The Complexity of All-switches Strategy Improvement*

John Fearnley† and Rahul Savani‡

Abstract

Strategy improvement is a widely-used and well-studied class of algorithms for solving graph-based infinite games. These algorithms are parametrized by a switching rule, and one of the most natural rules is "all switches" which switches as many edges as possible in each iteration. Continuing a recent line of work, we study all-switches strategy improvement from the perspective of computational complexity. We consider two natural decision problems, both of which have as input a game G, a starting strategy s, and an edge e. The problems are: 1. The edge switch problem, namely, is the edge e ever switched by all-switches strategy improvement when it is started from s on game G? 2. The optimal strategy problem, namely, is the edge e used in the final strategy that is found by strategy improvement when it is started from s on game G? We show PSPACE-completeness of the edge switch problem and optimal strategy problem for the following settings: Parity games with the discrete strategy improvement algorithm of Vöge and Jurdziński; mean-payoff games with the gain-bias algorithm [11, 33]; and discounted-payoff games and simple stochastic games with their standard strategy improvement algorithms. We also show PSPACE-completeness of an analogous problem to edge switch for the bottom-antipodal algorithm for Acyclic Unique Sink Orientations on Cubes.

1 Introduction

In this paper we study strategy improvement algorithms for solving two-player games such as parity games, mean-payoff games, discounted games, and simple stochastic games [21,33,39]. These games are interesting both because of their important applications and their unusual complexity status. Parity games, for example, arise in several areas of theoretical computer science, for example, in relation to the emptiness problem for tree automata [5,19] and as an algorithmic formulation of model checking for the modal μ-calculus [6,37]. Moreover, all of these problems are in NP ∩ coNP, and even UP ∩ coUP [2,23], so they are unlikely to be NP-complete. However, despite much effort from the community, none of these problems are known to be in P, and whether there exists a polynomial-time algorithm to solve these games is a very important and long-standing open problem.

Strategy improvement is a well-studied method for solving these games [21,33,39]. It is an extension of the well-known policy iteration algorithms for Markov decision processes. The algorithm selects one of the two players to be the strategy improver. Each strategy of the improver has a set of *switchable* edges, and switching any subset of these edges produces a strictly better strategy. So, the algorithm proceeds by first choosing an arbitrary starting strategy, and then in each iteration, switching some subset of the switchable edges. Eventually this process will find a strategy with no switchable edges, and it can be shown that this strategy is an optimal strategy for the improver.

To completely specify the algorithm, a *switching rule* is needed to pick the subset of switchable edges in each iteration (this is analogous to the pivot rule used in the simplex method). Many switching rules have been proposed and studied [8,20,26,27,34]. One of the most natural rules, and the one that we consider in this paper, is the *all-switches* rule, which always switches a vertex if it has a switchable edge. In particular, we consider *greedy* all-switches, which chooses the best edge whenever more than one edge is switchable at a vertex (ties are broken arbitrarily). For a long time, the all-switches variant of the discrete strategy improvement algorithm of Vöge and Jurdzinski [39], was considered the best candidate for a polynomial-time algorithm to solve parity games. Indeed no example was known that required a super-linear number of iterations. However, these hopes were dashed when Friedmann showed an exponential lower bound for greedy all-switches strategy improvement for parity games [12]. In the same paper, he showed that his result extends to strategy improvement algorithms for discounted games, and simple stochastic games. Friedmann's lower bound triggered a flurry of work into related problems, that lead to exponential lower bounds on *policy iteration* algorithm for *Markov decision processes* [7], and ultimately to a number of new lower bounds for various simplex pivoting algorithms [13,15].

The computational power of pivot algorithms. In this paper, we follow a recent line of work that seeks to explain the poor theoretical performance of pivoting algorithms using a complexity theoretic point of view. The first results in this direction were proved

*This work was supported by EPSRC grant EP/L011018/1 "Algorithms for Finding Approximate Nash Equilibria." The full version of this paper, will full proofs, is available at http://arxiv.org/abs/1507.04500

†University of Liverpool
‡University of Liverpool

for problems in PPAD and PLS. It is known that, if a problem is *tight*-PLS-*complete* then computing the solution found by the natural improvement algorithm is PSPACE-complete [22]. Similarly, for the canonical PPAD-complete problem end-of-the-line, computing the solution that is found by the natural line following algorithm is PSPACE-complete [31]. This has recently been extended to show that is PSPACE-complete to compute any of the solutions that can be found by the Lemke-Howson algorithm [18], a pivoting algorithm that solves the PPAD-complete problem of finding a Nash equilibrium of a bimatrix game.

Until recently, results of this type were only known for algorithms for problems that are unlikely to lie in P. However, a recent series of papers has shown that similar results hold even for the simplex method for linear programming! Adler, Papadimitriou and Rubinstein [1] showed that there exists an artificially contrived pivot rule for which is PSPACE-complete to decide if a given basis is on the path of bases visited by the simplex algorithm with this pivot rule. Disser and Skutella [4] studied the natural pivot rule that Dantzig proposed when he introduced the simplex method, and they showed that it is NP-hard to decide whether a given variable enters the basis when the simplex method is run with this pivot rule. Finally, Fearnley and Savani strengthened both these results by showing that the decision problem that Disser and Skutella considered is actually PSPACE-complete [10], and they also showed that determining if a given variable is used in the *final optimal solution* found by the algorithm is PSPACE-complete. This result exploited a known connection between *single-switch* policy iteration for Markov Decision Processes (MDPs) and the simplex method for a corresponding LP. The result was first proved for a greedy variant of single-switch policy iteration, which then implied the result for the simplex method with Dantzig's pivot rule.

