The Complexity of the Simplex Method[*]

John Fearnley
University of Liverpool
Liverpool
United Kingdom
john.fearnley@liverpool.ac.uk

Rahul Savani
University of Liverpool
Liverpool
United Kingdom
rahul.savani@liverpool.ac.uk

ABSTRACT

The simplex method is a well-studied and widely-used pivoting method for solving linear programs. When Dantzig originally formulated the simplex method, he gave a natural pivot rule that pivots into the basis a variable with the most violated reduced cost. In their seminal work, Klee and Minty showed that this pivot rule takes exponential time in the worst case. We prove two main results on the simplex method. Firstly, we show that it is PSPACE-complete to find the solution that is computed by the simplex method using Dantzig's pivot rule. Secondly, we prove that deciding whether Dantzig's rule ever chooses a specific variable to enter the basis is PSPACE-complete. We use the known connection between Markov decision processes (MDPs) and linear programming, and an equivalence between Dantzig's pivot rule and a natural variant of policy iteration for average-reward MDPs. We construct MDPs and then show PSPACE-completeness results for single-switch policy iteration, which in turn imply our main results for the simplex method.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Linear programming

Keywords

Linear programming; The simplex method; Dantzig's pivot rule; Markov decision processes; Policy Iteration

1. INTRODUCTION

Linear programming is a fundamental technique that is used throughout computer science. The existence of a *strongly polynomial* algorithm for linear programming is a major open problem in this area, which was included, alongside P vs NP and the Riemann hypothesis, in Smale's list of great unsolved problems of the 21st century [24]. Of course, it is well known that linear programming problems can be solved in polynomial time by using algorithms such as the ellipsoid method or interior point methods, but all of these have running times that depend on the number of binary digits needed to represent the numbers used in the LP, and so they are not strongly polynomial.

A natural way to tackle this problem is to design a strongly polynomial *pivot rule* for the simplex method. The simplex method, originally proposed by Dantzig [5], is a local improvement technique for solving linear programs by pivoting between basic feasible solutions. In each step, the algorithm uses a pivot rule to determine which variable is pivoted into the basis. Dantzig's formulation of the simplex method used a particularly natural pivot rule, which we will call *Dantzig's pivot rule*: in each step, the variable with the most negative reduced cost is chosen to enter the basis [5].

Although the simplex method is known to work well in practice, Klee and Minty showed that Dantzig's pivot rule takes exponential time in the worst case [16]. There has been much subsequent work on developing pivot rules for the simplex method, but all known pivot rules have exponential worst case running time: Friedmann's recent exponential lower bound against Zadeh's pivot rule has eliminated the last major pivot rule for which the worst case running time was unknown [10].

If our goal is to design a strongly polynomial pivot rule, then we must understand what aspects of existing pivot rules make them unsuitable. In this paper, we take a complexity-theoretic approach to answering this question. This line of work was initiated by Disser and Skutella [6], who showed that the following problem is NP-hard. Given a linear program $\mathcal{L}$, a variable v, and a initial basic feasible solution b in which v is not basic, the problem $\textsc{BasisEntry}(\mathcal{L}, b, v)$ asks: if Dantzig's pivot rule is started at b, will it ever choose variable v to enter the basis? They also showed a variety of related results for Dantzig's network simplex algorithm applied to network flow problems.

Another recent result along these lines was given by Adler, Papadimitriou and Rubinstein [2], who studied a slightly different problem that they called the *path problem*: given an LP and a basic feasible solution b, decide whether the simplex method ever visits b. They show that there exists a highly artificial pivot rule for which this problem is in fact PSPACE-complete. They speculate that many other primal pivoting rules may share this property, and as a first step in that direction, they explicitly conjectured that a similar result could be shown for Dantzig's pivot rule, but noted that their techniques would not be sufficient to show this,

[*]A full version of this paper, with all proofs included, is available at `http://arxiv.org/abs/1404.0605`.

.

and left it as a challenging open problem.

In the first main theorem of this paper, we strengthen the result of Disser and Skutella, and (essentially) confirm the conjecture of Adler, Papadimitriou and Rubinstein. We show that the following theorem holds, regardless of the degeneracy resolution rule used by Dantzig's pivot rule.

Theorem 1. BasisEntry *is* PSPACE*-complete.*

Theorem 1 proves a property about the path taken by the simplex method during its computation, but we can also ask about the complexity of finding the solution that is produced by the simplex method. More precisely, we are interested in the following problem: given a linear program $\mathcal{L}$, an initial basic feasible solution b, and a variable v, the problem DantzigLpSol($\mathcal{L}, b, v$) asks: if Dantzig's pivot rule is started at basis b, and finds an optimal solution s, is variable v basic in s?

