *The winning region of Player 0 and Player 0's winning strategy in a parity game with n vertices, m edges, and $d \geq 2$ different priorities can be computed in time $O(d \cdot m \cdot \left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor})$ and space $O(dn \log n)$.*

Proof. First let us argue that the algorithm actually finds the winning region of Player 0. The computed game progress measure μ is the least simultaneous fixed point of all the lift operators. The strategy f_0^μ induced by μ is a winning strategy on the set of vertices $||\mu||$ due to Lemma 7.20. Therefore $||\mu||$ is a subset of Player 0's winning region. Furthermore $||\mu||$ is the largest set of vertices not assigned $\top$ over all game progress measures. Thus it must be Player 0's winning region due to Lemma 7.21.

Now let us calculate the complexity of the algorithm. The space is very easy to calculate. For each vertex we have to store an element of $M_{\mathcal{G}}^{\top}$, which consists of d numbers from the set $[n]$. Thus space used is $O(d \cdot n \log n)$.

The time can be bounded as follows. The $\text{Lift}(\rho, v)$ operator can be implemented in time $O(d \cdot \mathit{outdegree}(v))$. Every vertex may be lifted at most $|M_{\mathcal{G}}|$ times, so the time is upper bounded by $O(|M_{\mathcal{G}}| \cdot d \cdot \sum_v \mathit{outdegree}(v)) = O(md|M_{\mathcal{G}}|)$, if we ensure that we can always find a liftable vertex in constant time. This is possible by maintaining a queue of liftable vertices. In the beginning we insert all liftable vertices. Later we get a liftable vertex out of the queue, lift it, and test all predecessors of the vertex for liftability. Liftable vertices are marked liftable in an array, and if they change from non-liftable to liftable they are inserted into the queue. These operations are possible within the given time bound.

It remains now to estimate the size of $M_{\mathcal{G}}$. First assume that priority 0 is used, and also assume that there are vertices with priorities i for all $0 \leq i \leq d-1$. If some priority is missing, we can diminish the higher priorities by 2 without changing the game. Then

$$|M_{\mathcal{G}}| = \prod_{i=1}^{\lfloor d/2 \rfloor} (|V_{2i-1}| + 1).$$

We have

$$\sum_{i=1}^{\lfloor d/2 \rfloor} (|V_{2i-1}| + 1) \leq \sum_{i=0}^{d-1} |V_i| \leq n,$$

because there is at least one vertex with every even priority, and there are at most n vertices. Such a product is maximized when all the factors are equal, and can thus be bounded by

$$\left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}.$$

Now assume that priority 0 is not used. Then w.l.o.g. the priorities used are $\{1, 2, \ldots, d\}$. Inspection of the argument shows that it works in this case as well, by switching the roles of the players in the proof and in the algorithm. □

Now let us mention that one has indeed to specify in which order the Lift operators are applied, leading to a possible improvement by using a suitable such order. But Jurdziński has shown [93] that there is an example where for each such order policy the time bound is basically tight.

Exercise 7.10. Consider the following graph $H_{4,3}$ where quadratic vertices belong to Player 1 and all other vertices to Player 0. The numbers in the vertices are the priorities.

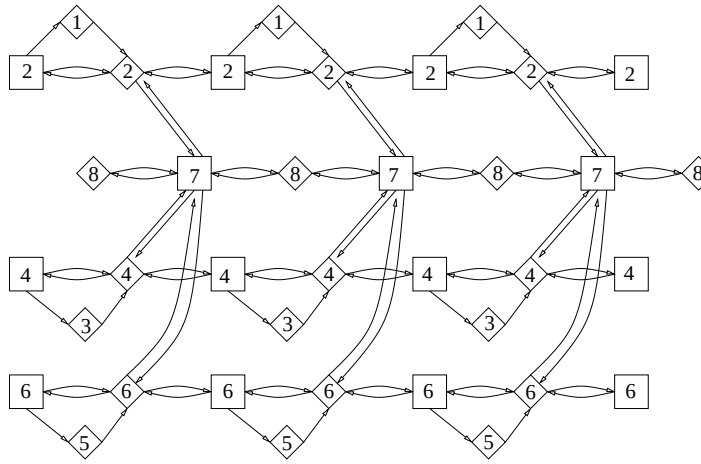

Fig. 7.1. The graph $H_{4,3}$

Show that the vertices with priority 7 are lifted 4^4 times, no matter what lifting policy is used. For this observe that for all vertices except those in the second, indented layer Player 0 has a winning strategy, for all vertices in the second layer Player 1 has a winning strategy, and hence $\top$ is the label assigned to these vertices in the least progress measure. Furthermore show that the Lift operator increases the labels of vertices with priority 7 only to their successors.

