A subexponential lower bound for the Least Recently Considered rule for solving linear programs and games

Oliver Friedmann
Department of Computer Science, University of Munich, Germany
`Oliver.Friedmann@gmail.com`

The *simplex* algorithm is among the most widely used algorithms for solving *linear programs* in practice. Most pivoting rules are known, however, to need an exponential number of steps to solve some linear programs. No non-polynomial lower bounds were known, prior to this work, for *Cunningham's* Least Recently Considered rule [5], which belongs to the family of history-based rules.

Also known as the ROUND-ROBIN rule, Cunningham's pivoting method fixes an initial ordering on all variables first, and then selects the improving variables in a round-robin fashion. We provide the first *subexponential* (i.e., of the form $2^{\Omega(\sqrt{n})}$) lower bound for this rule in a concrete setting.

Our lower bound is obtained by utilizing connections between pivoting steps performed by simplex-based algorithms and *improving switches* performed by *policy iteration* algorithms for 1-player and 2-player games. We start by building 2-player *parity games* (PGs) on which the policy iteration with the ROUND-ROBIN rule performs a subexponential number of iterations. We then transform the parity games into 1-player *Markov Decision Processes* (MDPs) which correspond almost immediately to concrete linear programs.

1 Introduction

The *simplex method*, developed by Dantzig in 1947 (see [6]), is among the most widely used algorithms for solving linear programs. One of the most important parameterizations of a simplex algorithm is the *pivoting rule* it employs. It specifies which non-basic variable is to enter the basis at each iteration of the algorithm. Although simplex-based algorithms perform very well in practice, essentially all *deterministic* pivoting rules are known to lead to an *exponential* number of pivoting steps on some LPs [23], [20], [1] and [17].

Kalai [21, 22] and Matoušek, Sharir and Welzl [24] devised *randomized* pivoting rules that never require more than an expected *subexponential* number of pivoting steps to solve any linear program. The most prominent randomized pivoting rules probably are RANDOM-FACET [21, 22, 24] as well as RANDOM-EDGE [4, 15, 16], for which, until recently [13], no non-trivial lower bounds given by concrete linear programs were known.

Along the same lines, we have shown a subexponential lower bound [14] for the rule suggested by Zadeh [27] (see also [8]). Also known as the LEAST-ENTERED rule, Zadeh's pivoting method belongs to the family of history-based improvement rules, which among all improving pivoting steps from the current basic feasible solution chooses one which has been entered least often.

Here, we consider now a related history-based improving rule, which is known as *Cunningham's* ROUND-ROBIN rule or as *Least Recently Considered* rule [5]. It fixes an initial ordering on all variables first, and then selects the improving variables in a round-robin fashion. We provide the first *subexponential* (i.e., of the form $2^{\Omega(\sqrt{n})}$) lower bound for the this rule.

Techniques used. The linear program on which ROUND-ROBIN performs a subexponential number of iterations is obtained using the close relation between simplex-type algorithms for solving linear programs and *policy iteration* (also known as *strategy improvement*) algorithms for solving certain 2-player and 1-player games.

This line of work was started by showing that standard strategy iteration [26] for parity games [18] may require an exponential number of iterations to solve them [10]. Fearnley [9] transferred the lower bound construction for parity games to *Markov Decision Processes* (MDPs) [19], an extremely important and well-studied family of stochastic 1-player games.

In [12], we recently constructed PGs on which the RANDOM-FACET algorithm performs an expected subexponential number of iterations. In [13], we applied Fearnley's technique to transform these PGs into MDPs, and included an additional lower bound construction for the RANDOM-EDGE algorithm.

The problem of solving an MDP, i.e., finding the optimal control *policy* and the optimal *values* of all states of the MDP, can be cast as a linear program. More precisely, the improving switches performed by the (abstract) ROUND-ROBIN algorithm applied to an MDP corresponds directly to the steps performed by the ROUND-ROBIN pivoting rule on the corresponding linear program.

Our results. We construct a family of concrete linear programs on which the number of iterations performed by ROUND-ROBIN is $2^{\Omega(\sqrt{n})}$, where n is the number of variables.

As the translation of our PGs to MDPs is a relatively simple step, we directly present the MDP version of our construction. (The original PGs from which our MDPs were derived can be found in Appendix B.) Hence, our construction can be understood without knowing anything about PGs.

The rest of this paper is organized as follows. In Section 2 we give a brief introduction to *Markov Decision Processes* (MDPs) and the primal linear programs corresponding to them. In Section 3 we review the policy iteration and the simplex algorithms, the relation between improving switches and pivoting steps, and Cunningham's ROUND-ROBIN pivoting rule. In Section 4, which is the main section of this paper, we describe our lower bound construction for ROUND-ROBIN. Many of the details are deferred, due to lack of space, to appendices. Particularly all proofs of Section 4 can be found in Appendix A. We end in Section 5 with some concluding remarks and open problems.

2 Markov Decision Processes and their linear programs

Markov decision processes (MDPs) provide a mathematical model for sequential decision making under uncertainty. They are employed to model stochastic optimization problems in various areas ranging from operations research, machine learning, artificial intelligence, economics and game theory. For an in-depth coverage of MDPs, see the books of Howard [19], Derman [7], Puterman [25] and Bertsekas [3].

Formally, an MDP is defined by its *underlying graph* $G=(V_0,V_R,E_0,E_R,r,p)$. Here, V_0 is the set of vertices (states) operated by the controller, also known as *player* 0, and V_R is a set of *randomization* vertices corresponding to the probabilistic actions of the MDP. We let $V = V_0 \cup V_R$. The edge set $E_0 \subseteq V_0 \times V_R$ corresponds to the actions available to the controller. The edge set $E_R \subseteq V_R \times V_0$ corresponds to the probabilistic transitions associated with each action. The function $r : E_0 \to \mathbb{R}$ is the immediate reward function. The function $p : E_R \to [0,1]$ specifies the transition probabilities. For every $u \in V_R$, we have $\sum_{v:(u,v)\in E_R} p(u,v) = 1$, i.e., the probabilities of all edges emanating from each vertex of V_R sum up to 1. As defined, the graph G is *bipartite*, but one can relax this condition and allow edges from V_0 to V_0 that correspond to *deterministic* actions.

A policy σ is a function $\sigma : V_0 \to V$ that selects for each vertex $u \in V_0$ a target node v corresponding to an edge $(u,v) \in E_0$, i.e. $(u,\sigma(u)) \in E_0$ wWe assume that each vertex $u \in V_0$ has at least one outgoing edge). There are several *objectives* for MDPs; we consider the *expected total reward objective* here. The *values* $\text{VAL}_\sigma(u)$ of the vertices under σ are defined as the unique solutions of the following set of linear equations:

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

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

All MDPs considered in this paper satisfy the *unichain* condition (see [25]). It states that the Markov chain obtained from each policy σ has a single irreducible recurrent class.