Questions of this nature have been widely studied for problems in PPAD and PLS. For example, the problem of finding an equilibrium in a bimatrix game is PPAD-complete [4], and this problem can be solved by applying the complementary pivoting approach of Lemke and Howson. It has been shown that it is PSPACE-complete to find any of the equilibria of a bimatrix game that can be computed by the Lemke-Howson algorithm [13]. For the local search problems in PLS [15], the notion of a *tight-PLS reduction* plays a similar role (see, e.g., [26]). If a problem is shown to be tight-PLS-complete, then it is PSPACE-complete to find the solution that is produced by the natural algorithm that follows the suggested local improvement in each step [19, 22]. Thus far, however, this type of result has only been proved for algorithms that solve problems where the complexity of finding any solution is provably hard: either PLS-complete or PPAD-complete.

The simplex method is both a complementary pivoting algorithm and a local search technique. Our second main theorem shows that, despite the fact that linear programming is in P, similar properties can be shown to hold. Once again, the following theorem holds regardless of the degeneracy resolution rule used by Dantzig's pivot rule.

Theorem 2. DantzigLpSol *is* PSPACE*-complete.*

This theorem suggests that PSPACE-completeness results for finding the solution produced by a particular algorithm may be much more widespread than previously thought. For example, when Monien and Tscheuschner showed a PSPACE-completeness result for the natural algorithm for finding a locally-optimal maximum cut on graphs of degree four [18], they saw this as evidence that the problem may be tight-PLS-complete, but this line of reasoning seems less appealing in light of Theorem 2.

Techniques.

In order to prove Theorems 1 and 2, we make use of a known connection between the simplex method for linear programming and *policy iteration* algorithms for Markov decision processes (MDPs), which are discrete-time stochastic control processes [21]. The problem of finding an optimal policy in an MDP can be solved in polynomial time by a reduction to linear programming. However, policy iteration is a local search technique that is often used as an alternative. Policy iteration starts at an arbitrary policy. In each policy it assigns each action an *appeal*, and if an action has positive appeal, then switching this action creates a strictly better policy. Thus, policy iteration proceeds by repeatedly switching a subset of switchable actions, until it finds a policy with no switchable actions. The resulting policy is guaranteed to be optimal.

We use the following connection: If a policy iteration algorithm for an MDP makes only a single switch in each iteration, then it corresponds directly to the simplex method for the corresponding linear program. In particular, Dantzig's pivot rule corresponds to the natural switching rule for policy iteration that always switches the action with highest appeal. We call this *Dantzig's rule.* This connection is well known, and has been applied in other contexts. Friedmann, Hansen, and Zwick used this connection in the expected total-reward setting, to show sub-exponential lower bounds for some randomized pivot rules [12]. Post and Ye have shown that Dantzig's pivot rule is strongly polynomial for *deterministic discounted* MDPs [20], while Hansen, Kaplan, and Zwick went on to prove various further bounds for this setting [14].

We define two problems for Dantzig's rule, which correspond to the two problems that we study in the linear programming setting. Let $\mathcal{M}$ be an MDP, σ be a starting policy, and a be an action. The problem ActionSwitch($\mathcal{M}, \sigma, a$) asks: if Dantzig's rule is started at some policy σ that does not use a, will it ever switch action a? The problem DantzigMdpSol($\mathcal{M}, \sigma, a$) asks: if σ^* is the optimal policy that is found when Dantzig's rule is started at σ, does σ^* use action a? We will show the following pair of theorems.

Theorem 3. ActionSwitch *is* PSPACE*-complete.*

Theorem 4. DantzigMdpSol *is* PSPACE*-complete.*

Policy iteration is a well-studied and widely-used method, so these theorems are of interest in their own right. Since we show that Dantzig's switching rule corresponds to Dantzig's pivot rule on a certain LP, these two theorems immediately imply that BasisEntry and DantzigLpSol are PSPACE-complete (Theorems 1 and 2, respectively). The majority of the paper is dedicated to proving Theorem 3; we then add one extra gadget to our construction to prove Theorem 4.

Related work.

There has been a recent explosion of interest in the complexity of pivot rules for the simplex method, and of switching rules for policy iteration. The original spark for this line of work was a result of Friedmann, which showed an exponential lower bound for the *all-switches* variant of strategy improvement for two-player *parity games* [8, 9]. Fearnley then showed that the second player in Friedmann's construction can be simulated by a probabilistic action, and used this to show an exponential lower bound for the all-switches variant of policy iteration for average-reward MDPs [7]. Friedmann, Hansen, and Zwick then showed a sub-exponential lower bound for the *random facet* strategy improvement algorithm for parity games [11], and then utilised Fearnley's construction to extend the bound to the random facet pivot rule for the simplex method [12].

It is generally accepted that the simplex algorithm performs well in practice. Our strong worst-case negative result should be understood in the context of a long line of work that has attempted to explain the good behaviour of the simplex algorithm. This started with probabilistic analyses of the expected running time of variants of the simplex method by Adler and Megiddo [1], Borgwardt [3], and Smale [23]. Later, in seminal work, Spielman and Teng [25] defined the concept of smoothed analysis and showed that the simplex algorithm has polynomial smoothed complexity.

2. PRELIMINARIES

Markov decision processes.

