THE COMPLEXITY OF ALL-SWITCHES STRATEGY IMPROVEMENT

JOHN FEARNLEY AND RAHUL SAVANI

University of Liverpool, Liverpool, United Kingdom
e-mail address: john.fearnley@liverpool.ac.uk

University of Liverpool, Liverpool, United Kingdom
e-mail address: rahul.savani@liverpool.ac.uk

ABSTRACT. Strategy improvement is a widely-used and well-studied class of algorithms for solving graph-based infinite games. These algorithms are parameterized by a switching rule, and one of the most natural rules is "all switches" which switches as many edges as possible in each iteration. Continuing a recent line of work, we study all-switches strategy improvement from the perspective of computational complexity. We consider two natural decision problems, both of which have as input a game G, a starting strategy s, and an edge e. The problems are: 1.) The edge switch problem, namely, is the edge e ever switched by all-switches strategy improvement when it is started from s on game G? 2.) The optimal strategy problem, namely, is the edge e used in the final strategy that is found by strategy improvement when it is started from s on game G? We show PSPACE-completeness of the edge switch problem and optimal strategy problem for the following settings: Parity games with the discrete strategy improvement algorithm of Vöge and Jurdziński; mean-payoff games with the gain-bias algorithm [14,37]; and discounted-payoff games and simple stochastic games with their standard strategy improvement algorithms. We also show PSPACE-completeness of an analogous problem to edge switch for the bottom-antipodal algorithm for finding the sink of an Acyclic Unique Sink Orientation on a cube.

1. Introduction

In this paper we study strategy improvement algorithms for solving two-player games such as parity games, mean-payoff games, discounted games, and simple stochastic games [24,37,43]. These games are interesting both because of their important applications and their unusual complexity status. Parity games, for example, arise in several areas of theoretical computer science, for example, in relation to the emptiness problem for tree automata [6, 22] and as an algorithmic formulation of model checking for the modal μ-calculus [7, 41]. Moreover, all of these problems are in NP ∩ coNP, and even UP ∩ coUP [3, 26], so they are unlikely to be NP-complete. However, despite much effort from the community, none of these problems are known to be in P, and whether there exists a polynomial-time algorithm to solve these games is a very important and long-standing open problem.

Strategy improvement is a well-studied method for solving these games [24, 37, 43]. It is an extension of the well-known policy iteration algorithms for solving Markov decision processes. The algorithm selects one of the two players to be the *strategy improver*. Each strategy of the improver has a set of *switchable* edges, and switching any subset of these edges produces a strictly better strategy. So, the algorithm proceeds by first choosing an arbitrary starting strategy, and then in each iteration, switching some subset of the switchable edges. Eventually this process will find a strategy with no switchable edges, and it can be shown that this strategy is an optimal strategy for the improver.

To completely specify the algorithm, a *switching rule* is needed to pick the subset of switchable edges in each iteration (this is analogous to the pivot rule used in the simplex method). Many switching rules have been proposed and studied [9, 23, 30, 31, 38]. One of the most natural rules, and the one that we consider in this paper, is the *all-switches* rule, which always switches a vertex if it has a switchable edge. In particular, we consider *greedy* all-switches, which chooses the best edge whenever more than one edge is switchable at a vertex (ties are broken arbitrarily). For a long time, the all-switches variant of the discrete strategy improvement algorithm of Vöge and Jurdziński [43] was considered the best candidate for a polynomial-time algorithm to solve parity games. Indeed, no example was known that required a super-linear number of iterations. However, these hopes were dashed when Friedmann showed an exponential lower bound for greedy all-switches strategy improvement for parity games [15]. In the same paper, he showed that his result extends to strategy improvement algorithms for discounted games and simple stochastic games.

The computational power of pivot algorithms. In this paper, we follow a recent line of work that seeks to explain the poor theoretical performance of pivoting algorithms using a complexity-theoretic point of view. The first results in this direction were proved for problems in the complexity classes PPAD and PLS. It is know that, if a problem is *tight-PLS-complete* then computing the solution found by the natural improvement algorithm is PSPACE-complete [25]. Similarly, for the canonical PPAD-complete problem End-of-the-Line, computing the solution that is found by the natural line following algorithm is PSPACE-complete [35]. This was extended to show that is PSPACE-complete to compute any of the solutions that can be found by the Lemke-Howson algorithm [21], a pivoting algorithm that solves the PPAD-complete problem of finding a Nash equilibrium of a bimatrix game.

Until recently, results of this type were only known for algorithms for problems that, due to known hardness results, are unlikely to lie in P. However, a recent series of papers has shown that similar results hold even for the simplex method for linear programming. Adler, Papadimitriou and Rubinstein [1] showed that there exists an artificially contrived

pivot rule for which is PSPACE-complete to decide if a given basis is on the path of bases visited by the simplex algorithm with this pivot rule. Disser and Skutella [5] studied the natural pivot rule that Dantzig proposed when he introduced the simplex method, and they showed that it is NP-hard to decide whether a given variable enters the basis when the simplex method is run with this pivot rule. Finally, Fearnley and Savani strengthened both these results by showing that the decision problem that Disser and Skutella considered is actually PSPACE-complete [12], and they also showed that determining if a given variable is used in the *final optimal solution* found by the algorithm is PSPACE-complete. This result exploited a known connection between *single-switch* policy iteration for Markov Decision Processes (MDPs) and the simplex method for a corresponding linear program (LP). The result was first proved for a greedy variant of single-switch policy iteration, which then implied the result for the simplex method with Dantzig's pivot rule.

All of the results on linear programming are motivated by the quest to find a *strongly polynomial* algorithm for linear programming, which was included in Smale's list of great unsolved problems of the 21st century [40]. One way of resolving this problem would be to design a polynomial-time pivot rule for the simplex method, and if we are to do this, then it is crucial to understand why existing pivot rules fail. The PSPACE-completeness indicates that they fail because in fact they can do something far more than is necessary, namely they are capable of solving any problem that can be computed in polynomial space.

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

Our contribution. Our main results are that, for greedy all-switches strategy improvement, determining whether the algorithm switches a given edge is PSPACE-complete, and determining whether the optimal strategy found by the algorithm uses a particular edge is PSPACE-complete. One of the key features that strategy improvement has is the ability to switch multiple switchable edges at the same time, rather than just one as in the simplex method. Our results show that naively using this power does not help to avoid the PSPACE-completeness results that now seem to be endemic among pivoting algorithms. The proof primarily focuses on the strategy improvement algorithm of Vöge and Jurdziński for solving parity games [43]. The following definition formalises the problem that we are interested in.

Definition 1.1. *Let G be a game, and let e be an edge and σ be a strategy profile of G. The problem* EDGESWITCH(G, σ, e) *is to decide if the edge e is ever switched by greedy all-switches strategy improvement when it is applied to G starting from σ.*

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

Theorem 1.2. EDGESWITCH *for Vöge and Jurdziński's algorithm is* PSPACE-*complete.*

We use this theorem to show similar results for other games. For mean-payoff games, our results apply to the gain-bias algorithm [14]; and for discounted and simple stochastic games our results apply to the standard strategy improvement algorithms [4,36]. We utilise the well-known polynomial-time reductions from parity games to mean-payoff games [36], mean-payoff games to discounted games, and from discounted games to simple stochastic games [44]. The parity games we construct have the property that when they are reduced to the other games mentioned above strategy improvement will behave in the same way (for discounted and simple stochastic games this was already observed by Friedmann [15]), so we get the following corollary of Theorem 1.2.

Corollary 1.3. EDGESWITCH *for the gain-bias algorithm, and the standard strategy improvement algorithms for discounted-payoff and simple stochastic games is* PSPACE-*complete.*

Theorem 1.2 proves a property about the path taken by strategy improvement during its computation. Previous results have also studied the complexity of finding the optimal strategy that is produced by strategy improvement, which we encode in the following problem.

Definition 1.4. *Let G be a game, and let e be an edge and σ be a strategy profile of G. The problem* OPTIMALSTRATEGY(G, σ, e) *is to decide if the edge e is used in the optimal strategy that is found by greedy all-switches strategy improvement when applied to G starting from σ.*

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

Theorem 1.5. OPTIMALSTRATEGY *for Vöge and Jurdziński's algorithm is* PSPACE-*complete.*

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

Corollary 1.6. OPTIMALSTRATEGY *for the gain-bias algorithm, and the standard strategy improvement algorithms for discounted-payoff and simple stochastic games is* PSPACE-*complete.*

Our results can also be applied to *unique sink orientations* (USOs). An *orientation* of an n-dimensional hypercube is a function that assigns a direction to each edge of the cube. The *faces* of an n-dimensional cube are the k-dimensional cubes that can be obtained by fixing $n - k$ of the dimensions and letting the other dimensions be free. An orientation is a USO if every face of the cube has a unique sink [42].

Recently, it was shown that recognising a USO is coNP-complete, and that recognising an *acyclic* USO (AUSO) is PSPACE-complete [20]. As we will see, the games that we consider will be guaranteed to induce AUSOs. The fundamental algorithmic problem for USOs is to find the global sink assuming oracle access to the edge orientation. The design and analysis of algorithms for this problem is an active research area [16, 17, 19, 23, 33, 39], in particular for AUSOs. The BOTTOMANTIPODAL algorithm [39] for AUSOs on cubes starts at an arbitrary vertex and in each iteration jumps to the antipodal vertex in the sub-cube spanned by the outgoing edges at the current vertex.

For binary games, where vertices have outdegree at most two, the valuation functions used by strategy improvement induce an AUSO on a cube, and all-switches strategy improvement corresponds to BOTTOMANTIPODAL on this AUSO. For non-binary games we instead get an AUSO on a grid [18], so our results immediately give a PSPACE-hardness result for grid USOs for a problem analogous to EDGESWITCH. To get a similar result for AUSOs on cubes we turn our construction into a binary parity game, and we get the following.

Corollary 1.7. *Let C be a d-dimensional cube AUSO, specified by a $poly(d)$-sized circuit that computes the edge orientations for each vertex of C. Given a dimension $k \in \{1, \ldots, d\}$, and a vertex v, it is PSPACE-complete to decide if BOTTOMANTIPODAL started at v, ever switches the kth coordinate.*

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

Related work. The best known algorithms for mean-payoff, discounted, and simple stochastic games have subexponential running time: the *random facet strategy improvement* algorithms combine strategy improvement with the random-facet algorithm for LPs [30–32]. Following the work of Friedmann [15] that we build on heavily in this paper, Friedmann, Hansen, and Zwick showed a sub-exponential lower bound for the random facet strategy improvement algorithm [16]. They also used a construction of Fearnley [8] to extend the bound to the random facet pivot rule for the simplex method [17].

For parity games, there are several algorithms that perform better than random facet strategy improvement. First, a *deterministic* subexponential-time algorithm was found [28]. Very recent work has improved this even further by producing an algorithm that uses quasi-polynomial time and space [2], and it has subsequently been shown that there are algorithms that use quasi-polynomial time and polynomial space [10, 27].

Roadmap. In Section 2, we give a formal definition of parity games, and more specifically the *one-sink* games used by Friedmann that we also use for our construction. We then give a high-level overview of how all-switches strategy improvement works. Our main reduction starts with an iterated circuit evaluation problem. In Section 3, we describe our main construction of a parity game that will implement iterated circuit iteration when strategy improvement is run on it. In Section 4, we describe the sequence of strategies that all-switches strategy improvement will go through as it implements the iterated circuit evaluation. In Section 5, we show that the construction works as claimed and thus prove that EDGESWITCH is PSPACE-hard for parity games. In Section 6, we show how this result for EDGESWITCH extends to strategy improvement algorithms for other games. In Section 7, we show how to augment our construction with an extra gadget to give PSPACE-hardness results for OPTIMALSTRATEGY. In Section 8, we state some open problems.

2. Preliminaries

2.1. **Parity games.** A parity game is defined by a tuple $G = (V, V_{\text{Even}}, V_{\text{Odd}}, E, \text{pri})$, where (V, E) is a directed graph. The sets V_{Even} and V_{Odd} partition V into the vertices belonging to

player Even, and the vertices belonging to player Odd, respectively. The *priority* function
pri : $V \to \{1, 2, \dots\}$ assigns a positive natural number to each vertex. We make the
standard assumption that there are no terminal vertices, which means that every vertex is
required to have at least one outgoing edge. The strategy improvement algorithm of Vöge
and Jurdziński also requires that we assume, without loss of generality, that every priority
is assigned to at most one vertex.

A strategy for player Even is a function that picks one outgoing edge for each Even
vertex. More formally, a *deterministic positional strategy* for Even is a function $\sigma : V_{\text{Even}} \to V$ such that, for each $v \in V_{\text{Even}}$ we have that $(v, \sigma(v)) \in E$. Deterministic positional
strategies for player Odd are defined analogously. Throughout this paper, we will only
consider deterministic positional strategies, and from this point onwards, we will refer to
them simply as *strategies*. We use Σ_{Even} and Σ_{Odd} to denote the set of strategies for players
Even and Odd, respectively.

A *play* of the game is an infinite path through the game. More precisely, a play is a
sequence $v_0, v_1, \dots$ such that for all $i \in \mathbb{N}$ we have $v_i \in V$ and $(v_i, v_{i+1}) \in E$. Given a pair
of strategies $\sigma \in \Sigma_{\text{Even}}$ and $\tau \in \Sigma_{\text{Odd}}$, and a starting vertex v_0, there is a unique play that
occurs when the game starts at v_0 and both players follow their respective strategies. So,
we define $\text{Play}(v_0, \sigma, \tau) = v_0, v_1, \dots$, where for each $i \in \mathbb{N}$ we have $v_{i+1} = \sigma(v_i)$ if $v_i \in V_{\text{Even}}$,
and $v_{i+1} = \tau(v_i)$ if $v_i \in V_{\text{Odd}}$.

Given a play $\pi = v_0, v_1, \dots$ we define

$$\text{MaxIo}(\pi) = \max\{p \ : \ \exists \text{ infinitely many } i \in \mathbb{N} \text{ s.t. } \text{pri}(v_i) = p\},$$

to be the maximum priority that occurs *infinitely often* along π. We say that a play π
is *winning* for player Even if $\text{MaxIo}(\pi)$ is even, and we say that π is winning for Odd if
$\text{MaxIo}(\pi)$ is odd. A strategy $\sigma \in \Sigma_{\text{Even}}$ is a *winning strategy* for a vertex $v \in V$ if, for
every strategy $\tau \in \Sigma_{\text{Odd}}$, we have that $\text{Play}(v, \sigma, \tau)$ is winning for player Even. Likewise,
a strategy $\tau \in \Sigma_{\text{Odd}}$ is a winning strategy for v if, for every strategy $\sigma \in \Sigma_{\text{Even}}$, we have
that $\text{Play}(v, \sigma, \tau)$ is winning for player Odd. The following fundamental theorem states that
parity games are *positionally determined*.

Theorem 2.1 ([6,34]). *In every parity game, the set of vertices V can be partitioned into
winning sets (W_0, W_1), where Even has a positional winning strategy for all $v \in W_0$, and
Odd has a positional winning strategy for all $v \in W_1$.*

The computational problem that we are interested in is, given a parity game, to determine
the partition (W_0, W_1).

2.2. **Strategy improvement.** We now describe the strategy improvement algorithm of
Vöge and Jurdziński [43] for solving parity games, which will be the primary focus of this
paper.

Valuations. The algorithm assigns a *valuation* to each vertex v under every pair of strate-
gies $\sigma \in \Sigma_{\text{Even}}$ and $\tau \in \Sigma_{\text{Odd}}$. Since both of these strategies are positional, we know that
$\text{Play}(v, \sigma, \tau)$ consists of a finite initial path followed by an infinitely repeated simple cycle.
Let p be the largest priority that is seen infinitely often along $\text{Play}(v, \sigma, \tau)$. Since every
priority is assigned to at most one vertex, there is a unique vertex u with $\text{pri}(u) = p$. We
use this vertex to decompose the play: let $P(v, \sigma, \tau)$ be the finite simple path that starts at
v and ends at u, and let $C(v, \sigma, \tau)$ be the infinitely-repeated cycle that starts at u and ends

at u. We can now define the valuation function $\mathrm{Val}_{\mathrm{VJ}}^{\sigma,\tau}(v) = (p, S, d)$ where p is as above and:

- S is the set of priorities on the finite path that are strictly greater than p:

$$S = \{\mathrm{pri}(u) \ : \ u \in P(v, \sigma, \tau) \text{ and } \mathrm{pri}(u) > p\}.$$

- d is the length of the finite path: $d = |P(v, \sigma, \tau)|$.

We now define an order over valuations. First we define an order $\preceq$ over priorities: we have that $p \prec q$ if one of the following holds:

- p is odd and q is even.
- p and q are both even and $p < q$.
- p and q are both odd and $p > q$.

Furthermore, we have that $p \preceq q$ if either $p \prec q$ or $p = q$.

Next we define an order of the sets of priorities that are used in the second component of the valuation. Let $P, Q \subset \mathbb{N}$. We first define:

$$\mathrm{MaxDiff}(P, Q) = \max\big((P \setminus Q) \cup (Q \setminus P)\big).$$

If $d = \mathrm{MaxDiff}(P, Q)$ then we define $P \sqsubset Q$ to hold if one of the following conditions holds:

- d is even and $d \in Q$.
- d is odd and $d \in P$.

Furthermore, we have that $P \sqsubseteq Q$ if either $P = Q$ or $P \sqsubset Q$.

Finally, we can provide an order over valuations. We have that $(p, S, d) \prec (p', S', d')$ if one of the following conditions holds:

- $p \prec p'$.
- $p = p'$ and $S \sqsubset S'$.
- $p = p'$ and $S = S'$ and p is odd and $d < d'$.
- $p = p'$ and $S = S'$ and p is even and $d > d'$.

Furthermore, we have that $(p, S, d) \preceq (p', S', d')$ if either $(p, S, d) \prec (p', S', d')$ or $(p, S, d) = (p', S', d')$.

Best responses. Given a strategy $\sigma \in \Sigma_{\mathrm{Even}}$, a *best response* against σ is a strategy $\tau^* \in \Sigma_{\mathrm{Odd}}$ such that, for every $\tau \in \Sigma_{\mathrm{Odd}}$ and every vertex v we have $\mathrm{Val}_{\mathrm{VJ}}^{\sigma,\tau}(v) \preceq \mathrm{Val}_{\mathrm{VJ}}^{\sigma,\tau^*}(v)$. Vöge and Jurdziński proved the following properties.

Lemma 2.2 ([43]). *For every $\sigma \in \Sigma_{Even}$ a best response τ^* can be computed in polynomial time.*

We define $\mathrm{Br}(\sigma)$ to be an arbitrarily chosen best response strategy against σ. Furthermore, we define $\mathrm{Val}_{\mathrm{VJ}}^{\sigma}(v) = \mathrm{Val}_{\mathrm{VJ}}^{\sigma,\mathrm{Br}(\sigma)}(v)$.

Switchable edges. Let σ be a strategy and $(v, u) \in E$ be an edge such that $\sigma(v) \neq u$. We say that (v, u) is *switchable* in σ if $\mathrm{Val}_{\mathrm{VJ}}^{\sigma}(\sigma(v)) \prec \mathrm{Val}_{\mathrm{VJ}}^{\sigma}(u)$. Furthermore, we define a *most appealing* outgoing edge at a vertex v to be an edge (v, u) such that, for all edges (v, u') we have $\mathrm{Val}_{\mathrm{VJ}}^{\sigma}(u') \preceq \mathrm{Val}_{\mathrm{VJ}}^{\sigma}(u)$.

There are two fundamental properties of switchable edges that underlie the strategy improvement technique. The first property is that switching any subset of the switchable edges will produce an improved strategy. Let σ be a strategy, and let $W \subseteq E$ be a set of

switchable edges in σ such that, for each vertex v, there is at most one edge of the form $(v, u) \in W$. *Switching W in σ creates a new strategy $\sigma[W]$ where for all v we have:*

$$\sigma[W](v) = \begin{cases} u & \text{if } (v, u) \in W, \\ \sigma(v) & \text{otherwise.} \end{cases}$$

We can now formally state the first property.

Lemma 2.3 ([43]). *Let σ be a strategy and let $W \subseteq E$ be a set of switchable edges in σ such that, for each vertex v, there is at most one edge of the form $(v, u) \in W$. We have:*

- *For every vertex v we have $\mathrm{Val}_{VJ}^{\sigma}(v) \preceq \mathrm{Val}_{VJ}^{\sigma[W]}(v)$.*
- *There exists a vertex v for which $\mathrm{Val}_{VJ}^{\sigma}(v) \prec \mathrm{Val}_{VJ}^{\sigma[W]}(v)$.*

The second property concerns strategies with no switchable edges. A strategy $\sigma \in \Sigma_{\text{Even}}$ is *optimal* if for every vertex v and every strategy $\sigma' \in \Sigma_{\text{Even}}$ we have $\mathrm{Val}_{VJ}^{\sigma'}(v) \preceq \mathrm{Val}_{VJ}^{\sigma}(v)$.

Lemma 2.4 ([43]). *A strategy with no switchable edges is optimal.*

Vöge and Jurdziński also showed that winning sets for both players can be extracted from an optimal strategy. If σ is an optimal strategy, then W_0 contains every vertex v for which the first component of $\mathrm{Val}_{VJ}^{\sigma}(v)$ is even, and W_1 contains every vertex v for which the first component of $\mathrm{Val}_{VJ}^{\sigma}(v)$ is odd. Hence, to solve the parity game problem, it is sufficient to find an optimal strategy.

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

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

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

The algorithm given above does not specify a complete algorithm, because it does not specify *which* subset of switchable edges should be chosen in each iteration. Indeed, there are many variants of the algorithm that use a variety of different *switching rules*. In this paper, we focus on the *greedy all-switches* switching rule. This rule switches every vertex that has a switchable edge, and if there is more than one switchable edge, it arbitrarily picks one of the most appealing edges.

