An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it

Oliver Friedmann
Institut für Informatik, LMU München

E-mail: Oliver.Friedmann@googlemail.com

Abstract

This paper presents a new lower bound for the discrete strategy improvement algorithm for solving parity games due to Vöge and Jurdziński. First, we informally show which structures are difficult to solve for the algorithm. Second, we outline a family of games on which the algorithm requires exponentially many strategy iterations, answering in the negative the long-standing question whether this algorithm runs in polynomial time. Additionally we note that the same family of games can be used to prove a similar result w.r.t. the strategy improvement variant by Schewe as well as the strategy iteration for solving discounted payoff games due to Puri.

1. Introduction

Parity games are simple two-player games of perfect information played on directed graphs whose nodes are labeled with natural numbers, called priorities. A play in a parity game is an infinite sequence of nodes whose winner is determined by all priorities occurring infinitely often. In fact, it depends on the parity of the highest priority that occurs infinitely often, giving parity games their name.

Parity games occur in several fields of theoretical computer science, e.g. as solution to the problem of complementation or determinisation of tree automata [5, 2] or as algorithmic backend to the model checking problem of the modal μ-calculus [3, 15].

There are many algorithms that solve parity games, such as the recursive decomposing algorithm due to Zielonka [18] and its recent improvement by Jurdziński, Paterson and Zwick [9], the small progress measures algorithm due to Jurdziński [8] with its recent improvement by Schewe [12], the model-checking algorithm due to Stevens and Stirling [14] and finally the two strategy improvement algorithms by Vöge and Jurdziński [17] and Schewe [13].

All mentioned algorithms except for the two strategy improvement algorithms have been shown to have a super-polynomial worst-case runtime complexity at best or there is at least little doubt that their worst-case runtime complexity is super-polynomial or even exponential.

Solving parity games is one of the few problems that belongs to the complexity class $\mathtt{NP} \cap \mathtt{coNP}$ and that is not (yet) known to belong to $\mathtt{P}$ [3]. It has also been shown that solving parity games belongs to $\mathtt{UP} \cap \mathtt{coUP}$ [7]. The currently best known upper bound on the deterministic solution of parity games is $\mathcal{O}(|E| \cdot |V|^{\frac{1}{3}|\mathtt{ran}\Omega|})$ due to Schewe's big-step algorithm [12].

Parity games are closely related to other games of infinite duration, in particular mean, and discounted payoff as well as simple stochastic games [7, 15]. The worst-case complexity is unknown in all cases, but since parity games are the simplest among these games, lower bounds on solving parity games can have far reaching implications.

The *strategy improvement*, *strategy iteration* or *policy iteration* technique is the most general approach that can be applied as a solving procedure for all of these game classes. It was introduced by Howard [6] for solving problems on Markov decision processes and has been adapted by several other authors for solving discounted and mean payoff games [10, 19] as well as parity games [17].

Strategy iteration is an algorithmic scheme that is parameterized by the *improvement policy* which basically defines how to select a successor strategy in the iteration process. There are two major kinds of improvement policies: deterministic and randomized approaches; we will investigate the deterministic approaches in this paper.

For discounted payoff games, there is the deterministic algorithm due to Puri [10] that can also be used to solve mean payoff games as well as parity games by reduction [19, 17]. Vöge's improvement algorithm is a refined ver-

sion of Puri's on parity games that omits the use of high-precision rational numbers; there are at least two reasonable improvement policies for Vöge's procedure appearing in the literature such as Vöge's original *locally optimizing policy* and Schewe's *globally optimizing policy*.

An example has been known for some time for which a sufficiently poor choice of switching policy causes an exponential number of iterations of the strategy improvement algorithm [1], but there have been no games known so far on which the policies due to Vöge or Schewe require more than linearly many iterations.

In this paper, we particularly investigate the locally optimizing policy for solving parity games by Vöge and Jurdziński. We present a family of games comprising a linear number of nodes and a quadratic number of edges such that the strategy improvement algorithm using this policy requires an exponential number of iterations on them. These games can be refined in such a way that they only comprise a linear number of edges resulting in an undeniable exponential lower bound, but the refined construction obfuscates the main idea behind it and is therefore not presented here.

It should be noted that these games can be also applied to show exponential lower bounds for Puri's algorithm for solving mean and discounted payoff games by using the standard reductions as well as to prove an exponential lower bound for Schewe's variant. In order to prove the latter result, some structures of the presented games have to be altered; we omit this construction due to page restrictions.

Section 2 defines the basic notions of parity games and some notations that are employed throughout the paper. Section 3 recaps the strategy improvement algorithm by Vöge and Jurdziński. In Section 4, we present two graph structures that are tricky to be solved by strategy iteration algorithms. Section 5 outlines a family of games on which the algorithm requires an exponential number of iterations.

2. Parity Games

A *parity game* is a tuple $G = (V, V_0, V_1, E, \Omega)$ where (V, E) forms a directed graph whose node set is partitioned into $V = V_0 \cup V_1$ with $V_0 \cap V_1 = \emptyset$, and $\Omega : V \to \mathbb{N}$ is the *priority function* that assigns to each node a natural number called the *priority* of the node. We assume the graph to be total, i.e. for every $v \in V$ there is a $w \in V$ s.t. $(v, w) \in E$.

In the following we will restrict ourselves to finite parity games. W.l.o.g. we assume Ω to be injective, i.e. there are no two different nodes with the same priority.

We also use infix notation vEw instead of $(v, w) \in E$ and define the set of all *successors* of v as $vE := \{w \mid vEw\}$. The size $|G|$ of a parity game $G = (V, V_0, V_1, E, \Omega)$ is defined to be the cardinality of E, i.e. $|G| := |E|$; since we assume parity games to be total w.r.t. E, this is a reasonable way to measure the size.

The game is played between two players called 0 and 1: starting in a node $v_0 \in V$, they construct an infinite path through the graph as follows. If the construction so far has yielded a finite sequence $v_0 \dots v_n$ and $v_n \in V_i$ then player i selects a $w \in v_n E$ and the play continues with $v_0 \dots v_n w$.

Every play has a unique winner given by the *parity* of the greatest priority that occurs infinitely often. The winner of the play $v_0 v_1 v_2 \dots$ is player i iff $\max\{p \mid \forall j \in \mathbb{N}\, \exists k \geq j : \Omega(v_k) = p\} \equiv_2 i$ (where $i \equiv_k j$ holds iff $|i - j| \mod k = 0$). That is, player 0 tries to make an even priority occur infinitely often without any greater odd priorities occurring infinitely often, player 1 attempts the converse.

We depict parity games as directed graphs where nodes owned by player 0 are drawn as circles and nodes owned by player 1 are drawn as rectangles; all nodes are labelled with their respective priority, and - if needed - with their name.

A *strategy* for player i is a – possibly partial – function $\sigma : V^* V_i \to V$, s.t. for all sequences $v_0 \dots v_n$ with $v_{j+1} \in v_j E$ for all $j = 0, \dots, n-1$, and all $v_n \in V_i$ we have: $\sigma(v_0 \dots v_n) \in v_n E$. A play $v_0 v_1 \dots$ *conforms* to a strategy σ for player i if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ then $v_{j+1} = \sigma(v_0 \dots v_j)$. Intuitively, conforming to a strategy means to always make those choices that are prescribed by the strategy. A strategy σ for player i is a *winning strategy* in node v if player i wins every play that begins in v and conforms to σ.

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

With G we associate two sets $W_0, W_1 \subseteq V$ such that W_i is the set of all nodes v s.t. player i wins the game G starting in v. Here we restrict ourselves to positional strategies because it is well-known that these suffice. In fact, parity games enjoy positional determinancy meaning that for every node v in the game either $v \in W_0$ or $v \in W_1$ [2]. Hence, we consider a strategy for player i as a function $\sigma : V_i \to V$, s.t. for all $v \in V_i$ holds that $vE\sigma(v)$. Furthermore, it is not difficult to show that, whenever player i has winning strategies σ_v for all $v \in U$ for some $U \subseteq V$, then there is also a single strategy σ that is winning for player i from every node in U.