All of the results on linear programming are motivated by the quest to find a *strongly polynomial* algorithm for linear programming, which was included in Smale's list of great unsolved problems of the 21st century [36]. One way of resolving this problem would be to design a polynomial-time pivot rule for the simplex method, and if we are to do this, then it is crucial to understand why existing pivot rules fail. The PSPACE-completeness indicate that they fail not because they are stupid, but in fact because they are capable of solving any problem that can be computed in polynomial space.

We face a similar quest to find a polynomial-time algorithm for the games studied in this paper. Strategy improvement is a prominent algorithm for solving these games, and indeed it is one of the only algorithms for solving discounted and simple stochastic games. So, devising a polynomial-time switching rule is an obvious direction for further study. It may in fact be easier to devise a polynomial time switching rule, because there is a lot more freedom in each step of the algorithm: simplex pivot rules correspond to switching rules *that can only switch a single edge*, whereas strategy improvement rules can switch *any subset of edges*. Indeed, it may be the case that the polynomial Hirsch conjecture fails, ruling out a strongly polynomial simplex method, but the analogue of the Hirsch conjecture for strategy improvement is known to be true: one can always reach an optimal strategy in at most n strategy improvement iterations, where n is the number of vertices.

Our contribution. Our main results are that, for greedy all-switches strategy improvement, determining whether the algorithm switches a given edge is PSPACE-complete, and determining whether the optimal strategy found by the algorithm uses a particular edge is PSPACE-complete. One of the key features that strategy improvement has is the ability to switch multiple switchable edges at the same time, rather than just one as in the simplex method. Our results show that naively using this power does not help to avoid the PSPACE-completeness results that now seem to be endemic among pivoting algorithms.

The proof primarily focuses on the strategy improvement algorithm of Vöge and Jurdziński for solving parity games [39]. We now give a short overview of the algorithm so that we can explain our results, while a formal definition will be given in Section 2. A parity game is played by two players on a finite graph. The vertices are partitioned between the two players, who are called *Odd* and *Even*. A strategy for one of the two players chooses an outgoing edge at each vertex belonging to that player. The strategy improvement algorithm takes the point of view of player Even. It starts with an arbitrarily chosen player Even strategy. In each iteration it computes a set of *switchable edges*, and as we have mentioned, switching any subset of switchable edges yields an improved strategy. The greedy all-switches variant of strategy improvement always switches every vertex that has a switchable edge, and if a vertex has more than one switchable edge, it selects the "best" switchable edge (according to a valuation function that we will be defined formally later.) The following definition formalises the problem that we are interested in.

DEFINITION 1.1. *Let G be a game, and let e be an edge and σ be a player Even strategy. The problem* EDGESWITCH(G, σ, e) *is to decide if the edge e is ever switched by greedy all-switches strategy improvement*

when it is applied to G starting from σ.

The main technical contribution of the paper is to show the following theorem.

THEOREM 1.1. EDGESWITCH *for Vöge and Jurdziński's algorithm is* PSPACE*-complete.*

We use this theorem to show similar results for other games. For mean-payoff games, our results apply to the gain-bias algorithm [11]; and for discounted and simple stochastic games our results apply to the standard strategy improvement algorithms [3, 32]. We utilise the well-known polynomial-time reductions from parity games to mean-payoff games [32], mean-payoff games to discounted games, and from discounted games to simple stochastic games [40]. The parity games we construct have the property that when they are reduced to the other games mentioned above strategy improvement will behave in the same way (for discounted and simple stochastic games this was already observed by Friedmann [12]), so we get the following corollary of Theorem 1.1.

COROLLARY 1.1. EDGESWITCH *for the gain-bias algorithm, and the standard strategy improvement algorithms for discounted-payoff and simple stochastic games is* PSPACE*-complete.*

Theorem 1.1 proves a property about the path taken by strategy improvement during its computation. Previous results have also studied the complexity of finding the optimal strategy that is produced by strategy improvement, which we encode in the following problem.

DEFINITION 1.2. *Let G be a game, and let e be an edge and σ be a player Even strategy. The problem* OPTIMALSTRATEGY(G, σ, e) *is to decide if the edge e is used in the optimal strategy that is found by greedy all-switches strategy improvement when applied to G starting from σ.*

Augmenting our construction for parity games with an extra gadget gives the following theorem.

THEOREM 1.2. OPTIMALSTRATEGY *for Vöge and Jurdziński's algorithm is* PSPACE*-complete.*

This result requires that the parity game has multiple optimal solutions because otherwise the PSPACE hardness of OPTIMALSTRATEGY would imply NP $\cap$ coNP = PSPACE. With further modifications, we can again extend this result to strategy improvement algorithms for other games.

COROLLARY 1.2. OPTIMALSTRATEGY *for the gain-bias algorithm, and the standard strategy improvement algorithms for discounted-payoff and simple stochastic games is* PSPACE*-complete.*

Our results can also be applied to *unique sink orientations* (USOs.) A USO of an n-dimensional hypercube is an orientation of its edges such that every face has a unique sink [38]. The fundamental algorithmic problem for USOs is to find the global sink assuming oracle access to the edge orientation. The design and analysis of algorithms for this problem is an active research area [14,15,17,20,29,35], in particular for *acyclic* orientations (AUSOs). The BOTTOMANTIPODAL algorithm [35] for AUSOs on cubes starts at an arbitrary vertex and in each iteration jumps to the antipodal vertex in the sub-cube spanned by the outgoing edges at the current vertex.

For binary games, where vertices have outdegree at most two, and no two strategies have the same value, the valuation functions used by strategy improvement induce an AUSO on a cube, and all-switches strategy improvement corresponds to BOTTOMANTIPODAL on this AUSO. For non-binary games we instead get an AUSO on a grid [16], so our results immediately give a PSPACE-hardness result for grid USOs for a problem analogous to EDGESWITCH. To get a similar result for AUSOs on cubes we turn our construction into a binary parity game, and we get the following.