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

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

The variable $x(u,v)$, for $(u,v) \in E_0$, stands for the probability (frequency) of using the edge (action) (u,v). The constraints of the linear program are *conservation constraints* that state that the probability of entering a vertex u is equal to the probability of exiting u. It is not difficult to check that the *basic feasible solutions* (bfs's) of (P) correspond directly to policies of the MDP. For each policy σ we can define a feasible setting of primal variables $x(u,v)$, for $(u,v) \in E_0$, such that $x(u,v) > 0$ only if $\sigma(u) = (u,v)$. Conversely, for every bfs $x(u,v)$ we can define a corresponding policy σ. It is well known that the policy corresponding to an optimal bfs of (P) is an optimal policy of the MDP. (See, e.g., [25].)

It should be noted that *all* pivoting steps performed on these linear programs are *non-degenerate*, due to the fact the we consider the expected total reward criterion here. The lower bound construction of this paper also works when applied to the *discounted reward criterion* (for large enough discount factors), and also for the *limiting average reward criterion*. However, in the latter case, all pivoting steps performed on the induced linear programs are *degenerate*.

3 Policy iteration algorithms and simplex algorithms

Howard's [19] *policy iteration* algorithm is the most widely used algorithm for solving MDPs. It is closely related to the simplex algorithm.

The algorithm starts with some initial policy σ_0 and generates an improving sequence $\sigma_0, \sigma_1, \ldots, \sigma_N$ of policies, ending with an optimal policy σ_N. In each iteration the algorithm first *evaluates* the current policy σ_i, by computing the values $\text{VAL}_{\sigma_i}(u)$ of all vertices. An edge $(u,v') \in E_0$, such that $\sigma_i(u) \neq v'$ is then said to be an *improving switch* if and only if either $\text{VAL}_{\sigma_i}(v') > \text{VAL}_{\sigma_i}(u)$. Given a policy σ, we denote the *set of improving switches* by I_σ.

A crucial property of policy iteration is that σ is an optimal policy if and only if there are no improving switches with respect to it (see, e.g., [19], [25]). Furthermore, if $(u,v') \in I_\sigma$ is an improving switch w.r.t. σ, and σ' is defined as $\sigma[(u,v')]$ (i.e., $\sigma'(u) = v'$ and $\sigma'(w) = \sigma(w)$ for all $w \neq u$), then σ' is *strictly* better than σ, in the sense that for every $u \in V_0$, we have $\text{VAL}_{\sigma'}(u) \geq \text{VAL}_\sigma(u)$, with a strict inequality for at least one vertex $u \in V_0$.

Policy iteration algorithms that perform a single switch at each iteration – like ROUND-ROBIN– are, in fact, simplex algorithms. Each policy σ of an MDP immediately gives rise to a feasible solution $x(u,v)$ of the primal linear program (P); use σ to define a Markov chain and let $x(u,v)$ be the 'steady-state' probability that the edge (action) (u,v) is used. In particular, if $\sigma(u) \neq v$, then $x(u,v) = 0$.

Cunningham's ROUND-ROBIN pivoting rule is a *deterministic*, *memorizing* improvement rule which fixes an initial ordering on all variables first, and then selects the improving variables in a round-robin fashion. When applied to the primal linear program of an MDP, it is equivalent to the variant of the policy iteration algorithm, in which an initial ordering on the edges is fixed first, and the improving switch is chosen among all improving switches in a round-robin fashion. This is the foundation of our lower bound for the ROUND-ROBIN rule.

We describe Cunningham's pivoting rule now formally in the context of MDPs. We assume that we are given a total ordering $\prec$ on the player 0 edges of the MDP. As a memorization structure, we remember the last edge that has been applied.

Given a non-empty subset of player 0 edges $\emptyset \neq F \subseteq E_0$ and a player 0 edge $e \in E_0$, we define a *successor operator* as follows:

$$\operatorname{succ}_{\prec}(e,F) := \begin{cases} \min_{\prec}\{e' \in F \mid e \preceq e'\} & \text{if } \{e' \in F \mid e \preceq e'\} \neq \emptyset \\ \min_{\prec}\{e' \in F \mid e' \preceq e\} & \text{otherwise} \end{cases}$$

See Algorithm 1 for a pseudo-code specification of the ROUND-ROBIN pivoting rule for solving MDPs.

Algorithm 1 Cunningham's Improvement Algorithm

```
1: procedure ROUND-ROBIN($G$,$\sigma$,$\prec$,$e$)
2:     while $I_\sigma \neq \emptyset$ do
3:         $e \leftarrow \operatorname{succ}_{\prec}(e, I_\sigma)$
4:         $\sigma \leftarrow \sigma[e]$
5:     end while
6: end procedure
```

Let (σ_1, e_1), …, (σ_n, e_n) be a trace of the algorithm w.r.t. some selection ordering $\prec$. We write $(\sigma, e) \rightsquigarrow_{\prec} (\sigma', e')$ iff there are $i < j$ s.t. $(\sigma, e) = (\sigma_i, e_i)$ and $(\sigma', e') = (\sigma_j, e_j)$.

In the original specification of Cunningham's algorithm [5], there are no clear objectives how to select the ordering on the edges or how to select the initial edge e. In fact, we know that the asymptotic behavior of Cunningham's improvement rule highly depends on the method that is used to find the ordering, at least in the world of MDPs, PGs and policy iteration for games in general. We have the following theorem which is easy to verify (the idea is that there is at least one improving switch towards the optimal policy in each step).

Theorem 1. *Let G be an MDP with n nodes and σ_0 be a policy. There is a sequence policies $\sigma_0, \sigma_1, \ldots, \sigma_N$ and a sequence of* different *switches $e_1, e_2, \ldots, e_N$ with $N \leq n$ s.t. σ_{N-1} is optimal, $\sigma_{i+1} = \sigma_i[e_{i+1}]$ and e_{i+1} is an σ_i-improving switch.*

Since all switches are different in the sequence, it follows immediately that there is always a way to select an ordering that results in a linear number of pivoting steps to solve an MDP with Cunningham's improvement rule. However, there is no obvious method on how to efficiently find such an ordering. The question whether Cunningham's pivoting rule solves MDPs (and LPs) in polynomial time should therefore be phrased *independently* of the heuristic of finding the ordering. In other words, we as "lower bound designers" are the ones that choose a particular ordering selection rule.

4 Lower bound for Round-Robin

We start with a high-level description of the MDPs on which Round-Robin performs a subexponential number of iterations[1]. The construction may be seen as an implementation of a full binary counter. A schematic description of the lower bound MDPs is given in Figure 1. Circles correspond to vertices of V_0, i.e., vertices controlled by player 0, while rectangles correspond to the randomization vertices of V_R.

The MDP of Figure 1 emulates an 3-bit counter. It is composed of 3 essentially identical levels, each corresponding to a single bit of the counter. The MDP includes one *source* u and one *sink* t.