One-sink games. Friedmann observed that, for the purposes of showing lower bounds, it is possible to simplify the Vöge-Jurdziński algorithm by restricting the input to be a *one-sink game* [15].

A one-sink parity game contains a sink vertex s such that $\mathrm{pri}(s) = 1$. An even strategy $\sigma \in \Sigma_{\text{Even}}$ is called a *terminating strategy* if, for every vertex v, the first component of $\mathrm{Val}_{VJ}^{\sigma}(v)$ is 1. This means that, when the opponent plays optimally against σ, the play will terminate in the sink s. Formally, a parity game is a one-sink parity game if:

- There is a vertex $s \in V$ such that $\mathrm{pri}(s) = 1$, and (s,s) is the only outgoing edge from s. Furthermore, there is no vertex v with $\mathrm{pri}(v) = 0$.
- All optimal strategies are terminating.

Now, suppose that we apply the Vöge-Jurdziński algorithm, and furthermore suppose that the initial strategy is terminating. Since the initial and optimal strategies are both terminating, we have that, for every strategy σ visited by the algorithm and every vertex v, the first component of $\mathrm{Val}^{\sigma}_{\mathrm{VJ}}(v) = 1$, and so it can be ignored. Furthermore, since there is no vertex with priority 0, the second component of $\mathrm{Val}^{\sigma}_{\mathrm{VJ}}(v)$ must be different from the second component of $\mathrm{Val}^{\sigma}_{\mathrm{VJ}}(u)$, for every $v, u \in V$. Therefore, the third component of the valuation can be ignored.

Thus, for a one-sink game, we can define a simplified version of the Vöge-Jurdziński algorithm that only uses the second component. So, we define $\mathrm{Val}^{\sigma}(v)$ to be equal to the second component of $\mathrm{Val}^{\sigma}_{\mathrm{VJ}}(v)$, and we carry out strategy improvement using the definitions given above, but with $\mathrm{Val}^{\sigma}(v)$ substituted for $\mathrm{Val}^{\sigma}_{\mathrm{VJ}}(v)$. Note, in particular, that in this strategy improvement algorithm, and edge (v, u) is switchable in σ if $\mathrm{Val}^{\sigma}(\sigma(v)) \sqsubset \mathrm{Val}^{\sigma}(u)$.

In our proofs, we will frequently want to determine the maximum difference between two valuations. For this reason, we introduce the following notation. For every strategy σ, and every pair of vertices $v, u \in V$, we define $\mathrm{MaxDiff}^{\sigma}(v, u) = \mathrm{MaxDiff}(\mathrm{Val}^{\sigma}(v), \mathrm{Val}^{\sigma}(u))$.

2.3. **Circuit iteration problems.** To prove our PSPACE-completeness results, we will reduce from two *circuit iteration problems*, which we now define.

The 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 $1 \le z \le n$.

We use standard notation for function iteration: given a bit-string $B \in \{0, 1\}^n$, we recursively define $F^1(B) = F(B)$, and $F^i(B) = F(F^{i-1}(B))$ for all $i > 1$.

We now define two problems that will be used as the starting point for our reduction. Both are decision problems that take as input a circuit iteration instance (F, B, z).
- BITSWITCH(F, B, z): decide whether there exists an even $i \le 2^n$ such that the z-th bit of $F^i(B)$ is 1.

- CIRCUITVALUE(F, B, z): decide whether the z-th bit of $F^{2^n}(B)$ is 1.

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

The fact that these problems are PSPACE-complete should not be too surprising, because F can simulate a single step of a space-bounded Turing machine, so when F is iterated, it simulates a run of the space-bounded Turing machine. The following result was shown in [12].

Lemma 2.5. *[12, Lemma 7]* BITSWITCH *and* CIRCUITVALUE *are* PSPACE-*complete.*

Circuits. For the purposes of our reduction, we must make some assumptions about the format of the circuits that represent F. Let C be a boolean circuit with n input bits, n output bits, and k gates. We assume, w.l.o.g., that all gates are or-gates or not-gates. The circuit will be represented as a list of gates indexed 1 through $n + k$. The indices 1 through n represent the n *inputs* to the circuit. Then, for each $i > n$, we have:

- If gate i is an or-gate, then we define $I_1(i)$ and $I_2(i)$ to give the indices of its two inputs.
- If gate i is a not-gate, then we define $I(i)$ to give the index of its input.

The gates $k + 1$ through $k + n$ correspond to the n *output bits* of the circuit, respectively. For the sake of convenience, for each input bit i, we define $I(i) = k + i$, which indicates that, if the circuit is applied to its own output, input bit i should copy from output bit $I(i)$. Moreover, we assume that the gate ordering is topological. That is, for each or-gate i we assume that $i > I_1(i)$ and $i > I_2(i)$, and we assume that for each not-gate i we have $i > I(i)$.

For each gate i, let $d(i)$ denote the *depth* of gate i, which is the length of the longest path from i to an input bit. So, in particular, the input bits are at depth 0. Observe that we can increase the depth of a gate by inserting dummy or-gates: given a gate i, we can add an or-gate j with $I_1(j) = i$ and $I_2(j) = i$, so that $d(j) = d(i) + 1$. We use this fact in order to make the following assumptions about our circuits:

- For each or-gate i, we have $d(I_1(i)) = d(I_2(i))$.
- There is a constant c such that, for every output bit $i \in \{k + 1, k + n\}$, we have $d(i) = c$.

From now on, we assume that all circuits that we consider satisfy these properties. Note that, since all outputs gates have the same depth, we can define $d(C) = d(k + 1)$, which is the depth of all the output bits of the circuit.

Given an input bit-string $B \in \{0, 1\}^n$, the output of each gate in C can be determined. We define $\mathrm{Eval}(B, i) = 1$ if gate i is true for input B, and $\mathrm{Eval}(B, i) = 0$ if gate i is false for input B.

Given a circuit C, we define the *negated form* of C to be a transformation of C in which each output bit is negated. More formally, we transform C into a circuit C' using the following operation: for each output bit $n + i$ in C, we add a NOT gate $n + k + i$ with $I(n + k + i) = n + i$.

3. The Construction

Overview. We will show that EDGESWITCH is PSPACE-complete by reducing from the circuit iteration problem BITSWITCH. Figure 1 gives a high level picture of the construction. Given a circuit F that is to be iterated, we create a gadget that is capable of computing F on a given input. Our construction will contain two copies of this gadget, which will be numbered 0 and 1. The two circuits alternate, with the output of one circuit being passed to the input of the other circuit. So, given an initial bit-string B, circuit 0 computes $F(B)$, then circuit 1 computes $F(F(B))$, then circuit 0 computes $F(F(F(B)))$, and so on. The technical reason for having two copies of the circuit is that our circuit gadget cannot handle the input bits being changed before the output bits are read, and so a single circuit gadget cannot feed its own outputs back into its inputs.

Figure 1 also shows the *clocks* which play a fundamental role in driving the construction. Each copy of the circuit is equipped with its own clock, which controls the timing of that circuit. In particular, each clock has two states r and s. Ordinarily, the valuation of r is larger than the valuation of s. Every so often, the clock produces a signal, which is transmitted by the valuation of s being larger than the valuation of r. This signal causes the associated circuit to begin computing based on its current input. Thus, the clocks plays
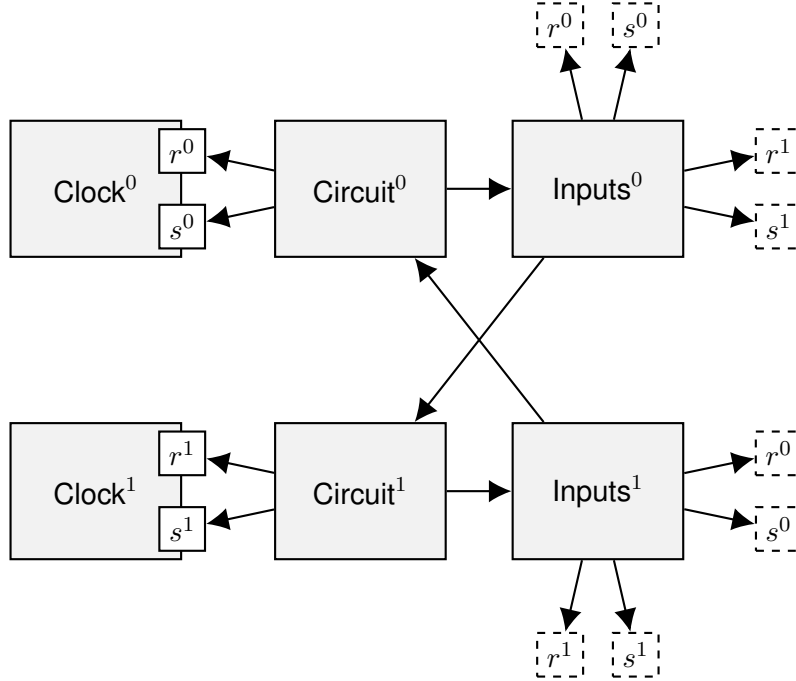
FIGURE 1. High-level overview of our construction. There are two copies of the underlying circuit, and two clocks. The two are synchronized via the nodes r^0, s^0, r^1, and s^1. In this diagram the directions of arrows are consistent with the directed edges in the corresponding parity game.

an important role in synchronising the two circuits, and ensuring that each circuit starts computing only after its partner has finished computing the previous iteration.

Each clock is implemented by a modified version of Friedmann's exponential-time example. Friedmann's examples are designed to force greedy all-switches strategy improvement to mimic a binary counter. The signal sent by the clock occurs when Friedmann's example increments the binary counter to the next number. Our modifications serve only to increase the number of strategy improvement iterations that take place between each increment.

Finally, Figure 1 shows the INPUT/OUTPUT gadgets. These gadgets are responsible for transmitting bit-strings between the two circuits, and so they are the most complex part of the construction. These gadgets have two modes. When they are in *output mode*, they are connected at the outputs of a circuit, where they read and store the outputs. When they are in *input mode* they are connected at the inputs of the other circuit, where they allow the stored bit-string to be read. For this reason, the INPUT/OUTPUT gadgets must be connected to both clocks, so that they are able to transition between the circuits at the correct time.

The reduction. Formally, let (F, B, z) be the input to the circuit iteration problem, and let C be the negated form of the circuit that computes F. Throughout this section, we will use n as the bit-length of B, and $k = |C|$ as the number of gates used in C. We will use OR, NOT, and INPUT/OUTPUT to denote the set of or-gates, not-gates, and input/output-gates, respectively.
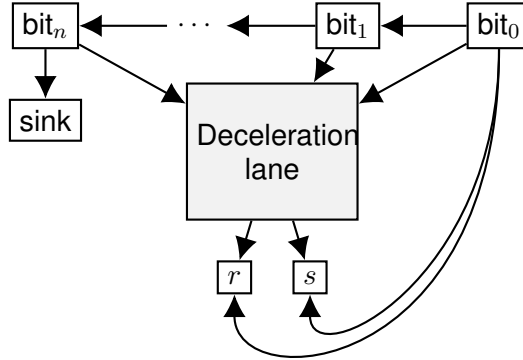
FIGURE 2.   High-level overview of a clock.

In the rest of this section, we describe the construction. We begin by giving an overview of Friedmann's example, both because it plays a key role in our construction, and because the NOT gate gadgets in our circuits are a modification of the bit-gadget used by Friedmann. We then move on to describe the gate gadgets, and how they compute the function F.

Priorities. As we have mentioned, the strategy improvement algorithm that we consider requires that every priority is assigned to at most one vertex. This is unfortunately a rather cumbersome requirement when designing more complex constructions. To help with this, we define a shorthand for specifying priorities. Let $c \in \mathbb{N}$, let $i, l \in \{1, \ldots, |V|\}$, let $j \in \{0, 1, 2\}$, and let $e \in \{0, 1\}$. We define

$$\mathrm{P}(c, i, l, j, e) = 6 \cdot |V|^2 \cdot c + 6 \cdot |V| \cdot i + 6 \cdot l + 2 \cdot j + e.$$

The first four parameters should be thought of as a lexicographic ordering, which determines how large the priority is. The final number e determines whether the priority is odd or even. Note that $\mathrm{P}(c, i, l, j, e)$ is an injective function, so if we ensure that the same set of arguments are never used twice, then we will never assign the same priority to two different vertices. One thing to note is that, since this priority notation is rather cumbersome, it is not possible to use it in our diagrams. Instead, when we draw parts of the construction, we will use *representative* priorities, which preserve the order and parity of the priorities used in the gadgets, but not their actual values.

3.1. **Friedmann's exponential-time example.** In this section, we give an overview of some important properties of Friedmann's exponential-time examples. In particular, we focus on the properties that will be important for our construction. A more detailed description of the example can be found in Friedmann's original paper [15].

A high level view of Friedmann's construction is shown in Figure 2. It works by forcing greedy all-switches strategy improvement to simulate an n bit binary counter. It consists of two components: a *bit gadget* that is used to store one of the bits of the counter, and a *deceleration lane* that is used to ensure that the counter correctly moves from one bit-string to the next.
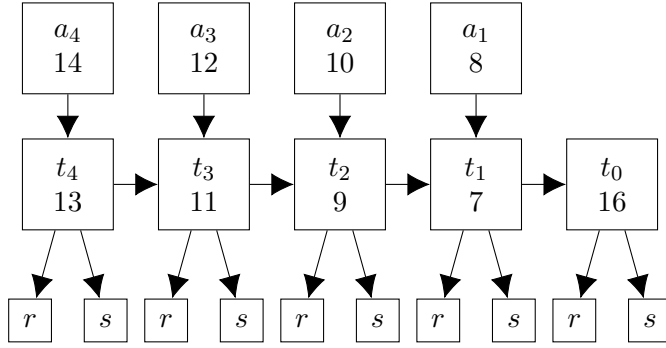
FIGURE 3. Friedmann's deceleration lane of length 4. Each t_i vertex with $i > 0$ has an odd priority, while each a_i vertex has an even priority that is equal to $\mathrm{pri}(t_i) + 1$. The vertex t_0 has an even priority that is larger than any of the priorities assigned to the a_i vertices.

The deceleration lane. Friedmann's example contains one copy of the deceleration lane. The deceleration lane has a specified length m, and Figure 3 shows an example of a deceleration lane of length 4. Friedmann's construction contains one copy of the deceleration lane of length $2n$. Remember that our diagrams use representative priorities, which preserve the order and parity of the priorities used, but not their values.

A key property of the deceleration lane is that greedy all-switches strategy improvement requires m iterations to find the optimal strategy. Consider an initial strategy in which each vertex t_i uses the edge to r, and that the valuation of r is always larger than the valuation of s. First note that, since there is a large even priority on t_0, the optimal strategy is for every vertex t_i, with $i \geq 1$, to use the edge to t_{i-1}. However, since the vertices t_i with $i \geq 1$ are all assigned odd priorities, in the initial strategy only the edge from t_1 to t_0 is switchable. Furthermore, once this edge has been switched, only the edge from t_2 to t_1 is switchable. In this way, the gadget ensures that m iterations are required to move from the initial strategy to the optimal strategy for this gadget.

Another important property is that the gadget can be *reset*. This is achieved by having a single iteration in which the valuation of s is much larger than the valuation of r, followed by another iteration in which the valuation of r is much larger than the valuation of s. In the first iteration all vertices t_i switch to s, and in the second iteration all vertices switch back to r. Note that after the second iteration, we have arrived back at the initial strategy described above.

The bit gadget. The bit gadget is designed to store one bit of a binary counter. The clock construction will contain n copies of this gadget, which will be indexed 1 through n. Figure 4 gives a depiction of a bit gadget with index i.

The current value of the bit for index i is represented by the choice that the current strategy makes at d_i. More precisely, for every strategy σ we have:

- If $\sigma(d_i) = e_i$, then bit i is 1 in σ.
- If $\sigma(d_i) \neq e_i$, then bit i is 0 in σ.

The Odd vertex e_i plays a crucial role in this gadget. If $\sigma(d_i) = e_i$, then Odd's best response is to use edge (e_i, h_i), to avoid creating the even cycle between d_i and e_i. On the other
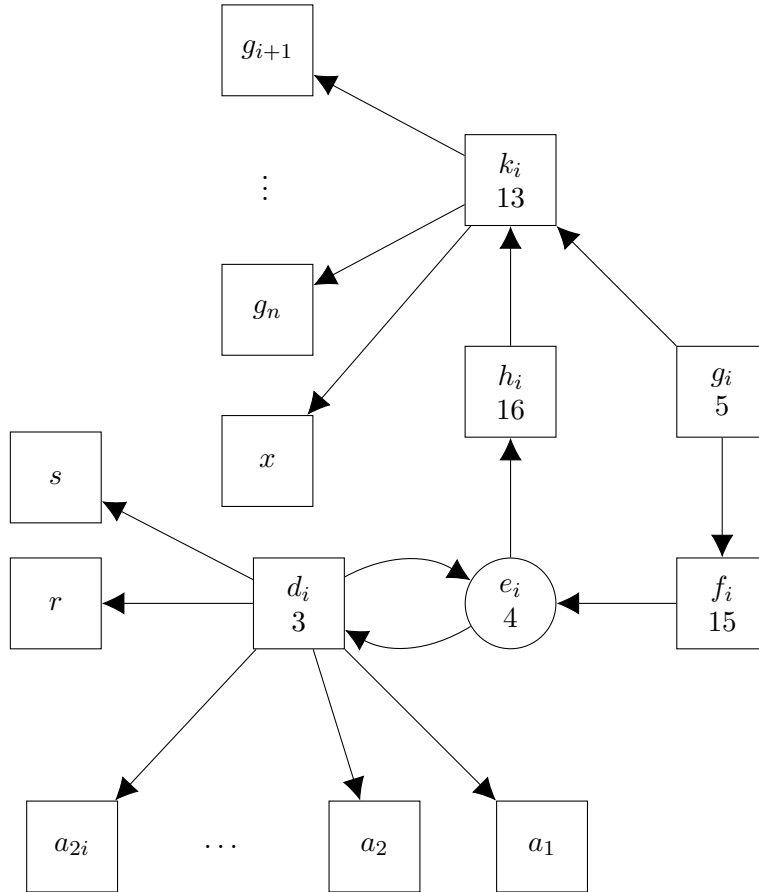
FIGURE 4. An example of Friedman's bit gadget. The vertex d_i has a small odd priority, while the vertex e_i has an even priority that is equal to $\mathrm{pri}(d_i) + 1$. The vertex f_i has a large odd priority, while the vertex h_i has an large even priority that is equal to $\mathrm{pri}(h_i)+1$. The priorities assigned to k_i and g_i are not relevant to the operation of Friedmann's construction.

hand, if $\sigma(d_i) \neq e_i$, then Odd's best response is to use (e_i, d_i), to avoid seeing the large even priority at h_i.

One thing to note is that, in the case where $\sigma(d_i) \neq e_i$, the edge to e_i is always switchable. To prevent d_i from immediately switching to e_i, we must ensure that there is always a more appealing outgoing edge from e_i, so that the greedy all-switches rule will switch that edge instead. The edges from d_i to the deceleration lane provide this. Once t_1 has switched to t_0, the edge from d_i to a_1 becomes more appealing than the edge to e_i, once t_2 has switched to t_1, the edge from d_i to a_2 becomes more appealing than the edge to e_i, and so on. In this way, we are able to prevent d_i from switching to e_i for $2i$ iterations by providing outgoing edges to the first $2i$ vertices of the deceleration lane.

The vertices s and r. The vertex s has outgoing edges to every vertex f_i in the bit gadgets, and the vertex r has outgoing edges to every vertex g_i in the bit gadgets. If i is the index of the least significant 1 bit, then s chooses the edge to f_i and r chooses the edge

to g_i. The priority assigned to r is larger than the priority assigned to s, which ensures that the valuation of r is usually larger than the valuation of s, as required to make the deceleration lane work.

When the counter moves from one bit-string to the next, the index of the least significant 1 changes to some $i' \neq i$. The vertex s switches to $f_{i'}$ one iteration before the vertex r switches to $g_{i'}$. This creates the single iteration in which the valuation of s is larger than the valuation of r, which resets the deceleration lane.

Simulating a binary counter. To simulate a binary counter, we must do two things. Firstly, we must ensure that if the counter is currently at some bit-string $K \in \{0, 1\}^n$, then the *least significant zero* in K must be flipped to a one. Secondly, once this has been done, all bits whose index is smaller than the least significant zero must be set to 0. If these two operations are always performed, then strategy improvement will indeed count through all binary strings.

The least significant zero is always flipped because each bit i has $2i$ edges to the deceleration lane. Since the purpose of the deceleration lane is to prevent the vertex d_i switching to e_i, the vertex $d_{i'}$ where i' is the index of the least significant zero, is the first to run out of edges, and subsequently switch to $e_{i'}$.

Once this has occurred, all bits with index smaller than the least significant zero are set to 0 due to the following chain of events. The vertex s switches $f_{i'}$, and then the vertex $d_{i''}$ in all bits with index $i'' < i'$ will be switched to s. Since $d_{i''}$ no longer uses the edge to $e_{i''}$, the bit has now been set to 0.

Our modifications to Friedmann's example. In order to use Friedmann's example as a clock, we make a few minor adjustments to it. Firstly, we make the deceleration lane longer. Friedmann's example uses a deceleration lane of length $2n$, but we use a deceleration lane of length $2k + 4n + 6$. Furthermore, while the vertex d_i has outgoing edges to each a_j with $j \leq 2i$ in Friedmann's version, in our modified version the vertex d_i has outgoing edges to each a_j with $j \leq 2i + 2k + 2n + 6$.

The reason for this is that Friedmann's example can move from one bit-string to the next in as little as four iterations, but we need more time in order to compute the circuit F. By making the deceleration lane longer, we slow down the construction, and ensure that there are at least $2k + 2n + 6$ iterations before the clock moves from one bit-string to the next.

The second change that we make is to change the priorities, because we need to make room for the gadgets that we add later. However, we have not made any fundamental changes to the priorities: the ordering of priorities between the vertices and their parity is maintained. We have simply added larger gaps between them.