COROLLARY 1.3. *Let C be a d-dimensional cube AUSO, specified by a poly(d)-sized circuit that computes the edge orientations for each vertex of C. Given a dimension $k \in \{1, \ldots, d\}$, and a vertex v, it is* PSPACE*-complete to decide if* BOTTOMANTIPODAL *started at v, ever switches the kth coordinate.*

Since a USO has a unique solution, by definition, we cannot hope to get a result for AUSOs that is analogous to OPTIMALSTRATEGY, since, as noted above, PSPACE-hardness of OPTIMALSTRATEGY requires multiple optimal solutions under standard complexity-theoretic assumptions.

Related work. The techniques used in this work can be compared to the techniques used to show PSPACE-hardness results for the simplex algorithm equipped with Dantzig's pivoting rule [10]. In that work, the lower bound was ultimately shown for *policy iteration* algorithms for *Markov decision processes.* The reduction used there shares some similarities with the construction that we present in this paper: both constructions are based on simulating a circuit iteration problem using NOT and OR gates, and in both cases the whole construction is driven by a *clock* gadget.

The main difference is that, while Dantzig's pivot rule only switches one vertex in each iteration, the greedy all-switches rule switches every vertex that has a switchable edge. This necessitates a radically different construction, where, as we will explain, the circuit gadgets must be capable of being switched in every iteration. Also, we must use *two* separate clock gadgets, which must be carefully synchronised.

We now discuss other related work. The best known algorithms for parity games games have subexponential running time: the *random facet strategy improvement* algorithms combine strategy improvement with the random-facet algorithm for LPs [26–28]. Following the work of Friedmann [12] that we build on heavily in this paper, Friedmann, Hansen, and Zwick showed a sub-exponential lower bound for the random facet strategy improvement algorithm [14]. They also used a construction of Fearnley [7] to extend the bound to the random facet pivot rule for the simplex method [15]. For parity games, a deterministic subexponential-time algorithm is known [24].

2 Preliminaries

2.1 Parity games

A parity game is a tuple $G = (V, V_{\text{Even}}, V_{\text{Odd}}, E, \text{pri})$, where (V, E) is a directed graph. The sets V_{Even} and V_{Odd} partition V into the vertices belonging to player Even, and the vertices belonging to player Odd, respectively. The *priority* function $\text{pri} : V \to \mathbb{N}$ assigns a natural number to each vertex. We assume that every vertex has at least one outgoing edge. The algorithm of Vöge and Jurdziński also requires that we assume that every priority is assigned to at most one vertex.

A strategy for Even is a function $\sigma : V_{\text{Even}} \to V$ such that, for each $v \in V_{\text{Even}}$ we have that $(v, \sigma(v)) \in E$. Strategies for player Odd are defined analogously. We use Σ_{Even} and Σ_{Odd} to denote the set of strategies for players Even and Odd, respectively.

A *play* is a sequence $v_0, v_1, \ldots$ such that for all $i \in \mathbb{N}$ we have $v_i \in V$ and $(v_i, v_{i+1}) \in E$. Given a pair of strategies $\sigma \in \Sigma_{\text{Even}}$ and $\tau \in \Sigma_{\text{Odd}}$, and a starting vertex v_0, there is a unique play that starts at v_0 and follows both of the strategies. So, we define $\text{Play}(v_0, \sigma, \tau) = v_0, v_1, \ldots$, where for each $i \in \mathbb{N}$ we have $v_{i+1} = \sigma(v_i)$ if $v_i \in V_{\text{Even}}$, and $v_{i+1} = \tau(v_i)$ if $v_i \in V_{\text{Odd}}$.

Given a play $\pi = v_0, v_1, \ldots$ we define $\text{MaxIo}(\pi) = \max\{p \ : \ \exists \text{ infinitely many } i \in \mathbb{N} \text{ s.t. } \text{pri}(v_i) = p\}$, to be the maximum priority that occurs *infinitely often* along π. We say that a play π is *winning* for player Even if $\text{MaxIo}(\pi)$ is even, and we say that π is winning for Odd if $\text{MaxIo}(\pi)$ is odd. A strategy $\sigma \in \Sigma_{\text{Even}}$ is a *winning strategy* for a vertex $v \in V$ if, for every strategy $\tau \in \Sigma_{\text{Odd}}$, we have that $\text{Play}(v, \sigma, \tau)$ is winning for player Even. Likewise, a strategy $\tau \in \Sigma_{\text{Odd}}$ is a winning strategy for v if, for every strategy $\sigma \in \Sigma_{\text{Even}}$, we have that $\text{Play}(v, \sigma, \tau)$ is winning for player Odd. The following fundamental theorem states that parity games are *determined*.

THEOREM 2.1. ([5, 30]) *In every parity game, the set V can be partitioned into sets (W_0, W_1), where Even has a winning strategy for all $v \in W_0$, and Odd has a winning strategy for all $v \in W_1$.*

2.2 Strategy improvement

Our main theorem is about the strategy improvement algorithm of Vöge and Jurdziński [39] for solving parity games. Friedmann observed that, for the purposes of showing lower bounds, the algorithm can be greatly simplified. Here, we present only the simplified version. A description of the complete algorithm can be found in the full version of the paper.

One-sink games. The simplified algorithm only works if the input is a *one-sink game*. These games contain a sink vertex $s \in V$ such that $\text{pri}(s) = 1$ and $(s, s) \in E$ is the only outgoing edge from s. An even strategy $\sigma \in \Sigma_{\text{Even}}$ is a *terminating strategy* if, for every vertex v, the path that starts at v and follows σ eventually reaches the sink, no matter how the opponent plays. A parity game is a one-sink parity game if the following conditions hold. Firstly, there should be a sink vertex s which satisfies the properties given above. Secondly, there should be no vertex v with $\text{pri}(v) = 0$. Thirdly, there should exist at least one terminating strategy. Finally, player Even should not win the parity game, ie., $W_0 = \emptyset$.

