Exponential Lower Bounds for Policy Iteration

John Fearnley

Department of Computer Science, University of Warwick, UK
john@dcs.warwick.ac.uk

Abstract. We study policy iteration for infinite-horizon Markov decision processes. It has recently been shown policy iteration style algorithms have exponential lower bounds in a two player game setting. We extend these lower bounds to Markov decision processes with the total reward and average-reward optimality criteria.

1 Introduction

The problem of finding an optimal policy for infinite-horizon Markov decision process has been widely studied [8]. Policy iteration is one method that has been developed for this task [5]. This algorithm begins by choosing an arbitrary policy, and then iteratively improves that policy by modifying the policy so that it uses different actions. For each policy, the algorithm computes a set of actions that are switchable, and it then chooses some subset of these actions to be switched. The resulting policy is guaranteed to be an improvement.

The choice of which subset of switchable actions to switch in each iteration is left up to the user: different variants of policy iteration can be created by giving different rules that pick the subset. Traditionally, policy iteration algorithms use a greedy rule that switches every state with a switchable action. Greedy policy iteration will be the focus of this paper.

Policy iteration has been found to work well in practice, where it is used as an alternative to linear programming. Linear programming is known to solve the problem in polynomial time. However, relatively little is known about the complexity of policy iteration. Since each iteration yields a strictly improved policy, the algorithm can never consider the same policy twice. This leads to a natural exponential bound on the number of iterations before the algorithm arrives at the optimal policy. The best upper bounds have been provided by Mansour and Singh [6], who showed that greedy policy iteration will terminate in $O(k^n/n)$ iterations, where k is the maximum number of outgoing actions from a state.

Melekopoglou and Condon have shown exponential lower bounds for some simple variants of policy iteration [7]. The policy iteration algorithms that they consider switch only a single action in each iteration. They give a family of examples upon which these policy iteration algorithms take $2^n - 1$ steps. It has been a long standing open problem as to whether exponential lower bounds could be shown for greedy policy iteration. The best lower bound that has been shown so far is $n + 6$ iterations [2].

S. Abramsky et al. (Eds.): ICALP 2010, Part II, LNCS 6199, pp. 551–562, 2010.

Policy iteration is closely related to the technique of strategy improvement for two player games. Friedmann [4] has recently found a family of parity games upon which the strategy improvement algorithm of Vöge and Jurdziński [9] takes an exponential number of steps. It has been shown that these examples can be generalized to obtain exponential lower bounds for strategy improvement algorithms on other prominent types of two player game [1].

Our contribution. Friedmann's example relies on the fact that there are two players in a parity game. We show how Friedmann's example can be adapted to provide exponential lower bounds for policy iteration on Markov decision processes. We present an example that provides an exponential lower bound for the total reward criterion, and we also argue that the same example provides an exponential lower bound for the average-reward criterion.

2 Preliminaries

A Markov decision process consists of a set of states S, where each state $s \in S$ has an associated set of actions A_s. For a given state $s \in S$ and action $a \in A_s$ the function $\mathrm{r}(s, a)$ gives an integer reward for when the action a in chosen at the state s. Given two states s and s', and an action $a \in A_s$, the function $\mathrm{p}(s'|s, a)$ gives the probability of moving to state s' when the action a is chosen in state s. This is a probability distribution, so $\sum_{s' \in S} \mathrm{p}(s'|s, a) = 1$ for all s and $a \in A_s$.

A deterministic memoryless policy $\pi : S \to A_s$ is a function that selects one action at each state. It has been shown that there is always a policy with this form that maximizes the optimality criteria that we are interested in. Therefore, we restrict ourselves to policies of this form for the rest of the paper. For a given starting state s_0, a run that is consistent with a policy π is an infinite sequence of states $\langle s_0, s_1, \dots \rangle$ such that $p(s_{i+1}|s_i, \pi(s_i)) > 0$ for all i. The set $\Omega^\pi_{s_0}$ contains every consistent run from s_0 when π is used. A probability space can be defined over these runs using the σ-algebra that is generated by the cylinder sets of finite paths starting at s_0. The cylinder set of a finite path contains every infinite path that has the finite path as a prefix. If we fix the probability of the cylinder set of a finite path $\langle s_0, s_1, s_2, \dots s_k \rangle$ to be $\prod_{i=0}^{k-1}(s_{i+1}|s_i, \pi(s_i))$, then standard techniques from probability theory imply that there is a unique extension to a probability measure $\mathbb{P}^\pi_{s_0}(\cdot)$ on the σ-algebra [3]. Given a function that assigns a value to each consistent run $f : \Omega \to \mathbb{R}$, we define $\mathbb{E}^\pi_{s_0}\{f\}$ to be the expectation of this function in the probability space.