The following table specifies the version of the construction that we use. Observe that two copies are specified: one for $j = 0$ and the other for $j = 1$. Furthermore, observe that the vertex x will be the sink in our one-sink game.
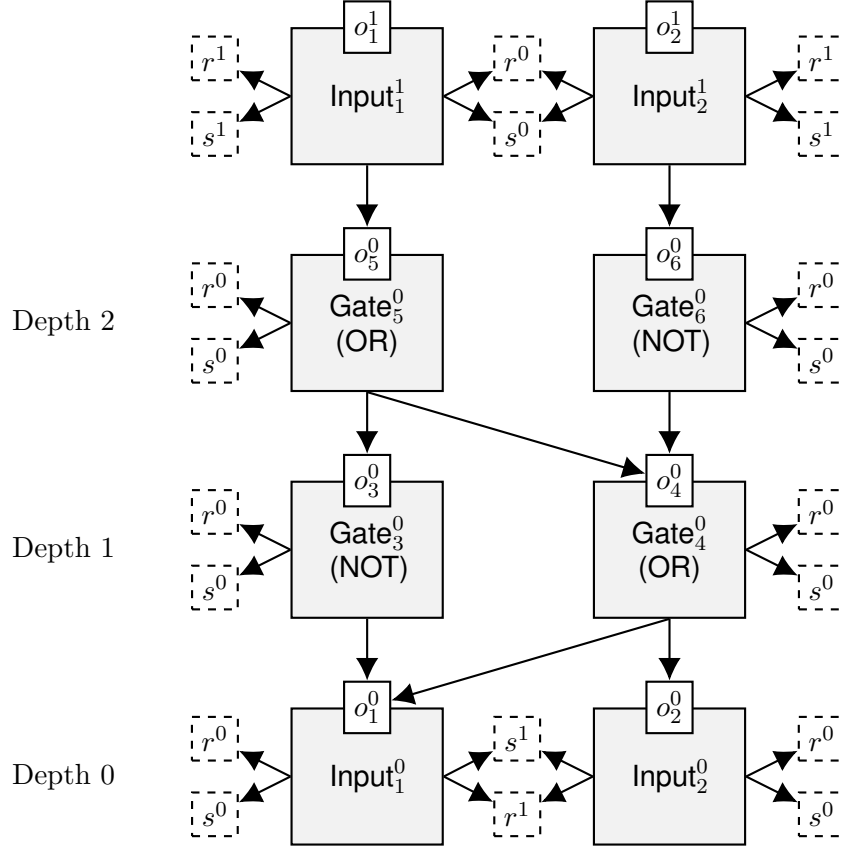
FIGURE 5. Example of how we implement a specific circuit with three gates.

Vertex	Conditions	Edges	Priority	Player
t_0^j	$j \in \{0,1\}$	r^j, s^j	$P(2,0,2k+4n+4,j,0)$	Even
t_l^j	$j \in \{0,1\}$, $1 \le l \le 2k+4n+6$	r^j, s^j, t_{l-1}^j	$P(2,0,l,j,1)$	Even
a_l^j	$j \in \{0,1\}$, $1 \le l \le 2k+4n+6$	t_l^j	$P(2,0,l+1,j,0)$	Even
d_i^j	$j \in \{0,1\}, 1 \le i \le n$	e_i^j, s^j, r^j, a_l^j for $1 \le l \le 2k+2n+6+2i$	$P(1,i,0,j,1)$	Even
e_i^j	$j \in \{0,1\}, 1 \le i \le n$	h_i^j, d_i	$P(1,i,1,j,0)$	Odd
g_i^j	$j \in \{0,1\}, 1 \le i \le n$	f_i^j	$P(1,i,2,j,1)$	Even
k_i^j	$j \in \{0,1\}, 1 \le i \le n$	x, g_l^j, for $i < l \le n$	$P(8,i,0,j,1)$	Even
f_i^j	$j \in \{0,1\}, 1 \le i \le n$	e_i^j	$P(8,i,1,j,1)$	Even
h_i^j	$j \in \{0,1\}, 1 \le i \le n$	k_i^j	$P(8,i,2,j,0)$	Even
s^j	$j \in \{0,1\}$	x, f_l^j for $1 \le l \le n$	$P(7,0,0,j,0)$	Even
r^j	$j \in \{0,1\}$	x, g_l^j for $1 \le l \le n$	$P(7,0,1,j,0)$	Even
x		x	$P(0,0,0,0,1)$	Even

3.2. **Our construction.**

Circuits. Given a circuit, we will produce a gadget that simulates that circuit. An example is given in Figure 5. For each gate in the circuit, we design a gadget that computes the output of that gate. The idea is that greedy all-switches strategy improvement will compute these gates in depth order. Starting from an initial strategy, the first iteration will compute the outputs for all gates of depth 1, the next iteration will use these outputs to compute the outputs for all gates of depth 2, and so on. In this way, after k iterations of strategy improvement, the outputs of the circuit will have been computed. We then use one additional iteration to store these outputs in an input/output gadget.

Strategy improvement valuations will be used to represent the output of each gate. Each gate i has a state o_i^j, and the valuation of this state will indicate whether the gate evaluates to true or false. In particular the following rules will be followed.

Property 3.1. *In every strategy σ we have the following properties.*
  (1) *Before the gate has been evaluated, we will have $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.*
  (2) *If the gate has been evaluated to false, we will continue to have $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.*
  (3) *If the gate has been evaluated to true, then we will instead have $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(o_i^j)$, and $\mathrm{MaxDiff}^\sigma(r^j, o_i^j)$ will be a large even priority.*

The input/output gadgets are connected to both circuits, and these gadgets have two modes.
  (1) When circuit j is computing, the gadget is in *output mode*, where it reads the output of circuit j and stores it.
  (2) When circuit $1 - j$ is computing, the gadget is in *input mode*, where it outputs the value that was stored from the previous computation into circuit $1 - j$.

Therefore, the gates of depth 1 in circuit j read their input from the input/output gadgets in circuit $1-j$, while the input/output gadgets in circuit j read their input from the outputs of circuit j. To formalise this, we introduce the following notation. For every NOT-gate, we define $\mathrm{InputState}(i, j)$ as follows:

$$\mathrm{InputState}(i, j) = \begin{cases} o_{I(i)}^{1-j} & \text{if } d(i) = 1, \\ o_{I(i)}^{j} & \text{if } d(i) > 1. \end{cases}$$

For every OR-gate, and every $l \in \{1, 2\}$, we define $\mathrm{InputState}(i, j, l)$ as follows:

$$\mathrm{InputState}(i, j, l) = \begin{cases} o_{I_l(i)}^{1-j} & \text{if } d(i) = 1, \\ o_{I_l(i)}^{j} & \text{if } d(i) > 1. \end{cases}$$

The clocks. As we have mentioned, we use two copies of Friedmann's example to act as clocks in our construction. These clocks will be used to drive the computation. In particular, the vertices r^j and s^j will play a crucial role in synchronising the two circuits. As described in the previous section, when the clock *advances*, i.e., when it moves from one bit-string to the next, there is a single iteration in which the valuation of s^j is much larger than the valuation of r^j. This event will trigger the computation.

  • The iteration in which the valuation of s^0 is much larger than the valuation of r^0 will trigger the start of computation in circuit 0.
  • The iteration in which the valuation of s^1 is much larger than the valuation of r^1 will trigger the start of computation in circuit 1.

In order for this approach to work, we must ensure that the two clocks are properly synchronised. In particular, the gap between computation starting in circuit j and computation starting in circuit $1 - j$ must be at least $k + 3$, to give enough time for circuit j to compute the output values, and for these values to be stored. We now define notation for this purpose. First we define the number of iterations that it takes for a clock to move from bit-string K to $K + 1$. For every bit-string $K \in \{0, 1\}^n$, we define $\mathrm{Lsz}(K)$ to be the index of the *least significant zero* in K: that is, the smallest index i such that $K_i = 0$. For each $K \in \{0, 1\}^n$, we define:

$$\mathrm{Length}(K) = \Big(2k + 2n + 6\Big) + 2\,\mathrm{Lsz}(K) + 5.$$

This term can be understood as the length of the deceleration lane to which all bits in the clock have edges, plus the number of extra iterations it takes to flip the least-significant zero, plus five extra iterations needed to transition between the two bit-strings.

Next we introduce the following *delay* function, which gives the amount of time each circuit spends computing. For each $j \in \{0, 1\}$ and each $K \in \{0, 1\}^n$, we define:

$$\mathrm{Delay}(j, K) = \begin{cases} \Big(d(C) + 3\Big) + 2n & \text{if } j = 0, \\ \Big(d(C) + 3\Big) + 2 \cdot \mathrm{Lsz}(K) + 5 & \text{if } j = 1. \end{cases}$$

Circuit 1 starts computing $\mathrm{Delay}(0, K)$ iterations after Circuit 0 started computing, and Circuit 0 starts computing $\mathrm{Delay}(1, K)$ iterations after circuit 1 started computing. Observe that $\mathrm{Delay}(0, K) + \mathrm{Delay}(1, K) = \mathrm{Length}(K)$, which ensures that the two circuits do not drift relative to each other. The term $d(C) + 3$ in each of the delays ensures that there is always enough time to compute the circuit, before the next circuit begins the subsequent computation.

Or gates. The gadget for a gate $i \in \mathrm{OR}$ is quite simple, and is shown in Figure 6. It is not difficult to verify that the three rules given in Property 3.1 hold for this gate. Before both inputs have been evaluated, the best strategy at o_i^j is to move directly to r^j, since the valuation of both inputs is lower than the valuation of r^j. Note that in this configuration the valuation of o_i^j is smaller than the valuation of r^j, since o_i^j has been assigned an odd priority.

Since, by assumption, both inputs have the same depth, they will both be evaluated at the same time. If they both evaluate to false, then nothing changes and the optimal strategy at o_i^j will still be r^j. This satisfies the second rule. On the other hand, if at least one input evaluates to true, then the optimal strategy at o_i^j is to switch to the corresponding input states. Since the valuation of this input state is now bigger than r^j, the valuation of o_i^j will also be bigger than r^j, so the third rule is also satisfied.

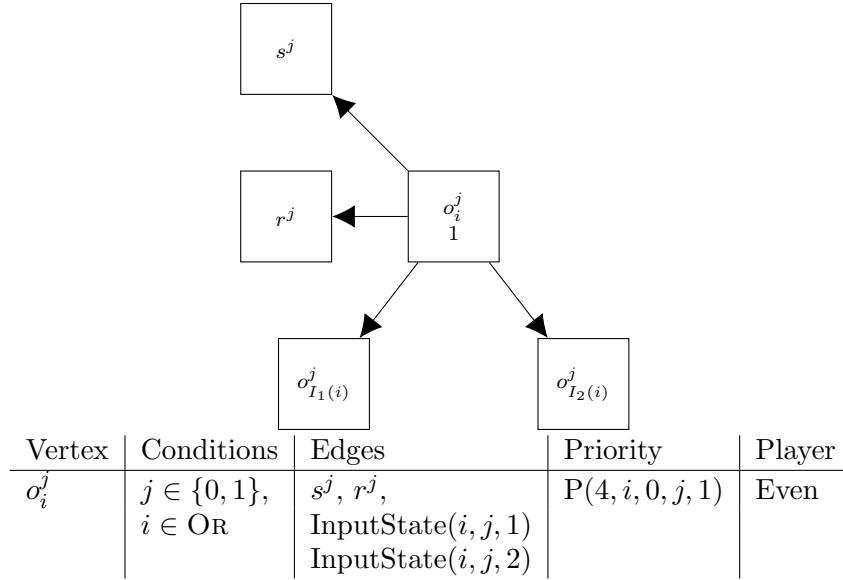| Vertex	Conditions	Edges	Priority	Player
o_i^j	$j \in \{0, 1\}$, $i \in \text{OR}$	s^j, r^j, $\text{InputState}(i, j, 1)$ $\text{InputState}(i, j, 2)$	$\text{P}(4, i, 0, j, 1)$	Even

FIGURE 6. The OR gate.

Not gates. The construction for a gate $i \in \text{NOT}$ is more involved. The gadget is quite similar to a bit-gadget from Friedmann's construction. However, we use a special *modified deceleration lane*, which is shown in Figure 7.

The modified deceleration lane is almost identical to Friedmann's deceleration lane, except that state $t_{i,d(i)}^j$ is connected to the output state of the input gate. The idea is that, for the first $d(i) - 1$ iterations the deceleration lane behaves as normal. Then, in iteration $d(i)$, the input gate is evaluated. If it evaluates to true then the valuation of $t_{i,d(i)}^j$ will be large, and the deceleration lane continues switching as normal. If it evaluates to false, then the valuation of $t_{i,d(i)}^j$ will be low, and the deceleration lane will stop switching.

The NOT gate gadget, which is shown in Figure 8 is a simplified bit gadget that is connected to the modified deceleration lane. As in Friedmann's construction, the strategy
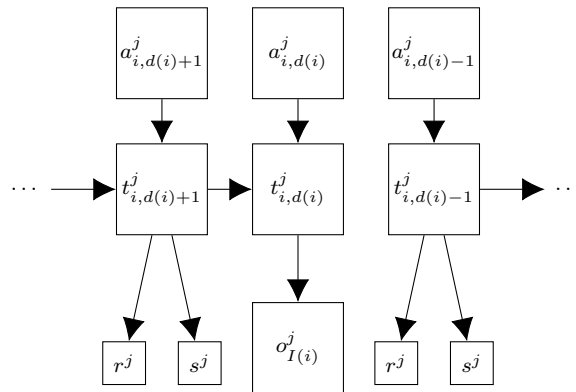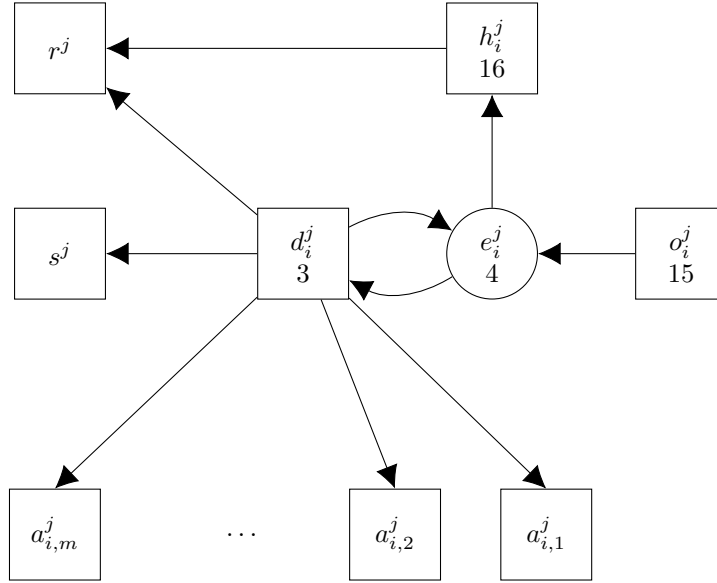

FIGURE 7. Modified deceleration lane for a NOT gate i in circuit j.

FIGURE 8. NOT gate with index i in circuit j.

chosen at d_i^j will represent the output of the gate. In a strategy σ, the gate outputs 1 if $\sigma(d_i^j) = e_i^j$, and it outputs 0 otherwise. As we know, Friedmann's bit gadget is distracted from switching d_i^j to e_i^j by the deceleration lane. By using the modified deceleration lane, we instead obtain a NOT gate. Since the deceleration lane keeps on switching if and only if the input gate evaluates to true, the state d_i^j will switch to e_i^j in iteration $d(i)$ if and only if the input gate evaluates to false. This is the key property that makes the NOT gate work.

To see that the three rules specified in Property 3.1 are respected, observe that there is a large odd priority on the state o_i^j, and an even larger even priority on the state h_i^j. This causes the valuation of o_i^j to only be larger than the valuation of r^j if and only if d_i^j chooses the edge to e_i^j, which only happens when the gate evaluates to true.

Finally, when the computation in circuit j begins again, the NOT-gate is reset. This is ensured by giving the vertex d_i^j edges to both s^j and r^j. So, when the clock for circuit j advances, no matter what strategy is currently chosen, the vertex d_i^j first switches to s^j, and then to r^j, and then begins switching to the deceleration lane.

The following table formally specifies the NOT gate gadgets that we use in the construction.
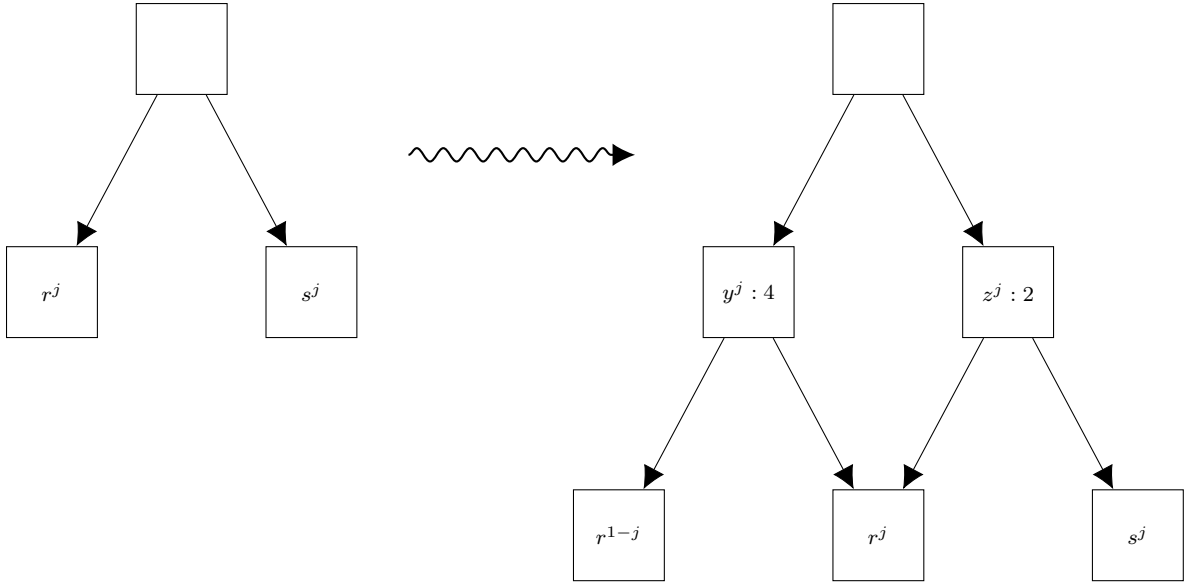
FIGURE 9. Circuit mover gadget for circuit j. Left: the edges to r^j and s^j in a vertex from a NOT-gate. Right: the replacement for these outgoing edges.

Vertex	Conditions	Edges	Priority	Player
$t_{i,0}^j$	$j \in \{0,1\}$, $i \in$ NOT	r^j, s^j	$P(5,i,2k+4n+4,j,0)$	Even
$t_{i,l}^j$	$j \in \{0,1\}$, $i \in$ NOT, $1 \le l \le 2k+4n+6$, and $l \ne d(i)$	r^j, s^j, $t_{i,l-1}^j$	$P(5,i,l,j,1)$	Even
$t_{i,d(i)}^j$	$j \in \{0,1\}$, $i \in$ NOT	InputState(i,j)	$P(5,i,d(i),j,1)$	Even
$a_{i,l}^j$	$j \in \{0,1\}$, $i \in$ NOT, $1 \le l \le 2k+4n+6$, and $l \ne d(i)$	$t_{i,l}$	$P(5,i,l+1,j,0)$	Even
$a_{i,d(i)}^j$	$j \in \{0,1\}$, $i \in$ NOT	$t_{i,d(i)}$	$P(4,i,0,j,0)$	Even
d_i^j	$j \in \{0,1\}$, $i \in$ NOT	s^j, r^j, e_i^j, $a_{i,l}^j$ for $1 \le l \le 2k+4n+6$	$P(4,i,0,j,1)$	Even
e_i^j	$j \in \{0,1\}$, $i \in$ NOT	h_i^j, d_i^j	$P(4,i,1,j,0)$	Odd
o_i^j	$j \in \{0,1\}$, $i \in$ NOT	e_i^j	$P(6,i,0,j,1)$	Even
h_i^j	$j \in \{0,1\}$, $i \in$ NOT	r^j	$P(6,i,1,j,0)$	Even

Input/output gates. For each input-bit in each copy of the circuit, have an input/output gate. Recall that these gadgets have two modes. When Circuit j is computing, the input/output gadgets in circuit j are in output mode, in which they store the output of the circuit, and the input/output gadgets in circuit $1 - j$ are in input mode, in which they output the value that was stored in the previous computation.