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

A strategy σ for player i induces a *strategy subgame* $G|_\sigma := (V, V_0, V_1, E|_\sigma, \Omega)$ where $E|_\sigma := \{(u, v) \in E \mid u \in dom(\sigma) \Rightarrow \sigma(u) = v\}$. Such a subgame $G|_\sigma$ is basically the same game as G with the restriction that whenever σ provides a strategy decision for a node $u \in V_i$ all transitions from u but $\sigma(u)$ are no longer accessible. The set of strategies for player i is denoted by $\mathcal{S}_i(G)$.

3. Strategy Improvement

First, we briefly recap the basic definitions of the strategy improvement algorithm. For a given parity game $G = (V,\ V_0,\ V_1,\ E,\ \Omega)$, the *reward* of node v is defined as follows: $\texttt{rew}_G(v) := \Omega(v)$ if $\Omega(v) \equiv_2 0$ and $\texttt{rew}_G(v) := -\Omega(v)$ otherwise. The set of *profitable nodes* for player 0 is defined to be $V_\oplus := \{v \in V \mid \Omega(v) \equiv_2 0\}$ and $V_\ominus := \{v \in V \mid \Omega(v) \equiv_2 1\}$ likewise for player 1.

The *relevance ordering* $<$ on V is induced by Ω: $v < u :\iff \Omega(v) < \Omega(u)$; additionally one defines the *reward ordering* $\prec$ on V by $v \prec u :\iff rew_G(v) < rew_G(u)$. Note that both orderings are total due to injectivity of the priority function.

A *loopless path* in G is an injective map $\pi : \{0, \dots, k-1\} \to V$ conforming with E, i.e. $\pi(i)E\pi(i+1)$ for every $i < k$. The length of a loopless path is denoted by $|\pi| := k$. The set of loopless paths π in a game G originating from the node v (i.e. $\pi(0) = v$) is denoted by $\Pi_G(v)$. We sometimes write $\pi = v_0 \dots v_{k-1}$ to denote the loopless path $\pi : i \mapsto v_i$.

A node v in G is called *dominating cycle node* iff there is a loopless path $\pi \in \Pi_G(v)$ s.t. $\pi(|\pi|-1)E\pi(0)$ and $\max\{\Omega(\pi(i)) \mid i < |\pi|\} = \Omega(v)$. The set of dominating cycle nodes is denoted by $\mathcal{C}_G$.

A key point of the strategy improvement algorithm is to assign to each node in the game graph a *valuation*. Basically, a valuation describes a loopless path originating from its node to a dominating cycle node. Such a valuation consists of three parts: the dominating cycle node, the set of more relevant nodes (w.r.t. the cycle node) on the loopless path leading to the cycle and the length of the loopless path (which measures the amount of less relevant nodes).

To compare the second component of a valuation - the set of nodes on the way to the cycle - we introduce a total ordering $\prec$ on 2^V: to determine which set of nodes is better w.r.t. $\prec$, one investigates the node with the highest priority that occurs in only one of the two sets. The set owning that node is greater than the other if and only if that node has an even priority. More formally:

$$M \prec N :\iff \begin{cases} (M \triangle N \neq \emptyset \wedge max_<(M \triangle N) \in N \cap V_\oplus) \vee \\ (M \triangle N \neq \emptyset \wedge max_<(M \triangle N) \in M \cap V_\ominus) \end{cases}$$

where $M \triangle N$ denotes the symmetric difference of both sets.

A loopless path $\pi = v_0 \dots v_k$ with $v_k \in \mathcal{C}_G$ induces a *node valuation* for the node v_0 as follows:

$$\vartheta_\pi := (v_k,\ \{v_i \mid v_k < v_i\}, k)$$

A node valuation ϑ for a node v is a triple $(c, M, l) \in V \times 2^V \times |V|$ such that there is a loopless path π with $\pi(0) = v$, $\pi(|\pi|-1) \in \mathcal{C}_G$ and $\vartheta_\pi = \vartheta$.

We extend the total ordering on sets of nodes to node valuations:

$$(u, M, e) \prec (v, N, f) :\iff \begin{cases} (u \prec v) \vee (u = v \wedge M \prec N) \vee \\ (u = v \wedge M = N \wedge e < f \wedge u \in V_\ominus) \vee \\ (u = v \wedge M = N \wedge e > f \wedge u \in V_\oplus) \end{cases}$$

A *game valuation* is a map $\Xi : V \to V \times 2^V \times |V|$ assigning each $v \in V$ a node valuation. A partial ordering on game valuations is defined as follows:

$$\Xi \lhd \Xi' :\iff (\forall v \in V : \Xi(v) \preceq \Xi'(v)) \wedge (\Xi \neq \Xi')$$

Game valuations are used to measure the performance of a strategy of player 0: for a fixed strategy σ of player 0 and a node v, the associated valuation basically states which is the worst cycle that can be reached from v conforming to σ as well as the worst loopless path leading to that cycle (also conforming to σ). Intuitively, the associated valuation reflects the best counter-strategy player 1 could play.

A strategy σ of player 0 therefore can be evaluated as follows:

$$\Xi_\sigma : v \mapsto \min_\prec\{\vartheta_\pi \mid \pi \in \Pi_{G|_\sigma}(v) \wedge \pi(|\pi|-1) \in \mathcal{C}_{G|_\sigma}\}$$

We also write $v \prec_\sigma u$ to compare the Ξ_σ-valuations of two nodes, i.e. to abbreviate $\Xi_\sigma(v) \prec \Xi_\sigma(u)$.

Lemma 1. *[17] A valuation of a strategy can be computed in polynomial time.*

A game valuation Ξ induces a counter-strategy τ_Ξ of player 1 by selecting the least profitable strategy decision with respect to the Ξ:

$$\tau_\Xi : v \in V_1 \mapsto \min_\prec U_\Xi(v)$$

where $U_\Xi(v) = \{u \in vE \mid \forall w \in vE : \Xi(u) \preceq \Xi(w)\}$.

If Ξ originates from a strategy σ, τ_Ξ can be seen as the best counter-strategy against σ; we also write τ_σ for τ_{Ξ_σ}.

A valuation Ξ originating from a strategy σ can be used to create a new strategy of player 0. The strategy improvement algorithm only allows to select new strategy decisions for player 0 occurring in the *improvement arena* $\mathcal{A}_{G,\sigma} := (V,\ V_0,\ V_1,\ E',\ \Omega)$ where

$$vE'u :\iff vEu \wedge (v \in V_1 \vee (v \in V_0 \wedge \sigma(v) \preceq_\sigma u))$$

Thus all edges performing worse than the current strategy are removed from the game. A strategy σ is *improvable* iff there is a node $v \in V_0$, a node $u \in V$ with vEu and $\sigma(v) \neq u$ s.t. $\sigma(v) \prec_\sigma u$.

An *improvement policy* now selects a strategy for player 0 in a given improvement arena w.r.t. a valuation originating from a strategy. More formally: an improvement policy is a map $\mathcal{I}_G : \mathcal{S}_0(G) \to \mathcal{S}_0(G)$ fulfilling the following two conditions for every strategy σ.

1. For every node $v \in V_0$ it holds that $(v, \mathcal{I}_G(\sigma)(v))$ is an edge in $\mathcal{A}_{G,\sigma}$.
2. If σ is improvable then there is a node $v \in V_0$ s.t. $\sigma(v) \prec_\sigma \mathcal{I}_G(\sigma)(v)$.

Jurdziński and Vöge proved in their work that every strategy that is improved by an improvement policy can only result in strategies with valuations being better (w.r.t. $\lhd$) than the valuation of the original strategy.