A Markov decision process (MDP) is defined by a tuple $\mathcal{M} = (S, (A_s)_{s \in S}, p, r)$, where S gives the set of states in the MDP. For each state $s \in S$, the set A_s gives the actions available at s. We also define $A = \bigcup_s A_s$ to be the set of all actions in $\mathcal{M}$. For each action $a \in A_s$, the function $p(s', a)$ gives the probability of moving from s to s' when using action a. Obviously, we must have $\sum_{s' \in S} p(s', a) = 1$, for every action $a \in A$. Finally, for each action $a \in A$, the function $r(a)$ gives a rational *reward* for using action a. A *policy* is a function $\sigma : S \to A$, which for each state s selects an action from A_s. We define Σ to be the set of all policies.

We use the *expected total reward* optimality criterion. Maximizing the expected total reward is equivalent to solving the system of *optimality equations*, where for each state $s \in S$ we have:

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

It has been shown that, if these equations have a solution, then $\mathrm{Val}(s)$ gives the maximal expected total reward that can be obtained when starting in state s [21].

Policy iteration.

Policy iteration is an algorithm for finding solutions to the optimality equations. For each policy $\sigma \in \Sigma$, we define the following system of linear equations:

$$\mathrm{Val}^\sigma(s) = r(\sigma(s)) + \sum_{s' \in S} p(s', \sigma(s)) \cdot \mathrm{Val}^\sigma(s'). \quad (2)$$

In a solution of this system, $\mathrm{Val}^\sigma(s)$ gives the expected total reward obtained by following σ from state s. For each policy $\sigma \in \Sigma$, each state s, and each action $a \in A_s$ we define:

$$\mathrm{Appeal}^\sigma(a) = \left(r(a) + \sum_{s' \in S} p(s', a) \cdot \mathrm{Val}^\sigma(s') \right) - \mathrm{Val}^\sigma(s). \quad (3)$$

We say that action $a \in A_s$ is *switchable* in σ if $\mathrm{Appeal}^\sigma(a) > 0$. Switching an action a at a state s in a policy σ creates a new policy σ' such that $\sigma'(s) = a$, and $\sigma'(s') = \sigma(s')$ for all states $s' \neq s$.

We define a partial order over policies according to their values. If $\sigma, \sigma' \in \Sigma$, then we say that $\sigma \prec \sigma'$ if and only if $\mathrm{Val}^{\sigma'}(s) \geq \mathrm{Val}^\sigma(s)$ for every state s, and there exists a state s' for which $\mathrm{Val}^{\sigma'}(s') > \mathrm{Val}^\sigma(s')$. We have the following theorem.

THEOREM 5 ([21]). *If σ is a policy and σ' is a policy that is obtained by switching a switchable action in σ then we have $\sigma \prec \sigma'$.*

Policy iteration starts at an initial policy σ_0. In iteration i, it switches a switchable action in σ_{i-1} to create a new policy σ_i. Since there are finitely many policies in Σ, Theorem 5 implies that it must eventually reach a policy σ^* with no switchable actions. By definition, a policy with no switchable actions is a solution to Equation (1), so σ^* is an optimal policy, and the algorithm terminates.

One technical complication of our result is that the MDPs that we construct do not actually fall into a class of MDPs for which total-reward policy iteration is known to work. For that reason, in the full version of this paper, we formally define our results in terms of the *expected average reward* optimality criterion. We show that, for the policies that we consider in our construction, policy iteration for total reward MDPs behaves in exactly the same way as policy iteration for average reward MDPs. Thus, it is valid to describe our result in terms of expected total reward.

Dantzig's rule.

Policy iteration uses a *switching rule* to determine which switchable action should be switched in each step. In this paper, we concentrate on *Dantzig's switching rule*, which always selects an action with maximal appeal. More formally, if σ is a policy with at least one switchable action, then Dantzig's switching rule selects an action a such that $\mathrm{Appeal}^\sigma(a)$ is equal to:

$$\max\{\mathrm{Appeal}^\sigma(a') : a' \in A \text{ and } \mathrm{Appeal}^\sigma(a') > 0\}. \quad (4)$$

If more than one action satisfies this equation, then Dantzig's switching rule selects one arbitrarily. Our `PSPACE` completeness results hold no matter how ties are broken. We will refer to the policy iteration algorithm that always follows Dantzig's switching rule as *Dantzig's rule*.

The connection with linear programming.

There is a strong connection between policy iteration for Markov decision processes, and the simplex method for linear programming. In particular, for a number of classes of MDPs there is a well-known reduction to linear programming, which essentially encodes the optimality equations as a linear program. In the full version of the paper, we show formally that maximizing the expected total reward in our construction can be written down as a linear program. Furthermore, we show that, on this LP, the simplex algorithm equipped with Dantzig's pivot rule corresponds exactly to policy iteration equipped with Dantzig's switching policy.