The value of a state s when a policy π is used varies according to the choice of optimality criterion. In the total reward criterion the value is $\mathrm{Val}^\pi(s) = \mathbb{E}^\pi_s\{\sum_{i=0}^{\infty} \mathrm{r}(s_i, s_{i+1})\}$, and for the average-reward criterion the value is $\mathrm{Val}^\pi_{\mathcal{A}}(s) = \mathbb{E}^\pi_s\{\liminf_{N \to \infty} \frac{1}{N} \sum_{i=0}^{N} \mathrm{r}(s_i, s_{i+1})\}$. It should be noted that the value of some policies may not exist under the total reward criterion. Our examples will be carefully constructed to ensure that the value does exist. The computational objective is to find the optimal policy π^*, which is the policy that maximizes the value function for every starting state. We define the value of a state to be

the value of that state when an optimal policy is being used. That is, we define $\mathrm{Val}(s) = \mathrm{Val}^{\pi*}(s)$ and $\mathrm{Val}_{\mathcal{A}}(s) = \mathrm{Val}^{\pi*}_{\mathcal{A}}(s)$ for every state s.

For each optimality criterion, it has been shown that the value of each state can be characterised by the solution of a system of optimality equations [8]. For the total reward criterion these optimality equations are, for every state s:

$$V(s) = \max_{a \in A_s} \left(\mathrm{r}(s,a) + \sum_{s' \in S} p(s'|s,a) \cdot V(s') \right) \tag{1}$$

For the average-reward criterion we have two types of optimality equation, which must be solved simultaneously. The first of these are called the gain equations:

$$G(s) = \max_{a \in A_s} \left(\sum_{s' \in S} p(s'|s,a) \cdot G(s') \right) \tag{2}$$

Secondly we have the bias equations. If $M_s = \{a \in A_s \ : \ G(s) = \sum_{s' \in S} p(s'|s,a) \cdot G(s')\}$ is the set of actions that satisfy the gain equation at the state s, then the bias equations are defined as:

$$B(s) = \max_{a \in M_s} \left(\mathrm{r}(s,a) - G(s) + \sum_{s' \in S} p(s'|s,a) \cdot B(s') \right) \tag{3}$$

It has been shown that the solution to these optimality equations is unique, and that this solution characterises the value of each state. That is, we have $\mathrm{Val}(s) = V(s)$ and $\mathrm{Val}_{\mathcal{A}}(s) = G(s)$, for every state s. We can also obtain an optimal policy by setting $\pi^*(s) = a$, where a is an action that achieves the maximum in the optimality equation.

3 Policy Iteration

Policy iteration is a method for solving the optimality equations presented in Section 2. We begin by describing policy iteration for the total reward criterion. For every policy π that the algorithm considers, it computes the value $\mathrm{Val}^{\pi}(s)$ of the policy at every state s, and checks whether this is a solution of the optimality equation (1). The value of the policy can be obtained by solving:

$$\mathrm{Val}^{\pi}(s) = \mathrm{r}(s, \pi(s)) + \sum_{s' \in S} p(s'|s, \pi(s)) \cdot \mathrm{Val}^{\pi}(s') \tag{4}$$

If the value of π satisfies the optimality equation (1) at every state, then a solution has been found and the algorithm terminates. Otherwise, we define the appeal for each action $a \in A_s$ in the policy π to be: $\mathrm{Appeal}^{\pi}(s,a) = \mathrm{r}(s,a) + \sum_{s' \in S} p(s'|s,a) \cdot \mathrm{Val}^{\pi}(s')$. If the policy π does not satisfy the optimality equation there must be at least one action a at a state s such that $\mathrm{Appeal}^{\pi}(s,a) > \mathrm{Val}^{\pi}(s)$. We say that an action with this property is switchable in π. Switching an action $a \in A_t$ in a policy π creates a new policy π' where $\pi'(s) = a$ if $s = t$, and $\pi'(s) = \pi(s)$ for all other states s. It can be shown that switching any subset of switchable actions will create an improved policy.