Generalize the graph to a family of graphs $H_{\ell,b}$ with $(\ell-1)\cdot(3b+1)+(2b+1)$ vertices and priorities from the set $\{1, \ldots, 2\ell\}$. Show that some vertices are lifted $(b+1)^\ell$ times no matter what lifting policy is used. Conclude that the running time bound of Theorem 7.25 is basically tight, in particular that the running time is exponential in d.

7.6 The Strategy Improvement Approach

In this section we briefly review another promising approach to solve parity games, which should also be useful in implementations. A rigorous theoretical analysis of this approach is missing, however.

The approach follows a quite general paradigm called **strategy improvement**. In this approach one starts with a pair of strategies for Player 0 and Player 1, and applies some simple operation on one player's strategy to "improve" it. Then the other player responds with an optimal strategy given the first player's strategy. This process is iterated. Of course it has to be made precise, what a better strategy is.

Strategy improvement was first proposed by Hoffman and Karp in 1966 [85] for stochastic games. Their algorithm proceeds basically by starting from any pair of strategies, and in each iteration considers a vertex, that can be "switched", in our context a vertex at which changing the strategy "improves" the solution. Then the player whose strategy is not yet changed responds with an optimal strategy according to the other player's strategy. This is done until no such iteration is possible. In this case both strategies are optimal. One has to show in particular, how to compute an optimal response strategy. Furthermore is must be made clear what an improved strategy is (this is easy for stochastic games). It is still unknown whether the Hoffmann-Karp algorithm terminates in polynomial time.

Strategy improvement algorithms for parity games have been proposed by Puri [147] and by Vöge and Jurdziński [191]. Both algorithms can do one iteration in polynomial time, but the actual number of iterations may be large. The algorithm due to Puri has the drawback that it is not a discrete algorithm, but involves linear programming and high precision arithmetic. So we discuss some of the ideas of the algorithm presented in [191]. Note that also the aforementioned algorithm of Ludwig for simple stochastic games [117] falls into this category.

The algorithm follows the approach described above. First a strategy of Player 0 is chosen randomly. Then an "optimal" response strategy is generated. After this the strategy of Player 0 is "improved" by some simple operation. This is done until both steps do not change the strategies anymore.

Instead of dealing with the strategies directly another combinatorial object is considered, and connected to strategies. This object is a *valuation*. Roughly speaking a valuation assigns to each vertex relevant information on a play starting from that vertex. Certain types of valuations correspond to strategy pairs. Furthermore an order is defined on valuations which captures whether one valuation is more valuable than another. This ordering allows to define optimal valuations. Furthermore simple improvement rules can be defined. This gives us all ingredients needed for a strategy improvement algorithm.

The first notion we need captures the kind of information we want to assign to single vertices. Note that we are again considering max-parity games, in which the vertex of highest priority occurring infinitely often is decisive. Furthermore, without loss of generality, we assume that no priority occurs twice.

Definition 7.26. Let $(\mathcal{A}, \Omega)$ be some parity game. Let w_π denote the vertex with highest priority occurring infinitely often in a play π. Let P_π denote the set of vertices encountered in play π before w_π appears first and having higher priority than w_π. Let $l(x)$ denote the size of the set of vertices encountered before w_π appears first.

The triple (w_π, P_π, l_π) is called the **play profile** of π.

A **valuation** is a mapping which assigns a play profile to every vertex.

A valuation is **induced** by a pair of strategies if it assigns to every vertex v the play profile of the play consistent with the strategies and starting at v.

Exercise 7.11. Construct a parity game and a valuation so that no pair of strategies corresponds to the valuation.

Construct a parity game and a valuation so that more than one pair of strategies corresponds to the valuation.

Obviously not all valuations are consistent with strategy pairs. We are looking for some nice conditions under which this is the case.