When Dantzig's pivot rule is applied in linear programming, a *degeneracy resolution* rule is required to prevent the algorithm from cycling. This rule picks the entering variable and leaving variable in the case of ties. In our formulation, the leaving variable is always unique. The entering variable is determined according to Equation (4). Since our `PSPACE`-completeness results for MDPs hold no matter how ties are broken, our `PSPACE`-completeness results for Dantzig's pivot rule will also hold no matter which degeneracy resolution rule is used. Thus, the main technical challenge of the paper is to show that the problems ACTIONSWITCH and DANTZIGMDPSOL, which were defined in the introduction, are `PSPACE`-complete problems.

The appeal reduction gadget.

We now describe the *appeal reduction gadget*, which will be frequently used in our construction. This gadget allows us to control the action that is switched by Dantzig's rule. Similar gadgets were used by Melekopoglou and Condon to show an exponential-time lower bound for Dantzig's rule [17], and by Fearnley to show an exponential-time lower bound against the all-switches rule [7].

The gadget is shown in Figure 1. Throughout the paper, we will use the following diagramming notation for MDPs. States are represented as boxes, and the name of the state is displayed in the center of the box. Each action is represented by an arrow that is annotated by a reward. If an action has more than one possible destination state then the arrow will split, and the reward is displayed before the split, while the transition probabilities are displayed after the split. The left half of Figure 1 diagram shows the gadget itself, and the right half shows our diagramming shorthand for the gadget: whenever we use this shorthand in our diagrams, we intend it to be replaced with the gadget in the left half of Figure 1.

To understand the purpose of this gadget, imagine that r_f and r_d are both 0. If this is the case, then we have the following two properties. Let a be the action from s to s'.

- If σ is a policy with $\sigma(s) = a$, then we have $\text{Val}^\sigma(s) = \text{Val}^\sigma(t)$.
- If σ is a policy with $\sigma(s) \neq a$, and if $\text{Val}^\sigma(t) = \text{Val}^\sigma(s) + b$, for some constant b, then we have $\text{Appeal}^\sigma(a) = p \cdot b$.

The first property states that the appeal reduction gadget acts like a normal action from s to t when it is used by a policy. However, the second property states that, if the action is not used, then the appeal of moving from s to t is reduced by the probability p. In short, an appeal reduction gadget can be used to make certain actions seem less appealing. Since Dantzig's rule always switches the action with highest appeal, this will allow us to control which action is switched. The rewards r_f and r_d give us further control on precisely how appealing the action is, and how rewarding the action is when it is used by a policy.

Circuit iteration problems.

The starting point for our PSPACE-hardness reductions will be a pair of circuit iteration problems. 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 define two different circuit iteration problems, which correspond to the two different theorems that we prove for Dantzig's rule. The problems are:

- BitSwitch(F, B, z): if the z-th bit of B is 1, then decide whether there exists an even $i \leq 2^n$ such that the z-th bit of $F^i(B)$ is 0.
- CircuitValue(F, B, z): decide whether the z-th bit of $F^{2^n}(B)$ is 0.

The requirement for i to be even in BitSwitch is a technical requirement that is necessary in order to make our reduction work. The fact that both of these problems are PSPACE-complete should not be too surprising, because we can use the circuit F to simulate a single step of a space-bounded Turing machine, so iterating F simulates a run of a space-bounded Turing machine.

In order to make our reduction work, we must make several assumptions about the format of the circuit C, all of which can be made without loss of generality. Firstly, we assume that the circuit only contains Not gates and Or gates. Secondly, we make an assumption about gate depths. For each gate i, we use $d(i)$ to denote the *depth* of the gate, which is the length of the longest path from i to an input bit. We assume that, for every Or gate, both inputs of the gate have the same depth. Finally, we assume that the circuits are presented in *negated form*, which means that all of the output bits are negated. That is, the z-th output bit will produce a 1 if and only if $F(B)$ is 0. We also make several further technical assumptions about the precise format of the circuit, which are explained in detail in the full paper, but which are not necessary for a high-level understanding of how the construction works.

3. THE CONSTRUCTION

Our main result is that BitSwitch and CircuitValue can be reduced to ActionSwitch and DantzigMdpSol, respectively. Both reductions use the same construction, but the reduction from CircuitValue to DantzigMdpSol uses one extra gadget that we will describe at the end. Let (F, B, z) be a circuit iteration instance, and let C be the circuit that implements F. In this section, we give a high-level overview of how our construction converts (F, B, z) into an MDP. A complete formal definition of our construction is available in the full version of the paper.

The construction is driven by a *clock*, which consists of a modified version of the exponential-time examples of Melekopoglou and Condon [17]. The clock has two output states c_0 and c_1, and the difference in value between these two states is what drives our construction. In particular, the clock alternates between two output *phases*: in phase 0 we have $\text{Val}^\sigma(c_1) \gg \text{Val}^\sigma(c_0)$, while in phase 1 we have $\text{Val}^\sigma(c_0) \gg \text{Val}^\sigma(c_1)$. The clock alternates between phase 0 and phase 1, and goes through exactly 2^n phases in total.