At its core, the input/output gadget is simply another copy of the NOT-gate gadget that is connected to the ith output bit of circuit j. However, we modify the NOT-gate gadget by adding in extra vertices that allow it to be *moved* between the two circuits. The most important part of this circuit mover apparatus is shown in Figure 9: all of the vertices
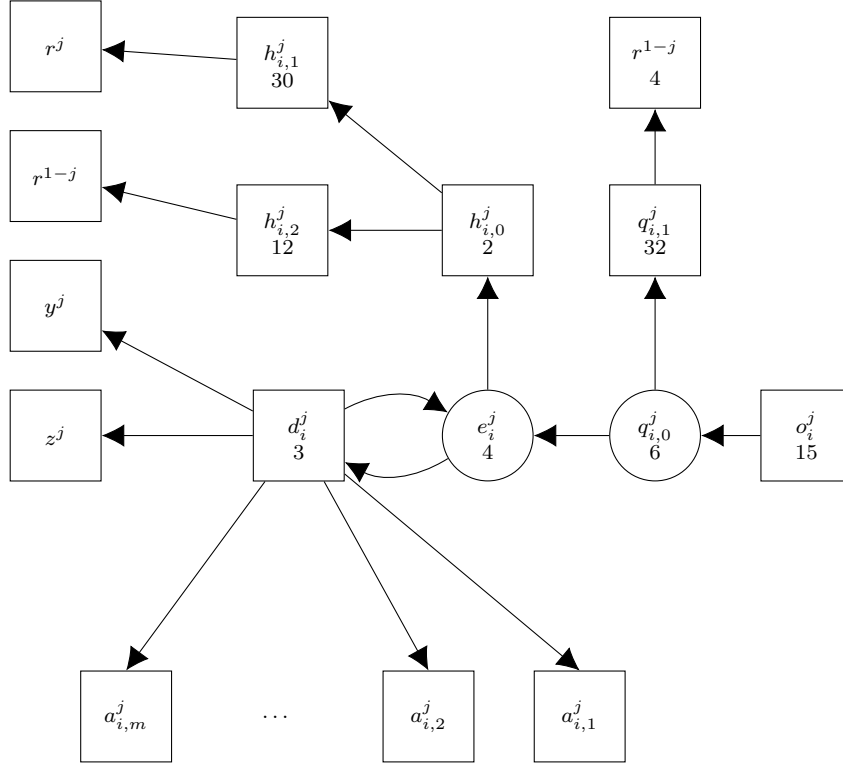
FIGURE 10. INPUT/OUTPUT gate with index i in circuit j.

in the NOT-gate gadget that have edges to r^j and s^j are modified so that they instead have edges to y^j and z^j. Figures 10 and 11 show the INPUT/OUTPUT gadget and its associated modified deceleration lane, respectively. There are three differences between this gadget and the NOT gate, which are the inclusion of the vertices $h_{i,\star}^j$ and $q_{i,\star}^j$ (shown in Figure 10), and the vertices p_i^j and $p_{i,1}^j$ (shown in Figure 11). All of these vertices are involved in the operation of moving the gadget between the two circuits.

When the gadget is in output mode, the vertex y^j chooses the edge to r^j, the vertex $h_{i,0}^j$ chooses the edge to $h_{i,1}^j$, and the vertex p_i^j chooses the edge to $o_{I(i)}^j$. When these edges are chosen, the gadget is essentially the same as a NOT-gate at the top of the circuit. So, once the circuit has finished computing, the vertex d_i^j chooses the edge to e_i^j (i.e., the stored bit is 1) if and only if the ith output from the circuit was a 0. Since the circuit was given in negated form, the gadget has therefore correctly stored the ith bit of $F(B)$.

Throughout the computation in circuit j, the valuation of r^j is much larger than the valuation of r^{1-j}. The computation in circuit $1-j$ begins when the clock in circuit $1-j$ advances, which causes the valuation of r^{1-j} to become much larger than the valuation of r^j. When this occurs, the input/output gate then transitions to input mode. The transition involves the vertex y^j switching to r^{1-j}, the vertex $h_{i,0}^j$ switching to $h_{i,2}^j$, and the vertex p_i^j switching to $p_{i,1}^j$. Moreover, the player Odd vertex $q_{i,0}^j$ switches e_i^j. This vertex acts as a circuit breaker, which makes sure that the output of the gadget is only transmitted to circuit $1-j$ when the gadget is in input mode.
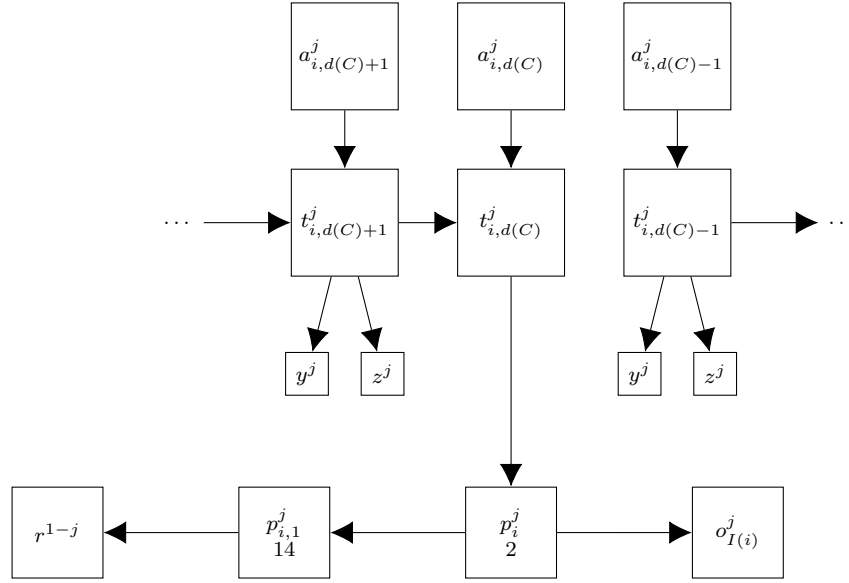
FIGURE 11. Modified deceleration lane for an INPUT/OUTPUT gate i in circuit j.

The key thing is that all of these switches occur *simultaneously* in the same iteration. Since strategy improvement only cares about the *relative* difference between the outgoing edges from the vertex, and since all edges leaving the gadget switch at the same time, the operation of the NOT-gate is not interrupted. So, the strategy chosen at d_i^j will continue to hold the ith bit of $F(B)$, and the gadget has transitioned to input mode.

When the gadget is in input mode, it can be viewed as a NOT at the bottom of circuit $1 - j$ that has already been computed. In particular, the switch from $h_{i,1}^j$ to $h_{i,2}^j$ ensures that, if the output is 1, then the gadget has the correct output priority. Moreover, the deceleration lane has enough states to ensure that, if the output is 0, then output of the gadget will not flip from 0 to 1 while circuit $1 - j$ is computing.

Finally, once circuit $1 - j$ has finished computing, the clock for circuit j advances, and the input/output gadget moves back to output mode. This involves resetting the NOT gate gadget back to its initial state. This occurs because, when the clock in circuit j advances, there is a single iteration in which the valuation of s^j is higher than the valuation of r^j. This causes z^j to switch to s^j which in turn causes a single iteration in which the valuation of z^j is higher than the valuation of y^j. Then, in the next iteration the vertex y^j switches to r^j, and so the valuation of y^j is then larger than the valuation of z^j. So, the valuations of y^j and z^j give exactly the same sequence of events as r^j and s^j, which allows the NOT-gate to reset.

The following table specifies the input/output gadget.

Vertex	Conditions	Edges	Priority	Player
y^j	$j \in \{0,1\}$	r^{1-j}, r^j	$P(3,0,1,j,0)$	Even
z^j	$j \in \{0,1\}$	r^j, s^j	$P(3,0,0,j,0)$	Even
$t_{i,0}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	y^j, z^j	$P(5,i,2k+4n+4,j,0)$	Even
$t_{i,l}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT},$ $1 < l \le 2k+4n+6,$ and $l \neq d(C)$	$y^j, z^j, t_{i,l-1}^j$	$P(5,i,l,j,1)$	Even
$t_{i,d(C)}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	p_i^j	$P(5,i,d(C),j,1)$	Even
p_i^j	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	$o_{I(i)}^j, p_{i,1}^j$	$P(3,i,2,j,0)$	Even
$p_{i,1}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	r^{1-j}	$P(5,i,2k+4n+5,j,0)$	Even
$a_{i,l}^j$	$j \in \{0,1\}$ $i \in \text{INPUT/OUTPUT},$ $1 \le l \le 2k+4n+6$	$t_{i,l}$	$P(5,i,l+1,j,0)$	Even
d_i^j	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	$y^j, z^j, e_i^j, a_{i,l}^j$ for $1 \le l \le 2k+4n+6$	$P(4,i,0,j,1)$	Even
e_i^j	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	$h_{i,0}^j, d_i^j$	$P(4,i,1,j,0)$	Odd
$q_{i,0}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	$e_i^j, q_{i,1}^j$	$P(4,i,2,j,0)$	Odd
$q_{i,1}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	r_i^{1-j}	$P(6,d(C)+2,0,j,0)$	Even
o_i^j	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	$q_{i,0}^j$	$P(6,i,0,j,1)$	Even
$h_{i,0}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	$h_{i,1}^j, h_{i,2}^j$	$P(3,i,3,j,0)$	Even
$h_{i,1}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	r^j	$P(6,d(C)+1,1,j,0)$	Even
$h_{i,2}^j$	$j \in \{0,1\}, i \in \text{INPUT/OUTPUT}$	r^{1-j}	$P(6,0,1,j,0)$	Even

One-sink game. If we are to use the simplified strategy improvement algorithm, we must first show that this construction is a one-sink game. We do so in the following lemma.

Lemma 3.2. *The construction is a one-sink game.*

Proof. In order to show that the construction is a one-sink game, we must show that the two required properties hold. Firstly, we must show that there is a vertex the satisfies the required properties of a sink vertex. It is not difficult to verify that vertex x does indeed satisfy these properties: the only outgoing edge from x is the edge (x, x), and we have $\text{pri}(x) = P(0,0,0,0,0) = 1$. Furthermore, no vertex is assigned priority 0.

Secondly, we must argue that all optimal strategies are terminating. Recall that a terminating strategy has the property that the first component of the Vöge-Jurdziński valuation is 1, which implies that all paths starting at all vertices eventually arrive at the sink x. So, consider a strategy σ that is not terminating, and let v be a vertex such that the first component of Val_{VJ}^σ is strictly greater than 1. Let C be the cycle that is eventually reached by following σ and $\text{Br}(\sigma)$ from v. There are two cases to consider:

- If C contains at least one vertex from a clock, then C must be entirely contained within that clock, because there are no edges that leave either of the two clocks. In this case we have that σ is not optimal because Friedmann has shown that his construction is a one-sink game.
- If C does not contain a vertex from a clock, then it is entirely contained within the circuits. First observe that C cannot be a two-vertex cycle using the vertices d_i^j and e_i^j, because it is not a best-response for Odd allow a cycle with an even priority to be formed, since he can always move to r^j, and from there eventually reach a cycle with priority $p \preceq 1$ (because the clock is a one-sink game). But, the only other way

to form a cycle in the circuits is to pass through both of the circuits. In this case, the highest priority on the cycle will be an odd priority assigned to the state o_i^j in either a NOT-gate (if there is one on the path), or an input/output gate (otherwise). Since this odd priority is strictly greater than 1, and since player Even can always assure a priority of 1 by, for example, moving to r^j in every input/output state d_i^j, we have that σ is not an optimal strategy.

Therefore, we have shown that the construction is a one-sink game. □

4. STRATEGIES

In this section, we define an initial strategy, and describe the sequence of strategies that greedy all-switches strategy improvement switches through when it is applied to this initial strategy. We will define strategies for each of the gadgets in turn, and then combine these into a full strategy for the entire construction.

It should be noted that we will only define *partial strategies* in this section, which means that some states will have no strategy specified. This is because our construction will work no matter which strategy is chosen at these states.

To deal with this, we must define what it means to apply strategy improvement to a partial strategy. If χ is a partial strategy and $\sigma \in \Sigma_{\text{Even}}$ is a strategy, then we say that σ *agrees* with χ if $\sigma(v) = \chi(v)$ for every vertex $v \in V$ for which χ is defined. So, if χ_1 and χ_2 are partial strategies, then we say that greedy all-switches strategy improvement switches χ_1 to χ_2 if, for every strategy $\sigma_1 \in \Sigma_{\text{Even}}$ that agrees with χ_1, greedy all-switches strategy improvement switches σ to a strategy σ_2 that agrees with χ_2.

We now describe the sequence of strategies. Each part of the construction will be considered independently.

The clock. We start by defining the sequence of strategies that occurs in the two clocks. For each *clock bit-string* $K \in \{0, 1\}^n$, we define a sequence of strategies $\kappa_1^K, \kappa_2^K, \ldots, \kappa_{\text{Length}(K)}^K$. Greedy all-switches strategy improvement switches through each of these strategies in turn, and then switches from $\kappa_{\text{Length}(K)}^K$ to κ_1^{K+1}, where $K + 1$ denotes the bit-string that results by adding 1 to the integer represented by K. The sequence begins in the first iteration after the valuation of s^j is larger than the valuation of r^j. We will first present the building blocks of this strategy, and the combine the building blocks into the full sequence.

We begin by considering the vertices t_l^j in the deceleration lane. Recall that these states switch, in sequence, from r^j to t_{l-1}^j. This is formalised in the following definition. For each $m \geq 1$, each l in the range $1 \leq l \leq 2k + 4n + 6$, and each $j \in \{0, 1\}$, we define:

$$\rho_m(t_0^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m > 1. \end{cases}$$

$$\rho_m(t_l^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m > 1 \text{ and } m \leq l + 1, \\ t_{l-1}^j & \text{if } m > l + 1. \end{cases}$$

We now move on to consider the vertices d_i^j, which represent the bits in the counter. We begin by defining a sequence of strategies for the bits that are 0. Recall that these vertices switch to the states a_i^j along the deceleration lane until they run out of edges, at

which point they switch to the vertex e_i^j. This is formalised in the following definition. For each i in the range $1 \leq i \leq n$, each $m \geq 1$, and each $j \in \{0, 1\}$, we define:

$$\rho_m(d_i^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m = 2, \\ a_{2k+2n+6+2i}^j & \text{if } m = 3, \\ a_{m-3}^j & \text{if } 4 \leq m \leq 2k + 2n + 6 + 2i + 3, \\ e_i^j & \text{if } m > 2k + 2n + 6 + 2i + 3. \end{cases}$$

Note that the first three iterations are special, because the edge to a_1^j only becomes switchable in the third iteration. The edge to r^j and the edge to $a_{2k+2n+6+2i}^j$ prevent the edge to e_i^j being switched before this occurs.

We now give a full strategy definition for the vertices d_i^j. The bits that are 0 follow the strategy that we just defined, and the bits that are 1 always choose the edge to e_i^j. For each bit-string $K \in \{0, 1\}^n$, each i in the range $1 \leq i \leq n$, each $m \geq 1$, and each $j \in \{0, 1\}$, we define:

$$\rho_m^K(d_i^j) = \begin{cases} \rho_m(d_i^j) & \text{if } K_i = 0, \\ e_i^j & \text{if } K_i = 1. \end{cases}$$

Finally, we consider the other vertices in the clock. To define strategies for these vertices, we must first define some notation. For each i in the range $1 \leq i \leq n$, we define $\text{NextBit}(K, i)$ to be a partial function that gives the index of the first 1 that appears higher than index i: that is, the smallest index $j > i$ such that $K_j = 1$. We now define the strategies. These strategies all depend on the current clock bit-string K, and have no dependence on how far the deceleration lane has switched, so the parameter m is ignored. For each bit-string $K \in \{0, 1\}^n$, each $m \geq 1$, each i in the range $1 \leq i \leq n$, and each $j \in \{0, 1\}$, we define:

$$\rho_m^K(g_i^j) = \begin{cases} k_i^j & \text{if } K_i = 0, \\ f_i^j & \text{if } K_i = 1. \end{cases}$$

$$\rho_m^K(k_i^j) = \begin{cases} g_{\text{NextBit}(K,i)}^j & \text{if } \text{NextBit}(K, i) \text{ is defined}, \\ x & \text{otherwise}. \end{cases}$$

$$\rho_m^K(r^j) = \begin{cases} g_{\text{NextBit}(K,0)}^j & \text{if } \text{NextBit}(K, 0) \text{ is defined}, \\ x & \text{otherwise}. \end{cases}$$

$$\rho_m^K(s^j) = \begin{cases} f_{\text{NextBit}(K,0)}^j & \text{if } \text{NextBit}(K, 0) \text{ is defined}, \\ x & \text{otherwise}. \end{cases}$$

When the clock transitions between two clock bit-strings, there is a single iteration in which the strategies defined above are not followed. This occurs one iteration after the vertex $d_{\text{Lsz}(K)}^j$ switches to $e_{\text{Lsz}(K)}^j$. In this iteration, the vertices $g_{\text{Lsz}(K)}^j$ and s^j switch to $f_{\text{Lsz }K}^j$, while every other vertex continues to use the strategies that were defined above. We now define a special *reset strategy* that captures this. For each bit-string $K \in \{0, 1\}^n$, and

every vertex v in either of the two clocks we define:

$$\rho_{\mathrm{Reset}}^{K}(v) = \begin{cases} f_{\mathrm{Lsz}(K)}^{j} & \text{if } v = g_{\mathrm{Lsz}(K)}^{j} \text{ or } v = s^{j}, \\ \rho_{\mathrm{Length}(K)}^{K}(v) & \text{otherwise.} \end{cases}$$

We can now combine the strategies defined above in order to define the full sequence of strategies that are used in the clocks. In the first $\mathrm{Length}(K) - 1$ iterations, we follow the sequence defined by the strategies $\rho_{m}^{K}(v)$, and in the final iteration we use the strategy $\rho_{\mathrm{Reset}}^{K}(v)$. Formally, for each bit-string $K \in \{0,1\}^{n}$, each m in the range $1 \leq m \leq \mathrm{Length}(K)$, and every vertex v in either of the two clocks, we define:

$$\kappa_{m}^{K}(v) = \begin{cases} \rho_{m}^{K}(v) & \text{if } m \leq \mathrm{Length}(K) - 1, \\ \rho_{\mathrm{Reset}}^{K}(v) & \text{if } m = \mathrm{Length}(K). \end{cases}$$

Friedmann showed the following lemma.

Lemma 4.1 ([15]). *Let $K \in \{0,1\}^{n}$. If we start all-switches strategy improvement at κ_{1}^{K} for clock j, then it will proceed by switching through the sequence $\kappa_{1}^{K}, \kappa_{2}^{K}, \ldots, \kappa_{\mathrm{Length}(K)}^{K}, \kappa_{1}^{K+1}$.*

The circuits. For each bit-string $B \in \{0,1\}^{n}$, we give a sequence of strategies $\sigma_{1}^{B}, \sigma_{2}^{B}, \ldots$, which describes the sequence of strategies that occurs when B is the input of the circuit. The sequence is indexed from the point at which the circuit's clock advances to the next bit-string. That is, σ_{1}^{B} occurs one iteration after the valuation of s^{j} exceeds the valuation of r^{j}.

Recall that all of the gates with the same depth are evaluated in the same iteration. We can now make this more precise: each gate i will be evaluated in the strategy $\sigma_{d(i)+2}^{B}$. After this iteration, there will then be two cases based on whether the gate evaluates to 1 or 0. To deal with this, we require the following notation. For each bit-string B and each gate i, we define $\mathrm{Eval}(B, i)$ to be 1 if gate i outputs true on input B, and 0 otherwise.

Or gates. Before the gate is evaluated, the state o_{i}^{j} chooses the edge to r^{j}. Once the gate has been evaluated, there are four possibilities. If both input gates evaluate to false, then the state o_{i}^{j} continues to use the edge to r^{j}. If one of the two inputs is true, then o_{i}^{j} will switch to the corresponding input state. The case where both inputs are true is the most complicated. Obviously, o_{i}^{j} will switch to one of the two input states, and in fact, it switches to the one with the highest valuation. Since the overall correctness of our construction does not care which successor is chosen in this case, we simply define $\mathrm{OrNext}(i, B, m)$ to be the successor with the highest valuation in step m of the sequence for bit-string B.

We can now formally define the sequence of strategies used by an OR-gate. For every gate $i \in \mathrm{OR}$, every pair of bit-strings $B \in \{0,1\}^{n}$, and every $m \geq 1$ we define:

$$\sigma_{m}^{B}(o_{i}^{j}) = \begin{cases} s_{i}^{j} & \text{if } m = 1, \\ r_{i}^{j} & \text{if } m > 1 \text{ and } m \leq d(i) + 2, \\ r_{i}^{j} & \text{if } m > d(i) + 2 \text{ and } \mathrm{Eval}(B, I_{1}(i)) = 0 \text{ and } \mathrm{Eval}(B, I_{2}(i)) = 0, \\ \mathrm{InputState}(i, j, 1) & \text{if } m > d(i) + 2 \text{ and } \mathrm{Eval}(B, I_{1}(i)) = 1 \text{ and } \mathrm{Eval}(B, I_{2}(i)) = 0, \\ \mathrm{InputState}(i, j, 2) & \text{if } m > d(i) + 2 \text{ and } \mathrm{Eval}(B, I_{1}(i)) = 0 \text{ and } \mathrm{Eval}(B, I_{2}(i)) = 1, \\ \mathrm{OrNext}(i, B, m) & \text{if } m > d(i) + 2 \text{ and } \mathrm{Eval}(B, I_{1}(i)) = 1 \text{ and } \mathrm{Eval}(B, I_{2}(i)) = 1. \end{cases}$$

$$(4.1)$$

Not gates. There are two components of the NOT-gate gadget: the modified deceleration lane and the state d_i^j. We begin by considering the modified deceleration lane.

