Nordic Journal of Computing 8(2001), 324–345.

A RANDOMIZED SUBEXPONENTIAL ALGORITHM FOR PARITY GAMES

VIKTOR PETERSSON
SERGEI VOROBYOV
Information Technology/Computing Science Department
Uppsala University, Box 337
751 05 Uppsala, Sweden
axel.viktor@swipnet.se
vorobyov@csd.uu.se

Abstract. We describe a randomized algorithm for PARITY GAMES (equivalent to the MU-CALCULUS MODEL CHECKING), which runs in expected time $2^{O(k^{1/(1+2\varepsilon)})}$ when k is $\Omega(n^{1/2+\varepsilon})$, n is the number of vertices, and $0 < \varepsilon \leq 1/2$. That is, our algorithm is *subexponential* in the number of colors k of the game graph provided that k is not too small. All previously known algorithms were *exponential* in the number of colors, with the best one taking time and space $O(k^2 \cdot n \cdot \sqrt{n}^k)$. Our algorithm does not rely on Linear Programming subroutines and uses a low-degree polynomial space.

CR Classification: F.2

Key words: parity games, mu-calculus model checking, decision complexity, subexponential decision algorithm

1. Introduction

The precise computational complexity (upper and lower bounds) of solving PARITY GAMES is a fundamental open problem in automata theory, temporal logic, and computer-aided program verification.

PARITY GAMES is the following decision problem about determining a winner in two-person adversary full-information *infinite* games on *finite* graphs. Given a finite directed bi-partite graph without sinks, painted in k colors, two players alternate moves along edges. Player 1 tries to make sure that in any infinite play, independently of how intelligent Player 2 is, the *maximal color* hit *infinitely often* is *even*, whereas Player 2 tries to make it *odd*.

The fundamental importance of the problem stems from the fact that it is inter-reducible, by linearly bounded polynomial reductions, with the so-called MU-CALCULUS MODEL CHECKING problem (given a Kripke structure M, a state s, and a μ-calculus formula ϕ, decide whether $M, s \models \phi$?), and both problems belong to the complexity class $\mathrm{NP} \cap \mathrm{coNP}$ [Emerson *et al.* 1993, Thm 4.1, 4.2, 4.3]. The Mu-calculus is the 'ultimate' logic expressively subsuming virtually all known propositional temporal programming logics,

Received December 19, 2000; revised May 8, 2001; accepted August 13, 2001.

so characterizing precisely its model checking complexity is a challenging problem.

Despite a long-standing commonly believed conjecture that both PARITY GAMES and MU-CALCULUS MODEL CHECKING are in PTIME, all the best previously known decision procedures were *exponential* in the number of colors. The original iterative fixpoint algorithm [Emerson and Lei 1986], [Emerson 1997, Section 5] runs in time $O(n^k)$ and space $O(n)$, where n is the size of the game graph and k is the number of colors. The best previously known algorithm [Browne *et al.* 1997] takes exponential time and space $O(k^2 \cdot n \cdot (\sqrt{n})^k)$. No subexponential algorithms were known so far. It is an embarrassing and tantalizing open problem to find a polynomial, or at least a subexponential algorithm. This paper, besides providing better (in some cases) algorithms for parity games and the μ-calculus model checking also contributes to understanding the structure of a quite rare problem from NP $\cap$ coNP.

The upper bounds mentioned above show that the problem becomes more complicated as the number k of colors grows. In this paper we suggest the first randomized algorithm, which is *subexponential* in the number of colors k, when this number is *quite large* compared to the size of the graph n, namely, $k = \Omega(n^{1/2+\varepsilon})$ for $0 < \varepsilon \leq 1/2$. In this case our algorithm runs in subexponential time $2^{O(k^{1/(1+2\varepsilon)})}$ in the number of colors, and always uses a small polynomial space. Summarizing, for $k = 1$ or $k = 2$ it is better to use the algorithms of Emerson *et al.* [1993], then, up to $k = \sqrt{n}$ the one of Browne *et al.* [1997] (provided you can afford exponential space), afterwards our algorithm is the best.

Theoretically, the problem (as to whether there is a subexponential algorithm) was open for *any* number of colors, and this paper gives a partial answer. Practically, parity game instances with the large number of colors may appear when formalizing systems with complicated hierarchies of possible failures and remedies, like complex control systems. Imagine a situation when a more complicated failure of color $2i - 1$ may be fixed by any of increasingly more expensive patches of colors $2j \leq k$ with $2j > 2i - 1$. Of course the developer would try to create the system winning against the environment a parity game with $k+1$ colors, additionally minimizing the costs of the system recoveries. This is best formalized as a parity game, because there is no need to apply the most expensive and powerful reparations for fixing minor problems. We give a small related example in Section 2.1 and explain why simpler Büchi-like games do not apply, while Rabin games are adequate but more difficult to solve.

Let us briefly relate PARITY GAMES (PGs) to other kinds of games taking active part in our approach, namely, MEAN PAYOFF GAMES (MPGs), DISCOUNTED PAYOFF GAMES (DPGs), and SIMPLE STOCHASTIC GAMES (SSGs), recently investigated from complexity-theoretic perspective [Gurvich *et al.* 1988, Condon 1992, Ludwig 1995, Zwick and Paterson 1996]. The common property unifying all these games is that their winner decision problems are in NP$\cap$coNP and the PTIME-memberships are open (although unanimously conjectured). There is a known chain of polynomial reductions [Puri 1995, Zwick and Paterson 1996, Jurdziński 1998]

$$PGs \leq MPGs \leq DPGs \leq SSGs,$$

which is believed to be strict, so Parity Games are the simplest. In a major breakthrough Kalai [1992] proved a subexponential upper bound for a variant of the randomized simplex method, and Ludwig [1995] transferred Kalai's ideas to SSGs suggesting a subexponential randomized algorithm for them. However, until recently the only known reduction $PGs \leq MPGs$ was quite inefficient, which did not allow to use decision algorithms for MPGs to decide PGs as efficiently as the best immediate decision procedures for PGs. In Vorobyov [2000a, 2001] we suggested a better reduction, which allows us to get in some cases better (but still exponential) decision procedures for PGs based on reduction to MPGs. Also in Vorobyov [2000a, 2001] we first reported on the indirect subexponential decision algorithm for PGs based on the full chain of reductions $PGs \leq ... \leq SSGs$ and using the algorithm of Ludwig [1995] (which also uses the Linear Programming procedure as a subroutine). However, using a long inefficient chain of reductions in conjunction with the algorithm for a(n intuitively) more complicated problem (SSGs, plus LP) does not lead to a very effective algorithm. In this paper we give a more direct and efficient algorithm, based on graph optimization. Although it resembles Ludwig's and ultimately the one due to Kalai [1992], it does not use reduction to SSGs, neither does it rely on LP. We still use the reductions $PGs \leq MPGs \leq DPGs$ (which do not change the game graphs), and formally our algorithm applies to DPGs. But this is only needed to compute numerical target function values guiding the search and making sure that every step is profitable (which determines a subexponential behavior). This seems unavoidable since PGs are 'very discrete' and it is difficult to say what is a 'strategy improvement' for PGs and MPGs, whereas it is possible for DPGs. Otherwise, our algorithm operates on the initial PG graph, moreover, we take advantage of making powerful simplifications considerably reducing the search space, possible due to the relative simplicity of PGs compared with MPGs, DPGs, SSGs.