Valuations. The algorithm assigns a *valuation* to each vertex v under every terminating strategy $\sigma \in \Sigma_{\text{Even}}$ and every strategy $\tau \in \Sigma_{\text{Odd}}$. Since σ is terminating, for every vertex v we know that $\text{Play}(v, \sigma, \tau)$ consists of a finite path $P(v, \sigma, \tau)$ that ends at the sink. We define the valuation function $\text{Val}^{\sigma,\tau}(v) = \{\text{pri}(u) \ : \ u \in P(v, \sigma, \tau) \text{ and } \text{pri}(u) > 1\}$. That is, the valuation of v is the set of priorities on the path from v to the sink. Next we define an order on these sets. Let $P, Q \subset \mathbb{N}$. We first define: $\text{MaxDiff}(P, Q) = \max\big((P \setminus Q) \cup (Q \setminus P)\big)$. If $d = \text{MaxDiff}(P, Q)$ then we define $P \sqsubset Q$ to hold if one of the following conditions holds: d is even and $d \in Q$, or d is odd and $d \in P$. Furthermore, we have that $P \sqsubseteq Q$ if either $P = Q$ or $P \sqsubset Q$.

Intuitively, the $\sqsubseteq$ ordering describes the following preferences. Player Even prefers to see large even priorities, and prefers to avoid seeing large odd priorities. Thus, the valuation considers the largest priority that is not shared by the two valuations, and then decides based on its parity. Therefore, if we want to make a

path desirable according to the valuation, we can place a large even priority on it, while if we want to make it undesirable, we can place a large odd priority on it.

Best responses. Given a strategy $\sigma \in \Sigma_{\text{Even}}$, we can find a *best response* strategy for player Odd. To do so, we first eliminate the player Odd strategies τ in which, when played against σ, there is an cycle of even parity. These cannot be best responses, because in a terminating strategy player Odd can always reach an odd cycle by forcing play to the sink. Let $\Sigma^{\sigma}_{\text{Odd}}$ be the set of Odd strategies that do not form even cycles when played against σ. Then, we select the strategy $\tau^* \in \Sigma^{\sigma}_{\text{Odd}}$ such that, for every $\tau \in \Sigma^{\sigma}_{\text{Odd}}$ and every vertex v we have: $\text{Val}^{\sigma,\tau}(v) \sqsubseteq \text{Val}^{\sigma,\tau^*}(v)$. Vöge and Jurdziński showed that a best response can always be computed in polynomial time. We define $\text{Br}(\sigma)$ to be an arbitrarily chosen best response strategy against σ. Furthermore, we define $\text{Val}^{\sigma}(v) = \text{Val}^{\sigma,\text{Br}(\sigma)}(v)$.

Switchable edges. Let $\sigma \in \Sigma_{\text{Even}}$ be a strategy for the Even player and $(v,u) \in E$ be an edge such that $\sigma(v) \neq u$. We say that (v,u) is *switchable* in σ if $\text{Val}^{\sigma}(\sigma(v)) \sqsubset \text{Val}^{\sigma}(u)$. Furthermore, we define a *most appealing* outgoing edge at a vertex v to be an edge (v,u) such that, for all edges (v,u') we have $\text{Val}^{\sigma}(u') \sqsubseteq \text{Val}^{\sigma}(u)$.

There are two fundamental properties of switchable edges. Firstly, switching any subset of the switchable edges produces an improved strategy. Let σ be a strategy, and let $W \subseteq E$ be a set of switchable edges in σ such that, for each vertex v, there is at most one edge of the form $(v,u) \in W$. *Switching* W in σ creates a new strategy $\sigma[W]$ where for all v we have: $\sigma[W](v) = u$ if $(v,u) \in W$, and $\sigma[W](v) = \sigma(v)$ otherwise. Secondly, a strategy with no switchable edges is optimal: a strategy $\sigma \in \Sigma_{\text{Even}}$ is *optimal* if for every vertex v and every strategy $\sigma' \in \Sigma_{\text{Even}}$ we have $\text{Val}^{\sigma'}(v) \sqsubseteq \text{Val}^{\sigma}(v)$.

LEMMA 2.1. ([39]) *We have:*

1. *Let σ be a terminating strategy and let $W \subseteq E$ be a set of switchable edges in σ such that, for each vertex v, there is at most one edge of the form $(v,u) \in W$. Then, $\sigma[W]$ is terminating, and for every vertex v we have $\text{Val}^{\sigma}(v) \sqsubseteq \text{Val}^{\sigma[W]}(v)$. Moreover, there exists a vertex v for which $\text{Val}^{\sigma}(v) \sqsubset \text{Val}^{\sigma[W]}(v)$.*
2. *A strategy with no switchable edges is optimal.*

The algorithm. These properties give rise to an obvious *strategy improvement* algorithm that finds an optimal strategy. The algorithm begins by selecting an arbitrary terminating strategy $\sigma \in \Sigma_{\text{Even}}$. In each iteration, the algorithm performs the following steps:

1. If there are no switchable edges, then terminate.
2. Otherwise, select a set $W \subseteq E$ of switchable edges in σ such that, for each vertex v, there is at most one edge of the form $(v,u) \in W$. Set $\sigma := \sigma[W]$ and go to step 1.

By the first property, each iteration of this algorithm produces a strictly better strategy according to the $\sqsubseteq$ ordering, and therefore the algorithm must eventually terminate. However, the algorithm can only terminate when there are no switchable edges, and therefore the second property implies that the algorithm will always find an optimal strategy.