All edges have an immediate reward of 0 associated with them (such 0 rewards are not shown explicitly in the figure) unless stated otherwise as follows: Some of the vertices are assigned integer *priorities*. If a vertex v has priority $\Omega(v)$ assigned to it, then a reward of $\langle v \rangle = (-N)^{\Omega(v)}$ is *added* to all edges emanating from v, where N is a sufficiently large integer. We use $N \geq 0.5n^2 + 5.5n + 4$ (the number of nodes in the game) and $\varepsilon \leq N^{-(2n+9)}$ (the highest priority in the game will be $2n+8$). Priorities, if present, are listed next to the vertex name. Note that it is profitable for the controller, to move through vertices of even priority and to avoid vertices of odd priority, and that vertices of higher numerical priority dominate vertices of lower priority (the idea of using priorities is inspired, of course, by the reduction from parity games to mean payoff games).

Each level i contains an instances of a *cycle gadget* that consists of a randomization vertex B_i and an increasing number of player 0 controlled nodes $b_{i,j}$. To simplify notation, we identify B_i with $b_{i,0}$.

From B_i, the edge $\mathrm{B}_i \to b_{i,i}$, is chosen with probability $1-\varepsilon$, while the edge $\mathrm{B}_i \to y_i$ (called "escape edge") is chosen with probability ε. Thus, if $\sigma(b_{i,j}) = b_{i,j-1}$ for all $j \leq i$, the MDP is guaranteed to eventually move from B_i to y_i (this is similar to the use of randomization by Fearnley [9]). We say that a cycle gadget is

- *closed* iff $\sigma(b_{i,j}) = b_{i,j-1}$ for all $j \leq i$,
- *open* iff $\sigma(b_{i,j}) \neq b_{i,j-1}$ for some $j \leq i$, and
- *completely open* iff $\sigma(b_{i,j}) \neq b_{i,j-1}$ for all $j \leq i$.

The i-th level of the MDP corresponds to the i-th bit. A set bit is represented by a *closed* cycle gadget.

Our proof is conceptually divided into two parts. First we investigate the improving switches that can be performed from certain policies of the MDP. This allows us to prove the *existence* of a sequence of improving switches that indeed generates the sequence of policies $\sigma_{0\ldots00}, \sigma_{0\ldots01}, \sigma_{0\ldots10}, \ldots, \sigma_{1\ldots11}$. A transition from $\sigma_{\mathfrak{b}}$ to $\sigma_{\mathfrak{b}+1}$ involves many intermediate improvement steps. We partition the path leading from $\sigma_{\mathfrak{b}}$ to $\sigma_{\mathfrak{b}+1}$ into four sub-paths which we refer to as *phases*. In the following, we first give an informal description of the phases. The second part of our proof will be to show that the way we want to apply the improving switches is compliant with some selection ordering.

Before starting to describe what happens in the different phases, we describe the "ideal" configuration of a policy, which belongs to phase 1: (1) all cycles corresponding to set bits are closed, (2) all other cycles are completely open, moving either to s or u_1, depending on the setting of the first bit, (3) all nodes w_i and u_i move to d_i iff the corresponding bit i is set and to w_{i+1} resp. u_{i+1} otherwise.

Now, we are ready to informally describe all phases:

1. At the beginning of the first phase, we only have open cycle gadgets that are competing with each other to close. It is improving to close every open cycle, however, there is only one improving edge per open cycle moving into the cycle in each iteration.

[1] See http://files.oliverfriedmann.de/add/round_robin_animation.pdf for an animated run of the policy iteration algorithm on the presented MDPs.

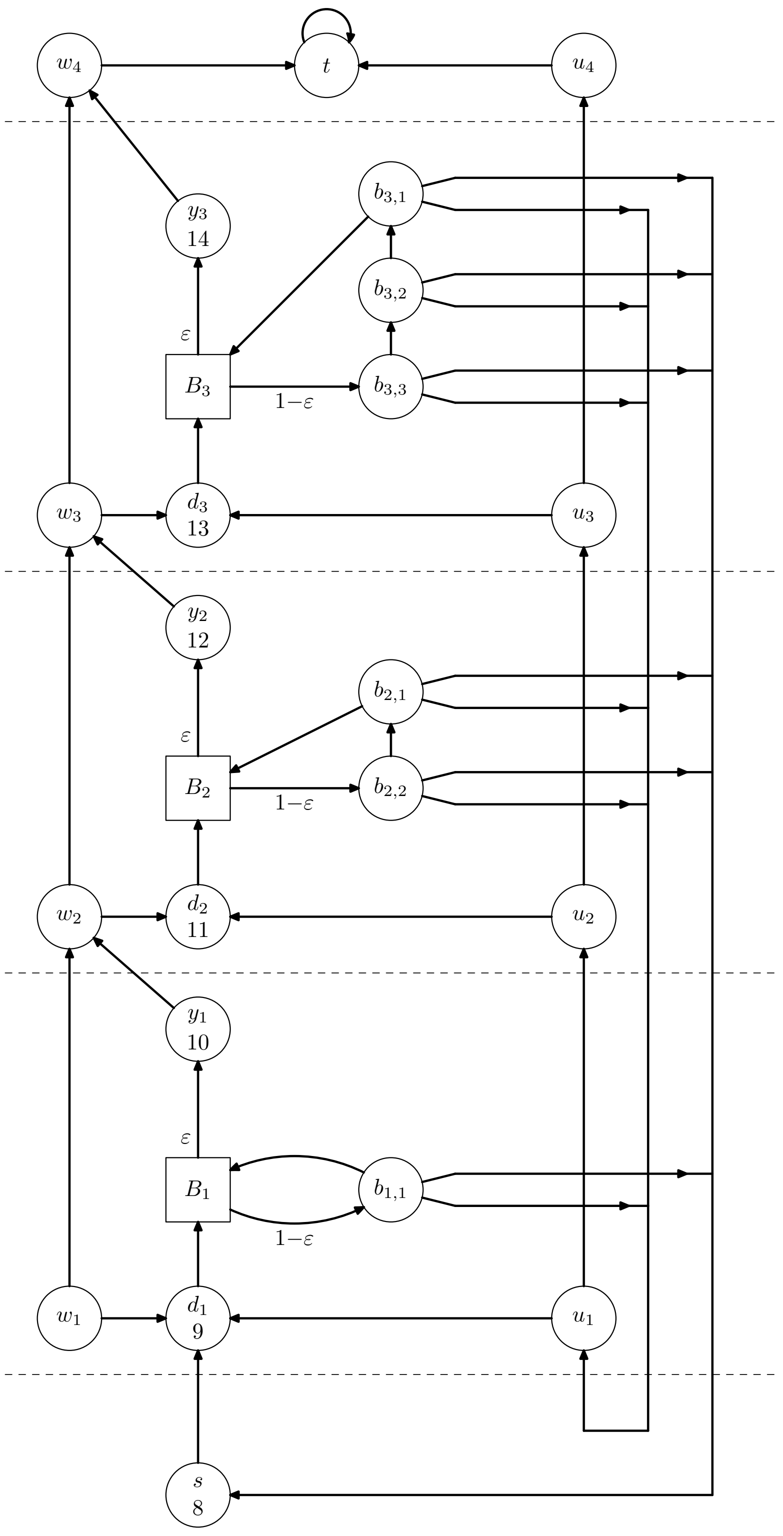

Figure 1: ROUND-ROBIN MDP Construction

By selecting improving edges in a round-robin fashion, it must be the case that eventually the least unset cycle closes, as it is composed of less nodes than cycles corresponding to higher bits.