The paper is organized as follows. In Section 2 we review PGs and MPGs. Section 3 is devoted to DPGs, finding discounted values of the vertices and optimal counterstrategies when the strategy of one of the adversaries is fixed. These are important constituents of our decision algorithm, replacing the Linear Programming subroutines for SSGs. Section 4 discusses the simplification techniques we use throughout our main algorithm to reduce the search space. Our main algorithm is basically the recursive case analysis with backtrackings, and some cases appear to be immediately solvable (in the PG sense) without further recursion necessary for general DPGs and SSGs. This leads to a considerable (in practice) reduction of the search space, although we currently do not have yet a precise complexity-theoretic characterization of this simplification (whether it reduces complexity even further?). Section 5 describes the main algorithm, Section 6 proves its termination and goal-orientedness (the target value or profit increases as we proceed), and Section 7 provides a subexponential upper bound on the runtime of the algorithm.

As the main problems for future investigation we would mention the development of a direct subexponential (independent of the ratio between the number of colors and the graph size) decision procedure for PARITY GAMES, without auxiliary translation to MPGs and DPGs.

Some results from this paper were briefly reported in a short communication at LICS'2000 [Vorobyov 2000b] and we thank E. Allen Emerson for comments, useful remarks, and encouragement. Partially, the material of this paper was reported at research seminars at Tel-Aviv University and the Max-Planck-Institut für Informatik. We are grateful Alex Rabinovich, Nachum Dershowitz, Yoram Hirshfeld, Boris Trakhtenbrot, Arnon Avron, as well as to Harald Ganzinger and his group for stimulating discussions and feedback. The contents of this paper were presented at the Nordic Workshop on Programming Theory 2000 [Petersson and Vorobyov 2000]. Anonymous referees contributed to the improvement of this paper.

2. Parity and mean payoff games

2.1 Parity games

DEFINITION 1. (PARITY GAMES) *A* parity game *(PG for short, in other terminology, a* parity tree automaton, *PTA) is a 5-tuple*

$$A = (V_1, V_2, E, s_0, F), \tag{1}$$

where:

(1) V_1 *is a set of vertices in which Player 1 selects a move*[1]*;*

(2) V_2 *is a set of vertices in which Player 2 selects a move*[2]*;*

(3) $E \subseteq V_1 \times V_2 \cup V_2 \times V_1$ *is a set of directed edges, such that every* $v \in V_1 \cup V_2$ *has an outgoing edge (i.e., the graph is* bi-partite *with* no sinks*);*

(4) $s_0 \in V_1 \cup V_2$ *is the initial vertex;*

(5) $F = (F_0, F_1, \ldots, F_{k-1}, F_k)$ *is the* winning condition *to be explained shortly, where* F_i*, called* colors*, form a partition*[3] *of* V_1*.*

A PG starts from the initial vertex and Players 1, 2 alternate moves by selecting outgoing edges and moving a pebble along those edges. Player 1 wins the PG (1) if she has a selection[4] of *exactly one* outgoing edge from

[1] Called *states* in automata terminology.

[2] Called *transitions* in automata terminology.

[3] This does not lead to the loss of generality.

[4] Such selections will further be called *stationary, positional, or memoryless strategies.*

each vertex in V_1 such that independently of the choices made by Player 2 (of outgoing edges leaving vertices in V_2) the maximal index of the color visited by the pebble *infinitely often* is *even*. Symmetrically, Player 2 wins if he has a choice of *exactly one* outgoing edge from each vertex in V_2 such that independently of the choices made by Player 1 the maximal index of the color visited by the pebble *infinitely often* is *odd*.

Emerson and Jutla [1991, Thm 4.4, Cor 4.3] proved that in any PG either Player 1 or Player 2 wins (the so-called *memoryless determinacy*). Consequently, by restricting attention to positional or stationary strategies one does not lose generality.

The graph in Fig. 1 represents a 'Car Battery' Parity Game between the car owner and the environment. Although very simple, this game illustrates the two important points:

(1) in practice many colors (5 in our small example) may be useful and cannot be avoided or diminished without complicated system transformations and nontrivial assumptions,

(2) even from the practical viewpoint, solving a Parity Game cannot be substituted by solving simpler Büchi-like games.

The owner and the environment select actions in round and rectangular vertices respectively. The labels in vertices have the following intuitive meanings:

- 0 - battery OK,
- 1 - battery low,
- 2 - battery recharged,
- 3 - battery dead,
- 4 - battery just replaced,
- A - the owner tries to start the car, and finding out that the battery is OK, low on power, or dead (as chosen by the environment),
- B - the owner ignores the low power warning, the battery might be recharged automatically or drains completely (as chosen by the environment).

We want to check whether the owner by appropriate actions can ensure that all problems with the battery are resolved so that he always has a functioning car, i.e., wins the game in Fig. 1. One possibility is to verify some property like

$$\Box(1 \Rightarrow \Diamond(2 \vee 4) \wedge 3 \Rightarrow \Diamond 4).$$

No such property holds in our model without adding some fairness constraints however, and limiting the owner's actions with fairness constraints makes no sense. Another possibility is to check the model as three separate Büchi games with inhibitions with the following winning conditions:

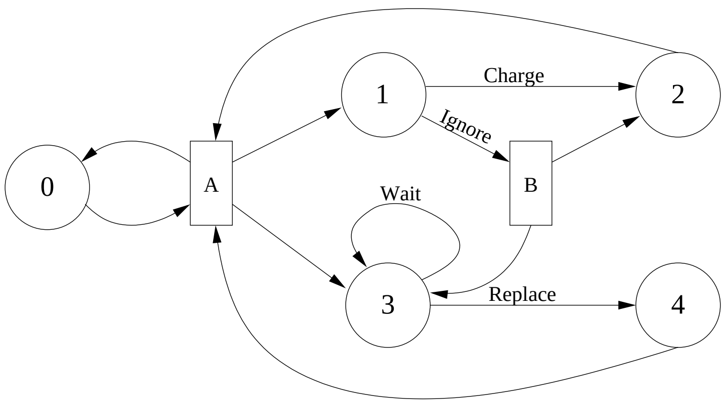


Fig. 1: Car battery game.