The construction also contains *two* full copies of the circuit C, which we will call circuit 0 and circuit 1. At the start of the computation, circuit 0 will hold the initial bit-string B in its input bits. During phase 0, circuit 0 will compute $F(B)$, and circuit 1 will copy $F(B)$ into its input bits. The clock then moves to phase 1, which causes circuit 1 to compute $F(F(B))$, and circuit 0 to copy $F(F(B))$ into its input bits. This pattern is then repeated until 2^n phases have been encountered, and so when the clock terminates we will have computed $F^{2^n}(B)$.

Each copy of the circuit is built from gadgets. We design gadgets to model the input bits, Or gates, and Not gates. In particular, for each gate i in the circuit, and for each $j \in \{0, 1\}$, there is a state o_i^j, which represents the output of the gate i in copy j of the circuit. The value of this state will indicate whether the gate is true or false. We use a family of constants H_k, L_k, and b_k, which have the following properties: for each k we have $H_k > L_k$, $H_k = L_k + b_k$, and $H_k = L_{k+1}$. The idea is that H_k and L_k give high and low values, which will be used by gates of depth k to indicate whether they are true or false. More precisely, the truth values in circuit j will be given relative to the value of the clock state c_j. Given a policy σ, we say that:

- Gate i is false in σ in phase j if $\text{Val}^\sigma(o_i^j) = \text{Val}^\sigma(c_j) + L_{d(i)}$.
- Gate i is true in σ in phase j if $\text{Val}^\sigma(o_i^j) = \text{Val}^\sigma(c_j) + H_{d(i)}$.

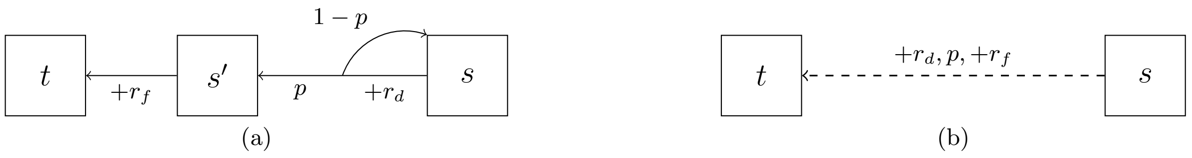

Figure 1: Left: The appeal reduction gadget with rewards r_f and r_d, and probability p. Right: our shorthand for the appeal reduction gadget.

Throughout this high-level description, we will use "gate i outputs x" as a shorthand for saying that $\text{Val}^\sigma(o_i^j) = \text{Val}^\sigma(c_j) + x$.

We now describe the gadgets used in the construction. Throughout our exposition, we will only describe the parts of the construction that are necessary for understanding the high-level picture. In particular, we will not define the probabilities used in the appeal reduction gadgets, because knowing the precise probabilities is not necessary for understanding the construction.

The clock.

Figure 2 shows the clock. We have used special notation to simplify this diagram. The circle states have a single outgoing action a. Each circle state has two outgoing arrows, which both have probability 0.5 of being taken when a is used. The clock is an adaptation of the exponential-time lower bound of Melekopoglou and Condon [17], but with several modifications. Most importantly, while they considered minimising the reachability probability of state 0, we consider maximizing the expected total reward.

Note that the two states c_0 and c_1 are the two clock states that we described earlier. In the initial policy for the clock, all states select the action going right. In the optimal policy, all states select the action going right, except state 1, which selects its downward action. However, to move from the initial to final policy, Dantzig's rule makes an exponential number of switches. In each step, the probability of reaching si' increases. The large payoff of $T \cdot 2^{n+1}$, where T is a large positive constant, ensures that the difference in value between c_0 and c_1 is always T.

Or gates.

Figure 3a shows the gadget that will be used for each Or gate i in circuit $j \in \{0, 1\}$, where $\text{I}_1(i)$ and $\text{I}_2(i)$ are the two inputs for gate i. As the circuit is computing in phase j we will have that x_i^j takes the action towards c_j. The behaviour of this gadget is comparatively straightforward. If one of the two input gates outputs a high signal, then the state v_i^j will switch to the corresponding output state, and the output of gate i will be $H_{d(i)-1} + b_d(i) = H_{d(i)}$. On the other hand, if both input gates output a low signal, then o_i^j will switch to x_i^j, so in this case the gate outputs $L_{d(i)}$, which is the correct output for this case.

Not gates.

Figure 3b shows the gadget that will be used to represent a Not gate i in circuit $j \in \{0, 1\}$, where $\text{I}(i)$ is the input to the gate. The state a_i^j is used to activate the gadget. At the start of clock phase j, we have that a_i^j takes the action towards c_j. The probability p_1 will be carefully chosen to ensure that a_i^j can only switch to c_{1-j} after the output of all gates with depth strictly less than i has been computed. The gate activates when a_i^j switches to c_{1-j}. While the gadget is not activated, the state o_i^j will use the action towards $o_{\text{I}(i)}^j$. Once the gadget has been activated, there are then two possible outcomes, depending on whether the input bit is low or high.