To get a complete algorithm, we must specify a *switching rule*, that chooses which subset of switchable edges should be chosen in each iteration. In this paper, we focus on the *greedy all-switches* switching rule. This rule switches every vertex that has a switchable edge, and if there is more than one switchable edge, it arbitrarily picks one of the most appealing edges.

2.3 Circuit iteration problems

The problems. To prove our PSPACE-completeness results, we will reduce from *circuit iteration problems*, which we now define. A *circuit iteration* instance is a triple (F,B,z), where: $F : \{0,1\}^n \to \{0,1\}^n$ is a function represented as a boolean circuit C, $B \in \{0,1\}^n$ is an initial bit-string, and z is an integer such that $1 \leq z \leq n$. We use standard notation for function iteration: given a bit-string $B \in \{0,1\}^n$, we recursively define $F^1(B) = F(B)$, and $F^i(B) = F(F^{i-1}(B))$ for all $i > 1$.

We now define two problems that will be used as the starting point for our reduction. Both are decision problems that take as input a circuit iteration instance (F,B,z).

- BITSWITCH(F,B,z): decide whether there exists an even $i \leq 2^n$ such that the z-th bit of $F^i(B)$ is 1.
- CIRCUITVALUE(F,B,z): decide whether the z-th bit of $F^{2^n}(B)$ is 1.

The requirement for i to be even in BITSWITCH is a technical requirement that is necessary in order to make our reduction to strategy improvement work.

It has been shown that both of these problems are PSPACE-complete [10]. This should not be too surprising, because F can simulate a single step of a space-bounded Turing machine, so when F is iterated, it simulates a run of the space-bounded Turing machine.

For our reduction, we will make a number of technical assumptions about the circuits used to represent F, which are discussed in the full version of the paper.

Here we highlight some of the most important assumptions. Firstly, we assume that the circuits contain only NOT and OR gates. Secondly, we assume that the circuits are presented in *negated form*, which means that the ith output of the circuit is 1 on input B if and only if $F(B)_i = 0$.

3 The Construction

The main technical result of the paper is to show that EDGESWITCH is PSPACE-complete by reducing from the circuit iteration problem BITSWITCH. Let (F, B, z) be a circuit iteration instance, and let C be the negated form of the circuit that computes F. We construct a parity game that forces greedy all-switches strategy improvement to compute $F^i(B)$ for each i.

The construction will contain two copies of the circuit C, which are numbered 0 and 1. The circuits take turns in computing F. First circuit 0 computes $F(B)$, then circuit 1 computes $F^2(B)$, then circuit 0 computes $F^3(B)$, and so on.

Each circuit is equipped with its own *clock,* which dictates when the circuit should be computing. To do this, each clock has two special vertices r and s. Most of the time, the valuation of r is much larger than the valuation of s. However, every so often the clock *ticks*, and there is a single iteration in which the valuation of s is much larger than the valuation of r. This signal causes the circuit to start computing.

When the circuits compute, the gates are evaluated in *depth* order. In particular, we transform our circuits so that both inputs to an OR-gate have the same depth. Then, in the first iteration the gates that read from inputs are evaluated, in the second iteration the gates of depth 2 are evaluated, and so on. Thus, if $d(C)$ denotes the depth of the circuit, then after $d(C)$ strategy improvement iterations the output of the circuit will be evaluated. We then spend one extra iteration to store each output bit in a special INPUT/OUTPUT gadget.

The INPUT/OUTPUT gadgets play a crucial role in the construction, because they *move between the circuits.* As we have described so far, the clock for Circuit 0 ticks, and after spending $d(C) + 1$ iterations computing, the output of the circuit will be stored in the INPUT/OUTPUT gadgets for Circuit 0. Then, in some later iteration, the clock for Circuit 1 ticks. At this point, the INPUT/OUTPUT gadgets in Circuit 0 move to Circuit 1, where they act as input gates, which output their stored values to the gates of depth 1. In other words, during a computation in Circuit j, we have that the INPUT/OUTPUT gadgets in Circuit $1-j$ are in *input mode*, where they feed their stored values into Circuit j, meanwhile the INPUT/OUTPUT gadgets in Circuit j are in *output mode*, where they read the output and then store it.

In this way, we force greedy all-switches strategy improvement to compute $F^i(B)$ for increasingly large values of i. Each INPUT/OUTPUT gadget contains a specific edge that is switched only when the bit stored by the gadget is a 1. Thus, we can choose the particular edge e that corresponds to the zth output bit, and use this edge to monitor whether $F^i(B)_z$ is ever 1. Note that, since there are two circuits, each INPUT/OUTPUT gadget only holds $F^i(B)_z$ for either every even i, or every odd i. This is why we needed the technical restriction on BITSWITCH to only consider $F^i(B)$ for every even i. Hence, we show that EDGESWITCH is PSPACE-complete. In what follows, we give more details on how each of the gadgets is implemented.

The clock. For the clocks, we adapt Friedmann's construction [12]. In order to show that greedy-all switches strategy improvement can take exponential time, Friedmann constructed an example that forces greedy all-switches strategy improvement to simulate a binary counter. The example contains n *bit gadgets*, each of which stores a single bit. Each strategy in the example encodes a binary string of length n, and Friedmann showed that greedy all-switches strategy improvement must pass through one strategy for each possible bit-string, thus giving a 2^n lower bound on the running time of the algorithm.

One key property of Friedmann's construction is that the number of strategy improvement iterations between one bit-string and the next can be made arbitrarily large. We exploit this in order to use the construction as a clock. We setup each clock so that it ticks precisely when it moves from one bit-string to the next, and the delay between bit-strings is configured so that there is enough time for both circuits to compute their output.