Consider the play profiles of vertices u, v with $u = f_0(v)$ in a valuation induced by strategies f_0, f_1. Call the plays originating at those vertices $\pi(u), \pi(v)$. Now obviously the most relevant vertex occurring infinitely often in the plays starting at u and at v is the same. We can distinguish three cases.

(1) $w_{\pi(v)}$ has larger or equal priority than v, but is not equal to v. In this case $P_{\pi(u)} = P_{\pi(v)}$ and $l_{\pi(u)} = l_{\pi(v)} - 1$.
(2) $w_{\pi(v)}$ has smaller priority than v. In this case $P_{\pi(v)} = P_{\pi(u)} \cup \{v\}$ and $l_{\pi(u)} = l_{\pi(v)} - 1$.
(3) $w_{\pi(v)} = v$. In this case $P_{\pi(v)} = \emptyset$ and $l_{\pi(v)} = 0$. Furthermore $P_{\pi(u)} = \emptyset$, since there are no vertices on the cycle, which are more relevant than v.

These conditions allow us to define what we call a **progress ordering**. We say that two vertices v, u obey a progress relation with respect to a valuation ϕ if the above conditions hold for the play profiles assigned to the vertices, and write $v \lhd_\phi u$.

The following is straightforward.

Lemma 7.27. *Let ϕ be a valuation satisfying $v \lhd_\phi f_0(v)$ resp. $v \lhd_\phi f_1(v)$ for all $v \in V$, then (f_0, f_1) induces ϕ.*

Our goal is still to give sufficient conditions for valuations which are induced by some pair of strategies.

We call a valuation ϕ **locally progressive**, if

$$\forall u \in V \exists v \in V : v \in uE \wedge u \lhd_\phi v.$$

This characterizes those valuations induced by strategies.

Lemma 7.28. *A valuation is locally progressive iff there exists a strategy pair inducing the valuation.*

Exercise 7.12. Prove the lemma. In particular, first show how to extract a strategy pair from a locally progressive valuation so that the strategy pair induces the valuation. Then show how to compute a locally progressive valuation when given a pair of strategies. Analyze the time needed to do so.

We now have a close connection between strategy pairs and locally progressive valuations. Our original goals were to find a way to get an "optimal" response strategy, and a way to "improve" strategies by some simple operations. We now define these with respect to valuations.

The first thing we need is a total ordering on the valuations. Since we assume that no priority occurs twice, we simply take the order on the priorities.

Next we define another ordering on vertices. Let $u \prec v$, if the priority of u is smaller than the priority of v and v has even priority, and if v has smaller priority than u and v has odd priority. So this ordering tells us how *valuable* vertices are from the point of view of Player 0.

This can be extended to sets of vertices P, Q, saying that $P \prec Q$ if $P \neq Q$ and the highest vertex in the symmetric difference between P and Q is in Q, if even, and in P, if odd.

Now extend the order to play profiles. Let (u, P, l) and (v, Q, r) be two play profiles. Then $(u, P, l) \prec (v, Q, r)$ if $u \prec v$, or if $u = v$ and $P \prec Q$ or if $u = v$ and $P = Q$ and $[l < r$ iff v has odd priority$]$.

This captures how advantageous a play profile may be for Player 0 compared to another play profile. If the most relevant vertex is advantageous, then so is the profile. If the most relevant vertex is the same, then the sets of vertices more relevant but occurring only finitely often is decisive. If these are the same, then the delay until the most relevant vertex appears decides. This is as much as we can see from the play profile, and the profile has been designed to let us see that much from it.

We are now able to state what optimal and improved valuations are.

Definition 7.29. A valuation ϕ is **optimal for Player 0**, if two vertices u and $v \in uE$ satisfy the progress relation $u \lhd_\phi v$ only if v is the $\prec$-maximal successor of u or if $\phi(u) = (u, \emptyset, 0)$ and $v = (u, \emptyset, k)$.

A symmetric definition **optimal for Player 1**. A valuation is **optimal** if it is optimal for both players.

In other words, regarding the above defined value ordering the progress relation increases only on optimal edges. Strategies inducing the valuation send vertices to optimal neighbors.

Definition 7.30. A locally progressive valuation ϕ is **improved** for Player 0 in the following way: first a strategy for Player 0 is extracted from ϕ so that for each vertex a successor is chosen which is maximal with respect to the $\prec$-order on profiles with respect to ϕ, then a valuation is constructed which is compatible with this strategy.