- If the input bit is low, then the appeal of switching o_i^j to a_i^j is large. So, o_i^j will be switched to a_i^j, and the rewards on the action from o_i^j to a_i^j and on the action from a_i^j to c_{1-j} ensure that the gate outputs $H_{d(i)-1} + b_{d(i)} = H_{d(i)}$, which is the correct output for this case.

- If the input bit is high, then the appeal of switching o_i^j to a_i^j is smaller, because the state $o_{\text{I}(i)}^j$ has a higher value. In fact, we ensure that the appeal is so small that the action from o_i^j to a_i^j will never be switched, because its appeal is smaller than the appeal of the action that advances the clock to the next phase. So, o_i^j never switches away from $o_{\text{I}(i)}^j$, and therefore the gate will output $H_{d(i)-1} = L_{d(i)}$, which is the correct output for this case. Here we can see why the relationship between $H_{d(i)-1}$ and $L_{d(i)}$ is needed in order for our reduction to work.

Input bits.

Figure 4 shows the gadget that we will use for every input bit i and every $j \in \{0, 1\}$, where $\text{I}(i)$ gives the output bit that corresponds to input bit i. Since the input bits lie at the interface between the two circuits, they are the most complicated part of the construction. The input bit gadget has two distinct modes. During phase j, when circuit j is computing, the input bits in circuit j are in *output mode*, which means that they output the values that they are storing. During phase $1-j$, when circuit j is copying, the input bits in circuit j are in *copy mode*, which means that they must copy the output of gate $\text{I}(i)$ from circuit $1-j$.

We start by describing the output mode. In this mode, states l_i^j and r_i^j both take the action towards c_j. The output is determined by the action chosen by state o_i^j: if the action to l_i^j is chosen, then the gadget outputs the high signal H_0, and if the action to r_i^j is chosen, then the gadget outputs the low signal L_0.

When the gadget is in copy mode, the state l_i^j takes the action towards c_{1-j} and the state r_i^j takes the action towards $o_{\text{I}(i)}^{1-j}$. In this mode, the state o_i^j initially takes the action towards l_i^j. If at any point the gate $\text{I}(i)$ outputs a high signal, then o_i^j switches to r_i^j, if $\text{I}(i)$ only ever outputs a low

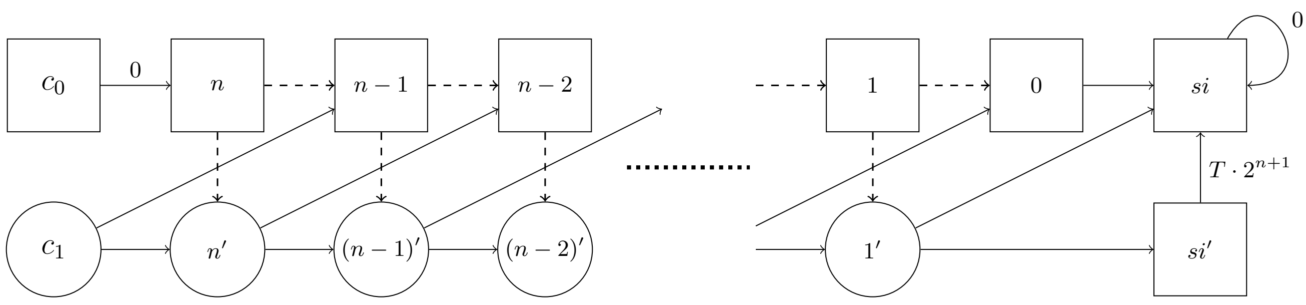

Figure 2: The clock construction.

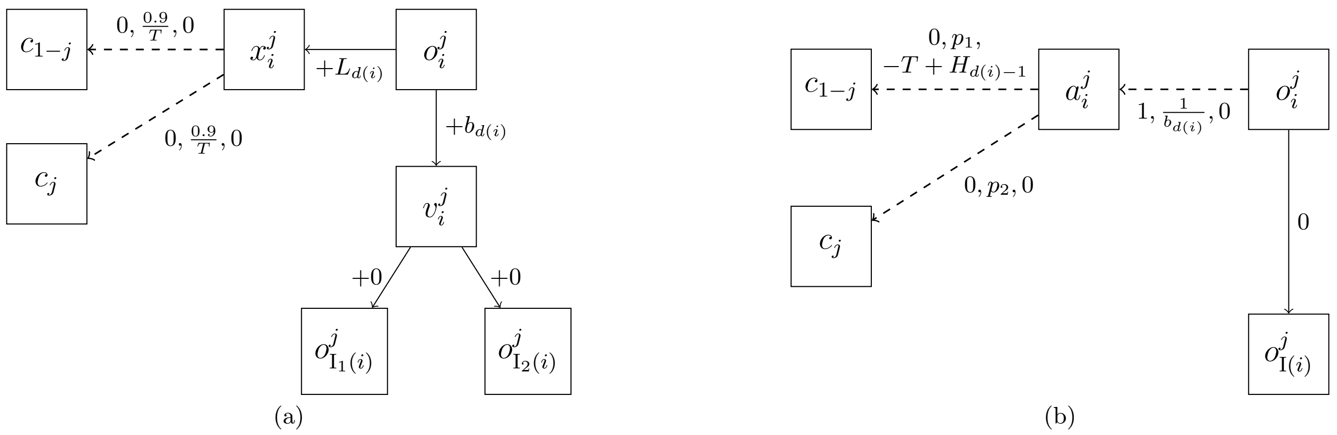

Figure 3: Left: The gadget for an Or gate i in circuit j. Right: A gadget for a Not gate i in circuit j.