As we have mentioned, the construction contains two independent clocks, which control their respective circuits. In order for the construction to work, these two clocks must be carefully synchronised. In particular, Clock 0 must always tick at least $k + 1$ iterations after Clock 1, and likewise Clock 1 must always tick at least $k+1$ iterations after Clock 0. If this is not the case, then there will not be enough time for a circuit to compute its output before the next circuit starts computing.

Circuits. Each gate in each circuit will be represented by a gadget. The gadget for gate i in circuit j has an *output state* that will be denoted by o_i^j. The valuation of this output state will represent the output of the gate. In particular, the valuations of these output vertices will follow three rules.

- Before the gate is evaluated, the valuation of o_i^j will be small.

- If the gate is evaluated to false, then the valuation of o_i^j will remain small.

- If the gate is evaluated to true, then the valuation of o_i^j will become large.

Figure 1 shows all of the gadgets used in our construction. Box vertices belong to player Even while circle vertices belong to player Odd. Since we are required to assign each priority to at most one vertex, the actual construction must use a complicated function to determine priorities. In the diagrams we use *representative priorities,* which maintain the order and parity of the priorities used in the gadgets. A complete formal specification of the construction, and a proof of correctness, can be found in the full version of the paper. We now give a high level overview of the design of these gadgets.

The or gate. The OR-gate gadget has a simple design with a single state o_i^j. It has edges to the clock vertices r and s, and to the output-vertices of its two inputs. Before the gate is evaluated, the state o_i^j chooses the edge to r. If both inputs evaluate to false, then o_i^j does not switch away from r. In both of these cases the odd priority on o_i^j ensures that the valuation of the state remains low. On the other hand, if at least one of the two inputs evaluates to true, then o_i^j switches to the corresponding input state, and the large valuation of the input then causes the valuation of o_i^j to also be large.

The not gate. For the NOT-gate gadget, we use a modified version of the bit-gadget used by Friedmann in his examples. The value of the bit is represented by the strategy chosen at d_i^j: the bit is 1 if d_i^j chooses the edge to e_i^j, and it is 0 otherwise. The edge from d_i^j to e_i^j is always switchable, but to prevent it from switching, Friedmann uses a gadget that he calls a *deceleration lane*. This gadget provides a series of *distracting* switchable edges (to the vertices $a_{i,l}^j$) that are each more appealing than the edge from d_i^j to e_i^j. Since greedy all-switches strategy improvement always switches the most appealing edge, as long as there is a distracting edge to the deceleration lane, the vertex d_i^j is prevented from switching to e_i^j.

To turn this into a NOT-gate, we use a modified deceleration lane. This deceleration lane provides distracting switchable edges until the gate is evaluated. Then, if the input gate is evaluated to true, the modified deceleration lane continues providing distracting switchable edges, but if the input gate is evaluated to false, no more distracting switchable edges are produced. In this way, the edge from d_i^j to e_i^j is switched (ie., the bit flips to 1) if and only if the input gate evaluated to false. This gives us a NOT-gate.

To see that the gadget outputs the correct values, observe that there is a large odd priority on o_i^j, but an even larger even priority on h_i^j. If the edge from d_i^j to e_i^j is not switched, then player Odd prefers not to use the edge to h_i^j, and the odd priority causes the valuation of o_i^j to be small. On the other hand, if the edge is switched, Odd is forced to escape from the even cycle, so then the even priority on h_i^j causes the valuation of o_i^j to be large.

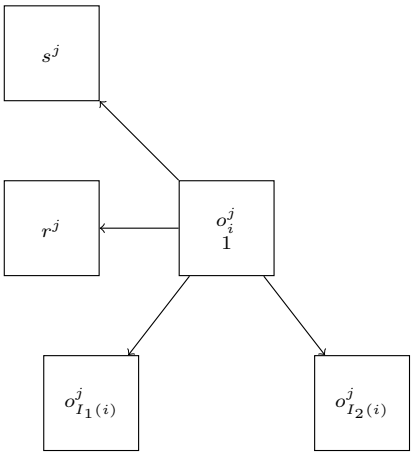
The input/output gadget. The input/output gadget is the most complicated part of the construction, because as we have mentioned, it plays two different roles. At a high level, the gadget is a *relocatable* NOT-gate. When the gadget is in output mode, it is a NOT-gate of depth $d(C)+1$ in circuit j, and when the gadget is in input mode, it is a NOT-gate of depth 0 in circuit $1-j$. The extra vertices added to the NOT-gate and a special modified deceleration lane perform the transition between these two modes.

Since the gadget is used in both circuits, it must be connected to both clocks. The special circuit mover gadget plays an important role here. The vertices y^j and z^j replace the clock-vertices r^j and s^j used in the NOT-gate gadget. The state y^j will switch to r^{1-j} when clock $1-j$ ticks, which changes the mode of the gadget.

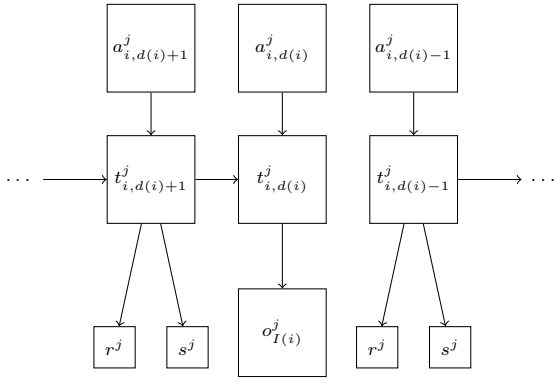
One critical property of the transition is that the operation of the internal NOT-gate should not be affected. This is achieved by ensuring that all of the vertices involved in the transition (y^j, $h_{i,0}^j$, and p_i^j) switch during the same iteration. Since the NOT-gate gadget only cares about the *relative* valuations of its outgoing edges, if all of these edges are switched at once, then the operation of the NOT-gate is not affected. This ensures that the data held in the NOT-gate is not destroyed.