We first define a strategy for the case where the gate evaluates to false. In this case, the input gate evaluates to true, which causes the modified deceleration lane to continue switching after iteration $d(i) + 2$. We formalise this in the following definition, which is almost identical to the definition given for the deceleration lane used in the clock. For each $i \in$ NOT, each l in the range $1 \le l \le 2k + 4n + 6$ with $l \ne d(i)$, each $j \in \{0, 1\}$, and each $m \ge 1$, we define:

$$\sigma_m(t_{i,0}^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m > 1. \end{cases} \tag{4.2}$$

$$\sigma_m(t_{i,l}^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m > 1 \text{ and } m \le l + 1, \\ t_{i,l-1}^j & \text{if } m > l + 1. \end{cases} \tag{4.3}$$

On the other hand, if the gate evaluates to false, then the deceleration lane stops switching. This is formalised in the following definition, which uses the previous definition to give the actual strategy used by the modified deceleration lane. For each $i \in$ NOT, each $B \in \{0, 1\}^n$, each l in the range $0 \le l \le 2k + 4n + 6$ with $l \ne d(i)$, each $j \in \{0, 1\}$, and each $m \ge 1$, we define:

$$\sigma_m^B(t_{i,l}^j) = \begin{cases} \sigma_m(t_{i,l}^j) & \text{if } l \le d(i) + 1, \text{ or } l > d(i) + 1 \text{ and } \mathrm{Eval}(B, I(i)) = 1, \\ s^j & \text{if } l > d(i) + 1 \text{ and } m = 1 \text{ and } \mathrm{Eval}(B, I(i)) = 0, \\ r^j & \text{if } l > d(i) + 1 \text{ and } m > 1 \text{ and } \mathrm{Eval}(B, I(i)) = 0. \end{cases}$$

We now turn out attention to the state d_i^j, where we again begin by considering the case where the gate evaluates to false. In this case, the state d_i^j continues switching to the modified deceleration lane. This is formalised in the following definition, which is almost identical to the definition given for the corresponding states in the clock. For all $i \in$ NOT $\cup$ INPUT/OUTPUT, all $B \in \{0, 1\}^n$, for all $m \ge 1$, and all $j \in \{0, 1\}$, we define:

$$\sigma_m(d_i^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m = 2, \\ a_{i,2k+4n+6}^j & \text{if } m = 3, \\ a_{i,m-3}^j & \text{if } 4 \le m \le 2k + 4n + 6 + 3, \\ e_i^j & \text{if } 2k + 4n + 6 + 3 < m. \end{cases}$$

On the other hand, if the gate evaluates to true, then after iteration $d(i) + 2$, the state d_i^j switches to e_i^j. This is formalised in the following definition, where the previous definition is used in order to give the actual sequence of strategies for the state d_i^j. For all $B \in \{0, 1\}^n$, all $i \in$ NOT, all $j \in \{0, 1\}$, and all $m \ge 1$ we define:

$$\sigma_m^B(d_i^j) = \begin{cases} \sigma_m(d_i^j) & \text{if } m \le d(i) + 2, \text{ or } m > d(i) + 2 \text{ and } \mathrm{Eval}(B, I(i)) = 1, \\ e_i^j & \text{if } m > d(i) + 2 \text{ and } \mathrm{Eval}(B, I(i)) = 0. \end{cases} \tag{4.4}$$

Input/output gates. We now describe the sequence of strategies used in the input/output gates. These strategies are almost identical to the strategies that would be used in a NOT-gates with depth $d(C) + 1$, but with a few key differences. Firstly, whereas the NOT-gates used edges to r^j and s^j, these have instead been replaced with the edges to y^j and z^j from the circuit movers. Secondly, the circuit movers cause a one iteration delay at the start of the sequence. Note, however, that despite this delay, the input/output gates are still evaluated on iteration $d(C) + 3$.

We begin by giving the strategies for the modified deceleration lane used in the input/output gates. For each $i \in$ NOT, each l in the range $1 \le l \le 2k + 4n + 6$ with $l \ne d(C)$, each $j \in \{0, 1\}$, and each $m \ge 1$, we define:

$$\sigma_m(t_{i,l}^j) = \begin{cases} z^j & \text{if } m = 2, \\ y^j & \text{if } m = 1 \text{ or } m > 1 \text{ and } m \le l + 2, \\ t_{i,l-1}^j & \text{if } m > l + 2. \end{cases}$$

$$\sigma_m(t_{i,0}^j) = \begin{cases} z^j & \text{if } m = 2, \\ y^j & \text{if } m = 1 \text{ or } m > 2. \end{cases}$$

Then, for each $i \in$ NOT, each $B \in \{0, 1\}^n$, each l in the range $0 \le l \le 2k + 4n + 6$ with $l \ne d(C)$, each $j \in \{0, 1\}$, and each $m \ge 1$, we define:

$$\sigma_m^B(t_{i,l}^j) = \begin{cases} \sigma_m(t_{i,l}^j) & \text{if } l < d(C) + 3, \text{ or } l \ge d(C) + 3 \text{ and } B_i = 0, \\ z^j & \text{if } l > d(C) + 3 \text{ and } m = 2 \text{ and } B_i = 1, \\ y^j & \text{if } l > d(C) + 3 \text{ and either } m = 1 \text{ or } m > 2 \text{ and } B_i = 1. \end{cases}$$

Finally, we give the strategy for the state d_i^j. We reuse the strategy σ_{m-1} from the NOT-gate definitions, but with a one iteration delay. For all $B \in \{0, 1\}^n$, all $i \in$ INPUT/OUTPUT, all $j \in \{0, 1\}$, and all $m > 1$ we define:

$$\sigma_m^B(d_i^j) = \begin{cases} \sigma_{m-1}(d_i^j) & \text{if } 1 < m \le d(C) + 3, \text{ or } m > d(C) + 3 \text{ and } B_i = 0, \\ e_i^j & \text{if } m > d(i) + 3 \text{ and } B_i = 1. \end{cases} \tag{4.5}$$

The circuit mover states. Finally, we describe the sequence of strategies used in the states that move the input/output gates between the circuits. These strategies do not depend on the current input bit-string to the circuit. Instead, they depend on the state of both of the clocks, and are parameterized by the value of the delay function that we defined earlier.

Formally, for every $m \geq 1$, every $i \in \text{INPUT/OUTPUT}$, and every clock-bit string $K \in \{0,1\}^n$ we define:

$$\sigma_m^K(z^j) = \begin{cases} s^j & \text{if } m = 1, \\ r^j & \text{if } m > 1. \end{cases} \tag{4.6}$$

$$\sigma_m^K(y^j) = \begin{cases} r^{1-j} & \text{if } m = 1 \text{ or } m \geq \text{Delay}(j, K) + 1 \\ r^j & \text{if } m > 1 \text{ and } m < \text{Delay}(j, K) + 1. \end{cases} \tag{4.7}$$

$$\sigma_m^K(p_i^j) = \begin{cases} p_{i,1}^j & \text{if } m = 1 \text{ or } m \geq \text{Delay}(j, K) + 1, \\ o_{I(i)}^j & \text{if } 1 < m \leq \text{Delay}(j, K) + 1. \end{cases} \tag{4.8}$$

$$\sigma_m^K(h_{i,0}^j) = \begin{cases} h_{i,2}^j & \text{if } m = 1 \text{ or } m \geq \text{Delay}(j, K) + 1, \\ h_{i,1}^j & \text{if } 1 < m < \text{Delay}(j, K) + 1. \end{cases} \tag{4.9}$$

Putting it all together. We can now define a combined sequence of strategies for the entire construction. We will defined a sequence of strategies $\chi_1^{B,K,j}$, $\chi_2^{B,K,j}$, ..., which describes a computation in circuit j under the following conditions:

- The clock for circuit j currently stores K in its binary counter.
- The input to circuit j is B.

Before stating the strategies, we first define some necessary notation. For every clock bit-string $K \in \{0,1\}$, and every $j \in \{0,1\}$ we define:

$$\text{OC}(K, j) = \begin{cases} K - 1 & \text{if } j = 0, \\ K & \text{if } j = 1. \end{cases}$$

This gives the bit-string used in the *other clock,* when circuit j is computing. Since clock 0 is ahead of clock 1, we have that $\text{OC}(K, 0)$ is the bit-string before K, while $\text{OC}(K, 1)$ is the same as K.

We can now define the sequence. For each bit-string $B \in \{0,1\}^n$, each bit-string $K \in \{0,1\}^n$, each $m \geq 1$, and every vertex v we define:

$$\chi_m^{B,K,j}(v) = \begin{cases} \kappa_m^K(v) & \text{if } v \text{ is in clock } j, \\ \kappa_{m+\text{Delay}(1-j,\text{OC}(K,j))}^{\text{OC}(K,j)}(v) & \text{if } v \text{ is in clock } 1 - j, \\ \sigma_m^B(v) & \text{if } v \text{ is in a NOT or OR gate in circuit } j, \\ \sigma_m^{F(B)}(v) & \text{if } v \text{ is in an input/output gate in circuit } j, \\ \sigma_{m+\text{Delay}(1-j,\text{OC}(K,j))}^B(v) & \text{if } v \text{ is an input/output gate in circuit } 1 - j, \\ \sigma_m^K(v) & \text{if } v \text{ is a circuit mover state in circuit } j, \\ \sigma_{m+\text{Delay}(1-j,\text{OC}(K,j))}^{\text{OC}(K,j)}(v) & \text{if } v \text{ is a circuit mover state in circuit } 1 - j. \end{cases}$$

The first two cases of this definition deal with the clocks: the clock in circuit j follows the sequence for bit-string K, while the clock in circuit $1 - j$ continues to follow the sequence for bit-string $\text{OC}(K, j)$. Observe that the clock for circuit $1 - j$ has already been running for $\text{Delay}(1 - j, \text{OC}(K, j))$ iterations, so the strategies for this clock start on iteration $1 + \text{Delay}(1 - j, \text{OC}(K, j))$. The next two cases deal with the gate gadgets in circuit j: the NOT and OR gates follow the sequence for bit-string B, and then the input/output gates for

circuit j, which are in output mode, store $F(B)$. The next case deals with the input/output gates in circuit $1 - j$ are in input mode, and so follow the strategy for bit-string B. The final two cases deal with the circuit mover states, which follow the strategies for the clock bit-string used in their respective clocks. Observe that no strategy is specified for the gate gadgets in circuit $1 - j$, because the strategy chosen here is irrelevant.

For technical convenience, we define:

$$\chi_{\text{Delay}(0,K)}^{B,K,0} = \chi_1^{F(B),K,1}$$
$$\chi_{\text{Delay}(1,K)}^{B,K,1} = \chi_1^{F(B),K+1,0}$$

Using this definition, we can now state the main technical claim of the paper.

Lemma 4.2. *Let $B \in \{0,1\}^n$ be a bit-string, let $C \in \{0,1\}^n$ be a bit-string such that $C \neq (1,1,\dots,1)$, and let $j \in \{0,1\}$. If greedy all-switches strategy improvement is applied to $\chi_1^{B,K,0}$, then it will pass through the sequence:*

$$\chi_1^{B,K,j}, \chi_2^{B,K,j}, \dots, \chi_{\text{Delay}(j,K)}^{B,K,j}.$$

Unfortunately, the proof of this lemma is quite long, and the vast majority of it is presented in the appendix. In Section 5, we give an overview of the proof, and describe how each of the individual appendices fit into the overall proof.

Best responses. Recall that, for each strategy considered, strategy improvement computes a best-response for the opponent. Now that we have defined the sequence of strategies, we can also define the best-responses to these strategies. For each strategy $\chi_i^{B,K,j}$, we define a strategy $\mu_i^{B,K,j} \in \Sigma_{\text{Odd}}$ that is a best-response to $\chi_i^{B,K,j}$. We will later prove that these strategies are indeed best-responses.

We begin by considering the vertices e_i^j for each NOT-gate i. Recall that these vertices only pick the edge to $h_{i,0}^j$ in the case where they are forced to by d_i^j selecting the edge to e_i^j. As defined above, this only occurs in the case where the NOT-gate evaluates to true. Formally, for each bit-string $B \in \{0,1\}^n$, each bit-string $K \in \{0,1\}^n$, each $m \geq 1$, and every $i \in$ NOT we define:

$$\mu_m^{B,K,j}(e_i^j) = \begin{cases} h_{i,0}^j & \text{if } m > d(i) + 2 \text{ and } \text{Eval}(B, I(i)) = 0, \\ d_i^j & \text{otherwise.} \end{cases}$$

For the input/output gadgets in circuit $1 - j$, which will provide the input to circuit j, the situation is the same. The vertex e_i^{1-j} chooses the edge to $h_{i,0}^{1-j}$ if and only if B_i is 1. Formally, for each bit-string $B \in \{0,1\}^n$, each bit-string $K \in \{0,1\}^n$, each $m \geq 1$, and every vertex $i \in$ INPUT/OUTPUT we define:

$$\mu_m^{B,K,j}(e_i^{1-j}) = \begin{cases} h_{i,0}^{1-j} & \text{if } B_i = 1, \\ d_i^{1-j} & \text{if } B_i = 0. \end{cases}$$

For the input/output gadgets in circuit $1 - j$, the situation is the largely the same as for a NOT-gate with depth $d(C) + 1$, and the edge chosen depends on $F(B)_i$. However, one difference is that we do not define a best-response for the case where $m = 1$, because the input/output gadget does not reset until the second iteration, and our proof does not depend

on the best response chosen in iteration one. Formally, for each bit-string $B \in \{0, 1\}^n$, each bit-string $K \in \{0, 1\}^n$, each $m > 1$, and every vertex $i \in \textsc{Input}/\textsc{Output}$ we define:

$$\mu_m^{B,K,j}(e_i^j) = \begin{cases} h_{i,0}^j & \text{if } m > d(i) + 3 \text{ and } F(B)_i = 1, \\ d_i^j & \text{otherwise.} \end{cases}$$

Finally, we define the best responses for the vertices q^j as follows. For each bit-string $B \in \{0, 1\}^n$, each bit-string $K \in \{0, 1\}^n$, each m in the range $1 \le m \le \mathrm{Delay}(j, K) - 1$, and every vertex $i \in \textsc{Input}/\textsc{Output}$ we define:

$$\mu_m^{B,K,j}(q_{i,0}^j) = \begin{cases} e_i^j & \text{if } m = 1, \\ q_{i,1}^j & \text{if } m > 1. \end{cases}$$

$$\mu_m^{B,K,j}(q_{i,0}^{1-j}) = \begin{cases} e_i^{1-j}. \end{cases}$$

5. The Proof

In this section we give the proof for Lemma 4.2. Let $B, K \in \{0, 1\}^n$ be two bit-strings, let $j \in \{0, 1\}$, and let m be in the range $1 \le m \le \mathrm{Delay}(j, K) - 1$. We must show that greedy all-switches strategy improvement switches $\chi_m^{B,K,j}$ to $\chi_{m+1}^{B,K,j}$.

Let σ be a strategy that agrees with $\chi_m^{B,K,j}$. Since we are using the all-switches switching rule, we can consider each vertex v independently, and must show that the most appealing outgoing edge at v is the one specified by $\chi_{m+1}^{B,K,j}$ (in our construction there will always be exactly one most appealing edge, so we do not care how ties are broken by the switching rule). Hence, the majority of the proof boils down to calculating the valuation of each outgoing edge of v, and then comparing these valuations.

To compare the valuation of two outgoing edges (v, u) and (v, w), we usually use the following technique. First we consider the two paths π_1 and π_2 that start at u and w, respectively, and follow σ and $\mathrm{Br}(\sigma)$. Then we find the first vertex v' that is contained in both paths. Since the $\sqsubseteq$ relation only cares about the maximum difference between the two paths, all priorities that are visited after v' are irrelevant, since they appear in both $\mathrm{Val}^\sigma(u)$ and $\mathrm{Val}^\sigma(w)$. On the other hand, since each priority is assigned to at most one vertex, all of the priorities visited by p_1 before reaching v' are contained in $\mathrm{Val}^\sigma(u)$ and not contained in $\mathrm{Val}^\sigma(w)$, and all of the priorities visited by p_2 before reaching v' are contained in $\mathrm{Val}^\sigma(w)$ and not contained in $\mathrm{Val}^\sigma(u)$. So it suffices to find the largest priority on the prefixes of p_1 before v' and the prefix of p_2 before v'. The parity of this priority then determines whether $\mathrm{Val}^\sigma(u) \sqsubseteq \mathrm{Val}^\sigma(w)$ according to the rules laid out in the definition of $\sqsubseteq$.

We now give an outline of the proof.

- The fact that the two clocks switch through their respective strategies follows from Lemma 4.1.
- The difference in valuation between the states r^j and s^j of the clock are the driving force of the construction. In Appendix A, we give two lemmas that formalize this difference.
- Next, in Appendix B, we prove that the best-response strategies defined in Section 4 are in fact the best responses. That is, we show that $\mu_m^{B,K,j}$ is a best response to every strategy σ that agrees with $\chi_m^{B,K,j}$.

- In Appendix B.4, we give two key lemmas that describe the valuations of the output stats o_i^j. These lemmas show three important properties. Firstly, if $m \le d(i) + 2$, then the valuation of o_i^j is low, so there is no incentive to switch to o_i^j before gate i is evaluated. Secondly, if $m > d(i) + 2$ and the gate evaluates to 0, then the valuation of o_i^j remains low. Finally, if $m > d(i) + 2$ and the gate evaluates to 1, then the valuation of o_i^j is high. These final two properties allow the gates with depth strictly greater than i to compute their outputs correctly.
- The rest of the proof consists of proving that all vertices switch to the correct outgoing edge. The states o_i^j in the OR gate gadgets are dealt with in Appendix C. The states $t_{i,l}^j$ in the NOT gates are dealt with in Appendix D. The states d_i^j in the NOT gates are dealt with in Appendix E. The states z^j and z^{1-j} are dealt with in Appendix F, and the states y^j and y^{1-j} are dealt with in Appendix G. The states p^j and p^{1-j} are dealt with in Appendix H and the states $h_{i,0}^j$ are dealt with in Appendix I. Finally, the states in the INPUT/OUTPUT gates, which behave in a largely identical way to the NOT gates, are dealt with in Appendix J.

All of the above combines to provide a proof for Lemma 4.2.

Having shown this Lemma, we can now give the reduction from BITSWITCH to EDGESWITCH. Given a circuit iteration instance (F, B, z), we produce the parity game G corresponding to F, we use $\chi_2^{B,1,0}$ as the initial strategy, and if $i \in$ INPUT/OUTPUT is the input/output gate corresponding to index z, then we will monitor whether the edge from d_i^0 to e_i^0 is ever switched by greedy all-switches strategy improvement. We therefore produce the instance EDGESWITCH$(G, (d_i^0, e_i^0), \chi_2^{B,1,0})$. We must take care to ensure that $\chi_2^{B,1,0}$ is a terminating strategy, which is proved in the following Lemma.

Lemma 5.1. *We have that $\chi_2^{B,1,0}$ is a terminating strategy.*

Proof. Firstly, since we use the same strategies as Friedmann in the clock, we do not need to prove that theses portions of the strategies are terminating, because this has already been shown by Friedmann. In particular, this implies that all paths starting at s^j and r^j for $j \in \{0, 1\}$ will eventually arrive at the sink x. Therefore, it is sufficient to show that the first component of $\mathrm{Val}_{\mathrm{VJ}}^{\chi_2^{B,1,0}}(v)$ is 1 for every vertex v in the circuits.

Observe that, in the strategy $\chi_2^{B,1,0}$, we have that the only possible cycles that player Odd can form in the best response are two-vertex cycles of the form d_i^j and e_i^j, but these cycles have an even priority, so the best response can not choose them. In particular, it is not possible to form a cycle that passes through the input/output gadgets in both circuits, because the path that starts at o_i^0 for every input/output gate i must eventually arrive at s^j. Thus, we have that all paths starting at all vertices in the circuits that follow $\chi_2^{B,1,0}$ and its best response will eventually arrive at this sink x. $\qquad\square$

Now all that remains is to argue that EDGESWITCH$(G, (d_i^0, e_i^0), \chi_2^{B,1,0})$ is true if and only if BITSWITCH(F, B, z) is true. To do this, we simply observe that the sequence of strategies used in Lemma 4.2 only ever specify that d_i^0 must be switched to e_z^0 in the case where there is some even j such that $F^j(B)_z = 1$. At all other times, the vertex d_i^0 chooses an edge other that e_i^j. Hence, the reduction is correct, and we have shown Theorem 1.2.

6. Other Algorithms

Other strategy improvement algorithms. As we mentioned in the introduction, Theorem 1.2 implies several results about other algorithms. In particular, discounted games and simple-stochastic games both have natural strategy improvement algorithms given by Puri [36], and Condon [4], respectively. Friedmann showed that if you take a one-sink parity game, and apply the natural reduction from parity games to either discounted or simple stochastic games, then the greedy variants of Puri's and Condon's algorithms will switch exactly the same edges as the algorithm of Vöge and Jurdziński [15, Corollory 9.10 and Lemma 9.12]. Hence, Theorem 1.2 also implies the discounted and simple-stochastic cases of Corollary 1.3.

One case that was missed by Friedmann was mean-payoff games. There is a natural strategy improvement algorithm [14] for mean-payoff games that adopts the well-known *gain-bias* formulation from average-reward MDPs [37]. In this algorithm, the valuations has two components: the *gain* of a vertex gives the long-term average-reward that can be obtained from that vertex under the current strategy, and the *bias* measures the short term deviation from the long-term average.

We argue that, if we apply the standard reduction from parity games to mean-payoff games, and then set the reward of x to 0, then the gain-bias algorithm for mean-payoff games switches exactly the same edges as the algorithm of Vöge and Jurdziński. The standard reduction from parity games to mean-payoff games [26, 36] replaces each priority p with the weight $(-m)^p$, where m denotes the number of vertices in the parity game. By setting the weight of x to 0, we ensure that the long-term average reward from each state is 0. Previous work has observed [12], that if the gain is 0 at every vertex, then the bias represents the *total reward* that can be obtained from each state. It is not difficult to prove that, after the standard reduction has been applied, the total reward that can be obtained from a vertex v is larger than the total reward from a vertex u if and only if $\mathrm{Val}^\sigma(u) \sqsubset \mathrm{Val}^\sigma(v)$ in the original parity game. This is because the rewards assigned by the standard reduction grow quickly enough so that only the largest priority visited matters. Hence, we also have the mean-payoff case of Corollary 1.3.

Björklund and Vorobyov have also devised a strategy improvement algorithm for mean-payoff games. Their algorithm involves adding an extra *sink* vertex, and then adding edges from every vertex of the maximizing player to the sink. Their valuations are also the total reward obtained before reaching the sink. We cannot show a similar result for their algorithm, but we can show a result for a *variant* of their algorithm that only gives additional edges to a subset of the vertices of the maximizing player. To do this, we do the same reduction as we did for the gain-bias algorithm, and then we only add an edge from x to the new sink added the Björklund and Vorobyov. The same reasoning as above then implies that the Björklund-Vorobyov algorithm will make the same switches as the Vöge-Jurdziński algorithm.