(1) $\Box\Diamond(0 \vee 2 \vee 4) \wedge \neg\Box\Diamond(1 \vee 3)$,

(2) $\Box\Diamond(2 \vee 4) \wedge \neg\Box\Diamond 3$,

(3) $\Box\Diamond 4$.

The owner winning one or more of those games would prove that he can ensure a functioning car but unfortunately that quite obviously is not the case. Short of using something more complicated, like a Rabin Game, the option that remains is to solve it as a Parity Game. Proving that the owner has a strategy such that the highest color hit infinitely often is always even adequately demonstrates that the model has the desired behavior.

One can easily imagine more complicated systems, like Space Shuttle or a nuclear power plant, with a more complicated (longer) hierarchy of colors, which demonstrates the practical usefulness of Parity games with many colors.

2.2 Mean payoff games

Our decision algorithm uses a reduction of PGs to the so-called MEAN PAYOFF GAMES, studied by Ehrenfeucht and Mycielski [1979], Gurvich *et al.* [1988], and Zwick and Paterson [1996]. The idea of the reduction is due to Puri [1995] [see also Jurdziński 1998], but the big weights he used made the decision procedure for PGs based on reduction of PGs to MPGs and subsequently deciding MPGs, impractical, always worse than previously known algorithms for the PG problem. In Vorobyov [2000a, 2001] we suggested a more economic weighting, allowing for competing with and outperforming (for a large number of colors) the best decision procedures. We do not apply decision algorithms for MPGs in this paper, so any weighting will be appropriate. We will describe the simplest such weighting [see Puri 1995, Jurdziński 1998] in Proposition 1 below.

DEFINITION 2. (MEAN PAYOFF GAMES) *A* mean payoff game *(MPG for short) is a 6-tuple:*

$$G = (S, T, E, s_0, w, W), \tag{2}$$

where (cf., Definition 1):

(1) V_1, V_2, E, s_0 *are the same as in PGs;*

(2) $W \in \omega$*;*

(3) $w : V_1 \to \{-W, \ldots, 0, \ldots, W\}$ *is a* weighting function *on Player 1 vertices.*

Starting from the initial vertex s_0 Players 1 and 2 alternate moves of a pebble along edges they select in turn. Any infinite play determines a sequence of visited vertices of $s_0, s_1, s_2, \ldots$. Player 1 tries to *maximize* (over all possible strategies of Player 2)

$$v_1 = \lim_{n \to \infty} \inf \frac{1}{n} \sum_{i=0}^{n} w(s_i),$$

whereas Player 2 tries to *minimize*

$$v_2 = \lim_{n \to \infty} \sup \frac{1}{n} \sum_{i=0}^{n} w(s_i).$$

Ehrenfeucht and Mycielski [1979] and Gurvich *et al.* [1988] proved that any MPG has a *value* v such that Player 1 has an optimal strategy guaranteeing her $v_1 \geq v$ and Player 2 has an optimal strategy guaranteeing him $v_2 \leq v$. Moreover, both players always have *stationary* (positional, memoryless) such strategies, in which every move depends only on the current vertex from which the player has to move.

MEAN PAYOFF GAMES is the following decision problem: *given an MPG and a number* v*, whether the value of the MPG is at least* v*?* Zwick and Paterson [1996, Thm 4.2, Thm 2.4] proved that: 1) MEAN PAYOFF GAMES is in NP$\cap$coNP; 2) Given an MPG (2) and an integer threshold T a decision whether the value of the game is $> T$, $= T$, or $< T$ can be made in time[5]

$$O((|V_1| + |V_2|)^2 \cdot |E| \cdot W). \tag{3}$$

Puri [1995], Jurdziński [1998] described the following simple polynomial reduction from PARITY GAMES to MEAN PAYOFF GAMES.

PROPOSITION 1. *Let an MPG* (2) *result from a PG* (1) *by assigning each vertex from* F_i *the weight* $(-1)^i \cdot n^i$*, where* $n = |V_1|$*, and 0 to all vertices from* V_2*. Player 1 wins the PG iff Player 1 has positive value in the MPG.*

[5] Note that this gives a pseudo-polynomial, not polynomial algorithm, since the maximum weight W may be *exponential* in the size of a game instance.

Together with Zwick-Paterson's result (3), this gives a decision algorithm for PARITY GAMES. Unfortunately, the weights in reduction used by Jurdziński [1998] were selected too big, up to $W = (|V_1| + |V_2|)^k$, which only gives an $O(|n_S + n_T|^{k+4})$ decision algorithm, unable to compete with the best $O(k^2 \cdot |n_S + n_T|^{1+k/2})$ previously known one. A better reduction is described in Vorobyov [2000a, 2001].

We will *not* rely on this algorithm, however, just use the reduction from PGs to MPGs as intermediate. Additionally, to avoid dealing with non-positive weighted paths and cycles (which leads to complications in computing shortest paths) we: 1) assume that the maximal color k is always *even* (possibly no vertices of this color) and 2) add n^k to the weight of each Player's 1 vertex in the MPG resulting (as described in Proposition 1) from the initial PG (1). Thus, in the *modified MPG* all vertices have *positive* weights, and the value of the MPG just increases by n^k. Player 1 wins an initial PG iff the value of the initial vertex of the corresponding MPG modified by adding n^k to the weight of each Player's 1 vertex is $> n^k$.

3. Discounted payoff games

DEFINITION 3. (DISCOUNTED PAYOFF GAMES) *A* discounted payoff game *(DPG for short) is a 7-tuple*

$$G = (V_1, V_2, E, s_0, w, W, \lambda), \tag{4}$$

where:

(1) V_1, V_2, E, s_0, w, W are the same as in MPGs (see Definition 2);

(2) $0 < \lambda < 1$ is a rational number called the discounting factor *of the game.*

Starting from the initial vertex s_0 Players 1 and 2 alternate moves of a pebble along edges they select in turn. Any infinite play determines a sequence of visited vertices $s_0, s_1, s_2, \dots \in V_1$. Player 1 tries to *maximize* (over all possible strategies of Player 2) the value

$$v_1 = \lim_{n\to\infty} \inf (1-\lambda) \sum_{i=0}^{n} \lambda^i w(s_i),$$

whereas Player 2 tries to *minimize* (over all possible strategies of Player 1) the value

$$v_2 = \lim_{n\to\infty} \sup (1-\lambda) \sum_{i=0}^{n} \lambda^i w(s_i).$$

Zwick and Paterson [1996, Section 5] proved that any DPG has a *value* v such that Player 1 has an optimal strategy guaranteeing her $v_1 \geq v$ and Player 2 has an optimal strategy guaranteeing him $v_2 \leq v$. Moreover, both players always have *stationary* such strategies, in which every move depends only on the current vertex from which the player has to move.