Theorem 2. *[17] Let G be a parity game, σ be an improvable strategy and $\mathcal{I}_G$ be an improvement policy. Let $\sigma' = \mathcal{I}_G(\sigma)$. Then $\Xi_\sigma \lhd \Xi_{\sigma'}$.*

If a strategy is not improvable, the strategy iteration comes to an end.

Theorem 3. *[17] Let G be a parity game and σ be a non-improvable strategy. Then the following holds:*

1. $W_0 = \{v \mid \Xi_\sigma(v) = (w, \_, \_) \wedge w \in V_\oplus\}$
2. $W_1 = \{v \mid \Xi_\sigma(v) = (w, \_, \_) \wedge w \in V_\ominus\}$
3. *σ is a winning strategy for player 0 on W_0*
4. *$\tau := \tau_\sigma$ is a winning strategy for player 1 on W_1*

The strategy iteration starts with an initial strategy ι_G and runs for a given improvement policy $\mathcal{I}_G$ as follows.

Algorithm 1 Strategy Iteration

```
σ ← ι_G
while σ is improvable do
    σ ← I_G(σ)
end while
return W_0, W_1, σ, τ as in Theorem 3
```

The initial strategy can be selected in several ways. We focus on a very easy method here, always selecting the node with the best reward:

$$\iota_G : v \in V_0 \mapsto \max_{\prec}\{u \mid vEu\}$$

The improvement policy we are following in this paper is the *locally optimizing policy* $\mathcal{I}^{\mathtt{loc}}_G$ due to Jurdziński and Vöge. It simply selects the most profitable strategy decision with respect to the current valuation:

$$\mathcal{I}^{\mathtt{loc}}_G(\sigma) : v \in V_0 \mapsto \max_{\prec} U_\sigma(v)$$

where $U_\sigma(v) = \{u \in vE \mid \forall w \in vE : w \preceq_\sigma u\}$.

Lemma 4. *[17] The locally optimizing policy can be computed in polynomial time.*

4. Critical Graphs

Every approach trying to construct a game family of polynomial size requiring exponentially many iterations to be solved, needs to focus on the second component of game valuations: there are only linearly many different values for the first and third component while there are exponentially many for the second.

Particularly, as there are at most linearly many different cycle nodes that can occur in valuations during a run, there is no real benefit in actually using different cycle nodes. Hence the basic layout of a game exploiting exponential behaviour consists of a (probably) complex structure leading to one single loop – the only cycle node that will occur in valuations[1]. Then, the whole strategy iteration simply tries to improve the paths leading to the cycle node.

More formally: a parity game G is a *1-sink game* iff the following two properties hold:

1. *Sink Existence*: There is a node v^* (called the *1-sink* of G) with v^*Ev^* and $\Omega(v^*) = 1$ reachable from all nodes; also, there is no other node w with $\Omega(w) \leq \Omega(v^*)$.
2. *Sink Seeking*: For each player 0 strategy σ with $\Xi_{\iota_G} \trianglelefteq \Xi_\sigma$ and each node w it holds that the cycle component of $\Xi_\sigma(w)$ equals v^*.

Obviously, a 1-sink game is won by player 1. Note that comparing node valuations in a 1-sink game can be reduced to comparing the path components of the respective node valuations for two reasons: first, the cycle component remains constant. Second, the path-length component equals the cardinality of the path component, because all nodes except the sink node are more relevant than the cycle node itself.

Lemma 5. *Let G be a parity game fulfilling the sink existence property w.r.t. v^*. G is a 1-sink game iff G is completely won by player 1 and for each node w it holds that the cycle component of $\Xi_{\iota_G}(w)$ equals v^*.*

All proofs have been put into the appendix.

In the case of a 1-sink game, we will therefore identify node valuations with their path component. When comparing the valuation of two nodes u and v w.r.t. a strategy σ in a 1-sink game, we additionally apply a more detailed relation which also states which node in the symmetric difference of the two paths is most significant:

$$v \prec^q_\sigma u :\iff v \prec_\sigma u \wedge \max_{<}(\Xi_\sigma(u) \triangle \Xi_\sigma(v)) = q$$

[1] Such structures can easily be identified by preprocessing, but obviously it is not very difficult to obfuscate the whole construction without really altering its effect on the strategy iteration.

Corollary 6. *Let G be a 1-sink game and a, b, c, p, q be nodes in G.*

1. *Let $a \succ_\sigma^p b \succ_\sigma^q c$. Then $a \succ_\sigma^r c$ where $r = \max(p, q)$.*
2. *Let $a \succ_\sigma^p b$, $c \succ_\sigma^q b$ and $p > q$. Then $a \succ_\sigma^p c$.*

There are certain graph structures confusing strategy iteration. We will combine two of them to finally construct a binary counter, leading to a family of games requiring an exponential number of iterations. Those two structures will be referred to as *deceleration lanes* and *simple cycles*.

We give an informal introduction to these structures in this chapter by considering them as parts of a (more) complex 1-sink game.

The *deceleration lane* is a family of structures that comprise two nodes, s and r, having outgoing edges to the rest of the game, a lane of nodes, $a_k, \ldots, a_0$ having incoming edges from the rest of the game, an internal parallel lane of nodes, $b_k, \ldots, b_0$, and finally an internal node c. See Figure 1 for an example of a deceleration lane.

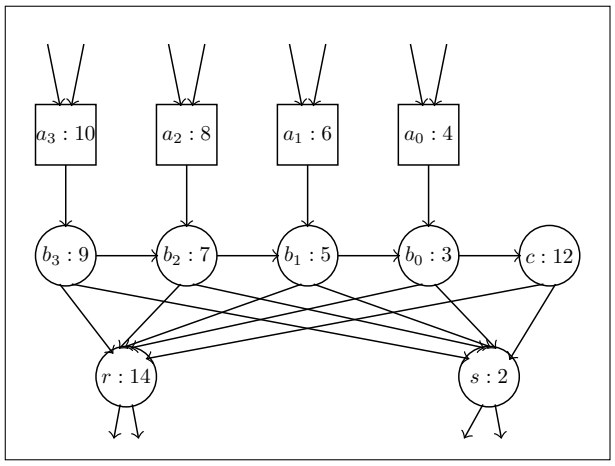

Figure 1. A Deceleration Lane

The initial setting for a deceleration lane would be a strategy that maps all player 0 nodes to r. Following a run of the strategy improvement algorithm on the whole graph, consider a setting in which the valuation of r remains greater than the one of s. In each such iteration, only one edge of the lane is a proper improvement edge: at first, the edge from b_0 to c, then the edge from b_1 to b_0 etc.

At the same time, after updating to the improvement edge, there is always a new node - meaning one in each iteration - accessible from the outside in the lane that has the highest valuation. In the beginning, the best accessible node is a_3, then a_0, then a_1 and after that a_2 etc.

There is another important feature of deceleration lanes: whenever the valuation of s gets better than the one of r, all nodes immediately switch to s.

Therefore, the whole strategy-structure of the deceleration lane can be reset simply be valuating s higher than r, even if it is only for the duration of one iteration. After that, the deceleration lane can restructure itself in the way described before.

σ	$s \prec_{\sigma'} r$	c	b_0	b_1	b_2	b_3
σ_0	Yes	r	r	r	r	r
σ_1	Yes	r	c	r	r	r
σ_2	Yes	r	c	b_0	r	r
σ_3	Yes	r	c	b_0	b_1	r
σ_4	Yes	r	c	b_0	b_1	b_2
σ_5	No	s	s	s	s	s
σ_6	Yes	r	r	r	r	r
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 2. Activity of a Deceleration Lane

Figure 2 illustrates the update activity of the deceleration lane: the first column shows the sequence of strategies associated with a run of the strategy iteration on a 1-sink game containing the deceleration lane. The second column shows which of the two nodes s and r has a better valuation according to the respective preceding strategy σ'; the other columns show the computed strategy decisions of the current strategy. Note that the external event that resets the lane simply valuates s better than r for only one iteration.