Note that if a locally progressive valuation is optimal for Player 0, then a strategy for Player 0 can be extracted from the valuation by mapping each vertex

to its successor in the progress ordering. This strategy leads from each vertex to a neighbor which is maximal in the value ordering. We can also extract a strategy for Player 1 from the valuation. If Player 1 wins in a play from some vertex v played as determined by those strategies, then Player 1 wins from v also if Player 0 choses a different strategy, since this other strategy cannot lead to vertices with a a more advantageous play profile for Player 0. Hence we can collect the following consequences.

Lemma 7.31. *Let ϕ be a locally progressive valuation which is optimal for Player 0 [Player 1]. Then the strategies which are compatible with ϕ are winning strategies for Player 1 [Player 0] on the set of vertices v whose play profile in ϕ is (w, P, l) with $\Omega(w)$ odd [even], against* all *strategies of Player 0 [Player 1].*

If ϕ is optimal (for both players) then all strategies compatible with ϕ are winning strategies (from the respective winning regions of the players).

So it suffices to find an optimal valuation! Now we note that improved valuations deserve their name.

Lemma 7.32. *If ϕ is a locally progressive valuation that is optimal for Player 1 and ϕ' is a locally progressive valuation that is improved for Player 0 with respect to ϕ, then $\phi(v) \preceq \phi'(v)$ for all $v \in V$.*

Hence improving a locally progressive valuation cannot lead to a less advantageous valuation. It is also strictly improved until it is optimal:

Exercise 7.13. Show that a locally progressive valuation that is optimal for Player 1, and which does not change when it is improved for Player 0, is already optimal for both players.

Improving valuations is defined in an algorithmic manner via extracting a improved strategies and computing a valuation induced by the strategies. Note that this is possible in an efficient manner due to Exercise 7.12.

Now let us briefly describe the structure of the algorithm.

The algorithm starts with a random strategy for Player 0. Then in each iteration first a locally progressive valuation is computed which is optimal for Player 1. Player 0 responds by improving his strategy as described in Definition 7.29. This is done until the iteration does not change the valuations anymore. Strategies are extracted from the valuations.

Theorem 7.33. *The above algorithm computes winning strategies for Player 0 and Player 1. It can be implemented so that each iteration runs in time $O(nm)$.*

Proof. The first statement follows from the previous lemmas. For the implementation we have to discuss the computation of an optimal valuation for Player 1 given a strategy of Player 0.

For this Player 1 fixes Player 0's strategy and then goes in ascending order over all the vertices in the resulting solitaire game using the "reward ordering" $\prec$. For such a vertex v Player 1 tests, if there is a cycle containing v and otherwise

only vertices of smaller priority. If so, then she computes the set of vertices from which v can be reached (and thus also the cycle). Then a valuation is computed on this component alone, and the component is removed, whereupon Player 1 continues with the next v.

To find an optimal valuation for the component from which the mentioned cycle is reachable, notice that it is optimal for Player 1 to go to the cycle, since v is the most profitable vertex which may occur infinitely often. It is her goal to find a path from each vertex that reaches the cycle giving the lowest reward for Player 0. All these computations are possible in time $O(nm)$.

For more details see [191]. □

So we have another approach to find winning regions and strategies in parity games. It is presently unknown how large the number of iterations may be in the worst case, except for an exponential upper bound. Neither examples with a high number of iterations nor good general upper bounds are known. Experiments suggest that the algorithm behaves quite good for some interesting inputs. One possible critique on this algorithm is that it does not make any use of a possibly bounded number of priorities, but rather expands the partial order on vertices induced by the priorities to a total order, resulting in n priorities used.

7.7 Conclusions

We have considered the problem of finding the winning regions of the two players in a given parity game and in several other graph-based games. We have seen that the problem can be solved in polynomial time, if the number of different priorities assigned to vertices is only a constant.

Our interest in the problem comes from its equivalence to model checking in the modal μ-calculus. Furthermore the problem is important as one of the few natural problems in UP $\cap$ co-UP. We have shown how to prove this complexity theoretic result. It is promising to investigate the complexity of the problem further. One message is at least that the problem is very unlikely to be NP-complete.

Furthermore we have discussed a simple, yet rather efficient algorithm, an algorithm with a quadratically improved time complexity compared to the first algorithm, and an attempt to solve the problem following the paradigm of strategy improvement.