The last switch that is performed in this phase is to move the remaining edge of the cycle associated with the least unset bit inward, and therefore close the gadget.

2. In this phase, we need to make the recently set bit i accessible by the rest of the MDP, which will be via the u_*-nodes. First, there will be an improving edge from u_i to d_i. After this edge has been applied, there will be an improving edge from u_{i-1} to u_i, then from u_{i-2} to u_{i-1} and so on.

   This phase ends when the last such edge starting from u_1 has been assigned properly.

   Note that it still improving in this phase to close the remaining open cycles.

3. In the third phase, we perform the major part of the *resetting* process. By resetting, we mean to unset lower bits again, which corresponds to reopening the respective cycles.

   Also, we want to reset all other "half-closed" (meaning not completely open) cycles of higher bits. It is improving for all nodes belonging to such cycles to directly move to either u_1 or s, depending on the recently set bit. If it was bit 1, then s is better, otherwise u_1.

   This phase ends when all cycle edges that should be reset have been reset.

4. In the fourth and last phase, we update the upper selection nodes w_j for all $j \leq i$. First, it will be improving to move from w_i to d_i, and then to move from w_{i-1} to w_i, and from w_{i-2} to w_{i-1} and so on.

   The phase ends when all upper selection nodes have been updated accordingly. The binary counter represented by policies has been increments by one bit.

4.1 Full Construction

In this subsection, we formally describe the full construction of our MDPs. We define an underlying graph $G_n = (V_0, V_R, E_0, E_R, r, p)$ of an MDP as shown schematically in Figure 1 as follows:

$$\begin{aligned} V_0 &:= \{b_{i,j} \mid 1 \leq j \leq i \leq n\} \cup \{y_i, d_i \mid i \in [n]\} \cup \{u_i, w_i \mid i \in [n+1]\} \cup \{t, s\} \\ V_R &:= \{\mathrm{B}_i \mid i \in [n]\} \end{aligned}$$

With G_n, we associate a large number $N \in \mathbb{N}$ and a small number $0 < \varepsilon$. We require N to be at least as large as the number of nodes with priorities, i.e. $N \geq 0.5n^2 + 5.5n + 4$ and ε^{-1} to be significantly larger than the largest occurring priority induced reward, i.e. $\varepsilon \leq N^{-(2n+9)}$. Remember that node v having priority $\Omega(v)$ means that the cost associated with every outgoing edge of v is $\langle v \rangle = (-N)^{\Omega(v)}$.

Table 1 defines the edge sets, the probabilities and the priorities (if present) of G_n. For convenience, we identify $b_{i,0} = \mathrm{B}_i$.

As designated *initial policy* σ^*, we use $\sigma^*(u_i) = u_{i+1}$, $\sigma^*(w_i) = w_{i+1}$, and $\sigma^*(b_{i,j}) = u_1$.

Lemma 2. *The Markov chains obtained by any policy reach the sink t almost surely (i.e. the sink t is the single irreducible recurrent class).*

It is not too hard to see that the absolute value of all nodes corresponding to policies are bounded by ε^{-1}. More formally we have:

Lemma 3. *Let $P = \{s, y_*, d_*\}$ be the set of nodes with priorities. For a subset $S \subseteq P$, let $\sum(S) = \sum_{v \in S} \langle v \rangle$. For non-empty subsets $S \subseteq P$, let $v_S \in S$ be the node with the largest priority in S.*

1. $|\sum(S)| < N^{(2n+9)}$ *and* $\varepsilon \cdot |\sum(S)| < 1$ *for every subset* $S \subseteq P$, *and*
2. $|v_S| < |v_{S'}|$ *implies* $|\sum(S)| < |\sum(S')|$ *for non-empty subsets* $S, S' \subseteq P$.

Node	Successors	Probability
B_i	$b_{i,i}$	ε
B_i	y_i	$1-\varepsilon$

Node	Successors	Priority
s	d_1	8
y_i	w_{i+1}	$2i+8$
d_i	B_i	$2i+7$

Node	Successors
u_{n+1}	t
$u_{i\le n}$	d_i, u_{i+1}
w_{n+1}	t
$w_{i\le n}$	d_i, w_{i+1}
$b_{i,j}$	$b_{i,j-1}, s, u_1$
t	t

Table 1: ROUND-ROBIN MDP Construction (with $b_{i,0}=\mathrm{B}_i$)

4.2 Phases and Improving Switches

In this subsection, we formally describe the different *phases* that a policy can be in, as well as the improving switches in each phase. The increment of the binary counter by one is realized by transitioning through all the phases.

First, we introduce notation to succinctly describe binary counters. It will be convenient for us to consider counter configurations with an *infinite* tape, where unused bits are zero. The set of n-bit configurations is formally defined as $\mathscr{B}_n = \{\mathfrak{b} \in \{0,1\}^\infty \mid \forall i > n : \mathfrak{b}_i = 0\}$.

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

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

Given a configuration $\mathfrak{b}$, we access the *least unset bit* by $\mu(\mathfrak{b}) = \min\{j \mid \mathfrak{b}_j = 0\}$. Let $\mathfrak{b}^\mu$ denote $\mathfrak{b}[\mu(\mathfrak{b})\mapsto 1]$.

We use two additional notations to denote certain sets of bit configurations. Let $\mathfrak{b}$ and $\mathfrak{b}'$ be two bit configurations. Then define:

$$[\mathfrak{b};\mathfrak{b}'] := \{\mathfrak{b}'' \mid \mathfrak{b}\le\mathfrak{b}''\le\mathfrak{b}'\} \qquad \langle\mathfrak{b};\mathfrak{b}'\rangle := \{(\mathfrak{b}_n,\ldots,\mathfrak{b}_{j+1},\mathfrak{b}'_j,\ldots,\mathfrak{b}'_1) \mid 1\le j\le n\}$$

In other words, $[\mathfrak{b};\mathfrak{b}']$ contains all bit configuration that lie between $\mathfrak{b}$ and $\mathfrak{b}'$, while $\langle\mathfrak{b};\mathfrak{b}'\rangle$ contains all bit configurations that can be obtained by merging $\mathfrak{b}$ and $\mathfrak{b}'$, with parts of $\mathfrak{b}$ building the upper and parts of $\mathfrak{b}'$ building the lower part of the merged bit configuration.
Given a bit configuration $\mathfrak{b}$, we denote the associated *source* as follows:

$$\mathrm{src}(\mathfrak{b}) = \begin{cases} s & \text{if } \mathfrak{b}_1 = 1 \\ d_1 & \text{if } \mathfrak{b}_1 = 0 \end{cases}$$

For any source $z\in\{s,d_1\}$, let $\bar{z}$ denote the *other* source, i.e. $\bar{s}=d_1$ and $\bar{d_1}=s$.