Theorem 1 ([8]). *If π is a policy and π' is a policy that is obtain by switching some subset of switchable actions in π then $\mathrm{Val}^{\pi'}(s) \geq \mathrm{Val}^{\pi}(s)$ for every state s, and there is some state in which the inequality is strict.*

Policy iteration begins at an arbitrary policy. In each iteration it computes the set of switchable actions, and picks some subset of these actions to switch. This creates a new policy to be considered in the next iteration. Since policy iteration only switches switchable actions, Theorem 1 implies that it cannot visit the same policy twice. This is because repeating a policy would require the value of some state to decrease. Since there are a finite number of policies, the algorithm must eventually arrive at a policy with no switchable actions. This policy satisfies the optimality equation (1), and policy iteration terminates.

Note that any subset of switchable actions can be chosen in each iteration of the algorithm, and the choice of subset affects the behaviour of the algorithm. In this paper we study the greedy policy iteration algorithm, which selects the most appealing switchable action at every state. For every state s where equation (1) is not satisfied, the algorithm will switch the action: $\mathrm{argmax}_{a \in A_s}(\mathrm{Appeal}^{\pi}(s, a))$.

Policy iteration for the average-reward criterion follows the same pattern, but it uses optimality equations (2) and (3) to decide which actions are switchable in a given policy. For each policy it computes a solution to:

$$G^{\pi}(s) = \sum_{s' \in S} p(s'|s, \pi(s)) \cdot G^{\pi}(s)$$

$$B^{\pi}(s) = \mathrm{r}(s, \pi(s)) - G^{\pi}(s) + \sum_{s' \in S} p(s'|s, \pi(s)) \cdot B^{\pi}(s')$$

An action $a \in A_s$ is switchable if either $\sum_{s' \in S} p(s'|s, a) \cdot G^{\pi}(s') > G^{\pi}(s)$ or if $\sum_{s' \in S} p(s'|s, a) \cdot G^{\pi}(s') = G^{\pi}(s)$ and: $\mathrm{r}(s, a) - G^{\pi}(s) + \sum_{s' \in S} p(s'|s, a) > B^{\pi}(s)$.

4 Exponential Lower Bounds for the Total Reward Criterion

In this section we will describe a family of examples that force greedy policy iteration for the total reward criterion to take an exponential number of steps. Due to the size and complexity of our examples, we will break them down into several component parts, which will be presented separately.

Our examples will actually contain very few actions that are probabilistic. An action $a \in A_s$ is deterministic if there is some state s' such that $\mathrm{p}(s'|s, a) = 1$. For the sake of convenience, we will denote actions of this form as (s, s'). We also overload our previous notations: the notation $\pi(s) = s'$ indicates that π chooses the deterministic action from s to s', the function $\mathrm{r}(s, s')$ gives the reward of this action, and $\mathrm{Appeal}^{\pi}(s, s')$ gives the appeal of this action under the policy π.

Since we are working with the total reward criterion, care must be taken to ensure that the value of a policy remains well defined. For this purpose, our examples will contain a sink state c_{n+1} that has a single action (c_{n+1}, c_{n+1}) with reward 0. This will be an absorbing state, in the sense that every run of the

MDP from every starting state will eventually arrive at the state c_{n+1}, for every policy that is considered by policy iteration. This will ensure that the value of each state remains finite throughout the execution of the algorithm.

We will give diagrams for parts our examples, such as the one given in Figure 1. States are drawn as boxes, and the name of a state is printed on the box. Actions are drawn as arrows: deterministic actions are drawn as an arrow between states, and probabilistic actions are drawn as arrows that split, and end at multiple states. The probability distribution is marked after the arrow has split, and the reward of the action is marked before the arrow has split.

Our goal is to construct a set of examples that force policy iteration to mimic the behaviour of a binary counter. Each policy will be associated with some configuration of this counter, and the exponential lower bound will be established by forcing policy iteration to consider one policy for every configuration of the counter. In the rest of this section, we construct an example that forces policy iteration to mimic the behaviour of a binary counter with n bits.

If the bits of our counter are indexed 1 through n, then there are two conditions that are sufficient enforce this behaviour. Firstly, a bit with index i should become 1 only after all bits with index $j < i$ are 1. Secondly, when the bit with index i becomes 1, every bit with index $j < i$ must be set to 0. Our exposition will follow this structure: in section 4.1 we describe how each policy corresponds to a configuration of a binary counter, in section 4.2 we show how the first condition is enforced, and in section 4.3 we show how the second condition is enforced.