signal, then o_i^j does not switch away from l_i^j. Hence, when the gadget moves back into output mode, it will output a high signal if and only if I(i) gave a low signal in the previous phase. Since we require that the circuit is given in negated form, this is the correct behaviour.

The most complicated part of the construction is ensuring that the input bits switch modes correctly at the end of each clock phase. This is difficult because we must ensure that the data held by the states o_i^j is not lost, and because this operation happens simultaneously in both circuits. Ultimately, we show that there are probabilities p_3 through p_7 that ensure that the mode transitions occur correctly.

ActionSwitch is PSPACE-complete.

We have now completed the high-level description of all of the gadgets in the construction. Suppose that we are given an instance BitSwitch(F, B, z). We define an initial policy σ_{init} such that the input bits in circuit 0 output the bit-string B. The state o_z^0 is the output state of the input-bit gadget corresponding to the z-th input bit. When we execute Dantzig's rule starting from σ_{init}, we have that the state o_z^0 switches to r_z^0 if and only if there exists an even $i \leq 2^n$ such that the z-th bit of $F^i(B)$ is 1. So, we have a reduction from BitSwitch to ActionSwitch, which proves that ActionSwitch is PSPACE-complete.

DantzigMdpSol is PSPACE-complete.

To show that DantzigMdpSol is PSPACE-complete, we use one additional gadget, which is shown in Figure 5. Suppose that we have an instance CircuitValue(F, B, z). We create the MDP as before, but the additional gadget interfaces with the states l_z^0 and r_z^0 from the z-th input bit gadget. The purpose of this gadget is to make sure that the z-th output bit computed in phase 2^n is not destroyed by any subsequent switching.

In the initial policy, the state b_2 uses the action that goes directly to the state si from the clock. In this configuration, b_2 has value 0, so the states l_z^0 and r_z^0 will not switch to b_2. Therefore, the behaviour of policy iteration will be unchanged, and Dantzig's rule will compute $F^{2^n}(B)$.

The appeal reduction gadget between b_2 and b_1 is configured so that b_2 switches to b_1 immediately after the 2^n-th clock phase. Once this has occurred, the state b_2 will have value $2 \cdot W$, where the payoff W is chosen to be extremely large, so both l_z^0 and r_z^0 will switch to b_2. After both states have switched, the state o_z^0 will be indifferent between its two successors, so it can never be switched. Thus, when policy iteration terminates with an optimal policy, the action chosen by o_z^0 will be determined by the z-th bit of $F^{2^n}(B)$, which completes the reduction from CircuitValue to DantzigMdpSol.

4. CONCLUSION

There are several research directions that arise from this work. Firstly, and most obviously, is to extend the PSPACE-completeness results to other pivot rules. A first step here

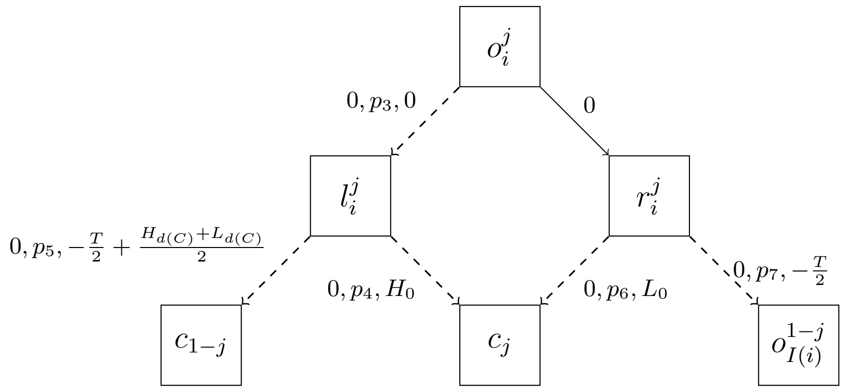

Figure 4: The gadget for input bit i in circuit j.