A deceleration lane is used to absorb the update activity of other nodes in such a way that wise strategy updates are postponed.

One scenario would be a *simple cycle* consisting of a player 0 and a player 1 node: assume that a wise strategy for player 0 is to move to the player 1 node s.t. player 1 is forced to leave the cycle; see Figure 3 for an example of such a situation.

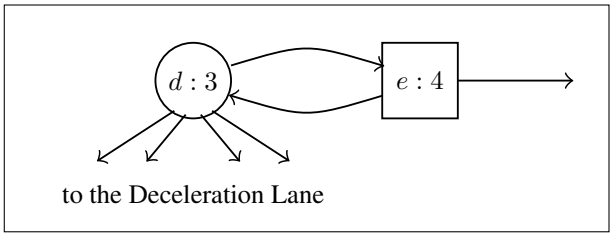

Figure 3. Simple Cycle

Player 0 will update the strategy in d to move to e iff there is no edge leading out of the cycle that is better than the edge used before. A deceleration lane thus is a device to fulfill these needs with the addition to be reusable due to its ability to reset itself.

5. Exponential Lower Bound

We present a family of parity games requiring exponentially many iterations to be solved by the strategy improvement algorithm. The games are denoted by $\mathcal{G}_n$. The set of nodes are $\mathcal{V}_n := \mathcal{V}_n^0 \cup \mathcal{V}_n^1$, where $\mathcal{V}_n^i$ denote the sets of nodes owned by player i:

$$\mathcal{V}_n^0 := \{s,\ c,\ r,\ b_0,\ \ldots,\ b_{2n-1},\ d_0,\ \ldots,\ d_{n-1}\ g_0,\ \ldots,\ g_{n-1},\ k_0,\ \ldots,\ k_{n-1}\}$$
$$\mathcal{V}_n^1 := \{p,\ q,\ a_0,\ \ldots,\ a_{2n-1},\ e_0,\ \ldots,\ e_{n-1},\ f_0,\ \ldots,\ f_{n-1},\ h_0,\ \ldots,\ h_{n-1}\}$$

Please refer to Figure 4 for the priority function and the edges of $\mathcal{G}_n$. The game $\mathcal{G}_3$ is depicted in Figure 5.

	Node	Priority	Successors
Decel.Lane	s	2	$\{p\} \cup \{f_j \mid j < n\}$
	b_0	$4n+3$	$\{s,\ r,\ c\}$
	$b_{i>0}$	$4n+2i+3$	$\{s,\ r,\ b_{i-1}\}$
	a_i	$4n+2i+4$	$\{b_i\}$
	c	$8n+4$	$\{s,\ r\}$
	r	$8n+6$	$\{p\} \cup \{g_j \mid j < n\}$
Cycles	d_i	$4i+3$	$\{s,\ e_i,\ r\} \cup \{a_j \mid j < 2i+2\}$
	e_i	$4i+4$	$\{d_i,\ h_i\}$
Backend	g_i	$4i+6$	$\{f_i,\ k_i\}$
	k_i	$8n+4i+7$	$\{p\} \cup \{g_j \mid i < j < n\}$
	f_i	$8n+4i+9$	$\{e_i\}$
	h_i	$8n+4i+10$	$\{k_i\}$
End	q	1	$\{q\}$
	p	$12n+8$	$\{q\}$

Figure 4. The Game $\mathcal{G}_n$

Fact 7. *The game $\mathcal{G}_n$ has $10{\cdot}n{+}5$ nodes, $1.5{\cdot}n^2{+}20.5{\cdot}n{+}6$ edges and $12{\cdot}n{+}8$ as highest priority. In particular, $|\mathcal{G}_n| = \mathcal{O}(n^2)$.*

Again, we note that $\mathcal{G}_n$ can be refined in such a way that it only comprises a linear number of edges. This can be basically achieved by replacing the edges going from the cycle nodes to the deceleration lane as well as the edges going from the backend nodes k_* to g_* respectively by an inductive ladder-style construction; the overall structure of these refined games is still the same, but more obscure and therefore omitted here.

First, we note some easy properties regarding $\mathcal{G}_n$, particularly that $\mathcal{G}_n$ is indeed a 1-sink game.

Lemma 8.

1. *The game $\mathcal{G}_n$ is completely won by player 1.*
2. *q is the 1-sink of $\mathcal{G}_n$ and the cycle component of $\Xi_{\iota_{\mathcal{G}_n}}(w)$ equals q for all w.*

By Lemma 5 it follows that $\mathcal{G}_n$ is a 1-sink game, hence it is safe to identify the valuation of a node with its path component from now on.

The game $\mathcal{G}_n$ implements a binary counter which is represented by n simple cycles that are connected to the deceleration lane. A bit i is considered to be set iff player 0 moves from d_i to e_i by the current strategy. The main idea is to absorb the update activity of cycles representing bits which are not set by using the deceleration lane.

The backend structure of the game basically connects all bits while it preserves the following idea: each k_i-node moves to the structure associated with the lowest bit greater than i which is set, because this is the most profitable strategy decision.

The strategy decision of g_i depends on whether bit i is set or not: if it is, then g_i moves to f_i by the current strategy and otherwise it moves to k_i. To put it simple: the most profitable path (w.r.t. the current strategy σ and the associated counter-strategy τ_σ) starts in the backend structure of the lowest bit which is set and advances to the second lowest bit which is set etc. to the highest bit which is set and thereafter to p.

The deceleration lane itself also eventually reaches this path over the node r which connects the lane by the current strategy to the backend structure of the lowest bit which is set.

Cycles which represent lower bits have less edges leading to the deceleration lane. Therefore, the cycle representing the lowest bit which is not set, gets set (i.e. player 0 moves into the simple cycle forcing player 1 to move out) after some iterations, because it has the least number of edges connecting it to the deceleration lane.

Next, the deceleration lane gets reset and all cycles representing lower bits than the one that just has been set, get unset again for the following reasons: after setting bit i, player 1 is forced to move from e_i to h_i which always moves to k_i.

This has two immediate consequences: first, it is profitable for s to update to f_i implying that s will have a better valuation than r still moving to the former most profitable g_i; hence, the deceleration lane gets reset. Second, all nodes d_j representing lower bits than the one that just has been set, also update to move to r in order to directly reach the backend structure of the newly set bit.

Cycles representing higher bits which are set, remain set, because updating to lower backend structures would be a degradation due to the node f_j having a high unprofitable priority which can be omitted when moving directly into its backend structure (from the point of view of d_j).

One iteration after that, s will also improve to the newly lowest set bit backend structure and therefore the whole deceleration process will start over again.