4.1 A Bit

The example will contain n instances of structure shown in Figure 1, which will represent the bits. We will represent the configuration of a binary counter as a set $B \subseteq \{1, 2, \ldots n\}$ that contains the indices of the bits that are 1. A policy π represents a configuration B if $\pi(b_i) = a_i$ for every index $i \in B$, and $\pi(b_i) \neq a_i$ for every every index $i \notin B$. For a set of natural numbers B we define $B^{>i}$ to be the set $B \setminus \{k \in \mathbb{N} \ : \ k \leq i\}$. We define analogous notations for $<$, $\geq$, and $\leq$.

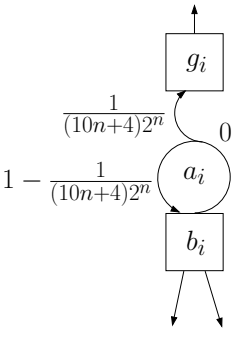

Fig. 1. The structure for the bit with index i

The actions a_i are the only probabilistic actions in the example. When the action a_i is chosen at b_i the effect, under the total reward criterion, is identical to a deterministic action (b_i, g_i) with reward 0. The fact that it takes an expected $(10n + 4)2^n$ steps to move from b_i to the g_i using the action a_i is irrelevant because the reward of a_i is 0, and these steps have no effect on the total reward.

Proposition 2. *For every policy π, if $\pi(b_i) = a_i$ then $\mathrm{Val}^\pi(b_i) = \mathrm{Val}^\pi(g_i)$.*

The reason why the given probabilities have been chosen for the action a_i is that the value of the state g_i will never exceed $(10n + 4)2^n$. Therefore, we make the following assumption, which we will later show to be true.

Assumption 3. *For every policy π we have $\mathrm{Val}^\pi(b_i) > 0$ and $\mathrm{Val}^\pi(g_i) \leq (10n + 4)2^n$.*

Although the action a_i behaves like a deterministic action when it is chosen at b_i, it behaves differently when it is not chosen. A deterministic action (b_i, g_i) would have $\mathrm{Appeal}^\pi(b_i, g_i) = \mathrm{Val}^\pi(g_i)$ in every policy. By contrast, when a_i is not chosen by a policy π, we can show that the appeal of a_i is at most $\mathrm{Val}^\pi(b_i) + 1$.

Proposition 4. *Suppose that Assumption 3 holds. If π is a policy such that $\pi(b_i) \neq a_i$ then $\mathrm{Appeal}^\pi(b_i, a_i) < \mathrm{Val}^\pi(b_i) + 1$.*

This is the key property that will allow us to implement a binary counter. The value of the state g_i could be much larger than the value of b_i. However, we are able to prevent policy iteration from switching the action a_i by ensuring that there is always some other action x such that $\mathrm{Appeal}(b_i, x) \geq \mathrm{Val}^\pi(b_i) + 1$.

4.2 Switching the Smallest Bit

In this section we fully describe the example, and we show how policy iteration can only switch a_i at b_i after it has switched a_j at every state b_j with $j < i$.

Figure 2 shows a key structure, which is called a deceleration lane. Previously we argued that an action (b_i, x) with $\mathrm{Appeal}^\pi(b_i, x) \geq \mathrm{Val}^\pi(b_i) + 1$ is required in every policy π with $\pi(b_i) \neq a_i$ to prevent policy iteration from switching the action a_i. The deceleration is the structure that ensures these actions will exist.

The states x and y both have outgoing actions that will be specified later. For now, we can reason about the behaviour of the deceleration lane by assuming that the value of y is larger than the value of x.

Assumption 5. *For every policy π we have $\mathrm{Val}^\pi(y) > \mathrm{Val}^\pi(x)$.*

The initial policy for the deceleration lane is the one in which every state d_k chooses (d_k, y). It is not difficult to see that the only switchable action in this policy is (d_1, d_0). This is a general trend: the action (d_j, d_{j-1}) can only be switched after every action (d_k, d_{k-1}) with $1 \leq k < j$ has been switched. Therefore, policy