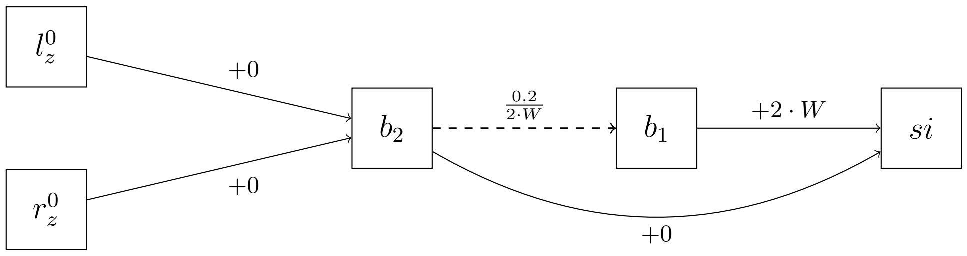

Figure 5: The extra gadget needed in Constr(C, z).

might be to consider Bland's rule, which assigns an index to each variable, and always pivots the variable with negative reduced cost that has the least index. It is possible that our construction might be applied in a rather direct way to show results for Bland's rule, since our construction essentially relies on controlling the order in which actions are switched in the MDP.

Providing results for other pivot rules will probably be more challenging, and require different constructions. The steepest edge pivot rule is the one that is most commonly used in practice, so proving PSPACE-completeness results for this rule might be the next logical step. It is not even known if the steepest edge switching policy can take exponential time for MDPs. There are also other rules, such as the one that in each step gives the biggest overall improvement in the objective function, where PSPACE-completeness results would be interesting.

Moving away from linear programming, our results indicate that we may be able to show PSPACE-completeness results for a wide variety of different algorithms. As mentioned in the introduction, this type of result was only previously known to hold for algorithms that solve PPAD-hard or PLS-hard problems, but our results indicates that this phenomenon may be more widespread. In particular, parity games, mean-payoff games, and discounted games are three problems that lie at the intersection of PPAD and PLS, but which are not known to lie in P. Strategy improvement algorithms, which are a generalisation of policy iteration, are used to solve these games, and these algorithms are an obvious target. Note that our results already hold for strategy improvement algorithms that solve *stochastic* mean-payoff games, because these games are a superset of average-reward MDPs. It seems likely that this can be used directly to show results for stochastic discounted games, and simple stochastic games, using reductions similar to the ones used by Friedmann in the non-stochastic setting [9].

Another target might be Lemke's algorithm for solving linear complementarity problems (LCPs). For the special case of *P-matrix* LCPs, the problem of finding a solution is known to lie at the intersection of PPAD and PLS, but not known to lie in P. At first glance, such results may not seem possible for P-matrix LCPs, because these problems are known to always have a unique solution. However, there is some leeway for a result here, because the solution can lie at the boundary between multiple convex cones, and the algorithm will only return one of these cones as a solution.

Ultimately, the end goal of this line of research is to produce a “recipe” for PSPACE-hardness than can be applied to a wide variety of simplex pivoting rules. For example, Adler, Papadimitriou, and Rubintstein ask whether such a result holds for all pivoting rules that use only primal feasible bases [2]. Any such recipe would shed light on the requirements for a strongly polynomial-time pivot rule. While we believe our work has made an important first step, it is clear that much further research will be necessary to achieve this goal.

5. ACKNOWLEDGEMENTS

This work was supported by EPSRC grant EP/L011018/1 “Algorithms for Finding Approximate Nash Equilibria.”

6. REFERENCES

[1] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two

quadratic functions of the smaller dimension. *J. ACM*, 32(4):871–895, Oct. 1985.

[2] I. Adler, C. Papadimitriou, and A. Rubinstein. On simplex pivoting rules and complexity theory. In *Proc. of IPCO*, 2014.

[3] K. H. Borgwardt. *A Probabilistic Analysis of the Simplex Method*. Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[4] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3):14:1–14:57, May 2009.

[5] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press, 1965.

[6] Y. Disser and M. Skutella. In defense of the simplex algorithm's worst-case behavior. *CoRR*, abs/1311.5935, 2013.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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. Preliminary version: *FOCS 2011*.

[14] T. D. Hansen, H. Kaplan, and U. Zwick. Dantzig's pivoting rule for shortest paths, deterministic MDPs, and minimum cost to time ratio cycles. In *Proc. of SODA*, pages 847–860, 2014.

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

[16] V. Klee and G. Minty. How good is the simplex algorithm? In *Inequalities, III*, pages 159–âĂŞ175, 1972.

[17] M. Melekopoglou and A. Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *INFORMS Journal on Computing*, 6(2):188–192, 1994.

[18] B. Monien and T. Tscheuschner. On the power of nodes of degree four in the local max-cut problem. In *Proc. of CIAC*, pages 264–275, 2010.

[19] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. On the complexity of local search. In *Proc. of STOC*, pages 438–445, 1990.

[20] I. Post and Y. Ye. The simplex method is strongly polynomial for deterministic Markov decision processes. In *Proc. of SODA*, pages 1465–1473, 2013.

[21] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[22] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991.

[23] S. Smale. On the average number of steps of the simplex method of linear programming. *Mathematical Programming*, 27(3):241–262, 1983.

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

[25] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, May 2004.

[26] M. Yannakakis. Computational complexity. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley, 1997.