We first introduce notation to succinctly describe policies. Let σ be a policy. Then define bit configurations $U(\sigma),W(\sigma),B(\sigma)\in\mathscr{B}_n$ as follows:

$$U(\sigma)_i = 1 \iff \sigma(u_i)=d_i; \qquad W(\sigma)_i = 1 \iff \sigma(w_i)=d_i; \qquad B(\sigma)_i = 1 \iff \text{cycle } i \text{ closed}$$

We say that a strategy σ is *forward-consecutive w.r.t.* $z \in \{s, u_1\}$ — and write Fwd-Con$(\sigma) = z$ to indicate that σ satisfies this condition — iff

$$\forall 1 \leq i \leq n \; \exists 0 \leq j \leq i \; : (\sigma(b_{i,1}), \ldots, \sigma(b_{i,i})) = (b_{i,0}, \ldots, b_{i,j-1}, \underbrace{z, \ldots, z}_{i-j})$$

We say that a strategy σ is *backward-consecutive w.r.t.* $z \in \{s, u_1\}$ — and write Bwd-Con$(\sigma) = z$ to indicate that σ satisfies this condition — iff

$$\forall 1 \leq i \leq n \; \exists 0 \leq k \leq i \; \exists k \leq j \leq i \; : (\sigma(b_{i,1}), \ldots, \sigma(b_{i,i})) = (\underbrace{\bar{z}, \ldots \bar{z}}_{k}, b_{i,k}, \ldots, b_{i,j-1}, \underbrace{z, \ldots, z}_{i-j})$$

We are now ready to formulate the conditions for policies that fulfill one of the four phases along with the improving edges. See Table 2 for a complete description (with respect to a bit configuration $\mathfrak{b}$). We say that a strategy σ is a *phase p strategy with configuration* $\mathfrak{b}$ iff every node is mapped by σ to a choice included in the respective cell of the table.

Phase	$B(\sigma)$	$U(\sigma)$	$W(\sigma)$	Fwd-Con(σ)	Bwd-Con(σ)
1	$\mathfrak{b}$	$\mathfrak{b}$	$\mathfrak{b}$	$\mathrm{src}(\mathfrak{b})$	-
2	$\mathfrak{b}^{\mu}$	$\langle \mathfrak{b}^{\oplus}; \mathfrak{b} \rangle$	$\mathfrak{b}$	$\mathrm{src}(\mathfrak{b})$	-
3	$[\mathfrak{b}^{\oplus}; \mathfrak{b}^{\mu}]$	$\mathfrak{b}^{\oplus}$	$\mathfrak{b}$	-	$\mathrm{src}(\mathfrak{b})$
4	$\mathfrak{b}^{\oplus}$	$\mathfrak{b}^{\oplus}$	$\langle \mathfrak{b}^{\oplus}; \mathfrak{b} \rangle$	$\mathrm{src}(\mathfrak{b}^{\oplus})$	-

Table 2: Policy Phases

The following lemma tells us that all occurring values in the policy iteration are *small* compared to $N^{(2n+9)}$. Particularly, ε-times values are almost negligible. It follows immediately from Table 2 and Lemma 3 that:

Lemma 4. *Let σ be a policy belonging to one of the phases specified in Table 2. Then $|\mathrm{VAL}_{\sigma}(v)| < N^{(2n+9)}$ and $\varepsilon \cdot |\mathrm{VAL}_{\sigma}(v)| < 1$ for every node v.*

Next, we specify the improving switches in each phase. In order to unify the notation, we define some sets of edges first:

$$\begin{aligned}
\mathrm{BCloseI}(\sigma) &= (\{(b_{i,j}, b_{i,j-1}) \mid j = 1 \vee \sigma(b_{i,j-1}) = b_{i,j-2}\}) \setminus \sigma \\
\mathrm{BOpenI}_{\mathfrak{b}}(\sigma) &= (\{(b_{i,j}, \mathrm{src}(\mathfrak{b})) \mid \mathfrak{b}_i = 0\} \cup \\
&\quad \{(b_{i,j}, b_{i,j-1}) \mid \mathfrak{b}_i = 0 \wedge \sigma(b_{i,j}) \neq \mathrm{src}(\mathfrak{b}) \wedge (j = 1 \vee \sigma(b_{i,j-1}) \in \{b_{i,j-2}, \mathrm{src}(\mathfrak{b})\})\}) \setminus \sigma \\
\mathrm{UI}_i(\sigma) &= (\{(u_i, d_i)\} \cup \{(u_j, u_{j+1}) \mid j < i \wedge U(\sigma)_i = 1 \wedge (j = i-1 \vee U(\sigma)_{j+1} = 0)\}) \setminus \sigma \\
\mathrm{WI}_i(\sigma) &= (\{(w_i, d_i)\} \cup \{(w_j, w_{j+1}) \mid j < i \wedge W(\sigma)_i = 1 \wedge (j = i-1 \vee W(\sigma)_{j+1} = 0)\}) \setminus \sigma
\end{aligned}$$

Table 3 specifies the sets of improving switches for each phase p by unioning all columns labelled "Yes". We finally arrive at the following main lemma describing the improving switches.

Lemma 5. *The improving switches from policies that belong to the phases in Table 2 are exactly those specified in Table 3.*

Phase	$\text{BCloseI}(\sigma)$	$\text{BOpenI}_{\mathfrak{b}^\oplus}(\sigma)$	$\text{UI}_{\mu(\mathfrak{b})}(\sigma)$	$\text{WI}_{\mu(\mathfrak{b})}(\sigma)$
1	Yes	No	No	No
2	Yes	No	Yes	No
3	No	Yes	No	Yes
4	Yes	No	No	Yes

Table 3: Improving Switches

4.3 Lower bound Proof

Finally, we outlint the specific ordering selection on the edges of the game s.t. all four phases are fully traversed in the right ordering. From a macroscopic point of view, we have the following ordering:

$$\underbrace{(b_{*,*}, b_{*,*})}_{\text{Phase 1}} \prec \underbrace{(u_*, *)}_{\text{Phase 2}} \prec \underbrace{(b_{*,*}, [s|d_1])}_{\text{Phase 3}} \prec \underbrace{(w_*, *)}_{\text{Phase 4}}$$

The detailed ordering for every phase is as follows:

$$\begin{aligned}
\text{Phase 1}: &\quad (b_{i,*}, b_{*,*}) \prec (b_{i+1,*}, b_{*,*}) \quad \text{and} \quad (b_{i,j}, b_{*,*}) \prec (b_{i,j-1}, b_{*,*}) \\
\text{Phase 2}: &\quad (u_i, d_i) \prec (u_i, u_{i+1}) \prec (u_{i-1}, d_{i-1}) \\
\text{Phase 3}: &\quad (b_{i,*}, [s|d_1]) \prec (b_{i-1,*}, [s|d_1]) \quad \text{and} \quad (b_{i,j}, d_1) \prec (b_{i,j}, s) \prec (b_{i,j+1}, d_1) \\
\text{Phase 4}: &\quad (w_i, d_i) \prec (w_i, w_{i+1}) \prec (w_{i-1}, d_{i-1})
\end{aligned}$$

We are now ready to formulate our main lemma describing the transitioning from an initial phase 1 policy corresponding to $\mathfrak{b}$ to a successor initial phase 1 policy corresponding to $\mathfrak{b}'$, complying with the given ordering selection.