Unique sink orientations. As mentioned in the introduction, there is a relationship between strategy improvement algorithms and sink-finding algorithms for unique sink orientations. Our result already implies a similar lower bound for sink-finding algorithms applied to unique sink orientations. However, since the vertices in our parity game have more than two outgoing edges, these results only hold for unique sink orientations of *grids*. The more commonly studied model is unique sink orientations of *hypercubes*, which correspond to
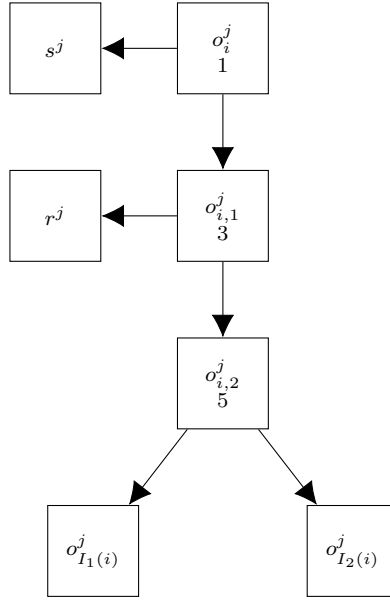
FIGURE 12. The binary OR gate.

binary parity games, where each vertex has at most two outgoing edges. We argue that our construction can be formulated as a binary parity game.

Friedmann has already shown that his construction can be formulated as a binary parity game [15], so we already have that the clocks can be transformed so that they are binary. Furthermore, since our NOT-gates and our INPUT/OUTPUT-gates are taken directly from Friedmann's bit gadget, we can apply Friedmann's reduction to make these binary. In particular, note that all of the extra states that we add to the input/output gate are binary, so these states do not need any modification.

The only remaining part of the construction is the OR-gate, which has four outgoing edges. We replace the existing gadget with a modified gadget, shown in Figure 12.

Vertex	Conditions	Edges	Priority	Player
o_i^j	$j \in \{0,1\}$, $i \in \text{OR}$	s^j, $o_{i,1}^j$	$P(4,i,0,j,1)$	Even
$o_{i,1}^j$	$j \in \{0,1\}$, $i \in \text{OR}$	r^j, $o_{i,2}^j$	$P(4,i,1,j,1)$	Even
$o_{i,2}^j$	$j \in \{0,1\}$, $i \in \text{OR}$	$\text{InputState}(i,j,1)$, $\text{InputState}(i,j,2)$	$P(4,i,2,j,1)$	Even

This gadget replaces the single vertex of the original OR-gate, with three binary vertices. The only significant difference that this gadget makes to the construction is that now it can take up to two strategy improvement iterations for the OR-gate to compute its output. This is because, we may have to wait for $o_{i,2}^j$ to switch before $o_{i,1}^j$ can switch. The vertex o_i^j always chooses the edge $o_{i,1}^j$ during the computation, because the valuation of r^j is larger than the valuation of s^j.

To deal with this, we can redesign the construction so that each NOT-gate i is computed on iteration $2i$ rather than iteration i, and each OR-gate is computed before iteration $2i$. This involved making the following changes:

- The length of the deceleration lane in the two clocks must be extended by $2k$, to account for the $2k$ extra iterations it takes for the circuits to compute (k extra iterations for circuit 0 and k extra iterations for circuit 1). Moreover, the delays for both of the clocks must be increased by k.
- For the same reason, the length of the modified deceleration lanes in the NOT and INPUT/OUTPUT gates must be increased by $2k$.
- Finally, the edge to $o^j_{\text{InputState}(i,j)}$ must be moved from $t_{i,d(i)}$ to $t_{i,2d(i)}$.

Once these changes have been made, we have produced a binary parity game.

One final thing we must be aware of is that we only get a unique-sink orientation if there is never a tie between the valuation of two vertices. This, however, always holds in a one-sink game where every vertex has a distinct priority, because all paths necessarily contain a distinct set of priorities, which prevents ties in the $\sqsubseteq$ ordering. Therefore, we have the PSPACE-completeness result for the BOTTOMANTIPODAL algorithm claimed in Corollary 1.7.

7. THE OPTIMAL STRATEGY RESULT

We are also able to prove a result about the complexity of determining which optimal strategy is found by the Vöge-Jurdziński algorithm. However, we cannot formulate this in the context of a one-sink game, because any result of this nature must exploit *ties in valuations.* In a one-sink game, since every vertex has a different priority, no two paths can have the same set of priorities, so ties are not possible. Hence, for a one-sink game, there will be a unique optimal strategy, and so the complexity of finding it can be no harder than solving the parity game itself, and this problem is not PSPACE-complete unless PSPACE = UP $\cap$ coUP.

On the other hand, ties in valuations are possible in the original Vöge-Jurdziński algorithm. This is because the first component of their valuation is not necessarily 1, and so the second component does not necessarily contain every priority along the relevant path (recall that priorities smaller than the first component are not included in the second component). These facts mean that it is possible to construct parity games that have multiple optimal strategies under the Vöge-Jurdziński valuation.

Our modified construction. We will use a slight modification of our construction to show that computing the optimal strategy found by the Vöge-Jurdziński algorithm is PSPACE-complete. The key difference is the addition of a *third* clock with $n + 1$ bits, which will be indexed by 2. We remove a single edge from this clock: the edge from e^2_{n+1} to h^2_{n+1} is removed.

Recall that in the clock construction, the odd vertices e^j_i do not use the edge to h^j_i unless they are forced to by the vertex d^j_i selecting the edge to e^j_i. Hence, until the $n + 1$th bit is flipped, the third clock behaves like any other clock. When the $n + 1$th bit is flipped, after 2^n iterations have taken place, a new cycle is formed with a very large even priority.

We also modify the edges that leave d^1_z. For each edge $e = (d^1_z, u)$ we do the following:

(1) We delete e.
(2) We introduce a new vertex v_u owned by player Even. This vertex is assigned an insignificant priority that, in particular, is much smaller than the priorities assigned to e^2_{n+1} and d^2_{n+1}.
(3) We add the edges (d^1_z, v_u), (v_u, u), and (v_u, f^2_{n+1}).

The following table summarises the extra clock that we add to the construction, and the new outgoing edges from d_z^1. For ease of notation, we define $U = \{y^1, z^1, e_z^1\} \cup \{a_{z,l}^1 \ : \ 1 \leq l \leq 2k + 4n + 6\}$ to be the original outgoing edges from d_z^1 that will now be replaced. Moreover, we assume that each vertex u is represented by a number in the range $1 \leq u \leq |U|$, which will be used as part of the priority for v_u.

Vertex	Conditions	Edges	Priority	Player
t_0^2		r^2, s^2	$P(2, 0, 2k + 4n + 4, 2, 0)$	Even
t_l^2	$1 \leq l \leq 2k + 4n + 6$	r^2, s^2, t_{l-1}^2	$P(2, 0, l, 2, 1)$	Even
a_l^2	$1 \leq l \leq 2k + 4n + 6$	t_l^2	$P(2, 0, l + 1, 2, 0)$	Even
d_i^2	$1 \leq i \leq n$	$e_i^2, s^2, r^2,$	$P(1, i, 0, 2, 1)$	Even
		a_l^2 for $1 \leq l \leq 2k + 2n + 6 + 2i$		
e_i^2	$1 \leq i \leq n$	d_i	$P(1, i, 1, 2, 0)$	Odd
g_i^2	$1 \leq i \leq n$	f_i^2	$P(1, i, 2, 2, 1)$	Even
k_i^2	$1 \leq i \leq n$	$x, g_l^2,$ for $i < l \leq n$	$P(8, i, 0, 2, 1)$	Even
f_i^2	$1 \leq i \leq n$	e_i^2	$P(8, i, 1, 2, 1)$	Even
h_i^2	$1 \leq i \leq n$	k_i^2	$P(8, i, 2, 2, 0)$	Even
s^2		x, f_l^2 for $1 \leq l \leq n$	$P(7, 0, 0, 2, 0)$	Even
r^2		x, g_l^2 for $1 \leq l \leq n$	$P(7, 0, 1, 2, 0)$	Even
v_u	$u \in U$	u	$P(0, 0, 0, u, 0)$	Even
d_z^1		v_u for all $u \in U$	$P(4, i, 0, j, 1)$	Even

PSPACE-completeness. We now argue how this modified construction provides a PSPACE-hardness proof for the optimal strategy decision problem. Before the $n + 1$th bit of the third clock flips, the edge (v_u, f_{n+1}^2) is never switchable due to the large odd priority assigned to f_{n+1}^2, so this modification does not affect the computation of $F^{2^n}(B)$. On the other hand, once the $n + 1$th bit of the third clock flips, all edges of the form (v_u, f_{n+1}^2) immediately become switchable, because the first component of the valuation of f_{n+1}^2 is now a large even priority, and not 1. So all of these edges will be switched simultaneously.

The key thing to note is that, since the priorities assigned to the vertices v_u are insignificant, they do not appear in the second component of the valuation, and so vertex d_z^1 is now indifferent between all of its outgoing edges. Moreover, the vertices v_u never switch away from f_{n+1}^2 for the following reasons:

- The only even cycle that can be forced by player Even is the one that uses d_{n+1}^2 and e_{n+1}^2. So, these vertices must select a strategy that reaches this cycle eventually.
- All priorities used in the circuits are smaller than the priority of the cycle between d_{n+1}^2 and e_{n+1}^2. So, the second component of the valuation function is irrelevant, and the only way of improving the strategy would be to find a shorter path to the cycle.
- The vertices v_u are the only vertices that have edges to the third clock, so the only way a vertex v_u could reach the third clock would be to travel through both circuits to reach d_z^1, and then use a different vertex $v_{u'}$, but this would be a much longer path, and therefore this would have a lower valuation.

Hence, the vertices v_u will never switch away from f_{n+1}^2.

Observe that, after 2^n iterations, the input/output gadgets in circuit $1 - j$ store the value of $F^{2^n}(B)$, and therefore d_z^1 chooses the edge to e_z^j if and only if the zth bit of $F^{2^n}(B)$ is 1. The above argument implies that d_z^1 does not switch again, so in the optimal strategy found

by the algorithm, the vertex d_z^1 chooses the edge to e_z^j if and only if the zth bit of $F^{2^n}(B)$ is 1. Thus, we have that computing the optimal strategy found by the Vöge-Jurdziński strategy improvement algorithm is `PSPACE`-complete, as claimed in Theorem 1.5.

Other games. We also get similar results for the gain-bias algorithm for mean-payoff games, and the standard strategy improvement algorithms for discounted and simple stochastic games. For the most part, we can still rely on the proof of Friedmann for these results. This is because, although we do not have a one-sink game, the game behaves as a one-sink game until the $n + 1$th bit in the third clock is flipped. An easy way to see this is to reinstate the edge between e_{n+1}^2 and h_{n+1}^2 to create a one-sink game, and observe that, since the edge is not used in the best response until the $n + 1$th bit is flipped, it cannot affect the sequence of strategies visited by strategy improvement. Once the $n + 1$th bit has flipped, we only care about making d_z^1 indifferent between its outgoing edges, and in this section we explain how this is achieved.

For mean-payoff games, we use the same reduction as we did in Section 6 to our altered construction. After doing this, we set the weight of the vertices v_u to 0 to ensure that d_z^1 will be exactly indifferent between all of its outgoing edges once these vertices switch to f_{n+1}^2. This gives the result for the gain-bias algorithm in mean-payoff games.

For discounted games, once the standard reduction from mean-payoff to discounted games has been applied, the proof of Friedmann already implies that the discounted game algorithm makes the same decisions as the Vöge-Jurdziński algorithm for the vertices other than d_z^1. The only worry is that the discount factor may make the vertex d_z^1 not indifferent between some of its outgoing edges. However, it is enough to note that all paths from d_z^1 to f_{n+1}^2 have length 2, and therefore the vertex will be indifferent no matter what discount factor is chosen. This gives the result for the standard strategy improvement algorithm for discounted games.

Finally, after applying the standard reduction from discounted to simple-stochastic games, the proof of Friedmann can be applied to argue that the valuations in the simple stochastic game are related to the valuations in the discounted game by a linear transformation. Hence, d_z^1 will still be indifferent between its outgoing edges after the $n + 1$th bit is flipped. This gives the result for the standard strategy improvement algorithm for simple stochastic games. Thus, we have the claimed results from Corollary 1.6.

8. Open problems

Strategy improvement generalizes policy iteration which solves mean-payoff and discounted-payoff Markov decision processes [37]. The exponential lower bounds for greedy all-switches have been extended to MDPs. Fearnley showed that the second player in Friedmann's construction [15] can be simulated by a probabilistic action, and used this to show an exponential lower bound for the all-switches variant of policy iteration of average-reward MDPs [8]. This technique cannot be applied to the construction in this paper, because we use additional Odd player states (in particular the vertices $q_{i,1}^j$) that cannot be translated in this way. Can our `PSPACE`-hardness results be extended to all-switches strategy improvement for MDPs?

Also, there are other pivoting algorithms for parity games that deserve attention. It has been shown that Lemke's algorithm and the Cottle-Dantzig algorithm for the P-matrix linear complementarity problem (LCP) can be applied to parity, mean-payoff, and discounted

games [11, 29]. It would be interesting to come up with similar PSPACE-completeness results for these algorithms, which would also then apply to the more general P-matrix LCP problem.

References

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

[2] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. of STOC*, pages 252–263, 2017.

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

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

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

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

[7] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *Proc. of CAV*, pages 385–396, 1993.

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

[9] J. Fearnley. Non-oblivious strategy improvement. In *Proc. of LPAR*, pages 212–230, 2010.

[10] J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *Proc. of SPIN*, pages 112–121, 2017.

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

[12] J. Fearnley and R. Savani. The complexity of the simplex method. In *Proc. of STOC*, pages 201–208, 2015.

[13] J. Fearnley and R. Savani. The complexity of all-switches strategy improvement. In *Proc. of SODA*, pages 130–139, 2016.

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

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

[16] 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.

[17] 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.

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

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

[20] B. Gärtner and A. Thomas. The complexity of recognizing unique sink orientations. In *Proc. of STACS*, pages 341–353, 2015.

[21] 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.

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

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

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

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

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

[27] M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. In *Proc. of LICS*, 2017.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix A. Facts about the clock

In this section we prove two important lemmas about the clocks. The first lemma shows an important property about the difference in valuation between r^j and s^j for the clock used by circuit j. The second lemma considers the difference in valuation *across* the two clocks, by comparing the valuations of r^j, s^j, r^{1-j}, and s^{1-j}.

Lemma A.1. *Let σ be a strategy that agrees with κ_m^K for some m in the range $1 \leq m \leq$ Length(K), some clock-value $K \in \{0,1\}^n$, and some $j \in \{0,1\}$. We have:*
  (1) *If $m = $ Length$(K) - 1$ then $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(s^j)$.*
  (2) *If $m < $ Length$(K) - 1$ then $\mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.*
In both cases, we have that $\mathrm{MaxDiff}^\sigma(s^j, r^j) \geq \mathrm{P}(7,0,0,0,0)$.

Proof. We begin with the first case. In this case, by definition, we have that the path that starts at s^j and follows σ moves to f_i^j for some i, whereas the path that starts at r^j and follows σ moves to $g_{i'}^j$ for some $i' \neq i$. There are two possibilities.

  (1) If $i > i'$, then since i is the least significant zero in K, we must have that $i' = \mathrm{NextBit}(K, 0) = 1$. Hence, the path that starts at $g_{i'}^j$ passes through the bit gadgets for all bits strictly smaller than i before eventually arriving at $g_{\mathrm{NextBit}(K,i)}^j$ (or x if $\mathrm{NextBit}(K, i)$ is not defined). In particular, since the path does *not* pass through h_i^j the largest priority on the path is strictly smaller than $\mathrm{P}(8, i, 2, j, 0)$.

  On the other hand, the path that starts at f_i^j eventually arrives at $g_{\mathrm{NextBit}(K,i)}^j$ (or x if $\mathrm{NextBit}(K, i)$ is not defined) and it *does* pass through h_i^j. The largest priority on this path is $\mathrm{P}(8, i, 2, j, 0)$, and since the priority is even, we can conclude that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(s^j)$.

  (2) If $i < i'$, then since i' is the least significant one in K, we must have that $i = 1$. Hence, the path that starts at f_i^j eventually moves to k_i^j and then directly to $g_{i'}^j$. The largest priority on this path is $\mathrm{P}(8, i', 2, j, 0)$, and since this is even, we can conclude that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(s^j)$.

We now move on to the second case. In this case, by definition, we have that the path that starts at s^j moves to f_i^j for some i, whereas the path that starts at r^j moves to g_i^j and then to f_i^j. Since the priority on g_i^j is strictly smaller than $\mathrm{P}(7, 0, 1, j, 0)$, we have that $\mathrm{MaxDiff}^\sigma(r^j, s^j) = \mathrm{P}(7, 0, 1, j, 0)$, which the priority assigned to r^j. Since this priority is even, we have that $\mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, as required.

  Observe that in all cases considered above, we have shown that $\mathrm{MaxDiff}^\sigma(r^j, s^j) \geq \mathrm{P}(7, 0, 0, 0, 0)$ as required. Hence, we have completed the proof of this lemma. $\square$

Lemma A.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $m \geq 1$, some bit-strings $B, K \in \{0,1\}^n$, and some $j \in \{0,1\}$. We have:*
  (1) *If $m = \mathrm{Delay}(j, K) - 1$, then $\mathrm{Val}^\sigma(r^{1-j}) \sqsubset \mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(s^{1-j})$ and $\mathrm{MaxDiff}^\sigma(r^{1-j}, s^j) \geq \mathrm{P}(7, 0, 0, 0, 0)$ and $\mathrm{MaxDiff}^\sigma(r^{1-j}, r^j) \geq \mathrm{P}(7, 0, 0, 0, 0)$.*
  (2) *If $m < \mathrm{Delay}(j, K) - 1$, then $\mathrm{Val}^\sigma(r^{1-j}) \sqsubset \mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$ and $\mathrm{MaxDiff}^\sigma(r^{1-j}, s^j) \geq \mathrm{P}(7, 0, 0, 0, 0)$ and $\mathrm{MaxDiff}^\sigma(s^j, r^{1-j}) \geq \mathrm{P}(7, 0, 0, 0, 0)$.*

Proof. We begin with the second claim. The fact that $\mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$ follows from part 2 of Lemma A.1, so it is sufficient to show that $\mathrm{Val}^\sigma(r^{1-j}) \sqsubset \mathrm{Val}^\sigma(s^j)$. There are two cases to consider, based on whether $j = 0$ or $j = 1$.

(1) If $j = 0$, then clock j uses bit-string K, and clock $1 - j$ uses bit-string $K - 1$. Observe that the clock strategies specify that the path starting at s^j visits h_i^j if and only if $K_i = 1$. Similarly, the path starting at r^{1-j} visits h_i^{1-j} if and only if $(K - 1)_i = 1$. If i' is the index of the least significant 1 in K, then we have that the path that starts at s^j visits $h_{i'}^j$ and $k_{i'}^j$, and the path that starts at s^{1-j} does not visit these vertices. Moreover, these two paths are the same after this point. Hence, we have that $\text{MaxDiff}(s^j, r^{1-j})$ is $\text{P}(8, i', 2, j, 0)$, and since this priority is even, we can conclude that $\text{Val}^\sigma(r^{1-j}) \sqsubset \text{Val}^\sigma(s^j)$.

(2) If $j = 1$, then both clocks use bit-string K. Hence, the paths starting at s^j and r^{1-j} use the same path through their respective clocks. So, if i' is the index of the most significant 1 in K, then we have that $\text{P}(8, i', 2, 0, 0)$ is the largest priority on the path starting at $r^{1-j} = r^0$, and $\text{P}(8, i', 2, 1, 0)$ is the largest priority on the path starting at $s^j = s^1$. Thus, $\text{MaxDiff}^\sigma(s^j, r^{1-j}) = \text{P}(8, i', 2, 1, 0)$, and since this priority is even and contained in $\text{Val}^\sigma(s^j)$ we can conclude that $\text{Val}^\sigma(r^{1-j}) \sqsubset \text{Val}^\sigma(s^j)$.

We now move on to the first claim. Here the same reasoning as we gave for the second case can be used to prove that $\text{Val}^\sigma(r^{1-j}) \sqsubset \text{Val}^\sigma(s^j)$, and therefore Lemma A.1 implies that $\text{Val}^\sigma(r^{1-j}) \sqsubset \text{Val}^\sigma(r^j)$. What remains is to prove that $\text{Val}^\sigma(r^j) \sqsubset \text{Val}^\sigma(s^{1-j})$. Again there are two cases to consider.

(1) If $j = 0$, then clock j uses bit-string K, and clock $1 - j$ is about to transition from bit-string $K - 1$ to bit-string K. In fact, the path from s^{1-j} is already the path for bit-string K, so the proof from item 2 above can be reused.

(2) If $j = 1$, then clock j uses bit-string K, and clock $1 - j$ is about to transition from bit-string K to bit-string $K + 1$. In fact, the path from s^{1-j} is already the path for bit-string $K + 1$, so the proof from item 1 above can be reused.

Finally, we observe that all of the maximum difference priorities used in the proof are strictly larger than $\text{P}(7, 0, 0, 0, 0)$, which completes the proof. $\qquad\square$

Appendix B. Best responses

In this section, we prove that the best responses defined in Section 4 are indeed best responses to $\chi_m^{B,K,j}$. There are two types of odd vertices used in the construction: the vertices e_i^j used in the NOT and INPUT/OUTPUT gates, and the vertices $q_{i,0}^j$ used in the INPUT/OUTPUT gates. We begin by proving a general lemma concerning the vertices e_i^j used in the NOT and INPUT/OUTPUT gates.

Lemma B.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some bit-strings $B, K \in \{0, 1\}^n$, some m in the range $1 \leq m \leq \text{Delay}(j, K) - 1$, and some $j \in \{0, 1\}$. For every $i \in \text{NOT} \cup \text{INPUT/OUTPUT}$, and every $l \in \{0, 1\}$, we have that if $\sigma(d_i^l) = e_i^l$, then $\text{Br}(\sigma)(e_i^l) \neq d_i^l$.*

Proof. Note that if player Odd uses the edge from e_i^l to d_i^l, then this would create a cycle with largest priority $\text{P}(1, i, 1, j, 0)$, which is even. Since the game is a one-sink game, and since the initial strategy is terminating, we have that Odd can eventually reach the odd cycle at x^j from the vertices r^j, r^{1-j}, s^j, and s^{1-j}. Furthermore, Odd can reach one of these four vertices by moving to h_i^l. Since the odd cycle at x^j has priority smaller than $\text{P}(1, i, 1, j, 0)$, we can conclude that $\text{Br}(\sigma)(e_i^j) \neq d_i^l$. $\qquad\square$

We now proceed to prove individual lemma for each of the vertices that belong to player Odd. Each type of Odd vertex will be considered in a different subsection.