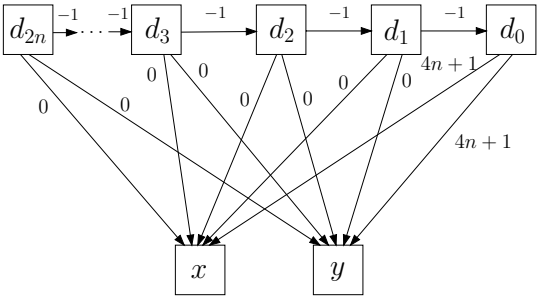

Fig. 2. The deceleration lane

iteration will take $2n$ steps to arrive at the optimal policy for the deceleration lane. Formally, for every j in the range $0 \leq j \leq 2n$ we define a partial policy:

$$\pi_j(s) = \begin{cases} d_{k-1} & \text{if } s = d_k \text{ and } 1 \leq k \leq j, \\ y & \text{otherwise.} \end{cases} \tag{5}$$

Proposition 6. *Suppose that Assumption 5 holds. Applying policy iteration to* π_0 *produces the sequence of policies* $\langle \pi_0, \pi_1, \ldots, \pi_{2n} \rangle$.

Figure 3 shows how each state b_i is connected to the deceleration lane. Of course, since we have not yet specified the outgoing actions from the states f_i, we cannot reason about their appeal. These actions will be used later to force the state b_i to switch away from the action a_i as the binary counter moves between configurations. For now, we can assume that these actions are not switchable.

Assumption 7. *We have* $\text{Appeal}^\pi(b_i, f_j) < \text{Val}^\pi(b_i)$ *for every policy* π *and every action* (b_i, f_j).

We now describe the behaviour of policy iteration for every index $i \notin B$. The initial policy for the state b_i will choose the action (b_i, y). In the first iteration the action (b_i, d_{2i}) will be switched, but after this the action chosen at b_i follows the

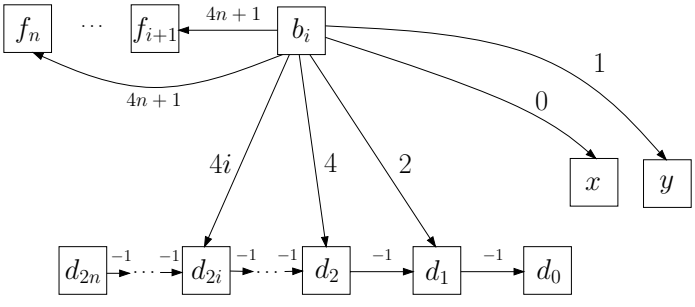

Fig. 3. The outgoing actions from the state b_i

deceleration lane: policy iteration will switch the action (b_i, d_k) in the iteration immediately after it switches the action (d_k, d_{k-1}). Since $\mathrm{r}(b_i, d_k) + \mathrm{r}(d_k, d_{k-1}) = \mathrm{r}(b_i, d_{k-1}) + 1$, this satisfies the condition that prevents the action a_i being switched at b_i. Formally, for every j in the range $0 \leq j \leq 2i + 1$ we define a partial policy:

$$\pi_j^o(s) = \begin{cases} \pi_j(s) & \text{if } s = d_k \text{ for some } k, \\ y & \text{if } j = 0 \text{ and } s = b_i \\ d_{2i} & \text{if } j = 1 \text{ and } s = b_i \\ d_{j-1} & \text{if } 2 \leq j \leq 2i + 1 \text{ and } s = b_i. \end{cases}$$

Proposition 8. *Suppose that Assumptions 3, 5, and 7 hold. When policy iteration is applied to π_0^o it will produce $\langle \pi_0^o, \pi_1^o, \ldots \pi_{2i+1}^o \rangle$.*

We can now see why a bit with index i can only be set to 1 after all bits with index j such that $j < i$ have been set to 1. Since each state b_i has $2i$ outgoing actions to the deceleration lane, policy iteration is prevented from switching the action a_i for $2i + 2$ iterations. Therefore, policy iteration can switch a_i at the state b_i at least two iterations before it can switch a_j at a state b_j with $j > i$.

The second important property of the deceleration lane is that it can be reset. If at any point policy iteration arrives at a policy π_j^o in which $\mathrm{Val}^{\pi_j^o}(x) > \mathrm{Val}^{\pi_j^o}(y) + 6n + 1$ then policy iteration will switch the actions (d_k, x) for all k and the action (b_i, x) for every $i \in B$, to create a policy π'. The reason why these actions must be switched is that the largest value that a state d_k or b_i can obtain in a policy π_j^o is $\mathrm{Val}^{\pi_j^o}(y) + 6n + 1$. Now suppose that $\mathrm{Val}^{\pi'}(y) > \mathrm{Val}^{\pi'}(x) + 4n$. If this is true, then policy iteration will switch the actions (d_k, y) and the action (b_i, y), and it therefore arrives at the policy π_0^o. The ability to force the deceleration lane to reset by manipulating the difference between the values of y and x will be used in the next section.