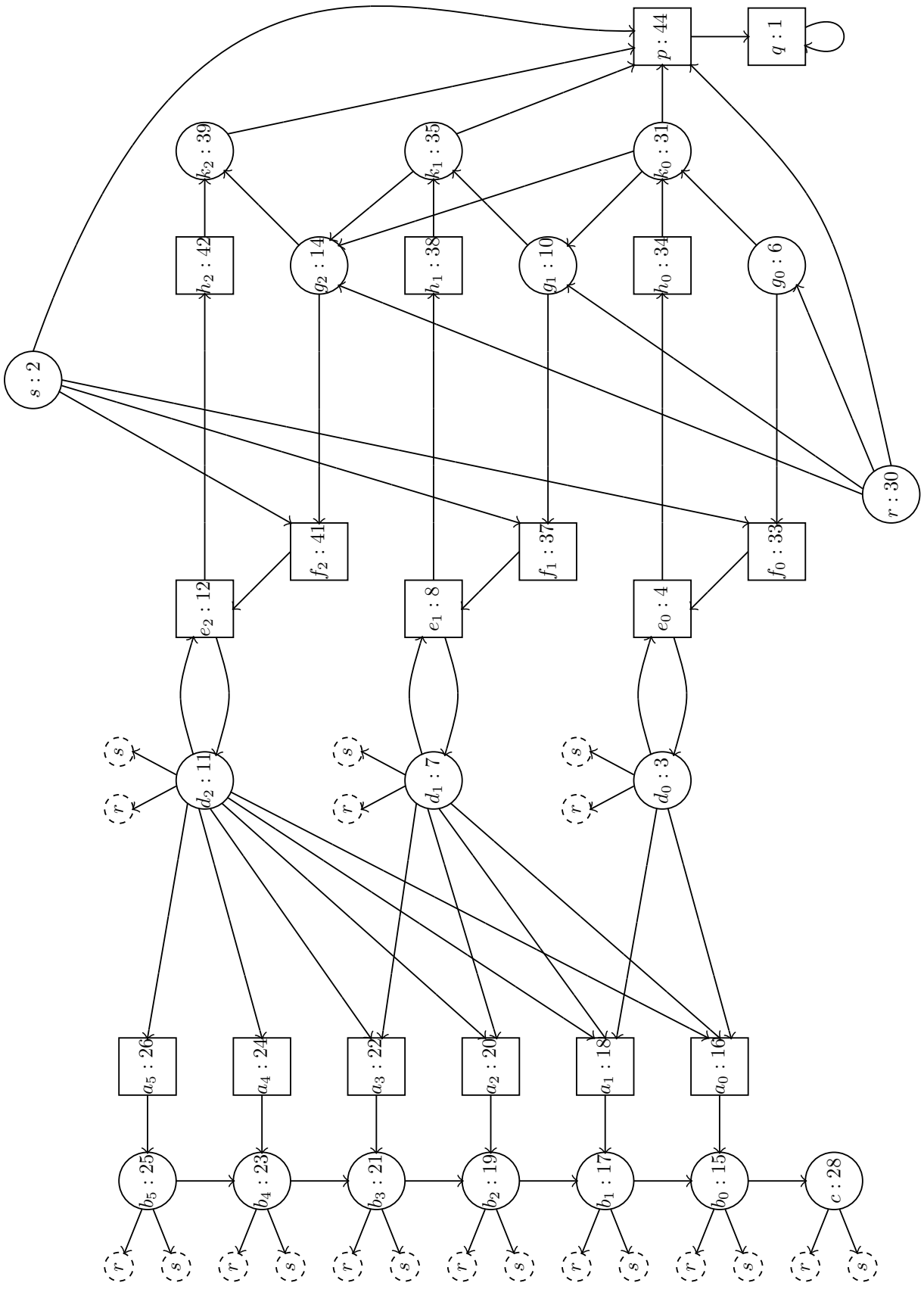

Figure 5. The Game $\mathcal{G}_3$

Now we will formally study the game. We will represent the state of the n-bit counter using elements $\alpha \in \{0,1\}^n$, where α_0 is considered to be the lowest and α_{n-1} the highest bit in α. Let $<_n$ denote the lexicographic ordering on $\{0,1\}^n$ valuating higher bits first; $\mathbf{0}_n := (0 \dots 0)$ and $\mathbf{1}_n := (1 \dots 1)$ represent the least and greatest possible bit states. For $\alpha \neq \mathbf{1}_n$, α^+ denotes the $<_n$-least bit state greater than α; accordingly, α^- denotes the $<_n$-greatest bit state less than α for $\alpha \neq \mathbf{0}_n$.

The lowest bit in α which is not set is denoted by $\mu_\alpha := \max\{j \leq n \mid \forall k < j.\alpha_k = 1\}$ and similarly the lowest bit in α which is set is denoted by $\nu_\alpha := \max\{j \leq n \mid \forall k < j.\alpha_k = 0\}$. The bit state that results from α when the first $j+1$ bits are cleared is denoted by $\alpha|_j := (\alpha_{n-1} \dots \alpha_{j+1} 0 \dots 0)$.

The number of strategy iterations that is needed to set the currently lowest bit which is not set directly depends on the index of the respective bit and will be bounded by $3 + \gamma_\alpha$, where

$$\gamma_\alpha := \begin{cases} 2 \cdot \mu_\alpha + 4 & \text{if } \alpha \neq \mathbf{1}_n \\ 2 \cdot n - 1 & \text{if } \alpha = \mathbf{1}_n \end{cases}$$

We split the correctness proof into separate lemmas. First, we define a family of partial strategies $\sigma^D_{(n,\beta)}$ for $-2 \leq \beta$ that will be proven to reflect the sequence of strategies w.r.t. the deceleration lane in a run of the improvement algorithm on $\mathcal{G}_n$. Define

$$\sigma^D_{(n,\beta)} := \begin{cases} b_0 & \mapsto \begin{cases} s & \text{if } \beta = -2 \\ r & \text{if } \beta = -1 \\ c & \text{otherwise} \end{cases} \\ b_{j>0} & \mapsto \begin{cases} s & \text{if } \beta = -2 \\ r & \text{if } -2 < \beta < j \\ b_{j-1} & \text{otherwise} \end{cases} \\ c & \mapsto \begin{cases} s & \text{if } \beta = -2 \\ r & \text{otherwise} \end{cases} \end{cases}$$

Lemma 9 (Deceleration Lane Behaviour)**.** *Let $-2 \leq \beta$, σ be a strategy s.t. $\sigma|_{D_n} = \sigma^D_{(n,\beta)}$ where $D_n = \mathtt{dom}(\sigma^D_{(n,\beta)})$ and $\sigma' := \mathcal{I}^{\mathtt{loc}}(\sigma)$. Then the following holds:*

1. *$s \prec_\sigma r$ implies $\sigma'|_{D_n} = \sigma^D_{(n,\beta+1)}$.*
2. *$s \succ_\sigma r$ and $\beta > -2$ implies $\sigma'|_{D_n} = \sigma^D_{(n,-2)}$.*
3. *$s \prec_\sigma r$ and $\beta = -2$ implies $r \succ_\sigma c \succ_\sigma a_{2n-1} \succ_\sigma \dots \succ_\sigma a_0 \succ_\sigma s$.*
4. *$s \prec_\sigma r$ and $\beta \geq -1$ implies $a_\beta \succ_\sigma \dots \succ_\sigma a_0 \succ_\sigma c \succ_\sigma a_{2n-1} \succ_\sigma \dots \succ_\sigma a_{\beta+1} \succ_\sigma r \succ_\sigma s$.*
5. *$s \succ_\sigma r$ and $\beta > -2$ implies $s \succ_\sigma v$ for all $v = c, r, a_0, \dots, a_{2n-1}$.*

Second, we define a family of partial strategies $\sigma^B_{(n,\alpha,\beta)}$ for $\alpha \in \{0,1\}^n$ and $-2 \leq \beta \leq 0$ that will be proven to reflect the sequence of strategies w.r.t. the backend structure in a run of the improvement algorithm. Define $\sigma^B_{(n,\alpha,\beta)} :=$

$$\begin{cases} g_j & \mapsto \begin{cases} f_j & \text{if } \alpha_j = 1 \vee (\beta = 0 \wedge j = \mu_\alpha) \vee \\ & \quad (\beta = -2 \wedge \alpha \neq \mathbf{0}_n \wedge \alpha_j^- = 1) \\ k_j & \text{otherwise} \end{cases} \\ k_j & \mapsto \begin{cases} g_{\nu_{\alpha|_j}} & \text{if } j < \nu_{\alpha|_j} < n \\ p & \text{otherwise} \end{cases} \end{cases}$$

Lemma 10 (Backend Behaviour)**.** *Let $-2 \leq \beta \leq 0$, $\alpha \in \{0,1\}^n$ σ be a strategy s.t. $\sigma|_{B_n} = \sigma^B_{(n,\alpha,\beta)}$ where $B_n = \mathtt{dom}(\sigma^B_{(n,\alpha,\beta)})$ and $\sigma' := \mathcal{I}^{\mathtt{loc}}(\sigma)$. Then the following holds:*