4 Other results

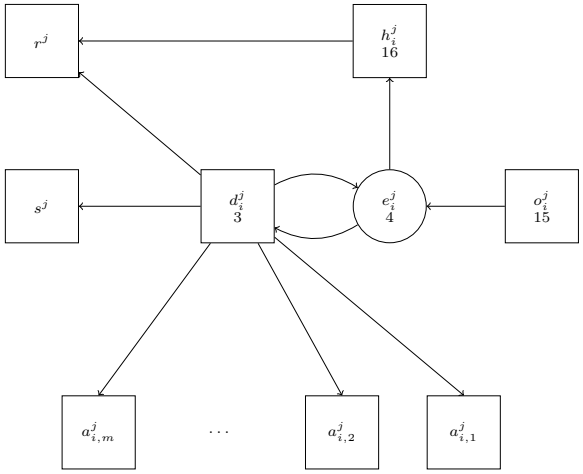
EdgeSwitch for other problems. Once we have shown that EDGESWITCH is PSPACE-complete, we then get several subsequent results. Firstly, we get that EDGESWITCH is PSPACE-complete for the gain-bias algorithm for mean-payoff games [11, 33], the standard strategy improvement algorithm for discounted games, and the standard strategy improvement algorithm for simple-stochastic games. The results concerning discounted and simple-stochastic games follow from a result of Friedmann [12], which shows that, for a one-sink parity game, after applying the standard reduction from parity games to discounted and simple-stochastic games, the standard strategy improvement algorithms for these games switch exactly the same edges as the parity game strategy improvement algorithm. For the result concerning mean-payoff games, we observe that



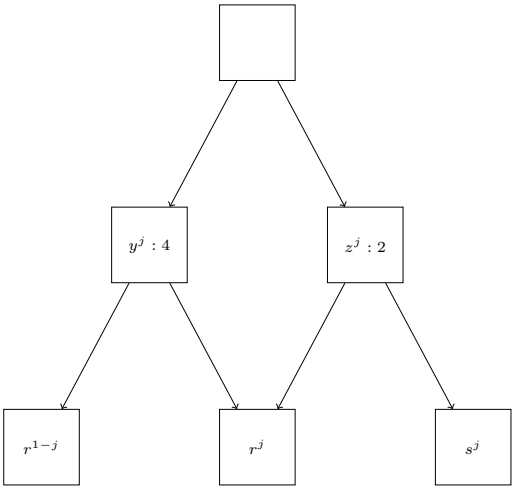
(a) The OR gate.



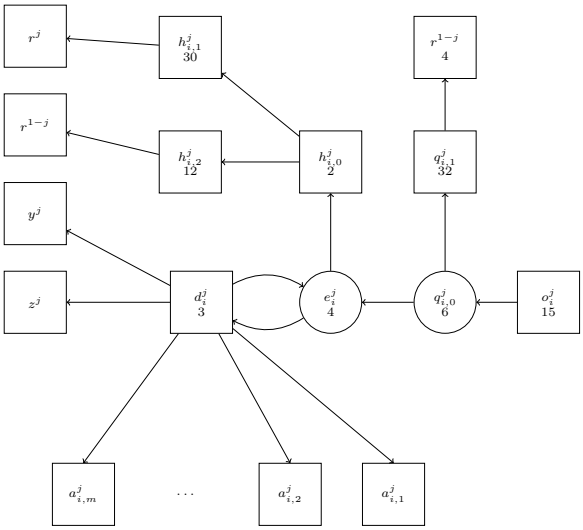
(b) Modified deceleration lane for NOT gate i in circuit j.



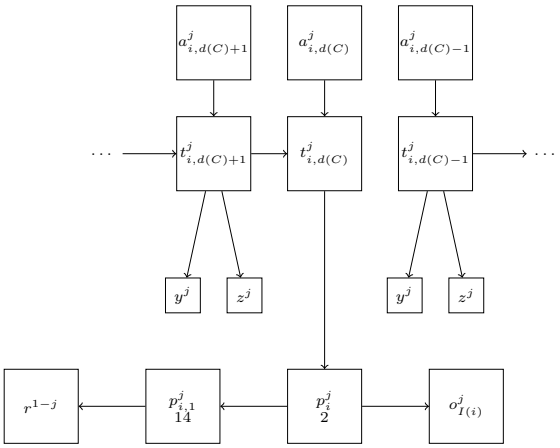
(c) NOT gate with index i in circuit j.



(d) Circuit mover gadget for circuit j.



(e) INPUT/OUTPUT gate with index i in circuit j.



(f) Modified deceleration lane for an INPUT/OUTPUT gate i in circuit j.

Figure 1: The gadgets used in our construction.

the same property holds after applying the standard reduction from parity games to mean-payoff games.

We also get a result for the bottom-antipodal algorithm for acyclic unique sink orientations on hypercubes. This result requires more work, because it requires us to transform the game into a *binary parity game*, which requires that every vertex has exactly two outgoing edges. Friedmann has already shown that his example can be transformed into a binary game. The clocks and NOT-gates in our construction can use the same transformation. For the OR-gates, we use a redesigned gadget. This gadget takes up to two-iterations to evaluate the gate after all of the inputs have been evaluated, so we must slightly alter the construction to allow for two-iterations to evaluate each level of gates. After applying this transformation, we obtain a binary parity game, which then gives the result for USOs on hypercubes.