Zwick and Paterson [1996, Section 5] describe a reduction from MPGs to DPGs. Given an MPG one can determine the discounting factor λ such that the value of MPG (recall that for our purposes it is enough to decide whether this value is $\geq n^k$) can be obtained from the value of the corresponding DPG (with the same graph and weights, and discounting factor λ). Thus, the discount value λ is determined by the necessity to find the value of the initial vertex in the modified MPG; we do not need to determine this factor here precisely [see Zwick and Paterson 1996, Section 5 for details].

In the sequel we will concentrate on describing randomized algorithms for DPGs, assuming that the discounting factor is determined from an MPG obtained using Proposition 1 and then adding n^k to the weight of each vertex in V_1, as discussed in the end of Section 2.2.

This algorithm will solve the initial PG as follows. We begin by translating an initial PG into an MPG and then a DPG. We apply our algorithm (Section 5.2) to determine the discounted values of all vertices (wrt the DPG). This allows us to define, by means of the reduction of Zwick and Paterson, the mean values of all vertices (wrt the MPG). This finally allows us to determine a winner for each vertex in the initial PG using Proposition 1.

From now on we only consider game graphs of outdegree at most 2. Any graph with a bounded outdegree may be transformed into a binary graph while only increasing the number of vertices with a constant factor.

DEFINITION 4. (STRATEGIES) *A strategy $\sigma = (\sigma_1, \ldots, \sigma_d)$ for Player 1 is a bit vector, where each component σ_i indicates the choice of an outgoing edge from the i-th vertex (0 - go left, 1 - go right). A strategy for Player 2 is defined analogously.*

3.1 Discounted values for fixed strategies

DEFINITION 5. *A (simple) trace is a simple loop together with a simple path to that loop. Any pair of stationary strategies defines a simple trace from each vertex.*

Let σ, τ be a pair of stationary strategies for Players 1, 2. Throw away all edges of the game graph that are not in the strategies. For every vertex in the graph there is a trace, which determines the discounted value of the vertex when both players stick to their respective strategies σ, τ. The value of a vertex v with respect to a pair of strategies σ, τ will be denoted $\nu_{\sigma,\tau}(v)$.

Computing values when strategies are fixed is an important constituent of the randomized subexponential algorithm for parity games we describe below in Section 5.

Suppose, a fixed pair of stationary strategies σ, τ determines for a vertex v_0 the trace

$$\underbrace{v_0, \ldots, v_{p-1}}_{\text{path}}, \underbrace{v_p, \ldots, v_{p+l} = v_p}_{\text{loop}}, \tag{5}$$

where all vertices $v_0, \ldots, v_{p+l-1}$ are different.

The discounted value of v_0 corresponding to the pair σ, τ is computed as follows:

$$\nu_{\sigma,\tau}(v_0) = (1-\lambda) \cdot \left[\sum_{i=0}^{p-1} \lambda^i w(v_i) + \lambda^p \cdot \sum_{i=0}^{l-1} \lambda^i w(v_{p+i}) \cdot \sum_{i=0}^{\infty} \lambda^{li} \right] =$$

$$= (1-\lambda) \left[\sum_{i=0}^{p-1} \lambda^i w(v_i) + \frac{\lambda^p}{1-\lambda^l} \sum_{i=0}^{l-1} \lambda^i w(v_{p+i}) \right], \tag{6}$$

since $\sum_{i=0}^{\infty} \lambda^{li} = \frac{1}{1-\lambda^l}$ (a geometric progression).

Therefore, for a fixed pair of strategies the discounted value of every vertex is immediately computable by expression (6).

Say that a vertex $v \in V_1$ is *stable* with respect to a pair of stationary strategies σ, τ for Players 1 and 2 if $\nu_{\sigma,\tau}(v) = w(v) + \lambda \cdot \max_{\{u|vEu\}} \nu_{\sigma,\tau}(u)$. Similarly, $v \in V_2$ is *stable* with respect to a pair of strategies for Players 1 and 2 σ, τ if $\nu_{\sigma,\tau}(v) = w(v) + \lambda \cdot \min_{\{u|vEu\}} \nu_{\sigma,\tau}(u)$.

Let σ, τ be a pair of strategies for Players 1 and 2. Say that τ is *optimal* with respect to σ if every vertex $v \in V_2$ is stable with respect to σ, τ.

In the sequel $\tau(\sigma)$ will denote an optimal strategy of Player 2 against the stationary strategy σ of Player 1.

3.2 Computing an optimal stationary counterstrategy against a fixed stationary strategy

In this section we describe an algorithm for computing an optimal stationary counterstrategy $\tau = \tau(\sigma)$ of Player 2 against a *fixed* stationary strategy σ of Player 1.

It will be easier to look first for an optimal counterstrategy, which is *not necessarily stationary*. For efficiency, our algorithms do not keep information about the paths and loops in the game graph (whether they are simple or not), they just optimize discounted distances (values of vertices). Ignoring a possible non-simplicity of paths allows us to keep algorithms polynomial. Otherwise, backtrackings with exponential resulting behavior seems unavoidable. The following lemma, however, will finally allow us to conclude that there exists a *stationary* optimal counterstrategy $\tau = \tau(\sigma)$, which can be found in polynomial time.

LEMMA 1. *Let one of the players in a DPG fix his stationary strategy. Then an optimal counterstrategy of his adversary may be assumed stationary.*

PROOF. Assume τ is an optimal *non-stationary* counterstrategy for Player 2 against a fixed stationary strategy σ of Player 1. (Recall that Player 2 wants to minimize the value of the game.) Suppose the game starts in an arbitrary vertex and Players 1, 2 stick to their strategies σ, τ. Let a vertex v be the first revisited vertex in which the choice of τ diverges from its previous choice at that vertex (clearly, τ is stationary if no such vertex exists.) Then some trace with value Y is chosen, whereas the original choice led to a loop with value X as shown in Fig. 2.

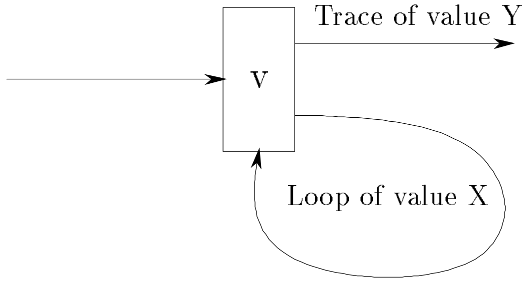


Fig. 2: Illustration to the proof of Lemma 1.

Under the strategy τ the value of v is $Z = X + \lambda^{|X|} \cdot Y$, whereas if Player 2 switches to the stationary strategy τ' repeating the loop X, the value of v will be equal to

$$Z' = X + \lambda^{|X|} \cdot X \cdot \frac{1}{1 - \lambda^{|X|}}. \tag{7}$$

Then we have the following three cases:

(1) $Z < Y$, which can be written as

$$Z = X + \lambda^{|X|} \cdot Y < Y. \tag{8}$$

Consequently, $X < Y \cdot (1 - \lambda^{|X|})$, hence $X \cdot \dfrac{1}{1 - \lambda^{|X|}} < Y$. Comparing the last terms in (7), (8), and using the last inequality we conclude that $Z' < Z$. This means that staying in the loop (switching to stationary τ') is better than diverging to the path with value Y, but this contradicts the optimality of τ.