1. *$\forall i : (\sigma(d_i) = e_i \iff \alpha_i = 1)$ and $\beta < 0$ implies:*
   (a) *$\sigma'|_{B_n} = \sigma^B_{(n,\alpha,-1)}$*
   (b) *If $\alpha = \mathbf{0}_n$ then $p \succ_\sigma g_i, f_i$ for all i, otherwise $g_{\nu_\alpha} \succ_\sigma f_{\nu_\alpha} \succ_\sigma p, g_i, f_i$ for all $i \neq \nu_\alpha$*
   (c) *$d_i \prec_\sigma h_i$ for all i with $\alpha_i = 0$*
   (d) *$e_i \succ^{f_i}_\sigma g_{\nu_\alpha}$ for all i with $\alpha_i = 1$*
2. *$\alpha \neq \mathbf{1}_n$, $\forall i : (\sigma(d_i) = e_i \iff \alpha_i = 1 \vee i = \mu_\alpha)$ and $\beta = -1$ implies:*
   (a) *$\sigma'|_{B_n} = \sigma^B_{(n,\alpha,0)}$*
   (b) *If $\alpha = \mathbf{0}_n$ then $f_{\mu_\alpha} \succ_\sigma p \succ_\sigma f_i$ for all $i \neq \mu_\alpha$, otherwise $f_{\mu_\alpha} \succ_\sigma g_{\nu_\alpha} \succ_\sigma p, f_i$ for all $i \neq \mu_\alpha$*
   (c) *$d_i \prec_\sigma h_i$ for all i with $\alpha_i = 0$ and $i \neq \mu_\alpha$*
   (d) *$e_i \succ^{f_i}_\sigma g_j$ for all j and all i with $\alpha_i = 1$ or $i = \mu_\alpha$*
3. *$\alpha \neq \mathbf{1}_n$, $\forall i : (\sigma(d_i) = e_i \iff \alpha_i = 1 \vee i = \mu_\alpha)$ and $\beta = 0$ implies:*
   (a) *$\sigma'|_{B_n} = \sigma^B_{(n,\alpha^+,-2)}$.*
   (b) *$g_{\mu_\alpha} \succ_\sigma f_{\mu_\alpha} \succ_\sigma p, g_i, f_i$ for all $i \neq \mu_\alpha$*
   (c) *$f_{\mu_\alpha} \succ^{h_{\mu_\alpha}}_\sigma g_i$ for all i*
   (d) *$d_i \prec_\sigma h_i$ for all i with $\alpha_i = 0$ and $i \neq \mu_\alpha$*
   (e) *$e_i \succ^{f_i}_\sigma f_{\mu_\alpha}$ for all $i \geq \mu_\alpha$ with $\alpha_i = 1$ or $i = \mu_\alpha$*
   (f) *$f_{\mu_\alpha} \succ^{h_{\mu_\alpha}}_\sigma e_i$ for all $i < \mu_\alpha$ with $\alpha_i = 1$*

Third, we define a family of strategies $\sigma_{(n,\alpha,\beta)}$ for $\alpha \in \{0,1\}^n$ and $-2 \leq \beta \leq \gamma_\alpha$ that will be proven to reflect the complete sequence of strategies in a run of the improvement

algorithm. Define $\sigma_{(n,\alpha,\beta)} :=$

$$\begin{cases} s & \mapsto \begin{cases} f_{\mu_\alpha} & \text{if } \alpha \neq \mathbf{1}_n \wedge \beta = \gamma_\alpha \\ f_{\nu_\alpha} & \text{if } \alpha \neq \mathbf{0}_n \wedge (\beta < \gamma_\alpha \vee \alpha = \mathbf{1}_n) \\ p & \text{otherwise} \end{cases} \\ r & \mapsto \begin{cases} g_{\nu_\alpha} & \text{if } \alpha \neq \mathbf{0}_n \\ p & \text{otherwise} \end{cases} \\ d_j & \mapsto \begin{cases} e_j & \text{if } \alpha_j = 1 \vee \\ & \quad (\beta \geq \gamma_\alpha - 1 \wedge \mu_\alpha = j) \\ s & \text{if } \beta = -2 \wedge \alpha_j = 0 \\ r & \text{if } \beta = -1 \wedge \alpha_j = 0 \\ a_{2j+1} & \text{if } \beta = 0 \wedge \alpha_j = 0 \\ a_{\beta-1} & \text{otherwise} \end{cases} \\ q \in D_n & \mapsto \sigma^D_{(n,\beta)}(q) \\ q \in B_n & \mapsto \begin{cases} \sigma^B_{(n,\alpha,-2)}(q) & \text{if } \beta = -2 \\ \sigma^B_{(n,\alpha,0)}(q) & \text{if } \beta = \gamma_\alpha \wedge \alpha \neq \mathbf{1}_n \\ \sigma^B_{(n,\alpha,-1)}(q) & \text{otherwise} \end{cases} \end{cases}$$

Lemma 11. *Let $n > 0$. Then the following holds:*

1. $\iota_{\mathcal{G}_n} = \sigma_{(n,\mathbf{0}_n,-1)}$
2. $\mathcal{I}^{\mathtt{loc}}(\sigma_{(n,\alpha,\beta)}) = \sigma_{(n,\alpha,\beta+1)}$ *for every* α *and* $-2 \leq \beta < \gamma_\alpha$
3. $\mathcal{I}^{\mathtt{loc}}(\sigma_{(n,\alpha,\gamma_\alpha)}) = \sigma_{(n,\alpha^+,-2)}$ *for every* $\alpha \neq \mathbf{1}_n$
4. $\mathcal{I}^{\mathtt{loc}}(\sigma_{(n,\mathbf{1}_n,\gamma_{\mathbf{1}_n})}) = \sigma_{(n,\mathbf{1}_n,\gamma_{\mathbf{1}_n})}$

By induction on n we can conclude that the strategy iteration using the locally optimizing policy requires at least exponential time in the worst case.

Theorem 12. *Let $n > 0$. The strategy improvement algorithm requires $9 \cdot 2^n - 8$ iterations on $\mathcal{G}_n$ using the locally optimizing improvement policy.*

We implemented an open-source parity game solver platform, the PGSOLVER Collection [4], that particularly contains implementations of the strategy iteration due to Vöge and Jurdziński [17] as well as the variant by Schewe [13]. Benchmarking both algorithms with $\mathcal{G}_n$ results in exponential run-time behaviour as can be seen in Figure 6 (note that the time-axis has logarithmic scale) [2].

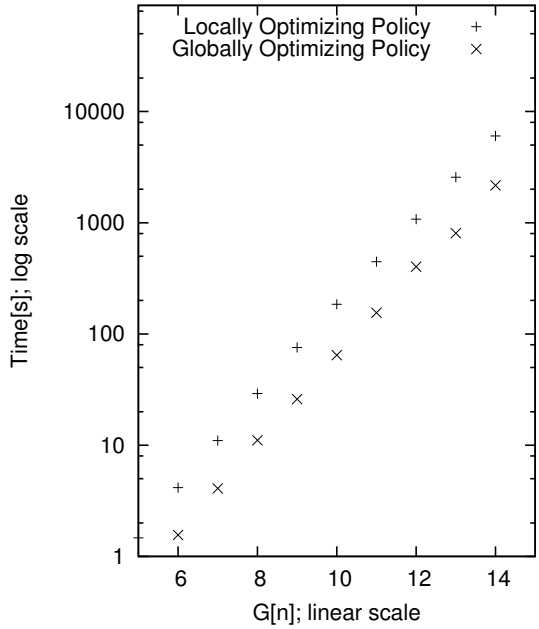

Figure 6. The Benchmark

6. Conclusion

We have presented a family of games on which the deterministic strategy improvement algorithm due to Vöge and Jurdziński requires exponentially many iterations. Although the games presented here comprise a quadratic number of edges, the construction can be refined s.t. there are only linearly many edges.