OptimalStrategy for parity games. Our construction can be modified to prove that OPTIMALSTRATEGY is PSPACE-complete for parity games. The idea here is to run the construction until $F^{2^n}(B)$ has been computed. At this point, the strategy choice made at a particular vertex d in one of the INPUT/OUTPUT gadgets will encode whether the zth bit of $F^{2^n}(B)$ is 1 or 0. We use an extra gadget that makes d permanently indifferent between all of its outgoing edges. Since strategy improvement never switches an indifferent edge, the strategy chosen at d in the optimal strategy will encode $F^{2^n}(B)$, as required. It should be noted that this result must be carried out in Vöge and Jurdziński's original algorithm, rather than the simplified one used for EDGESWITCH, because it is not possible for a vertex to be indifferent in a one-sink game.

The extra gadget consists of a third clock. This clock is modified so that, after 2^n iterations, a particular state f gets a very large valuation. Each edge (d, u) is replaced with three edges (d, v_u), (v_u, u), and (v_u, f), where v_u is a new state for player Even that can either move to u, or to f. The idea is that each state v_u chooses the edge to u as normal for the first 2^n iterations, and then once the valuation of f becomes large, they all simultaneously switch to f, and make d permanently indifferent.

OptimalStrategy for other games. The result for OPTIMALSTRATEGY can also be generalised to the gain-bias algorithm for mean-payoff games, and the standard strategy improvement algorithms for discounted and simple-stochastic games. Once again, the standard reduction from parity games is applied, but some small extra modifications are required (in particular, the rewards on the vertices v_u must be set to 0) to ensure that d^1_z remains indifferent in the valuations used by these algorithms.

5 Open problems

There are several open problems that arise from this work. Strategy improvement generalizes policy iteration which solves mean-payoff and discounted-payoff Markov decision processes [33]. The exponential lower bounds for greedy all-switches have been extended to MDPs: Fearnley showed that the second player in Friedmann's construction [12] can be simulated by a probabilistic action, and used this to show an exponential lower bound for the all-switches variant of policy iteration of average-reward MDPs [7]. This technique cannot be applied to the construction in this paper, because we use additional Odd player vertices (in particular the vertices $q^j_{i,1}$) that cannot be translated in this way. Can our PSPACE-hardness results be extended to all-switches strategy improvement for MDPs?

Also, there are other pivoting algorithms for parity games that deserve attention. It has been shown that Lemke's algorithm and the Cottle-Dantzig algorithm for the P-matrix linear complementarity problem (LCP) can be applied to parity, mean-payoff, and discounted games [9, 25]. It would be interesting to come up with similar PSPACE-completeness results for these algorithms, which would also then apply to the more general P-matrix LCP problem.

References

[1] I. Adler, C. H. Papadimitriou, and A. Rubinstein. On simplex pivoting rules and complexity theory. In *Proc. of IPCO*, pages 13–24, 2014.

[2] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[3] A. Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[4] Y. Disser and M. Skutella. The simplex algorithm is np-mighty. In *Proc. of SODA*, pages 858–872, 2015.

[5] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. of FOCS*, pages 368–377, 1991.

[6] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *Proc. of CAV*, 1993.

[7] J. Fearnley. Exponential lower bounds for policy iteration. In *Proc. of ICALP*, pages 551–562, 2010.

[8] J. Fearnley. Non-oblivious strategy improvement. In *Proc. of LPAR*, 2010.

[9] J. Fearnley, M. Jurdziński, and R. Savani. Linear complementarity algorithms for infinite games. In *Proc. of SOFSEM*, pages 382–393, 2010.

[10] J. Fearnley and R. Savani. The complexity of the

simplex method. In *Proc. of STOC*, pages 201–208, 2015.

[11] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

[12] O. Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Logical Methods in Computer Science*, 7(3), 2011.

[13] O. Friedmann. A subexponential lower bound for Zadeh's pivoting rule for solving linear programs and games. In *Proc. of IPCO*, pages 192–206, 2011.

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

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

[16] B. Gärtner, W. D. J. Morris, and L. Rüst. Unique sink orientations of grids. *Algorithmica*, 51(2):200–235, 2008.

[17] B. Gärtner and I. Schurr. Linear programming and unique sink orientations. In *Proc. of SODA*, pages 749–757, 2006.

[18] P. W. Goldberg, C. H. Papadimitriou, and R. Savani. The complexity of the homotopy method, equilibrium selection, and Lemke-Howson solutions. *ACM Trans. Economics and Comput.*, 1(2):9, 2013.

[19] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

[20] T. D. Hansen, M. Paterson, and U. Zwick. Improved upper bounds for random-edge and random-jump on abstract cubes. In *Proc. of SODA*, pages 874–881, 2014.

[21] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[22] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[23] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[24] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proc. of SODA*, pages 117–123, 2006.

[25] M. Jurdziński and R. Savani. A simple P-matrix linear complementarity problem for discounted games. In *Proc. of CiE*, pages 283–293, 2008.

[26] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. of STOC*, pages 475–482, 1992.

[27] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117(1):151–155, 1995.

[28] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4–5):498–516, 1996.

[29] J. Matoušek and T. Szabó. Random edge can be exponential on abstract cubes. In *Proc. of FOCS*, pages 92–100, 2004.

[30] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.

[31] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

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

[33] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. New York, NY, USA, 2005.

[34] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. of CSL*, pages 369–384, 2008.

[35] I. Schurr and T. Szabó. Jumping doesn't help in abstract cubes. In *Proc. of IPCO*, pages 225–235, 2005.

[36] S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20(2):7–15, 1998.

[37] C. Stirling. Local model checking games. In *Proc. of CONCUR*, pages 1–11, 1995.

[38] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *Proc. of FOCS*, pages 547–555, 2001.

[39] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proc. of CAV*, pages 202–215, 2000.

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