Lemma 6. *Let σ be a phase 1 policy with configuration $\mathfrak{b} < \mathbf{1}_n$. Let e be an edge with $e \preceq (b_{1,1}, \mathrm{B}_1)$ or $(u_n, d_n) \preceq e$. Then, there is a phase 1 policy σ' with configuration $\mathfrak{b}^\oplus$ and an edge e' with $e' \preceq (b_{1,1}, \mathrm{B}_1)$ or $(u_n, d_n) \preceq e'$ s.t. $(\sigma, e) \rightsquigarrow_\prec (\sigma', e')$.*

It follows immediately that the MDPs provided here indeed simulate a binary counter by starting with the designated initial policy and the $\prec$-minimal edge.

Theorem 7. *The number of improving steps performed by* Round-Robin *on the MDPs constructed in this section, which contain $\mathcal{O}(n^2)$ vertices and edges, is $\Omega(2^n)$.*

The primal linear programs corresponding to the MDPs constructed in this section are thus linear programs on which the simplex algorithm with Cunningham's pivoting rule performs a subexponential number of iterations.

5 Concluding remarks and open problems

We have shown that Cunningham's Round-Robin rule [5] may lead to a subexponential number of iterations by constructing explicit linear programs with n variables on which the expected number of iterations performed by Round-Robin is $2^{\Omega(\sqrt{n})}$.

The lower bound for linear programming has been obtained by constructing explicit parity games and subsequently MDPs on which we have the same expected number of iterations when solved by policy iteration. The lower bound result immediately transfers to mean payoff games, discounted payoff games and turned-based simple stochastic games [11].

Although now Zadeh's well-known LEAST-ENTERED rule [27] and Cunningham's ROUND-ROBIN rule are known to have subexponential lower bounds in concrete settings [14], there are still some history-based pivoting rules left for which no such bounds are known. First, there is the *least-recently basic rule* [5] which selects the improving variable that left the basis least-recently. Second, there is the *least-recently entered rule* [8] which selects the improving variable that entered the basis least-recently thus far. Third, there is the *least iterations in the basis rule* [2] which selects the improving variable that has been in the basis for the least number of iterations.

The most interesting open problems are, perhaps, whether linear programs can be solved in strongly polynomial time, whether the weak Hirsch conjecture holds, and whether there is a polynomial time algorithm for solving parity games or related game classes.

References

[1] D. Avis and V. Chvátal. Notes on Bland's pivoting rule. In *Polyhedral Combinatorics*, volume 8 of *Mathematical Programming Studies*, pages 24–34. Springer, 1978.

[2] D. Avis, S. Moriyama, and Y. Matsumoto. History based pivot rules and unique sink orientations. In *Japan-Canada Workshop*, 2009.

[3] D.P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, second edition, 2001.

[4] A.Z. Broder, M.E. Dyer, A.M. Frieze, P. Raghavan, and E. Upfal. The worst-case running time of the random simplex algorithm is exponential in the height. *Information Processing Letters*, 56(2):79–81, 1995.

[5] W. H. Cunningham. Theoretical properties of the network simplex method. In *Mathematics of Operations Research*, pages 196–208, 1979.

[6] G.B. Dantzig. *Linear programming and extensions*. Princeton University Press, 1963.

[7] C. Derman. *Finite state Markov decision processes*. Academic Press, 1972.

[8] Y. Fathi and C. Tovey. Affirmative action algorithms. *Mathematical Programming*, 34:292–301, 1986.

[9] J. Fearnley. Exponential lower bounds for policy iteration. In *Proc. of 37th ICALP*, pages 551–562, 2010.

[10] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proc. of 24th LICS*, pages 145–156, 2009.

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

[12] O. Friedmann, T.D. Hansen, and U. Zwick. A subexponential lower bound for the random facet algorithm for parity games. In *Proc. of 22nd SODA*, 2011.

[13] O. Friedmann, T.D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 283–292, New York, NY, USA, 2011. ACM.

[14] Oliver Friedmann. A subexponential lower bound for zadeh's pivoting rule for solving linear programs and games. In *IPCO*, pages 192–206, 2011.

[15] B. Gärtner, M. Henk, and G. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, 18(3):349–372, 1998.

[16] B. Gärtner, F. Tschirschnitz, E. Welzl, J. Solymosi, and P. Valtr. One line and n points. *Random Structures & Algorithms*, 23(4):453–471, 2003.

[17] D. Goldfarb and W.Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277 – 285, 1979.

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

[19] R.A. Howard. *Dynamic programming and Markov processes*. MIT Press, 1960.

[20] R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973.

[21] G. Kalai. A subexponential randomized simplex algorithm (extended abstract). In *Proc. of 24th STOC*, pages 475–482, 1992.

[22] G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997.

[23] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, New York, 1972.

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

[25] M.L. Puterman. *Markov decision processes*. Wiley, 1994.

[26] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In *International Conference on Computer-Aided Verification, CAV 2000*, volume 1855 of *LNCS*, pages 202–215, 2000.

[27] N. Zadeh. What is the worst case behavior of the simplex algorithm? Technical Report 27, Department of Operations Research, Stanford, 1980.

A Proofs

Theorem 1.

Proof. Let G be an MDP and σ^* be the optimal policy. We define a *distance* function d that maps every policy to a natural number, counting in how many edges it differs from $\sigma*$. Formally:

$$d(\sigma) = |\{v \mid \sigma(v) \neq \sigma^*(v)\}|$$

To show the claim of the theorem, it suffices to prove that we can improve any non-optimal strategy σ by an improving edge e in one step s.t. $d(\sigma) > d(\sigma[e])$. Let therefore σ be a strategy with $d(\sigma) > 0$. Consider now the MDP G' restricted to σ and σ^*, i.e. $E_0' = \{(v,w) \in E_0 \mid \sigma(v) = w \text{ or } \sigma^*(v) = w\}$.

It is easy to see that G' is a well-defined MDP, both σ and σ^* are policies in G' and that σ^* is still the optimal policy in G'. Since σ is not optimal in G', there must be an improving edge $e \in E_0' \setminus \sigma$. Hence, $d(\sigma) > d(\sigma[e])$. $\square$

We will specify a simple auxiliary lemma that describes the exact behavior of the cycles appearing in the construction.

The idea behind the cycles is to have a gate that controls the access of other nodes of the graph to the *escape node* of the cycle (y_i) to which the randomized node moves with very low probability.