We now turn our attention to the states b_i where $i \in B$. Policy iteration should never switch away from the action a_i at these states irrespective of the state of the deceleration lane. Since we have not yet specified the outgoing actions of g_i, we need to assume that the value of b_i is large enough to prevent the actions (b_i, d_k) being switchable.

Assumption 9. *For every policy π, if $i \in B$ then $\mathrm{Val}^{\pi}(b_i) > \mathrm{Val}^{\pi}(y) + 6n + 1$.*

When this assumption holds, the state b_i will not be switched away from the action a_i. Formally, for j in the range $2 \leq j \leq 2i$ we define a partial policy:

$$\pi_j^c(s) = \begin{cases} \pi_j(s) & \text{if } s = d_k \text{ for some } k, \\ a_i & \text{if } s = b_i. \end{cases}$$

Proposition 10. *Suppose that Assumptions 5, 7, and 9 hold. When policy iteration is applied to π_0^c it will produce the sequence $\langle \pi_0^c, \pi_1^c, \ldots \pi_{2i}^c \rangle$.*

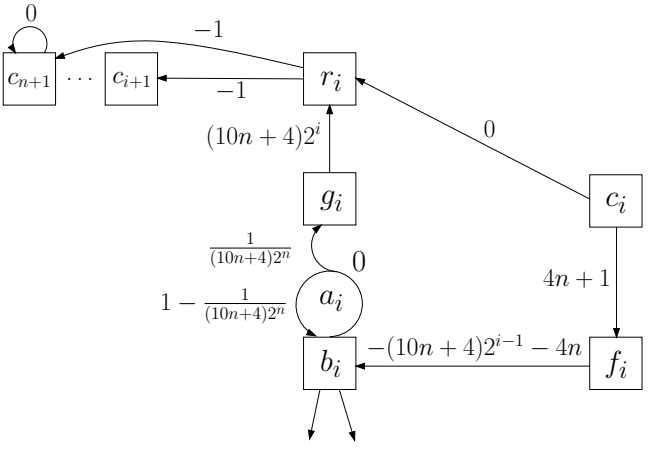

Fig. 4. The structure associated with the state b_i

Figure 4 shows the structure that is associated with each state b_i. We complete the example by specifying the outgoing actions from x and y: there is an action (y, c_i) with reward 0 for every i in the range $1 \leq i \leq n+1$, there is an action (x, f_i) with reward 0 for every i in the range $1 \leq i \leq n$, and there is an action (x, c_{n+1}) with reward -1.

The idea is that the state c_i should use the action (c_i, f_i) only when the index i is a member of B. Moreover, the state r_i should use the action (r_i, c_j) where $j \in B$ is the smallest bit that is both larger than i and a member of B. The state x should use the action (x, f_j) and the state y should use the action (y, c_j) where j is the smallest index that is a member of B.

Formally, for each configuration B we define a partial policy π^B for these states. We define $\pi^B(c_i) = (c_i, f_i)$ if $i \in B$ and $\pi^B(c_i) = (c_i, r_i)$ if $i \notin B$. We define $\pi^B(r_i) = (r_i, c_j)$ where $j = \min(B^{>i} \cup \{n+1\})$. We define $\pi^B(y) = (y, c_j)$ where $j = \min(B \cup \{n+1\})$. We define $\pi^B(x) = (x, f_j)$ where $j = \min(B)$ if $B \neq \emptyset$, and we define $\pi^B(x) = (x, c_{n+1})$ if $B = \emptyset$.

We can now define the sequence of policies that policy iteration will pass through for each configuration B. This definition combines the partial policies π_j, π_j^o, π_j^c, and π^B to give a complete policy π_j^B. If $i = \min(\{i \notin B \ : \ 1 \leq i \leq n\})$ then we define $\text{Sequence}(B) = \langle \pi_1^B, \pi_2^B, \ldots \pi_{2i+1}^B \rangle$, where:

$$\pi_j^B(s) = \begin{cases} \pi_j(s) & \text{if } s = d_k \text{ for some } k, \\ \pi_j^c(s) & \text{if } s = b_i \text{ where } i \in B, \\ \pi_j^o(s) & \text{if } s = b_i \text{ where } i \notin B. \\ \pi^B(s) & \text{otherwise.} \end{cases}$$