Vöge mentions in his PhD thesis [16] that it is probably much more convenient for strategy improvement algorithms to be performed on games with an outgoing edge degree limited by two. A simple transformation results in a family of games with out-degree limited by two that also requires exponential time to be solved.

There are other possibilities to select the initial strategy. Randomizing the initial strategy, for instance, is another popular choice: we note without proof that starting with a randomized strategy, the expected number of iterations on $\mathcal{G}_n$ is also exponential.

The games presented here can be also applied to show exponential lower bounds for Puri's algorithm for solving mean and discounted payoff games by using the standard reductions [10, 19].

Moreover these games can be refined s.t. an exponential lower bound for both variants [13, 11] of Schewe's strategy iteration can be shown. In particular, one has to replace the simple cycles representing the bits of the counter by larger cycles consisting of more player 0 nodes connected to the deceleration lane. The overall construction regarding deceleration lane and backend structure remains quite the same.

Although there are many preprocessing techniques that could be used to simplify the family of games presented here - e.g. decomposition into strongly connected compo-

[2] Both algorithms were benchmarked on a refined version of $\mathcal{G}_n$ that comprises only linearly many edges and also works for Schewe's improvement policy.

nents, compression of priorities, direct-solving of simple cycles, etc. - such procedures cannot be implemented to fix the bad performance of the strategy iteration on these games since all known preprocessing techniques can be fooled quite easily without really touching the inner structure of the game.

The same applies to simultaneous solving using different algorithms due to the fact that it is not very complicated to combine different worst-case games in such a way that each algorithm that tries to solve the whole game is slowed down by the part that belongs to its worst-case example.

Parity games are widely believed to be solvable in polynomial time, yet there is no algorithm known that is performing better than super-polynomially. Vöge presented his strategy iteration technique in his PhD thesis nine years ago, and this class of solving procedures is generally supposed to be the best candidate to give rise to an algorithm that solves parity games in polynomial time since then. Unfortunately the two most obvious improvement policies, namely the locally and the globally optimizing technique, are not capable of doing so.

We think that the strategy iteration still is a promising candidate for a polynomial time algorithm, although it is possibly necessary to alter more of it than just the improvement policy. The main problem of the algorithm (and the policies) is that reoccurring substructures are not handled in such a way that a combination of edges that was profitable before is applied again. The reason is that possibly not all edges belonging to that profitable combination are improvement edges, hence that combination cannot be selected in a single improvement step.

Therefore we believe that it would be an interesting approach to add some kind of memorization of profitable substructures that can be applied as a whole under certain conditions that are weaker than requiring all edges of the substructure to be improvement edges but strong enough to ensure the soundness of the algorithm.

Regarding randomized strategy improvement variants, we think that it should be possible to alter the games presented here in such a way that the expected run-time is super-polynomial. One has to ensure that very profitable strategy updates – like setting bits in $\mathcal{G}_n$ – are very unlikely to happen by chance. This should be achievable by replacing the simple cycles with cycles consisting of linearly many nodes having edges to the deceleration lane in combination with similar obfuscation techniques that are to be applied to the backend structure.

Acknowledgements. I am indebted to Martin Lange and Martin Hofmann for their guidance and numerous inspiring discussions on the subject.

References

[1] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Appl. Math.*, 155(2):210–229, 2007.

[2] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377, San Juan, 1991. IEEE.

[3] E. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th Conf. on CAV, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[4] O. Friedmann and M. Lange. The `PGSolver` collection of parity game solvers, 2009. University of Munich. Available from `http://www.tcs.ifi.lmu.de/pgsolver`.

[5] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, LNCS. Springer, 2002.

[6] R. Howard. *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.

[7] M. Jurdziński. Deciding the winner in parity games is in $\mathtt{UP} \cap \mathtt{coUP}$. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[8] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Proc. 17th Ann. Symp. on Theo. Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[9] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithm, SODA'06*, pages 117–123. ACM, 2006.

[10] A. Puri. *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, University of California, Berkeley, 1995.

[11] S. Schewe. An optimal strategy improvement algorithm for solving parity games. Reports of SFB/TR 14 AVACS 28, SFB/TR 14 AVACS, July 2007.

[12] S. Schewe. Solving parity games in big steps. In *Proc. FST TCS*. Springer-Verlag, 2007.

[13] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *17th Annual Conference on Computer Science Logic (CSL 2008)*, 2008.

[14] P. Stevens and C. Stirling. Practical model-checking using games. In B. Steffen, editor, *Proc. 4th Int. Conf. on Tools and Alg. for the Constr. and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[15] C. Stirling. Local model checking games. In *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[16] J. Vöge. *Strategiesynthese für Paritätsspiele auf endlichen Graphen*. PhD thesis, University of Aachen, 2000.

[17] J. Vöge and M. Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *Proc. 12th Int. Conf. on Computer Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.

[18] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1–2):135–183, 1998.

[19] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

A. Appendix

Lemma 5:

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

Corollary 6:

Proof. Let u_σ denote the path component of $\Xi_\sigma(u)$. Let $m_{(a,b)} := \max_<(a_\sigma \triangle b_\sigma)$ and $m_{(c,b)} := \max_<(c_\sigma \triangle b_\sigma)$. Then the following holds:

$$\begin{aligned} \max_<(a_\sigma \triangle c_\sigma) &= \max_<(\{m_{(a,b)}\} \triangle \{m_{(c,b)}\}) \\ &= \max_<(\{p\} \triangle \{q\}) =: r \end{aligned}$$

1. is obvious. Regarding 2., it holds by assumption that $p > q$, i.e. $r = p$. Since $p \in a_\sigma$ iff $p \in V_\oplus$ it follows that $a \succ_\sigma c$, hence $a \succ^p_\sigma c$. □

Lemma 8:

Proof.

1. Note that the only nodes owned by player 1 with an out-degree greater than 1 are $e_0, \ldots, e_{n-1}$. Consider the player 1 strategy τ which selects to move to h_i from e_i for all i. Now it is the case that $\mathcal{G}_n|_\tau$ contains exactly one cycle that is eventually to be reached no matter what player 0 does, namely the self-cycle at q which is won by player 1.

2. The self-cycle at q obviously is the 1-sink since it can be reached from all other nodes and has the smallest priority 1. Since qEq is the only cycle in $\mathcal{G}_n|_{\iota_{\mathcal{G}_n}}$, q must be the cycle component of each node valuation w.r.t. $\iota_{\mathcal{G}_n}$.

□

Lemma 9:

Proof.

1. Clearly, it is more profitable to move to r than to s. It is most profitable, however, to eventually reach r through c.

2. $\beta > -2$ implies that all nodes of the lane directly reach r instead of s. Note that $s \succ_\sigma r$ implies that s has a better valuation than all nodes in the deceleration lane. Therefore, all nodes of the deceleration lane improve to move to s.

3. It is most profitable to reach r by assumption, all other nodes move to s.

4. It is more profitable to eventually reach r through c than directly moving to c, therefore node a_β is the most profitable node in the lane.

5. Due to the fact that the most significant node in the symmetric difference of the path valuations of r and s has to have a higher priority than all nodes in the lane.

□

Lemma 10:

Proof. Let $i < n$. (i) Note that it is most profitable to move from g_i to f_i iff $\sigma(d_i) = e_i$. (ii) Also note that in order to reach a node w that is directly reachable by k_i, it is only profitable to move to node g_i (instead of directly moving to w) iff g_i moves to f_i and d_i moves to e_i.

1. Let $i < n$. (a) and (b) can be shown by using (i) and (ii).

   (c) The path associated with $\Xi_\sigma(d_i)$ eventually reaches the same node $\sigma(k_i)$ reaches, but passes on its way only priorities below $\Omega(h_i)$.