(2) $Z > Y$, which can be written as $Z = X + \lambda^{|X|} \cdot Y > Y$. This means that directly going left from v following the trace of value Y is strictly better than traversing the loop once before following Y. But this contradicts the optimality of τ.

(3) $Z = Y$, in which case we get, similarly to case 1, the following:

$$
\begin{aligned}
Z = X + \lambda^{|X|} \cdot Y &= Y, \\
X = Y \cdot (1 - \lambda^{|X|})&, \\
X \cdot \frac{1}{1 - \lambda^{|X|}} &= Y.
\end{aligned}
$$

Therefore, fixing the choice from v so that we forever remain in the loop of value X (i.e., switching to a stationary strategy) does not affect the value $\nu(v)$. This means that we get the stationary strategy τ', which is at least as good as τ.

In all cases τ is either non-optimal or an equally good stationary strategy is found. Since no other cases exist, this proves the lemma. □

3.2.1 Structure of the optimal counterstrategy algorithm

The major steps of the algorithm described below are as follows:

(1) Find strongly connected components (SCCs) of the game graph (recall that we fixed a strategy σ for Player 1; now Player 2 may stay within any SCC as long as he wants to). This leads to an optimization, minimizing the subsequent work.

(2) For every SCC C and every vertex $v \in C$, compute a (possibly non-simple[6]) optimal (i.e., minimal discounted value) loop from v to v (of length at most $|V|$).

The *discounted value* of a (possibly non-simple) loop $v_0, v_1, \ldots, v_l = v_0$ of length l is defined as

$$
\frac{1-\lambda}{1-\lambda^l} \cdot \sum_{i=0}^{l-1} \lambda^i w(v_i). \tag{9}
$$

This is the value of a DPG starting at v_0 and infinitely following the loop.

(3) For every vertex in $V_1 \cup V_2$ [7] compute the optimal (possibly non-simple) paths (of length at most $|V|$) to all loops found above, and select a trace (path + loop) minimizing (6).

(4) Finally we find an optimal stationary strategy of Player 2. For every vertex in V_2 we check which successor is consistent with the minimal values found above. This is done by repeating the steps above with a fixed successor; see Section 3.2.4.

[6] Recall that Lemma 1 will later allow us to conclude that all loops and paths may be considered simple.

[7] just V_2 is enough!

3.2.2 Finding minimal discounted value paths

We may assume (see the end of Section 2.2) that all vertices have *positive* weights (to avoid problems with negative-value loops).

Our algorithm is based on computing the list of *minimal discounted distances of various lengths to a given vertex.* This is basically the Bellman-Ford algorithm for finding shortest paths from a single source [Cormen *et al.* 1992, Section 25.3, p. 532], with the following differences:

(1) Instead of distances from the source we compute *discounted distances to the destination.*
More explicitly, let $v_0, v_1, \ldots, v_{p-1}, v_p = s$ be a (possibly non-simple) path from a vertex v_0 to the destination s. The *discounted distance* from v_0 to s along this path is defined as

$$(1-\lambda)\cdot\sum_{i=0}^{p}\lambda^i w(v_i).$$

(2) During the execution of the algorithm, with each vertex u we keep not just the minimal discounted distance $\delta(u)$ from u to the destination s, but rather the set of minimal discounted distances $\delta_i(u)$ from u to the destination s *in* i *steps* (i.e., traversing i edges). This is necessary, since there may be two different loops, and a longer loop with a larger discounted distance may give smaller value of (9), compared with a shorter loop of lower cost.

(3) Respectively, we modify the RELAX(u, v, w, λ) subroutine from Cormen *et al.* [1992, Section 25.1, p. 520], performing a relaxation step on edge (u, v) as follows (this simplest version could be considerably improved by avoiding unnecessary tests, but running in polynomial time, it is completely satisfactory for our purposes):

RELAX(u, v, w, λ)
1 **for** $i = 1$ **to** $|V|$ **do**
2   **if** $d_i[u] > w(u) + \lambda \cdot d_{i-1}(v)$
3   **then** $d_i[u] \leftarrow w(u) + \lambda \cdot d_{i-1}(v)$

Line 3 updates the minimal discounted value path of length i. (The lines handling successors and negative loops are of no interest to us.)

Using the modified Bellman-Ford's algorithm [Cormen *et al.* 1992, Section 25.3, p. 532] we find paths with shortest discounted distances of various lengths from every vertex to the destination vertex s. It is enough to make at most $|V|$ iterations of the above relaxation.

3.2.3 Finding minimal discounted values

With the above machinery, we now describe the procedure to compute the value of the minimal discounted value loop (possibly non-simple) of length at most $|V|$ from every vertex s (to itself) in a strongly connected component.

(1) Let $v_1, \ldots, v_q$ be all vertices reachable in one step from s.

(2) Compute the minimal discounted distances (via possibly non-simple paths) $d_i(v_j)$ from v_j to s for every $1 \leq i \leq |V|$, $1 \leq j \leq q$.

(3) Select a (possibly non-simple) loop minimizing

$$\frac{1-\lambda}{1-\lambda^i} \cdot (w(s) + \lambda \cdot d_i[v_j]).$$

It is possible that there are two or more equally good loops. We are only interested in the values of the loops, however, so that is not a problem. Now we will use the values determined by those loops to find the minimal discounted values of all vertices. Initially we have L, the set of vertices for which we have found minimal loops, and $\delta(v)$, the values of the loops for all $v \in L$.

(1) Set $\gamma(v) = \infty$ for all vertices.

(2) Remove a vertex s from L and find the minimal values $\varepsilon_s(v)$ for all vertices v based on the loop from s of value $\delta(v)$. This is done by computing the optimal paths to a single destination s, backwards, where the initial value of s is $\delta(s)$. (This is done with the modified Bellman-Ford algorithm described earlier except that we do not care about paths of different lengths.)

(3) For each vertex v compare $\varepsilon_s(v)$ with $\gamma(v)$ and update $\gamma(v)$ if $\varepsilon_s(v)$ is better.

(4) Repeat Steps 2-3 until L is empty.

For every vertex v, $\gamma(v)$ now contains the minimal discounted value achievable for that vertex with respect to the stationary strategy σ of Player 1.

LEMMA 2. *The algorithm is polynomial.*

PROOF. Both parts of the algorithm call a modified Bellman-Ford algorithm (which runs in some polynomial time $p(|V|)$) (this includes arithmetic operations on integers with $O(n^2)$ bits) at most $|V|$ times, so the algorithm runs in time $O(|V| \cdot p(|V|))$. □

3.2.4 Finding an optimal stationary counterstrategy

We will now find an optimal stationary counterstrategy $\tau = \tau(\sigma)$. As stated in Lemma 1, an optimal counterstrategy may be assumed stationary, and we know that all vertices should have the values found in Section 3.2.3 under (σ, τ).