B.1. **The vertices** $q_{i,0}^l$. We now consider the vertices $q_{i,0}^l$ for $i \in$ INPUT/OUTPUT. The first lemma considers the case where $l = j$, and the second lemma considers the case where $l = 1 - j$.

Lemma B.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some bit-strings $B, K \in \{0,1\}^n$, some m in the range $1 \le m \le \mathrm{Delay}(j,K) - 1$, and some $j \in \{0,1\}$. For every $i \in$ INPUT/OUTPUT, we have that $\mathrm{Br}(\sigma)(q_{i,0}^j) = \mu_m^{B,K,j}(q_{i,0}^j)$.*

Proof. There are two cases to consider.

- If $m = 1$, then then we must show that the edge to e_i^j is chosen by Odd in the best response. Consider a strategy τ where $\tau(e_i^j) = h_{i,0}^j$, and $\tau(q_{i,0}^j) = e_i^j$. When τ is played against σ, the path that starts at e_i^j eventually arrives at r^{1-j}, and the largest priority on this path is strictly smaller than $\mathrm{P}(6, d(C) + 2, 0, j, 0)$. On the other hand, taking the edge to $q_{i,1}^j$ leads directly to r^{1-j} while visiting the priority $\mathrm{P}(6, d(C) + 2, 0, j, 0)$. Since this priority is even, we can conclude that Odd would prefer to play τ than to use the edge from $q_{i,0}^j$ to $q_{i,1}^j$ in his best response. Therefore, we must have that $\mathrm{Br}(\sigma)(q_{i,0}^j) = e_i^j$, as required.

- If $m > 1$, then we must show that the edge to $q_{i,1}^j$ is the least appealing edge at $q_{i,0}^j$. Observe that the path that starts at e_i^j and follows σ eventually arrives at r^j, and every priority on this path that is strictly smaller than $\mathrm{P}(7, 0, 0, 0, 0)$. On the other hand, the path that starts at $q_{i,1}^j$ moves directly to r^{1-j}. Hence, we can apply Lemma A.2 (both parts) to argue that $\mathrm{Val}^\sigma(q_{i,1}^j) \sqsubset \mathrm{Val}^\sigma(e_i^j)$, as required. $\square$

Lemma B.3. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some bit-strings $B, K \in \{0,1\}^n$, some m in the range $1 \le m \le \mathrm{Delay}(j,K) - 1$, and some $j \in \{0,1\}$. For every $i \in$ INPUT/OUTPUT, we have that $\mathrm{Br}(\sigma)(q_{i,0}^{1-j}) = \mu_m^{B,K,j}(q_{i,0}^{1-j})$.*

Proof. We must show that the edge to e_i^{1-j} is the least appealing edge at $q_{i,0}^{1-j}$. Observe that the path that starts at e_i^{1-j} and follows σ will eventually arrive at either r^j or r^{1-j}. In either case, the largest priority on this path will be strictly smaller than $\mathrm{P}(6, d(C) + 2, 0, j, 0)$. On the other hand, the path that starts at $q_{i,1}^{1-j}$ moves directly to r^j, and the largest priority on this path is $\mathrm{P}(6, d(C) + 2, 0, j, 0)$. Since this priority is even, we can conclude that $\mathrm{Val}^\sigma(e_i^{1-j}) \sqsubset \mathrm{Val}^\sigma(q_{i,0}^{1-j})$, as required. $\square$

B.2. **The vertices** e_i^l **in Not gates.** The following lemma considers the vertices e_i^l for $l = j$ and $i \in$ NOT. We do not need to prove a lemma for the case where $l = 1 - j$ and $i \in$ NOT, because these vertices are in the non-computing circuit, and we do not specify strategies for the NOT and OR gadgets in the non-computing circuits.

Lemma B.4. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some bit-strings $B, K \in \{0,1\}^n$, some m in the range $1 \leq m \leq \mathrm{Delay}(j, K) - 1$, and some $j \in \{0, 1\}$. For every $i \in \mathrm{NOT}$, we have that $\mathrm{Br}(\sigma)(e_i^j) = \mu_m^{B,K,j}(e_i^j)$.*

Proof. There are three cases to consider.

   (1) If $m = 1$ then, the path that starts at d_i^j and follows σ moves directly to s^j. On the other hand, the path that starts at h_i^j moves directly to r^j. All of the priorities on these paths are strictly smaller than $\mathrm{P}(7, 0, 0, 0, 0)$, so we can apply Lemma A.1 part 2 to argue that $\mathrm{Val}^\sigma(d_i^j) \sqsubset \mathrm{Val}^\sigma(d_i^j)$, so therefore we have $\mathrm{Br}(\sigma)(e_i^j) = d_i^j$.
   (2) If $m > 1$ and either $m \leq d(i) + 2$, or $m > d(i) + 2$ and $\mathrm{Eval}(B, I(i)) = 1$, then observe that the path that starts at d_i^j and follows σ eventually arrives at r^j, and that the largest priority on this path is strictly smaller than $\mathrm{P}(6, i, 1, j, 0)$. On the other hand, the path that starts at h_i^j and follows σ moved directly to r^j, and the largest priority on this path is $\mathrm{P}(6, i, 1, j, 0)$. Since this priority is even, we can conclude that $\mathrm{Val}^\sigma(d_i^j) \sqsubset \mathrm{Val}^\sigma(e_i^j)$, so therefore we have $\mathrm{Br}(\sigma)(e_i^j) = d_i^j$.
   (3) If $m > d(i) + 2$ and $\mathrm{Eval}(B, I(i)) = 0$, then we have $\sigma(d_i^j) = e_i^j$, so we can apply Lemma B.1 to prove that $\mathrm{Br}(\sigma)(e_i^j) = h_i^j$.

$\square$

B.3. **The vertices e_i^j in input/output gates.** The following lemmas consider the vertices e_i^l when $i \in \mathrm{INPUT}/\mathrm{OUTPUT}$. The first lemma considers the case where $l = j$, and the second lemma considers the case where $l = 1 - j$.

Lemma B.5. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some bit-strings $B, K \in \{0,1\}^n$, some m in the range $1 \leq m \leq \mathrm{Delay}(j, K) - 1$, and some $j \in \{0, 1\}$. For every $i \in \mathrm{INPUT}/\mathrm{OUTPUT}$, we have that $\mathrm{Br}(\sigma)(e_i^j) = \mu_m^{B,K,j}(e_i^j)$.*

Proof. There are a number of cases to consider.

   (1) If $m > 1$ and $F^2(B)_i = 0$ and either $m \leq d(i) + 3$, or $m > d(i) + 3$ and $F^2(B)_i = 0$, then the path that starts at d_i^j eventually arrives at r^j, and the largest priority on this path is strictly smaller than $\mathrm{P}(6, d(C) + 1, 1, j, 0)$. On the other hand, the path that starts at $h_{i,0}^j$ and follows σ passes through $h_{i,1}^j$ and then arrives at r^j. The largest priority on this path is $\mathrm{P}(6, d(C) + 1, 1, j, 0)$, and since this priority is even, we can conclude that $\mathrm{Val}^\sigma(d_i^j) \sqsubset \mathrm{Val}^\sigma(h_{i,0}^j)$. Therefore, we have that $\mathrm{Br}(\sigma)(e_i^j) = d_i^j$.
   (2) If $m > 1$ and $m > d(i) + 3$ and $F^2(B)_i = 1$, then $\sigma(d_i^j) = e_i^j$, and so we can apply Lemma B.1 to argue that $\mathrm{Br}(\sigma)(e_i^j) = h_{i,0}^j$.

$\square$

Lemma B.6. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some bit-strings $B, K \in \{0,1\}^n$, some m in the range $1 \leq m \leq \mathrm{Delay}(j, K) - 1$, and some $j \in \{0, 1\}$. For every $i \in \mathrm{INPUT}/\mathrm{OUTPUT}$, we have that $\mathrm{Br}(\sigma)(e_i^{1-j}) = \mu_m^{B,K,j}(e_i^{1-j})$.*

Proof. There are a number of cases to consider.

   (1) If $m = 1$ and $B_i = 0$, then the path that starts at d_i^{1-j} and follows σ will eventually reach r^{1-j}, and the largest priority on this path is strictly smaller than $\mathrm{P}(6, d(C) + 1, 1, j, 0)$. On the other hand, the path that starts at $h_{i,0}^{1-j}$ moves to $h_{i,1}^{1-j}$ and

then arrives at r^{1-j}, and the largest priority on this path is $\mathrm{P}(6, d(C)+1, 1, j, 0)$. Since this priority is even, we have that $\mathrm{Val}^\sigma(d_i^{1-j}) \sqsubset \mathrm{Val}^\sigma(h_{i,0}^{1-j})$ and therefore $\mathrm{Br}(\sigma)(e_i^{1-j}) = d_i^{1-j}$.

(2) $m > 1$ and $B_i = 0$, then the path that starts at d_i^{1-j} moves to r^j, and the largest priority on this path is strictly smaller than $\mathrm{P}(6, 0, 1, j, 0)$. On the other hand, the path that starts at $h_{i,0}^{1-j}$ moves to $h_{i,2}^{1-j}$ and then arrives at r^j, and the largest priority on this path is $\mathrm{P}(6, 0, 1, j, 0)$. Since this priority is even, we have that $\mathrm{Val}^\sigma(d_i^{1-j}) \sqsubset \mathrm{Val}^\sigma(h_{i,0}^{1-j})$ and therefore $\mathrm{Br}(\sigma)(e_i^{1-j}) = d_i^{1-j}$.

(3) If $B_i = 1$, then we have $\sigma(d_i^{1-j}) = e_i^{1-j}$, and we can apply Lemma B.1 to argue that $\mathrm{Br}(\sigma)(e_i^{1-j}) = h_{i,0}^{1-j}$.

$\square$

B.4. Gate outputs.

In this section we give two key lemmas that describe the valuation of the output states o_i^j. The first lemma considers the case where $m = 1$ or $m = 2$, and the second lemma considers the case where $m \geq 3$.

Lemma B.7. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and for $m = 1$ or $m = 2$. For every gate i, we have $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.*

Proof. There are four cases to consider

(1) We begin by showing the claim for an input/output gate i from circuit $1 - j$ in the case where $m = 1$. Note that Lemma B.6 implies that $\mathrm{Br}(\sigma)(e_i^{1-j}) = d_i^{1-j}$. Observe that by definition, the path that starts at d_i^j and follows σ will trace a path through the gadgets for circuit $1 - j$, and will eventually reach r^{1-j}. Furthermore, the largest priority possible that can be seen along this path is $\mathrm{P}(6, d(C)+1, 1, j, 0) < \mathrm{P}(7, 0, 0, 0, 0)$. Hence, we can apply Lemma A.2 to argue that $\mathrm{Val}^\sigma(o_i^{1-j}) \sqsubset \mathrm{Val}^\sigma(r^j)$.

(2) Now we consider an input/output gate i from circuit $1 - j$ in the case where $m = 2$. Again, Lemma B.6 implies that $\mathrm{Br}(\sigma)(e_i^{1-j}) = d_i^{1-j}$, but in this case since $m = 2$, we have that the vertices y^{1-j} and p_i^{1-j} have both switched to r^j. Hence, the path that starts at o_i^{1-j} will eventually arrive at r^j. It can be verified that, whatever path is taken from o_i^{1-j} to r^j, the largest priority along this path is $\mathrm{P}(6, i, 0, j, 1)$ on the vertex o_i^{1-j}. Since this priority is odd, we have that $\mathrm{Val}^\sigma(o_i^{1-j}) \sqsubset \mathrm{Val}^\sigma(r^j)$.

(3) Next we consider the case where i is a OR-gate. If $m = 1$ then we have that $\sigma(o_i^j) = s^j$, and if $m = 2$ then $\sigma(o_i^j) = r^j$. In both cases, we can use Lemma A.1 and the fact that the priority assigned to o_i^j is odd, to conclude that $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.

(4) Finally, we consider the case where i is a NOT-gate. We can apply Lemma B.4 to argue that $\mathrm{Br}(\sigma)(e_i^j) = d_i^j$. If $m = 1$ then we have that $\sigma(d_i^j) = s^j$, and if $m = 2$ then $\sigma(d_i^j) = r^j$. In both cases, the highest priority on the path from o_i^j to either s^j or r^j is the odd priority from o_i^j, so we can use this fact, along with Lemma A.1 to conclude that $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.

$\square$

Lemma B.8. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and some m in the range $3 \le m \le \mathrm{Delay}(j, K$. For every gate i, we have:*

(1) *If $m \le d(i) + 2$, then $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.*

(2) *If $m > d(i) + 2$ and $\mathrm{Eval}(B, i) = 0$, then $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$.*

(3) *If $m > d(i) + 2$ and $\mathrm{Eval}(B, i) = 1$, then $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(o_i^j)$ and:*

$$\mathrm{P}(6,0,0,0,0) \le \mathrm{MaxDiff}^\sigma(r^j, o_i^j) \le \mathrm{P}(6, i, 1, j, 0).$$

Proof. We will prove this claim by induction over the depth of the gates. For the base case we consider an input/output gate i from circuit $1 - j$, which provide the input values for circuit j. Since we consider these gates to have depth 0, we always have $m > d(i) + 2$, so there are two cases to prove based on whether B_i is zero or one. First, observe that since $m \ge 3$, the circuit mover gadgets attached to the input/output gadget for bit i in circuit $1 - j$ have $\sigma(y^{1-j}) = r^j$. Since $\mathrm{Delay}(1 - j, K) \ge d(C) + 3$, the definition given in (4.5) implies that the strategy at d_i^{1-j} is determined by B_i. So we have the following two cases.

- If $B_i = 0$, then $\sigma(d_i^{1-j}) = a_{i,l}^{1-j}$ for some l. By definition, in the strategy σ, all paths from $a_{i,l}^{1-j}$ eventually arrive at r^j, and the maximum priority on any of these paths is smaller than $\mathrm{P}(6,0,0,j,1)$. Hence, the largest priority on the path from o_i^{1-j} to r^j is the priority $\mathrm{P}(6,0,0,j,1)$ on the vertex o_i^{1-j}, and since this is an odd priority, we have $\mathrm{Val}^\sigma(o_i^{1-j}) \sqsubset \mathrm{Val}^\sigma(r^j)$.

- If $B_i = 1$, then $\sigma(d_i^{1-j}) = e_i^{1-j}$ and Lemma B.6 then implies $\mathrm{Br}(\sigma)(e_i^{1-j}) = h_i^{1-j}$. So, the path that starts at o_i^{1-j} and follows σ passes through e_i^{1-j}, $h_{i,0}^{1-j}$, $h_{i,2}^{1-j}$, and then arrives at r^j. The largest priority on this path is $\mathrm{P}(6,0,1,j,0) > \mathrm{P}(6,0,0,j,1)$ on the state $h_{i,2}^{1-j}$, so we have $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(o_i^{1-j})$

Hence, the base case of the induction has been shown. The inductive step will be split into two cases, based on whether i is a NOT-gate or an OR-gate.

Suppose that the inductive hypothesis holds for all gates i with $d(i) < k$, and let i be a OR-gate with $d(i) = k$. We must prove three cases.

- The first two cases use the same proof. If $m \le d(i) + 2$, or if $m > d(i) + 2$ and $\mathrm{Eval}(B, i) = 0$, then by definition we have $\sigma(o_i^j) = r^j$. Since the priority assigned to o_i^j is odd, we have $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, as required.

- If $m > d(i) + 2$ and $\mathrm{Eval}(B, i) = 1$, then by definition we have that $\sigma(o_i^j) = \mathrm{InputState}(i, j, l)$ for some gate l with $l \in \{1, 2\}$, and we know that $\mathrm{Eval}(B, I_l(i)) = 1$. Hence, we can apply the inductive hypothesis to argue that $\mathrm{MaxDiff}^\sigma(r^j, \mathrm{InputState}(i, j, l))$ is even, and it satisfies:

$$\mathrm{P}(6,0,0,0,0) \le \mathrm{MaxDiff}^\sigma(r^j, \mathrm{InputState}(i, j, l)) \le \mathrm{P}(6, l, 1, j, 0).$$

Since the priority assigned to o_i^j is smaller than $\mathrm{P}(6,0,0,0,0)$, we have that the same two properties apply to $\mathrm{MaxDiff}^\sigma(r^j, o_i^j)$. Hence, $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(o_i^j)$, and the required bounds on $\mathrm{MaxDiff}^\sigma(r^j, o_i^j)$ hold because $\mathrm{P}(6, l, 1, j, 0) < \mathrm{P}(6, i, 1, j, 0)$.

Now suppose that the inductive hypothesis holds for all gates i with $d(i) < k$, and let i be a NOT-gate with $d(i) = k$. We must prove three cases.

- If $m \le d(i) + 2$, then by definition, the path that starts at o_i^j and follows σ passes through e_i^j, and Lemma B.4 implies that it then moves to d_i^j, a vertex of the form

$a_{i,l}^j$, a number of vertices of the form $t_{i,l}^j$, before finally arriving at r^j. It can easily be verified that the largest priority on this path is $\mathrm{P}(6, i, 0, j, 1)$ from the vertex o_i^j. So, we have $\mathrm{MaxDiff}^\sigma(r^j, o_i^j) = \mathrm{P}(6, i, 0, j, 1)$, and since this priority is odd, we have $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, as required.

- If $m > d(i) + 2$ and $\mathrm{Eval}(B, i) = 0$, then we must have $\mathrm{Eval}(B, I(i)) = 1$. Hence, by definition, the path that starts at o_i^j and follows σ first passes through e_i^j, and then Lemma B.4 implies that it then passes through d_i^j, and then some vertex of the form $a_{i,l}^j$, followed by a number of vertices of the form $t_{i,l}^j$, before finally arriving at $t_{i,d(i)}^j$ and then moving to $\mathrm{InputState}(i, j)$. The largest priority on this path is $\mathrm{P}(6, i, 0, j, 1)$ from the vertex o_i^j. By the inductive hypothesis, we have $\mathrm{MaxDiff}^\sigma(r^j, \mathrm{InputState}(i, j)) < \mathrm{P}(6, I(i), 1, j, 0)$, and since $I(i) < i$ we have that $\mathrm{P}(6, i, 0, j, 1) > \mathrm{P}(6, I(i), 1, j, 0)$. Hence, we have that $\mathrm{MaxDiff}^\sigma(r^j, o_{I(i)}^j) = \mathrm{P}(6, i, 0, j, 1)$, and since this is odd, we have that $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, as required.

- If $m > d(i) + 2$ and $\mathrm{Eval}(B, i) = 1$, then by definition we have that the path that starts at o_i^j and follows σ passes through e_i^j, and then Lemma B.4 implies that it passes through h_i^j, and then reaches r^j. The largest priority on this path is $\mathrm{P}(6, i, 1, j, 0)$ on the vertex h_i^j. Hence we have $\mathrm{MaxDiff}^\sigma(r^j, o_i^j) \leq \mathrm{P}(6, i, 1, j, 0)$, and since this priority is even, we have that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(o_i^j)$. Hence, we have shown both of the required properties for this case.

Now that we have shown the two versions of the inductive hypothesis, we have completed the proof. $\qquad\square$

Appendix C. Or gates

The following pair of lemmas show that the states o_i^j in the Or gate gadgets correctly switch to the outgoing edge specified by $\chi_{m+1}^{B,K,j}$. The first lemma considers the case where $1 \leq m < \mathrm{Delay}(j, K) - 1$, and the second lemma considers the case where $m = \mathrm{Delay}(j, K) - 1$.

Lemma C.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m < \mathrm{Delay}(j, K) - 1$. For each Or-gate i, greedy all-switches strategy improvement will switch o_i^j to $\chi_{m+1}^{B,K,j}(o_i^j)$.*

Proof. In this proof we will show the that the most appealing edge is the one that is specified in Equation (4.1). This boils down to a case analysis.

First suppose that $m < d(i) + 2$. We must prove that the edge to r^j is the most appealing edge at o_i^j. By Lemma A.1, we have that $\mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$. Furthermore, since $d(I_1(i)) = d(I_2(i)) = d(i) - 1$, part 1 of Lemma B.8 implies that $\mathrm{Val}^\sigma(\mathrm{InputState}(i, j, l)) \sqsubset \mathrm{Val}^\sigma(r^j)$ for $l \in \{1, 2\}$. Hence, we have that r^j is the most appealing edge at o_i^j, as required.

Now suppose that $d(i) + 2 \leq m \leq \mathrm{Delay}(j, K) - 1$. There are three cases to consider.

- If both input gates are false, then Lemma A.1 and part 2 of Lemma B.8 imply that r^j will continue to be the most appealing edge at o_i^j, as required.
- If $I_l(i)$ is true and $I_{1-l}(i)$ is false, for some $l \in \{1, 2\}$, then part 3 of Lemma B.8 implies that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(\mathrm{InputState}(i, j, l))$, and $\mathrm{Val}^\sigma(\mathrm{InputState}(i, j, 1 - l)) \sqsubset$

$\mathrm{Val}^\sigma(r^j)$. Therefore, the most appealing edge at o_i^j is the one to $\mathrm{InputState}(i, j, l)$, as required.

- Finally, if both input gates are true, then Lemma B.8 implies that the highest appeal edge at o_i^j is either $\mathrm{InputState}(i, j, 1)$ or $\mathrm{InputState}(i, j, 2)$. Since $\mathrm{OrNext}(i, B, m)$ is defined to be the successor with highest appeal, we have that the highest appeal edge at o_i^j is the one to $o_{I_{\mathrm{OrNext}(i)}(i, B, m)}^j$, as required.