Lemma 8. *Let σ be a policy belonging to one of the phases specified in Table 2. Let $1 \leq i \leq n$. Let $j = \max\{k \leq i \mid \sigma(b_{i,k}) \neq b_{i,k-1}\}$. The following holds:*

1. *cycle i closed $\Rightarrow \text{VAL}_\sigma(\mathrm{B}_i) = \text{VAL}_\sigma(y_i)$;*
2. *cycle i open $\Rightarrow \text{VAL}_\sigma(\mathrm{B}_i) = (1-\varepsilon)\cdot\text{VAL}_\sigma(\sigma(b_{i,j})) + \varepsilon\cdot\text{VAL}_\sigma(y_i)$.*

Finally, we prove that the improving switches are indeed exactly as specified. The simple but tedious proof uses Lemma 4 and Lemma 8 to compute the values of all important nodes in the game to determine whether a successor of V_0-controlled node is improving or not.

Lemma 5.

Proof. Let σ be a policy belonging to one of the phases with configuration $\mathfrak{b}$. We assume that σ is a phase 1 policy. The improving switches for the other phases can be shown the same way.

For $1 \leq i$ and $l \leq n$, let $S_i^l = \sum_{l \geq j \geq i, \mathfrak{b}_j = 1} (\langle d_j \rangle + \langle y_j \rangle)$ and $S_i = S_i^n$. Let $z = \mathrm{src}(\sigma)$. First, we apply Lemma 3 and Lemma 8, and compute the values of all nodes.

Node	t	w_i, u_i	y_i	z	
				$\mathfrak{b}_1$=1	$\mathfrak{b}_1$=0
Value	0	S_i	$\langle y_i \rangle + S_{i+1}$	$\langle s \rangle + S_1$	S_1

Node	s		B_i	
	$\mathfrak{b}_1$=0	$\mathfrak{b}_1$=1	$\mathfrak{b}_i$=0	$\mathfrak{b}_i$=1
Value	$\langle s \rangle + \langle d_1 \rangle + \varepsilon\langle y_1 \rangle + S_1$	$\langle s \rangle + S_1$	$\varepsilon(\langle y_i \rangle + S_{i+1}) + (1-\varepsilon)\text{VAL}_\sigma(z)$	$\langle y_i \rangle + S_{i+1}$

Node	d_i		$b_{i,j}$		
	$\mathfrak{b}_i$=0	$\mathfrak{b}_i$=1	$\mathfrak{b}_i=0, \sigma(b_{i,j}) = z$	$\mathfrak{b}_i=0, \sigma(b_{i,j}) = b_{i,j-1}$	$\mathfrak{b}_i$=1
Value	$\langle d_i \rangle + \varepsilon(\langle y_i \rangle + S_{i+1}) + (1-\varepsilon)\text{VAL}_\sigma(z)$	S_i	$\text{VAL}_\sigma(z)$	$\text{VAL}_\sigma(\mathrm{B}_i)$	$\langle y_i \rangle + S_{i+1}$

By computing the differences of the values, the claim follows immediately. $\square$

Lemma 6.

Proof. The proof of the lemma is ultimately based on the four phases described in Table 2, the corresponding improving switches given in Table 3 (proven correct in Lemma 5) and the introduced selection ordering.

We prove the lemma by outlining the complete sequence of switches that are applied to σ in order to obtain σ' (we do not explicitly describe the intermediate strategies which can be derived by applying all mentioned switches upto that point).

Let $i_1, \dots, i_k$ be the complete sequence of ascending indices s.t. $\mathfrak{b}_{i_j} = 0$ for $1 \le j \le k$. Let $z = \mathrm{src}(\mathfrak{b}^{\oplus})$. The following holds:

$$
\begin{aligned}
&\overset{\text{P1}}{\rightsquigarrow} (b_{i_1,1}, b_{i_1,0}) \rightsquigarrow (b_{i_2,1}, b_{i_2,0}) \rightsquigarrow \dots \rightsquigarrow (b_{i_k,1}, b_{i_k,0}) \\
&\rightsquigarrow (b_{i_1,2}, b_{i_1,1}) \rightsquigarrow (b_{i_2,2}, b_{i_2,1}) \rightsquigarrow \dots \rightsquigarrow (b_{i_k,2}, b_{i_k,1}) \\
&\qquad\qquad \vdots \\
&\rightsquigarrow (b_{i_1,i_1}, b_{i_1,i_1-1}) \overset{\text{P2}}{\rightsquigarrow} (b_{i_2,i_1}, b_{i_2,i_1-1}) \rightsquigarrow \dots \rightsquigarrow (b_{i_k,i_1}, b_{i_k,i_1-1}) \\
&\rightsquigarrow (u_{i_1}, d_{i_1}) \rightsquigarrow (u_{i_1-1}, u_{i_1}) \rightsquigarrow \dots \rightsquigarrow (u_1, u_2) \\
&\overset{\text{P3}}{\rightsquigarrow} (b_{i_k,1}, z) \rightsquigarrow (b_{i_k,2}, z) \rightsquigarrow \dots \rightsquigarrow (b_{i_k,i_k}, z) \\
&\rightsquigarrow (b_{i_{k-1},1}, z) \rightsquigarrow (b_{i_{k-1},2}, z) \rightsquigarrow \dots \rightsquigarrow (b_{i_{k-1},i_{k-1}}, z) \\
&\qquad\qquad \vdots \\
&\rightsquigarrow (b_{i_2,1}, z) \rightsquigarrow (b_{i_2,2}, z) \rightsquigarrow \dots \rightsquigarrow (b_{i_2,i_2}, z) \\
&\rightsquigarrow (b_{i_1-1,1}, z) \rightsquigarrow (b_{i_1-1,2}, z) \rightsquigarrow \dots \rightsquigarrow (b_{i_1-1,i_1-1}, z) \\
&\rightsquigarrow (b_{i_1-2,1}, z) \rightsquigarrow (b_{i_1-2,2}, z) \rightsquigarrow \dots \rightsquigarrow (b_{i_1-2,i_1-2}, z) \\
&\qquad\qquad \vdots \\
&\rightsquigarrow (b_{1,1}, z) \\
&\overset{\text{P4}}{\rightsquigarrow} (w_{i_1}, d_{i_1}) \rightsquigarrow (w_{i_1-1}, w_{i_1}) \rightsquigarrow \dots \rightsquigarrow (w_1, w_2) \overset{\text{P1}}{\rightsquigarrow}
\end{aligned}
$$

□

B Parity Games

In this section, we show how the lower bound graphs can be turned into a parity game to provide a lower bound for this class of games as well.

We just give a formal specification of parity games to fix the notation. For a proper description of parity games, related two-player game classes and policy iteration on these games, please refer to [12] and [11].

A *parity game* is a tuple $G = (V_0, V_1, E, \Omega)$, where V_0 is the set of vertices controlled by player 0, V_1 is the set of vertices controlled by player 1, $E \subseteq V \times V$, where $V = V_0 \cup V_1$, is the set of edges, and $\Omega : V \to \mathbb{N}$ is a function that assigns a *priority* to each vertex. We assume that each vertex has at least one outgoing edge.