For each vertex v we initially have the value $\gamma(v)$, which is the minimal discounted value for v found above. With this information an optimal stationary counterstrategy can be computed as follows:

(1) Select any strategy $\tau' = (\tau'_1, \ldots, \tau'_d)$ and set $i = 0$.

(2) Fix the successor of v_i to the one defined by τ'_i and repeat the algorithm from Section 3.2.3 to find the minimal values $\gamma'(v)$ under this choice.

(3) If $\gamma'(v_j) \leq \gamma(v_j)$ for all $0 \leq j < |V|$, then let $\tau_i = \tau'_i$, else let $\tau_i = 1 - \tau'_i$. (By Lemma 1, the optimal strategy may be assumed stationary, so one of the successors must be consistent with the values above.)

(4) Increase i by one and repeat Steps 2 – 3 if $i \leq d$.

Obviously, the strategy τ computed above is consistent with the optimal values found in Section 3.2.3. Further, it is found in polynomial time, since it calls the polynomial algorithm from Section 3.2.3 (see Lemma 2) d times, where $d \leq |V|$.

We thus avoided using general linear programming optimization procedures (in contrast to the case of simple stochastic games) for computing optimal counterstrategies in DPGs.

4. Necessary conditions to win

The following proposition is used to simplify games throughout the main recursive algorithms of Section 5, where the informal notation $\Box\Diamond F_i$ reads 'F_i is visited infinitely often'.

PROPOSITION 2. *1) If Player 1 can win a PG from a given vertex, then she can guarantee the following condition (when the game starts in this vertex):*

$$\Box\Diamond\, F_{2i+1} \Rightarrow \Box\Diamond \bigcup_{2i+1<2j\leq k} F_{2j}. \tag{10}$$

Conversely, if Player 1 cannot guarantee (10) *when the game starts in a given vertex, then she cannot win from this vertex.*

2) If Player 2 can win a PG from a given vertex, then he can guarantee the following condition (when the game starts in this vertex):

$$\Box\Diamond\, F_{2i} \Rightarrow \Box\Diamond \bigcup_{2i<2j+1\leq k} F_{2j+1}. \tag{11}$$

Conversely, if Player 2 cannot guarantee (11) *when the game starts in a given vertex, then he cannot win from this vertex.*

PROOF. Immediate from definitions. □

Although these and similar simplifications may considerably reduce the game graph and consequently lead to a substantial improvement in the running time of the subexponential randomized algorithm, we currently do not have a satisfactory theory in favor of a provably increased performance when simplifications are added. We temporarily suggest to use these simplifications as heuristics. We will investigate this problem in depth elsewhere.

5. The main algorithm

We run simultaneously (in quasi-parallel) two algorithms, one for Player 1 (described below) and one for Player 2 (completely analogous and symmetrical) [8]. The one which stops first provides a solution of the input parity game.

5.1 Simplifications

The following simplifications are applied to the game graph of the *Parity Game* before reducing it to a *Discounted Payoff Game* and applying the main algorithm. They (or other similar simplifications) could also be used to reduce the search space for the main algorithm, but the complexity of the resulting algorithm remains to be investigated.

5.1.1 Simple simplifications

During the run of the simplifications we maintain the sets **1** and **2** (initially empty) of vertices definitely winning for Players 1 and 2 respectively. Both sets only increase, and we may stop if they form a covering of V. Whenever one of the sets changes, the following 'forced reachability' algorithm is immediately performed.

(1) If a vertex $v \in V_1$ not yet in **1** has a successor $u \in \mathbf{1}$, then we add v to **1** and delete all outgoing edges from v, except (v, u).

(2) If a vertex $v \in V_2$ not yet in **2** has a successor $u \in \mathbf{2}$, then we add v to **2** and delete all outgoing edges from v, except (v, u).

This results, at a very small cost (linear in the size of the graph), in a maximally simplified game graph at every stage of the simplifications. This allows us to avoid applying more expensive and more general techniques described below (which could achieve the same effect, but at larger costs).

5.1.2 Simplifying components

Suppose, there are two vertices u, v in a game graph such that whenever u is visited, then v is also necessarily visited, independently of the intermediate choices of Players 1, 2.

As an example, think of a path from u to v, on which neither of the players has a choice. In a more complicated and realistic situation, both players may have numbers of choices between u and v linear in the size of the game graph (this gives exponential search space if treated straightforwardly), but v is always visited when u is visited. The latter condition can be easily checked for every pair of vertices in the graph in time linear in the size of the game graph. Indeed, this is a variant of the simple finite forced reachability game.

[8] The algorithm for Player 2 is based on a slightly different reduction from the Parity Game to a Mean Payoff Game, where we assign *negative (rather than positive) weights* to even-colored vertices.

Start from $S = \{v\}$; if all successors of a vertex w are in S, add w to S; continue as long as possible; finally, every $u \in S$ satisfies the desired property.

Once such a pair u, v is found, we may find in linear time the optimal strategies for both players when they start the game in u and finish in v. This is the usual finite min – max game solvable in the bottom-up manner, where in the beginning vertex v has value 0 and at each step:

(1) Player 1 (max) selects a successor with the *maximal even* value, if any; otherwise she selects a successor with a *minimal odd* value;

(2) Player 2 (min) selects a successor with the *maximal odd* value, if any; otherwise he selects a successor with a *minimal even* value.

This completely determines the optimal strategies for Players 1, 2 when they play from u to v, and the corresponding value $0 \leq z \leq k$ of the game from u to v. In fact, the whole subgraph from u to v may be replaced with a simplified subgraph, containing just one Player's 1 vertex assigned weight z.

This can lead to considerable simplifications in the game graph, leaving fewer choices to analyze simultaneously, thus considerably reducing the exponential search space.

5.1.3 More complex simplifications

After performing the simplifications above, for every edge e originating from a vertex $v \notin \mathbf{1} \cup \mathbf{2}$ of the Player 1 remaining in the game graph we may decide, in polynomial time:

- whether selecting this edge, and erasing all other edges from v Player 1 can guarantee the necessary condition (10) from this edge?

If this is not the case, the edge e may be safely erased[9], provided there are other edges leaving v; if there are no other outgoing edges, v can be immediately added to the set $\mathbf{2}$.

We apply these simplifications as much as possible, interleaving them with simplifications from Sections 5.1.1 and 5.1.2. Certain edges may be analyzed several times, since some edges may be eliminated only after some other edges are eliminated before.

Time used by these simplification is $O(m^2 \cdot poly(|G|))$, where m is the number of edges and $poly(|G|)$ is time needed to check (10).