We can now see why the assumptions that we have made are true in the full example. For example, in Assumption 3 we asserted that $\text{Val}^\pi(g_i) \leq (10n+4)2^n$. This holds for every policy π_j^B because by following this policy from the state g_i

we pass through r_i followed by c_j, f_j, b_j, g_j, and r_j for every index $j \in B^{>i}$, before arriving at the sink state c_{n+1}. Therefore, the value of the state g_i under the policy π_j^B can be at most $\sum_{l=i+1}^{n}(10n+4)(2^l - 2^{l-1}) + (10n+4)2^i = (10n+4)2^n$. The other assumptions that we have made can be also be shown to be true for every policy π_j^B.

Proposition 11. *For every configuration B we have that Assumptions 3, 5, 7, and 9 hold for every policy π in* Sequence(B).

Our previous propositions have done most of the work in showing that if policy iteration is applied π_0^B, then it will pass through Sequence(B). To complete the proof it is sufficient to note that policy iteration never switches away from the policy π^B at the states c_i, r_i, x, and y.

Proposition 12. *When policy iteration is applied to π_0^B policy iteration will pass through the sequence of policies given by* Sequence(B).

4.3 Moving between Configurations

In this section we will describe the behaviour of policy iteration after the final policy in Sequence(B) has been reached. Throughout this section we define $i = \min(\{j \notin B \ : \ 1 \leq j \leq n\})$ to be the smallest index that is not in the configuration B, and we define $B' = B \cup \{i\} \setminus \{1, 2, \ldots i-1\}$. Our goal is to show that policy iteration moves from the policy π_{2i+1}^B to the policy $\pi_0^{B'}$.

The first policy that policy iteration will move to is identical to the policy π_{2i+2}^B, with the exception that the state b_i is switched to the action a_i. We define:

$$\pi_{\mathrm{R1}}^B(s) = \begin{cases} a_i & \text{if } s = b_i, \\ \pi_{2i+2}^B(s) & \text{otherwise.} \end{cases}$$

This occurs because the state b_i only has $2i$ actions of the form (b_i, d_k). Therefore, once the policy π_{2i+1}^B is reached there will be no action of the form (b_i, d_k) to distract policy iteration from switching the action a_i. Since every other state b_j with $j \notin B$ has at least two actions (b_j, d_k) with $k > 2i$, they move to the policy π_{2i+2}^B.

Proposition 13. *Policy iteration moves from the policy π_{2i+1}^B to the policy π_{R1}^B.*

Since the action a_i has been switched the value of the state f_i is raised to $\mathrm{Val}^{\pi_{\mathrm{R1}}^B}(r_i) + (10n+4)(2^i - 2^{i-1}) - 4n$. The reward of $(10n+4)2^i$ is sufficiently large to cause policy iteration to switch the actions (c_i, f_i) and (x, f_i). It will also switch the actions (b_j, f_i) where for every index $j < i$. Since every index $j \notin B$ other than i has at least one action (b_j, d_k), these states can be switched to the policy $\pi_{2i+3}^B(s)$. Therefore, we define:

$$\pi_{\mathrm{R2}}^B(s) = \begin{cases} \pi_0^B(s) & \text{if } s = b_i \text{ or } s = r_i \text{ or } s \in \{c_j, b_j, r_j \ : \ j > i\}, \\ f_i & \text{if } s = x \text{ or } s = c_i \text{ or } s = b_j \text{ with } j < i, \\ \pi_{2i+3}^B(s) & \text{otherwise.} \end{cases}$$

The most important thing in this iteration is that every state b_j with index $j < i$ is switched away from the action a_j. This provides the critical property of reseting every bit that has a smaller index than i. Another important property is that, while the action (x, f_i) can be switched in this iteration, the action (y, c_i) cannot be switched until after the action (c_i, f_i) has been switched. This will provide a single iteration in which the value of x will exceed the value of y, which is the first of the two conditions necessary to reset the deceleration lane.