We say that G is a *sink parity game* iff there is a node $v \in V$ such that $\Omega(v) = 1$, $(v,v) \in E$, $\Omega(w) > 1$ for every other node $w \in V$, v is the only cycle in G that is won by player 1, and player 1 has a winning policy for the whole game.

Theorem 9 ([11])**.** *Let G be a sink parity game. Discrete policy iteration requires the same number of iterations to solve G as the policy iteration for the induced payoff games as well as turn-based stochastic games to solve the respective game G' induced by applying the standard reduction from G to the respective game class, assuming that the improvement policy solely depends on the ordering of the improving edges.*

Essentially, the parity game graph corresponding to our MDPs is the same. Randomization nodes are replaced by player 1 controlled nodes s.t. the cycles are won by player 0. We follow Fearnley's construction here [9]. We assign low unimportant priorities to all nodes that have currently no priority.

We define the underlying graph $G_n = (V_0, V_1, E, \Omega)$ of a parity game as shown schematically in Figure 2. More formally:

$$\begin{aligned} V_0 &:= \{b_{i,j} \mid 1 \le l \le i \le n\} \cup \{y_i, d_i \mid i \in [n]\} \cup \{u_i, w_i \mid i \in [n+1]\} \cup \{t, s\} \\ V_1 &:= \{\mathrm{B}_i \mid i \in [n]\} \end{aligned}$$

Table 4 defines the edge sets and the priorities of G_n (with $b_{i,0} = \mathrm{B}_i$).

Node	Successors	Priority	Node	Successors	Priority
u_{n+1}	t	3	t	t	1
$u_{i \le n}$	d_i, u_{i+1}	3	s	d_1	8
w_{n+1}	t	3	B_i	$b_{i,i}, y_i$	6
$w_{i \le n}$	d_i, w_{i+1}	3	y_i	w_{i+1}	$2i+8$
$b_{i,j}$	$b_{i,j-1}, s, u_1$	5	d_i	B_i	$2i+7$

Table 4: ROUND-ROBIN Parity Game Construction (with $b_{i,0} = \mathrm{B}_i$)

The first important observation to make is that the parity game is a sink game, which helps us to transfer our result to mean payoff games, discounted payoff games as well as turned-based simple stochastic games. The following lemma corresponds to Lemma 2 in the MDP world.

Lemma 10. *Starting with the designated initial policy of Section 4, we have that G_n is a sink parity game.*

All other definitions are exactly as in Section 4. Particularly, Table 2 and Table 3 become applicable again. The following lemma has the exact same formulation as Lemma 5 in the MDP world.

Lemma 11. *The improving switches from policies that belong to the phases in Table 2 are exactly those specified in Table 3.*

The reason why this lemma holds is that the valuations of the parity game nodes are essentially the same as the values in the MDP by dropping unimportant $\varepsilon \times *$ terms.

All other proofs in Section 4 rely on Table 2, Table 3 and Lemma 5, hence we transfer our main theorem to the parity game world.

Theorem 12. *The worst-case expected running time of the* ROUND-ROBIN *algorithm for n-state parity games, mean payoff games, discounted payoff games and turn-based simple stochastic games is subexponential.*

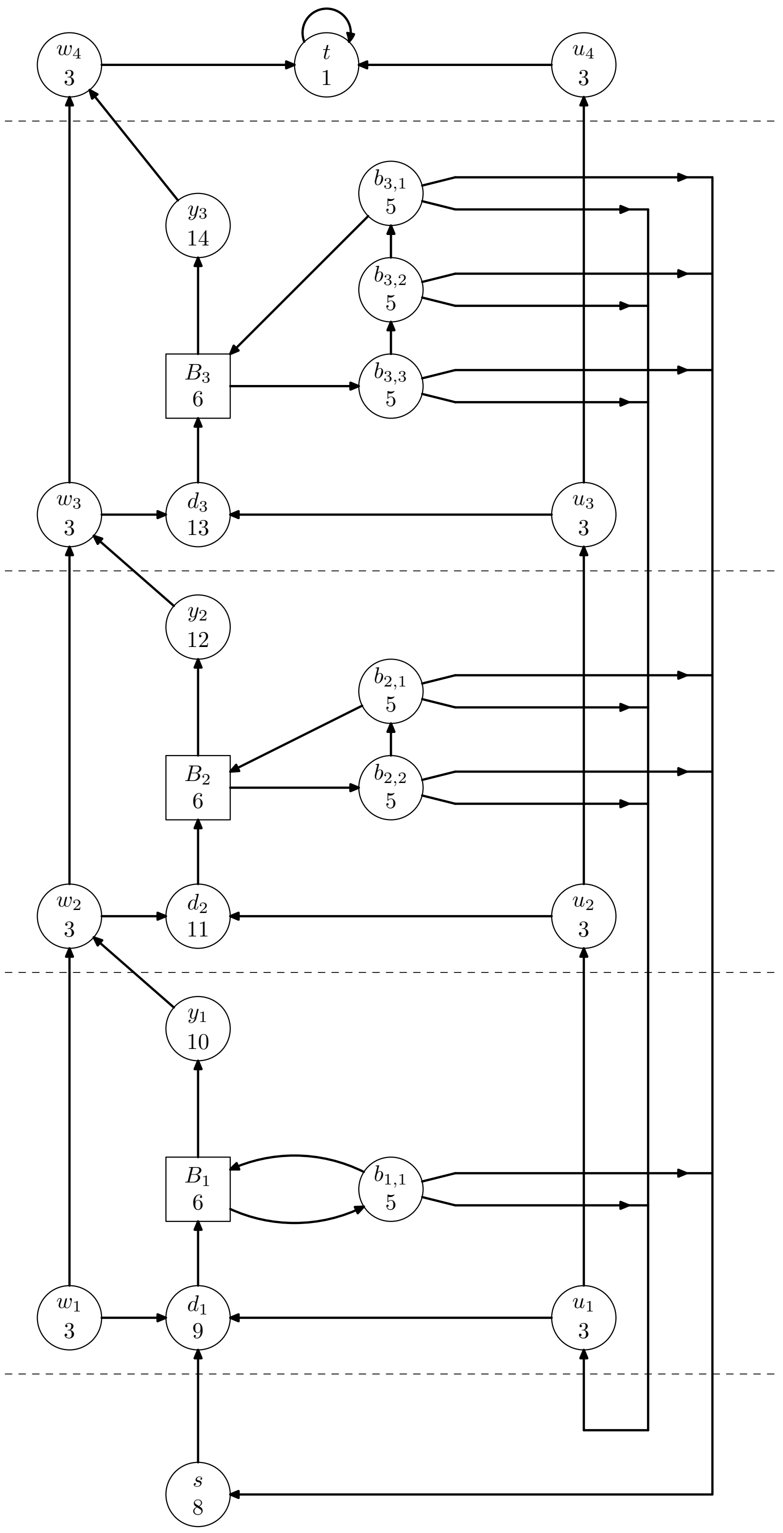

Figure 2: ROUND-ROBIN Parity Game Construction