After performing this stage we have a simplified graph with fewer edges and potentially increased sets $\mathbf{1}$, $\mathbf{2}$. If $V = \mathbf{1} \cup \mathbf{2}$ then the game is solved. Otherwise we apply the main algorithm below to the simplified graph (after performing the reductions described in Section 2).

[9] Since if Player 1 has a winning strategy from v, it should use another edge; if he looses anyway, the loosing choice is not important from the PG viewpoint.

5.2 Main part of the algorithm

(1) Select uniformly at random a vertex $v \in V_1$ of Player 1 such that $v \notin \mathbf{1} \cup \mathbf{2}$ and v has 2 outgoing edges[10].

(2) Select uniformly at random an outgoing edge e from this vertex. Throw away (temporarily, with subsequent possibility to backtrack) the other outgoing edge from this vertex.

(3) Apply the full algorithm recursively, this results in a complete optimal strategy σ containing edge e for Player 1.

(4) Find an optimal counterstrategy $\tau = \tau(\sigma)$ of Player 2 against the *fixed* strategy σ.

(5) If every vertex $v \in V_1$ is *stable* with respect to the pair of strategies σ, τ, then σ is an optimal strategy, and the algorithm returns it.

(6) Otherwise, switch the choice in the vertex v selected at Step 1 by choosing the other edge instead of e selected at Step 2, and apply the full algorithm to the resulting game.

Step 4 of the algorithm requires finding an optimal counterstrategy against a known strategy in a DPG. This can be done in polynomial time, as described in Section 3.2.

Step 5 of the algorithm requires determining the stability of vertices, this subsumes computing values (in the discounted sense) of all vertices, as described in Section 3.1.

6. Profitability of switches

DEFINITION 6. *Define the* profit function

$$h(\sigma) = \sum_{i=1}^{n} \nu_{\sigma,\tau(\sigma)}(v_i),$$

where $\{v_i\}_{i=1}^n = V_1$ *and* $\nu_{\sigma,\tau(\sigma)}(v_i)$ *is the DPG-value of* v_i *with respect to* σ *and* $\tau(\sigma)$ *(an optimal strategy of Player 2 against* σ*).*

A switch *is a change of a strategy for Player 1 at a single vertex. Say that a switch is* profitable *if it changes a strategy* σ *into* σ' *with* $h(\sigma') > h(\sigma)$*. (These notions for the Player 2 are defined symmetrically.)*

The following Lemma shows that the algorithm from Section 5.2 makes only profitable switches.

[10] When a player does not have any choices, we find in polynomial time an optimal counterstrategy for the opponent and immediately decide, who wins from each vertex. Thus, we assume a non-trivial case here, that there is at least one binary choice vertex at this stage of the algorithm.

LEMMA 3. *Let:*

(1) G be a PG (1)*, G' be a corresponding (shifted by adding weights n^k to all vertices in V_1) MPG* (2)*, and G'' be a corresponding DPG* (4)*.*

(2) σ be a strategy for Player 1 in G'' that is not optimal,

(3) $v \in V_1$ be a vertex unstable with respect to σ, $\tau(\sigma)$,

(4) σ' be a strategy obtained from σ by switching at vertex v, and $\tau(\sigma')$ be an optimal counterstrategy against σ'.

Then:

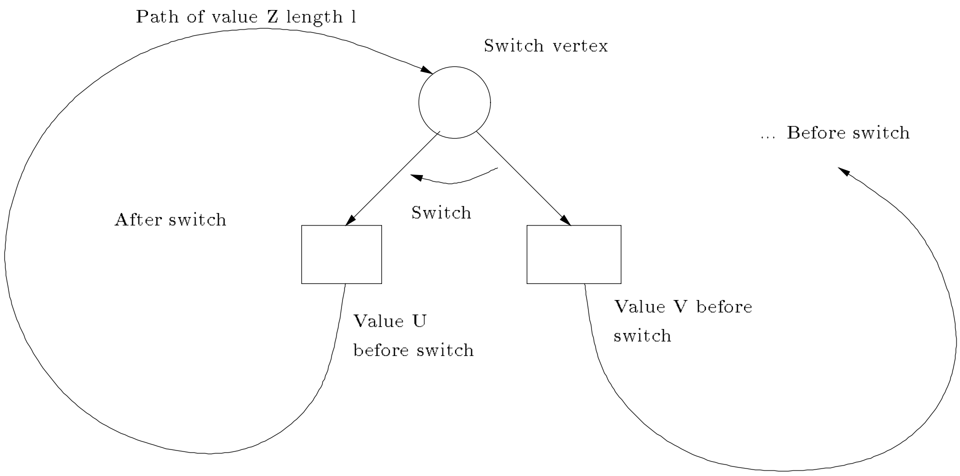
(1) for all $v \in V$ one has $\nu_{\sigma',\tau(\sigma')}(v) \geq \nu_{\sigma,\tau(\sigma)}(v)$, and

(2) for some $v \in V$ one has $\nu_{\sigma',\tau(\sigma')}(v) > \nu_{\sigma,\tau(\sigma)}(v)$.

PROOF. Suppose, towards a contradiction, that some vertex v decreases its value after the switch. Then, since $\tau(\sigma)$ was optimal with respect to σ, its new 'value trajectory' determined by σ', $\tau(\sigma')$ should necessarily:

- pass through the switch vertex v and include the newly selected edge (the opposite would contradict the optimality assumption for the strategy $\tau(\sigma)$ before the switch); consequently, we may assume that the switch vertex diminished its value;
- its loop should pass through the switch vertex and include the newly selected edge (otherwise it is impossible to get a smaller value).

Therefore, we have the following picture:



Let us analyze this picture. Since Player 1 decided to make a switch, $V < U$. From the optimality of $\tau(\sigma)$ before the switch, we have

$$U \leq Z + \lambda^{l+1} \cdot V.$$

Consequently, $V < Z + \lambda^{l+1} \cdot V$, or

$$V < \frac{Z}{1 - \lambda^{l+1}}. \tag{12}$$

On the other hand, the value of the left rectangular vertex after the switch will be the value of the left loop of length $l+1$, i.e.

$$\frac{Z}{1-\lambda^{l+1}} < V,$$

($< V$ by the decreasing assumption) which immediately yields a contradiction with (12). □

7. Runtime analysis

THEOREM 1. *1) The expected running time of the randomized algorithm for Player 1 from Section 5.2 is bounded by a subexponential function*

$$s(n) = 2^{O(\sqrt{n})},$$

where n is the number of Player's 1 vertices of outdegree > 1.

2) The expected running time of the two running in quasi-parallel symmetric algorithms, one for each player, is bounded by

$$s(n_1, n_2) = 2^{O(\sqrt{\min\{n_1,n_2\}})},$$

where n_1, n_2 are the numbers of vertices of outdegree > 1 of Players 1 and 2 respectively.