   (d) The path associated with $\Xi_\sigma(g_{\nu_\alpha})$ eventually reaches $\sigma(e_i)$ but on its way only passes priorities below $\Omega(f_i)$.

2. Let $i < n$. (a) and (b) can be shown by using (i) and (ii).

   (c) The path associated with $\Xi_\sigma(d_i)$ eventually reaches the same node $\sigma(k_i)$ reaches, but passes on its way only priorities below $\Omega(h_i)$.

   (d) The path associated with $\Xi_\sigma(g_{\nu_\alpha})$ eventually reaches $\sigma(e_i)$ but on its way only passes priorities below $\Omega(f_i)$.

3. Let $i < n$. (a), (b) and (c) can be shown by using (i) and (ii). Note that $\alpha_i = 1 \vee i = \mu_\alpha$ is equivalent to $\alpha_i^+ = 1 \vee \alpha_i = 1$.

   (d) The path associated with $\Xi_\sigma(d_i)$ eventually reaches the same node $\sigma(k_i)$ reaches, but passes on its way only priorities below $\Omega(h_i)$.

   (e) The path associated with $\Xi_\sigma(f_{\mu_\alpha})$ eventually reaches $\sigma(e_i)$ but on its way only passes priorities below $\Omega(f_i)$.

   (f) The path associated with $\Xi_\sigma(d_i)$ eventually reaches the same node $\sigma(k_{\mu_\alpha})$ reaches, but passes on its way only priorities below $\Omega(h_{\mu_\alpha})$.

□

Lemma 11:

Proof. We will implicitly make use of Corollary 6.

1. Obvious.

2. Let $\sigma := \sigma_{(n,\alpha,\beta)}$, $\sigma' := \mathcal{I}^{\mathtt{loc}}(\sigma)$ and $\sigma^* := \sigma_{(n,\alpha,\beta+1)}$.

   - Case $\beta < \gamma_\alpha - 1$ or $\alpha = \mathbf{1}_n$: Lemma 10 (1a) implies that $\sigma^*|_{B_n} = \sigma'|_{B_n}$, (1b) implies $s \succ_\sigma r$, $\sigma^*(r) = \sigma'(r)$ and $\sigma^*(s) = \sigma'(s)$. Hence, Lemma 9 (1) implies that $\sigma^*|_{D_n} = \sigma'|_{D_n}$.

     It remains to show that $\sigma^*(d_i) = \sigma'(d_i)$, hence let $i < n$.

     - If $\alpha_i = 1$, it follows by (1d) that $\sigma^*(d_i) = \sigma'(d_i)$.
     - Otherwise, if $\alpha_i = 0$, it follows by (1c) that $d_i \prec_\sigma^{e_i} e_i$. Then, if $\beta = -2$, $\sigma^*(d_i) = \sigma'(d_i)$ follows by Lemma 9 (3), and if $\beta \geq -1$, $\sigma^*(d_i) = \sigma'(d_i)$ follows by Lemma 9 (4).

   - Case $\beta = \gamma_\alpha - 1$ and $\alpha \neq \mathbf{1}_n$: Lemma 10 (2a) implies that $\sigma^*|_{B_n} = \sigma'|_{B_n}$, (2b) implies $r \succ_\sigma s$, $\sigma^*(r) = \sigma'(r)$ and $\sigma^*(s) = \sigma'(s)$. Hence, Lemma 9 (1) implies that $\sigma^*|_{D_n} = \sigma'|_{D_n}$.

     It remains to show that $\sigma^*(d_i) = \sigma'(d_i)$, hence let $i < n$.

     - If $\alpha_i = 1$ or $\mu_\alpha = i$, it follows by (2d) that $\sigma^*(d_i) = \sigma'(d_i)$.
     - Otherwise, if $\alpha_i = 0$ and $i \neq \mu_\alpha$, it follows by (2c) that $d_i \prec_\sigma^{e_i} e_i$. Then it follows by Lemma 9 (4) that $\sigma^*(d_i) = \sigma'(d_i)$.

3. Let $\alpha \neq \mathbf{1}_n$, $\sigma := \sigma_{(n,\alpha,\gamma_\alpha)}$, $\sigma' := \mathcal{I}^{\mathtt{loc}}(\sigma)$ and $\sigma^* := \sigma_{(n,\alpha^+,-2)}$. Lemma 10 (3a) implies that $\sigma^*|_{B_n} = \sigma'|_{B_n}$, (3b) implies $\sigma^*(r) = \sigma'(r)$ and $\sigma^*(s) = \sigma'(s)$ (since $\mu_\alpha = \nu_{\alpha^+}$) and (3c) implies that $s \succ_\sigma^{h_{\mu_\alpha}} r$. Hence, Lemma 9 (2) implies that $\sigma^*|_{D_n} = \sigma'|_{D_n}$.

   It remains to show that $\sigma^*(d_i) = \sigma'(d_i)$, hence let $i < n$. Since $s \succ_\sigma^{h_{\mu_\alpha}} r$, it clearly follows by Lemma 9 (5) that $s \succ_\sigma a_j$ for all j.

   - If $\alpha_i = 0$ and $i \neq \mu_\alpha$, it follows by (3d) that $e_i \succ_\sigma^{e_i} d_i$. Since $s \succ_\sigma^{h_{\mu_\alpha}} r$, it holds that $s \succ_\sigma^{h_{\mu_\alpha}} e_i$, hence $\sigma^*(d_i) = \sigma'(d_i)$.
   - If $i \geq \mu_\alpha$ and $\alpha_i = 1$ or $i = \mu_\alpha$, it follows by (3e) that $e_i \succ_\sigma^{f_i} g_{\nu_\alpha}$ and hence $e_i \succ_\sigma s$ which implies $\sigma^*(d_i) = \sigma'(d_i)$.
   - Finally, if $i < \mu_\alpha$ and $\alpha_i = 1$, it follows by (3f) that $f_{\mu_\alpha} \succ_\sigma^{h_{\mu_\alpha}} d_i$ implying that $s \succ_\sigma d_i$ and hence $\sigma^*(d_i) = \sigma'(d_i)$.

4. Let $\sigma := \sigma_{(n,\mathbf{1}_n,\gamma_{\mathbf{1}_n})}$ and $\sigma' := \mathcal{I}^{\mathtt{loc}}(\sigma)$. Lemma 10 (1a) implies that $\sigma|_{B_n} = \sigma'|_{B_n}$, (1b) implies $\sigma(r) = \sigma'(r)$, $\sigma(s) = \sigma'(s)$ and $r \succ_\sigma s$. Lemma 9 (1) implies that $\sigma|_{D_n} = \sigma'|_{D_n}$. Finally (1d) implies $\sigma(d_i) = \sigma'(d_i)$ for all i (due to the simple fact that f_i is more significant than all nodes belonging to the deceleration lane).

□

Theorem 12:

Proof. Let $f(n)$ denote the number of iterations the algorithm requires on $\mathcal{G}_n$. By induction on n and Lemma 11 the following can easily be shown:

$$f(n) = a + \sum_{\alpha \in I_n} d(\mu_\alpha) + e(n)$$

where $a = 6$, $e(n) = 2n + 2$, $d(i) = 2i + 7$ and $I_n = \{0,1\}^n \setminus \{\mathbf{0}_n, \mathbf{1}_n\}$.

It can be directly inferred that $f(1) = a + e(1) = 10$. Since $d(i)$ does not depend on n, $f(n+1)$ can be expressed in terms of $f(n)$:

$$\begin{aligned} f(n+1) &= a + 2(f(n) - a - e(n)) + \\ &\quad d(0) + d(n) + e(n+1) \\ &= 2f(n) + 8 \end{aligned}$$

By induction on n it is easy to see that $f(n) = 9 \cdot 2^n - 8$. □