Proposition 14. *Policy iteration moves from the policy π^B_{R1} to the policy π^B_{R2}.*

In the next iteration the deceleration lane begins to reset as policy iteration switches (d_k, x) for all k and (b_j, x) where $j > i$ and $j \notin B$. Policy iteration also switches (y, c_i) and (r_j, c_i) with $j < i$. We define:

$$\pi^B_{\mathrm{R3}}(s) = \begin{cases} \pi^B_0(s) & \text{if } s \in \{c_j, b_j, r_j \ : \ j \geq i\} \cup \{x\}, \\ c_i & s = y \text{ or } s = r_j \text{ with } j < i, \\ x & s = d_k \text{ for some } k \text{ or } s = b_j \text{ with } j \notin B \setminus \{x\}, \\ \pi^B_{2i+3}(s) & \text{if } s = c_j \text{ with } j < i. \end{cases}$$

The switching of (y, c_i) provides the second condition for the reset of the deceleration lane. After the action is switched the value of y will be $\mathrm{Val}^{\pi^B_{\mathrm{R2}}}(f_i) + 4n + 1$ whereas the value of x will be $\mathrm{Val}^{\pi^B_{\mathrm{R2}}}(f_i)$. Therefore, policy iteration will reset the deceleration lane in the next iteration. It is also important that the action (b_j, x) for $j \notin B$ is switchable in this iteration, since if $i + 1 \notin B$ then b_{i+1} will have run out of actions (b_{i+1}, d_k) to distract it from switching a_{i+1}. The is the reason why each state b_i must have $2i$ actions to the deceleration lane.

Proposition 15. *Policy iteration moves from the policy π^B_{R2} to the policy π^B_{R3}.*

Finally, once policy iteration has reached the policy π^B_{R3} it will move to the policy $\pi^{B'}_0$. This involves completing the reset of the deceleration lane by switching (d_k, y) for all k, and switching the actions (b_j, y) for every state b_j with index $j \notin B'$. It also makes the final step in transforming the policy π^B to the policy $\pi^{B'}$ by switching the actions (c_j, r_j) at every state c_j with $j < i$.

Proposition 16. *Policy iteration moves from the policy π^B_{R3} to the policy $\pi^{B'}_0$.*

When combined with Proposition 12, the propositions in this section imply that policy iteration will move from the policy π^B_0 to the policy $\pi^{B'}_0$. The optimal policy for the example is π^B_{2n+1} where $B = \{1, 2, \ldots n\}$. This is the policy that selects a_i at b_i for all i, and in which the deceleration lane has reached its optimal policy. Our results so far indicate that if we begin policy iteration at the policy $\pi^{\emptyset}_0$, then policy iteration must pass through a policy π^B_0 for every $B \subseteq \{1, 2, \ldots n\}$. Therefore, it will take at least 2^n iterations to terminate.

Theorem 17. *When policy iteration for the total reward criterion is applied to the policy $\pi^{\emptyset}_0$ it will take at least 2^n iterations to find the optimal policy.*

Finally, we can also argue that the example also provides an exponential lower bound for policy iteration for the average-reward criterion. The first thing to note that $G^\pi(s) = 0$ for every policy that we have specified. This is because all runs eventually reach the sink state c_{n+1}. Since the reward of the action (c_{n+1}, c_{n+1}) is 0, the long term average-reward of every policy will be 0. Note that when $G^\pi(s) = 0$, the bias optimality equation (3) becomes identical to the total reward optimality equation (1). This causes policy iteration for the average-reward criterion to behave identically to policy iteration for the total reward criterion on this example.

Theorem 18. *When policy iteration for the average-reward criterion is applied to the policy $\pi_0^\emptyset$ it will take at least 2^n iterations to find the optimal policy.*

Acknowledgements. I am indebted to Marcin Jurdziński for his guidance, support, and encouragement while preparing this paper.

References

1. Andersson, D.: Extending Friedmann's lower bound to the Hoffman-Karp algorithm. Preprint (June 2009)
2. Andersson, D., Hansen, T.D., Miltersen, P.B.: Toward better bounds on policy iteration. Preprint (June 2009)
3. Ash, R.B., Doléans-Dade, C.A.: Probability and Measure Theory. Academic Press, London (2000)
4. Friedmann, O.: A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it. In: Logic in Computer Science (LICS). IEEE, Los Alamitos (2009)
5. Howard, R.: Dynamic Programming and Markov Processes. Technology Press and Wiley (1960)
6. Mansour, Y., Singh, S.P.: On the complexity of policy iteration. In: Laskey, K.B., Prade, H. (eds.) UAI 1999: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 401–408. Morgan Kaufmann, San Francisco (1999)
7. Melekopoglou, M., Condon, A.: On the complexity of the policy improvement algorithm for Markov decision processes. ORSA Journal on Computing 6, 188–192 (1994)
8. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York (1994)
9. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games (Extended abstract). In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)