3) Consequently, for $0 < \varepsilon \leq 1/2$ and $k = \Omega(\min\{n_1, n_2\}^{1/2+\varepsilon})$ there is a subexponential in the number of colors *k algorithm for solving the* PARITY GAMES *problem of time complexity $2^{O(k^{1/(1+2\varepsilon)})}$.*

The proof of this theorem is based on the analysis due to Kalai [1992, pp. 478-479] and Ludwig [1995, p. 154], namely, on the following:

LEMMA 4. *The upper bound on the expected number of switches made by the algorithm from Section 5.2 to find a pair of optimal strategies for the game on n Player's 1 vertices of outdegree 2 is bounded from above by the function $s(n)$ satisfying the following conditions:*

$$s(n) = s(n-1) + 1 + \frac{1}{n} \cdot \sum_{i=1}^{n-1} s(i) \tag{13}$$

for $n > 1$ and $s(1) = 1$.

PROOF. Let σ^0 be the initial strategy for Player 1 formed by random choices in Steps 1 (vertex selection) and 2 (edge selection) in the initial recursion of the algorithm. Let $h^i = \max_\sigma\{h(\sigma) | \sigma_i = \sigma_i^0\}$ and $i_1, \ldots, i_n$ be the permutation of integers in the interval from 1 to n such that

$$h^{i_1} \geq h^{i_2} \geq \cdots \geq h^{i_n}. \tag{14}$$

Suppose at Step 1 the algorithm chooses at random the vertex indexed i_r. Then by solving recursively (at Step 3) the subproblem with $n-1$ Player's 1 vertices the algorithm finds a strategy σ' such that $h(\sigma') = h^{i_r}$. Since every switch the algorithm makes is profitable (by Lemma 3), from this point it will never make any switches to a strategy which agrees with σ_0 at a vertex v_{i_j} with $j > r$ (recall (14)); therefore, the strategies in all these vertices remain fixed until the algorithm terminates, and at most one switch may be made in vertex v_{i_r}. Consequently, after one top-level iteration making at most $s(n-1)+1$ switches the algorithm remains with the game on $r-1$ vertices to be solved. Since the choice of i_r is uniform and random, we have the last term in (13). □

The recurrence (13) is the key to a subexponential upper bound of the algorithm. Intuitively, the function satisfying (13) should grow slower than the one satisfying $f(n) = f(n-1)+1+f(n-1)$ and $f(1) = 1$ (which has an exponential solution). In fact, Kalai [1992, pp. 478-479] and Ludwig [1995, Lemma 9] proved the following

LEMMA 5. *The recurrence* (13) *has a* subexponential *solution* $s(n) = 2^{O(\sqrt{n})}$.

8. Conclusions

In this paper we described the first subexponential algorithm for the *Parity Games* and the *Mu-Calculus Model Checking* problems. This strengthens our belief that the problems are not *NP*-complete, and increases our hope that the problems might be polynomial. The following problems remain unsolved however:

(1) The recurrence (13) only holds on binary graphs. We have previously mentioned that any graph with a bounded outdegree can be reduced to a binary one without increasing the number of vertices by more than a constant factor. This is not true for graphs with unbounded outdegree (gives a quadratic blowup). Therefore, it is as of yet unknown whether such games can be solved in subexponential time by this or similar algorithms.

(2) Performing exact rational arithmetic leads to a considerable loss of efficiency in finding optimal counterstrategies in Discounted Payoff Games. Improving this algorithm remains an important practical problem. We would like to stress however that our algorithm for finding optimal counterstrategies in DPGs is *strongly polynomial*, in contrast to the one based on Linear Programming used for Simple Stochastic Games [Ludwig 1995].

(3) The simplifications described in Sections 4 and 5 considerably reduce the game graph and consequently lead to a substantial improvement in the running time of the subexponential randomized algorithm. However, we currently do not have a satisfactory theory in favor of a

provably increased performance when simplifications are added. We temporarily suggest to use these simplifications as heuristics. We will investigate this problem in depth elsewhere.

References

Browne, A., Clarke, E. M., Jha, S., Long, D. E, and Marrero, W. 1997. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science 178*, 237–255.

Condon, A. 1992. The complexity of stochastic games. *Information and Computation 96*, 203–224.

Cormen, Thomas H., Leiserson, Charles E., and Rivest, Ronald L. 1992. *Introduction to Algorithms*, 6th printing edition. The MIT electrical engineering and computer science series. MIT Press, Cambridge, MA.

Ehrenfeucht, A. and Mycielski, J. 1979. Positional strategies for mean payoff games. *International Joourn. of Game Theory 8*, 109–113.

Emerson, E. A. 1997. Model checking and the Mu-calculus. In *DIMACS Series in Discrete Mathematics*, Volume 31, 185–214.

Emerson, E. A., Jutla, C., and Sistla, A. P. 1993. On Model-Checking for Fragments of μ-Calculus. In *Computer Aided Verification, Proc. 5th Int. Conference*, Volume 697. Lecture Notes in Computer Science, 385–396.

Emerson, E. A. and Jutla, C. S. 1991. Tree automata, μ-calculus and determinacy. In *32nd Annual IEEE Symp. on Foundations of Computer Science*, 368–377.

Emerson, E. A and Lei, C.-L. 1986. Efficient Model Checking in fragments of the propositional Mu-calculus. In *1st Annual IEEE Symp. on Logic in Computer Science LICS'86*, 267–278.

Gurvich, V. A., Karzanov, A. V., and Khachiyan, L. G. 1988. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *U.S.S.R. Computational Mathematics and Mathematical Physics 28*, 5, 85–91.

Jurdziński, M. 1998. Deciding the winner in parity games is in UP$\cap$co-UP. *Information Processing Letters 68*, 119–124.

Kalai, G. 1992. A subexponential randomized simplex algorithm. In *24th ACM STOC*, 475–482.

Ludwig, W. 1995. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation 117*, 151–155.

Petersson, V. and Vorobyov, S. 2000. A Randomized Subexponential Algorithm for Parity Games. In *Nordic Workshop on Programming Theory (NWPT'2000)*.

Puri, A. 1995. *Theory of hybrid systems and discrete events systems*. PhD thesis, EECS Univ. Berkeley.

Vorobyov, S. 2000a. Better Decision Algorithms for Parity Games and the μ-Calculus Model Checking. Tech. Report 14, Uppsala University, Department of Information Technologies, www.it.uu.se/reports.

Vorobyov, S. 2000b. A randomized strategy improvement algorithm for parity games. In *IEEE Annual Symposium on Logic in Computer Science*.

Vorobyov, S. 2001. Better Decision Algorithms for Parity Games and the μ-Calculus Model Checking. *Information Processing Letters*, to appear.

Zwick, U. and Paterson, M. 1996. The complexity of mean payoff games on graphs. *Theoretical Computer Science 158*, 343–359.