This completes the proof that greedy all-switches strategy improvement will switch o_i^j to $\chi_{m+1}^{B, K, j}(o_i^j)$. $\square$

Lemma C.2. *Let σ be a strategy that agrees with $\chi_m^{B, K, j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and for $m = \mathrm{Delay}(j, K) - 1$. For each OR-gate i, greedy all-switches strategy improvement will switch o_i^{1-j} to $\chi_{m+1}^{B, K, j}(o_i^{1-j})$.*

Proof. We must show that the edge to s^{1-j} is the most appealing edge at o_i^{1-j}. It can be verified that all paths starting at $\mathrm{InputState}(i, j, 1)$ and $\mathrm{InputState}(i, j, 2)$ either reach r^{1-j}, s^{1-j}, or r^j. Furthermore, the largest possible priority on these paths is strictly smaller than $P(7, 0, 0, 0, 0)$. Hence, we can apply part 1 of Lemma A.1 and part 1 of Lemma A.2 to conclude that the edge to s^{1-j} is the most appealing outgoing edge at o_i^{1-j}. $\square$

APPENDIX D. THE STATES $t_{i,l}^j$ IN NOT GATES

In this section we show that the states $t_{i,l}^j$ in the NOT gate gadgets correctly switch to the outgoing edge specified by $\chi_{m+1}^{B, K, j}$. The first two lemmas consider the case where $1 \le m < \mathrm{Delay}(j, K) - 1$, and the third lemma considers the case where $m = \mathrm{Delay}(j, K) - 1$. The first lemma deals with the case where $l < d(i)$, while the second lemma deals with the case where $l > d(i)$. Observe that there is no need to deal with the case where $l = d(i)$, because $t_{i,d(i)}^j$ only has one outgoing edge.

Lemma D.1. *Let σ be a strategy that agrees with $\chi_m^{B, K, j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \le m < \mathrm{Delay}(j, K) - 1$. For each NOT-gate i, and each l int the range $1 \le l < d(i)$, greedy all-switches strategy improvement will switch $t_{i,l}^j$ to $\chi_{m+1}^{B, K, j}(t_{i,l}^j)$.*

Proof. Lemma A.1 implies that the state $t_{i,l}^j$ will not switch to s^j. In the rest of this proof we consider the two remaining outgoing edges at this state. We have two cases to consider.

(1) We first deal with the case where $m < l + 1$, where we must show that the edge to r^j is the highest appeal edge at $t_{i,l}^j$. Let $v = \sigma(t_{i,l}^j)$ be the successor of $t_{i,l}^j$ according to σ (if $m = 1$ then $v = s^j$, and otherwise $v = r^j$). Since $m \le l$, the definition in Equation (4.2) we have that $\sigma(t_{i,l-1}^j) = v$. Since the priority assigned to $t_{i,l-1}^j$ is odd, we therefore have that $\mathrm{Val}^\sigma(t_{i,l-1}^j) \sqsubset \mathrm{Val}^\sigma(v)$. Since we already know from Lemma A.1 that $\mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, we have therefore proved that the edge to r^j is the most appealing edge at $t_{i,l}^j$.

(2) Now we deal with the case where $m \geq l + 1$, where we must show that the most appealing edge at $t_{i,l}^j$ is the one to $t_{i,l-1}^j$. Since $m > l$, the definition in Equation (4.2) implies that the path that starts at $t_{i,l-1}^j$ and follows σ will pass through $t_{i,l'}^j$ for all l' in the range $0 \leq l' < l$ before arriving at r^j. The highest priority on this path is $\mathrm{P}(5, i, 2k + 4n + 4, j, 0)$ on the vertex $t_{i,0}^j$. Since this priority is even we have that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(t_{i,l-1}^j)$, and therefore the edge to $t_{i,l-1}^j$ is the most appealing edge at $t_{i,l}^j$.

□

Lemma D.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and some m in the range $1 \leq m < \mathrm{Delay}(j, K) - 1$. For each NOT-gate i, and each $l > d(i)$, greedy all-switches strategy improvement will switch $t_{i,l}^j$ to $\chi_{m+1}^{B,K,j}(t_{i,l}^j)$.*

Proof. Lemma A.1 implies that the state $t_{i,l}^j$ will not switch to s^j. In the rest of this proof we consider the two remaining outgoing edges at this state.

(1) First we deal with the case where $m < l+1$, where we must show that the edge to r^j is the most appealing edge at $t_{i,l}^j$. When $l > d(i)+1$, the proof of this fact is identical to Item 1 in the proof of Lemma D.1. For $l = d(i) + 1$, we invoke Lemma B.8 to argue that, since $m < l + 1 = d(i) + 2$, we have that $\mathrm{Val}^\sigma(o_i^j) \sqsubset \mathrm{Val}^\sigma(r^j)$. Since the priority assigned to $t_{i,d(i)}^j$ is odd, we therefore also have that $\mathrm{Val}^\sigma(t_{i,d(i)}^j) \sqsubset \mathrm{Val}^\sigma(r^j)$. Therefore, the edge to r^j is the most appealing edge at $t_{i,d(i)+1}^j$.

(2) Now we deal with the case where $m \geq l + 1$, and where $\mathrm{Eval}(B, I(i)) = 0$. Here we must show that the edge to r^j is the most appealing edge at $t_{i,l}^j$. For each $l > d(i)+1$, the proof is identical to the proof of the first case in the proof of this lemma. For $l = d(i) + 1$, we invoke Lemma B.8 to argue that, since $\mathrm{Eval}(B, I(i)) = 0$ we must have $\mathrm{Val}^\sigma(o_{I(i)}^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, and therefore the edge to r^j is the most appealing edge at $t_{i,l}^j$.

(3) Finally, we deal with the case where $m \geq l+1$ and where $\mathrm{Eval}(B, I(i)) = 1$. Here we must show that edge to $t_{i,l-1}^j$ is the most appealing edge at $t_{i,l}^j$. From the definition given in Equation (4.2), we have that the path that starts at $t_{i,l-1}^j$ and follows σ will pass through $t_{i,l'}^j$ for each l' in the range $d(i) \leq l' < l$ before moving to $o_{I(i)}^j$. Since $m \geq l + 1 > d(i)$, we have that $m \geq d(i) + 2$, and therefore Lemma B.8 implies that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(o_{I(i)}^j)$ and that $\mathrm{MaxDiff}^\sigma(r^j, o_{I(i)}^j) \geq \mathrm{P}(6, 0, 0, 0, 0)$. All priorities on the path from $t_{i,l-1}^j$ to $o_{I(i)}^j$ are smaller than $\mathrm{P}(6, 0, 0, 0, 0)$, so we can conclude that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(t_{i,l-1}^j)$, as required.

□

Lemma D.3. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and for $m = \mathrm{Delay}(j, K) - 1$. For each NOT-gate i, and each l int the range $1 \leq l < 2k+4n+6$, greedy all-switches strategy improvement will switch $t_{i,l}^{1-j}$ to $\chi_{m+1}^{B,K,j}(t_{i,l}^{1-j})$.*

Proof. We must show that the edge to s^{1-j} is the most appealing edge at $t_{i,l}^{1-j}$. It can be verified that all paths starting at $t_{i,l-1}^{1-j}$ reach one of r^{1-j}, s^{1-j}, or r^j. Furthermore, the

largest possible priority on these paths is strictly smaller than $P(7, 0, 0, 0, 0)$. Hence, we can apply part 1 of Lemma A.1 and part 1 of Lemma A.2 to conclude that the edge to s^{1-j} is the most appealing outgoing edge at $t_{i,l}^{1-j}$. $\square$

Appendix E. The state d_i^j in Not gates

In this section we show that the states d_i^j in the Not gate gadgets correctly switch to the outgoing edge specified by $\chi_{m+1}^{B,K,j}$. The first lemma considers the case where $m = 1$, the second lemma considers the case where $m = 2$, the third lemma considers the case where $3 \leq m < d(i) + 2$, the fourth lemma considers the case where $d(i) + 2 \leq m < \text{Delay}(j, K) - 1$ and the gate outputs 0, the fifth lemma considers the case where $d(i) + 2 \leq m < \text{Delay}(j, K)$ and the gate outputs 1, and the final lemma considers the case where $m = \text{Delay}(j, K) - 1$.

Lemma E.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and where $m = 1$. For each Not-gate i, greedy all-switches strategy improvement will switch d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. According to the definition given in Equation (4.4), we must show that the edge to r^j is the most appealing edge at d_i^j. We do so by a case analysis.

(1) First we consider the vertex s^j. Here we can apply part 2 of Lemma A.1 to argue that $\text{Val}^\sigma(s^j) \sqsubset \text{Val}^\sigma(r^j)$.

(2) Next we consider a vertex $a_{i,l}^j$ with $l \neq d(i)$. Here the definition given in Equation (4.2) implies that the path that starts at $a_{i,l}^j$ and follows σ passes through $t_{i,l}^j$ and then arrives at s^j. The largest priority on this path is $P(5, i, l + 1, j, 0)$ on the vertex $a_{i,l}^j$. However, since this priority is smaller than $P(7, 0, 0, 0, 0)$, we can apply part 2 of Lemma A.1 to prove that $\text{Val}^\sigma(a_{i,l}^j) \sqsubset \text{Val}^\sigma(r^j)$.

(3) Next we consider a vertex $a_{i,l}^j$ with $l = d(i)$. Here we can apply Lemma B.7 to argue that $\text{Val}^\sigma(o_{I(i)}^j) \sqsubset \text{Val}^\sigma(r^j)$. Furthermore, the largest priority on the path from $a_{i,l}^j$ to o_i^j is the odd priority on $t_{i,d(i)}^j$. Hence, we can conclude that $\text{Val}^\sigma(a_{i,d(i)}^j) \sqsubset \text{Val}^\sigma(r^j)$.

(4) Finally, we consider the vertex e_i^j. Lemma B.4 implies that $\text{Br}(\sigma)(e_i^j) = d_i^j$. Hence, the path that starts at e_i^j moves to d_i^j and then to s^j. The highest priority on this path is $P(4, i, 1, j, 0) < P(7, 0, 0, 0, 0)$. Therefore, we can apply Lemma A.1 to show that $\text{Val}^\sigma(e_i^j) \sqsubset \text{Val}^\sigma(r^j)$.

Hence, we have shown that the edge to r^j is the most appealing outgoing edge at d_i^j, so greedy all-switches strategy improvement will switch d_i^j to r^j. $\square$

Lemma E.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and where $m = 2$. For each Not-gate i, greedy all-switches strategy improvement will switch d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. According to the definition given in Equation (4.4), we must show that the edge to $a_{i,2k+4n+6}^j$ is the most appealing edge at d_i^j. We do so by a case analysis.

(1) First we consider the vertex r^j. Observe that the path that starts at $a_{i,2k+4n+6}^j$ and follows σ visits $t_{i,2k+4n+6}^j$ and then arrives at r^j. The highest priority on this path is the even priority assigned to $a_{i,2k+4n+6}^j$, so therefore we can conclude that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(a_{i,2k+4n+6}^j)$.

(2) Next we consider the vertex s^j. Here we can apply Lemma A.1 to argue that $\mathrm{Val}^\sigma(s^j) \sqsubset \mathrm{Val}^\sigma(r^j)$, and we have already shown that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(a_{i,2k+4n+6}^j)$.

(3) Next we consider a vertex $a_{i,l}^j$ with $l \neq d(i)$ and $l \neq a_{i,2k+4n+6}^j$. The path that starts at $a_{i,l}^j$ and follows σ passes through $t_{i,l}^j$ and then arrives at r^j. The largest priority on this path is the even priority assigned to $a_{i,l}^j$. However, since $l < 2k + 4n + 6$, we have that this priority is smaller than the even priority assigned to $a_{i,2k+4n+6}^j$. Therefore, we have $\mathrm{Val}^\sigma(a_{i,l}^j) \sqsubset \mathrm{Val}^\sigma(a_{i,2k+4n+6}^j)$.

(4) Next we consider the vertex $a_{i,d(i)}^j$. Here we can apply Lemma B.7 to argue that $\mathrm{Val}^\sigma(o_{I(i)}^j) \sqsubset \mathrm{Val}^\sigma(r^j)$. Furthermore, the largest priority on the path from $a_{i,l}^j$ to o_i^j is the odd priority on $t_{i,d(i)}^j$. Since the highest priority on the path from $a_{i,2k+4n+6}^j$ to r^j is even, we we can conclude that $\mathrm{Val}^\sigma(a_{i,d(i)}^j) \sqsubset \mathrm{Val}^\sigma(a_{i,2k+4n+6}^j)$.

(5) Finally, we consider the vertex e_i^j. Lemma B.4 implies that $\mathrm{Br}(\sigma)(e_i^j) = d_i^j$. Hence, the path that starts at e_i^j moves to d_i^j and then to r^j. The highest priority on this path is $\mathrm{P}(4, i, 1, j, 0)$ on the vertex e_i^j. However, this is smaller than the largest priority on the path from $a_{i,2k+4n+6}^j$ to r^j, so we can conclude that $\mathrm{Val}^\sigma(e_i^j) \sqsubset \mathrm{Val}^\sigma(a_{i,2k+4n+6}^j)$. $\qquad\square$

Lemma E.3. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and some m in the range $3 \leq m < d(i) + 2$. For each NOT-gate i, greedy all-switches strategy improvement will switch d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. Lemma A.1 implies that the state d_i^j will not switch to s^j. In the rest of this proof we consider the other outgoing edges at this state.

Since we are in the case where $m < d(i) + 2$ the definition from Equation (4.4) specifies that greedy all-switches strategy improvement must switch to $a_{i,m-2}^j$. Hence must argue that the edge to $a_{i,m-2}^j$ is the most appealing edge at d_i^j, and we will start by considering the appeal of this edge. The definition in Equation (4.2) implies that the path that starts at $a_{i,m-2}^j$ passes through $t_{i,l}^j$ for all $l \leq m - 2$ before arriving at r^j. The highest priority on this path is $\mathrm{P}(5, i, 2k + 4n + 4, j, 0)$ on the vertex t_0^j, and the second highest priority on this path is $\mathrm{P}(5, i, m - 1, j, 0)$ on the vertex $a_{i,m-2}^j$. We will now show that all other edges are less appealing.

(1) First we consider the vertex r^j. Since $\mathrm{P}(5, i, 2k+4n+4, j, 0)$ is even, we immediately get that $\mathrm{Val}^\sigma(r^j) \sqsubset \mathrm{Val}^\sigma(a_{i,m-2}^j)$.

(2) Next we consider a vertex $a_{i,l}^j$ with $l < m-2$. The path that starts at this vertex and follows σ passes through $t_{i,l'}^j$ for all $l' \leq l$ before arriving at r^j. The highest priority on this path is $\mathrm{P}(5, i, 2k + 4n + 4, j, 0)$ on the vertex t_0^j, and the second highest

priority on this path is $P(5, i, l + 1, j, 0)$ on the vertex $a_{i,l}^j$. Hence, we have that $\text{MaxDiff}^\sigma(a_{i,m-2}^j, a_{i,l}^j)$ is $P(5, i, m - 1, j, 0)$, and since this is even, we can conclude that $\text{Val}^\sigma(a_{i,l}^j) \sqsubset \text{Val}^\sigma(a_{i,m-2}^j)$.

(3) Next we consider a vertex $a_{i,l}^j$ with $l > m - 2$ and with $l \neq d(i)$. The path that starts at this vertex and follows σ passes through $t_{i,l}^j$ and then moves immediately to r^j. The highest priority on this path is $P(5, i, l + 1, j, 0)$ on the vertex $a_{i,l}^j$. Since $l + 1 < 2k + 4n + 4$ we have that $\text{MaxDiff}^\sigma(a_{i,m-2}^j, a_{i,l}^j)$ is $P(5, i, 2k + 4n + 4, j, 0)$, and since this priority is even, we can conclude that $\text{Val}^\sigma(a_{i,l}^j) \sqsubset \text{Val}^\sigma(a_{i,m-2}^j)$.

(4) Next we consider the vertex $a_{i,d(i)}^j$. The path that starts at this vertex passes through $t_{i,d(i)}^j$, and then moves to $\text{InputState}(i, j)$. Since $m < d(i) + 2$, we have that $m \leq d(I(i)) + 2$, and therefore the first case of Lemma B.8 tells us that $\text{Val}^\sigma(\text{InputState}(i, j)) \sqsubset \text{Val}^\sigma(r^j)$. Hence we can conclude that $\text{Val}^\sigma(a_{i,d(i)}^j) \sqsubset \text{Val}^\sigma(a_{i,m-2}^j)$.

(5) Finally, we consider the vertex e_i^j. By Lemma B.4, the path that starts at e_i^j and follows σ passes through d_i^j. If $m = 3$, it then moves to $a_{i,2k+4n+6}^j$, and if $m > 3$ it then moves to $a_{i,m-3}^j$. In either case, since the priority assigned to e_i^j is smaller than the priorities assigned to the vertices a_l^j and t_l^j, we can reuse the arguments made above to conclude that $\text{Val}^\sigma(e_i^j) \sqsubset \text{Val}^\sigma(a_{i,m-2}^j)$.

Therefore, we have shown that the edge to $a_{i,m-2}^j$ is the most appealing outgoing edge at d_i^j, and so this edge will be switched by greedy all-switches strategy improvement. $\qquad\square$

Lemma E.4. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and some m in the range $d(i) + 2 \leq m < \text{Delay}(j, K) - 1$. For each NOT-gate $\text{Eval}(B, I(i)) = 1$ then greedy all-switches strategy improvement will switch d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. Lemma A.1 implies that the state d_i^j will not switch to s^j. In the rest of this proof we consider the other outgoing edges at this state.

Since we are in the case where $m \geq d(i) + 2$ and $\text{Eval}(B, I(i)) = 1$, the definition from Equation (4.4) specifies that greedy all-switches strategy improvement must switch to $a_{i,m-2}^j$. Hence must argue that the edge to $a_{i,m-2}^j$ is the most appealing edge at d_i^j, and we will start by considering the appeal of this edge. The definition in Equation (4.2) implies that the path that starts at $a_{i,m-2}^j$ passes through $t_{i,l}^j$ for all l in the range $d(i) \leq l \leq m - 2$ before arriving at $\text{InputState}(i, j)$. Since $m \geq d(i) + 2$ we have $m > d(I(i) + 2)$, and so Lemma B.8 implies that $\text{MaxDiff}^\sigma(r^j, \text{InputState}(i, j)) \geq P(6, 0, 0, 0, 0)$. We now consider the other outgoing edges from d_i^j.

(1) First we consider r^j, where Lemma B.8 immediately gives that $\text{Val}^\sigma(r^j) \sqsubset \text{Val}^\sigma(a_{i,m-2}^j)$.

(2) Next we consider a vertex $a_{i,l}^j$ with $l < d(i)$. Using the same reasoning as Item 2 in the proof of Lemma E.3, we can conclude that the highest priority on the path from $a_{i,l}^j$ to r^j is $P(5, i, 2k + 4n + 4, j, 0) < P(6, 0, 0, 0, 0)$ and so therefore $\text{Val}^\sigma(a_{i,l}^j) \sqsubset \text{Val}^\sigma(a_{i,m-2}^j)$.

(3) Next we consider a vertex $a_{i,l}^j$ with l in the range $d(i) \leq l < m - 2$. The path that starts at $a_{i,l}^j$ and follows σ passes through $t_{i,l'}^j$ for all l' in the range $d(i) \leq l' \leq l$ and then arrives at InputState(i, j). The highest priority on this path is P$(5, i, l+1, j, 0)$. On the other hand, the largest priority on the path from $a_{i,m-2}^j$ to InputState(i, j) is P$(5, i, m - 1, j, 0)$. Since this priority is even, we can conclude that Val$^\sigma(a_{i,l}^j) \sqsubset$ Val$^\sigma(a_{i,m-2}^j)$.

(4) Next we consider a vertex $a_{i,l}^j$ with $l > m - 2$. Using the same reasoning as Item 3 in the proof of Lemma E.3 we can conclude that the highest priority on the path from $a_{i,l}^j$ to r^j is P$(5, i, l + 1, j, 0) <$ P$(6, 0, 0, 0, 0)$ and so therefore Val$^\sigma(a_{i,l}^j) \sqsubset$ Val$^\sigma(a_{i,m-2}^j)$.

(5) Finally, we consider the vertex e_i^j, where we can use the same reasoning as Item 5 in the proof of Lemma E.3 to conclude that Val$^\sigma(e_i^j) \sqsubset$ Val$^\sigma(a_{i,m-2}^j)$.

Therefore, we have shown that the edge to $a_{i,m-2}^j$ is the most appealing outgoing edge at d_i^j, and so this edge will be switched by greedy all-switches strategy improvement. $\qquad\square$

Lemma E.5. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $d(i) + 2 \leq m <$ Delay$(j, K) - 1$. For each* NOT*-gate* Eval$(B, I(i)) = 0$ *then greedy all-switches strategy improvement will switch d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. Lemma A.1 implies that the state d_i^j will not switch to s^j. In the rest of this proof we consider the other outgoing edges at this state.

Since $m \geq d(i) + 2$ and Eval$(B, I(i) = 0)$, the definition in Equation (4.4) specifies that the edge to e_i^j is the most appealing edge at d_i^j. To prove this, we first show that all other edges are less appealing than $a_{i,d(i)-1}^j$, and we will then later show that e_i^j is more appealing than $a_{i,d(i)-1}^j$. The definition in Equation (4.2) implies that the path that starts at $a_{i,d(i)-1}^j$ passes through $t_{i,l}^j$ for all l in the range $0 \leq l \leq d(i) - 1$ before arriving at r^j. The largest priority on this path is P$(5, i, d(i), j0)$ on the state $a_{i,d(i)-1}^j$. We now consider the other outgoing edges.

(1) First we consider the vertex r^j. Since P$(5, i, 2k+4n+4, j, 0)$ is even, we immediately get that Val$^\sigma(r^j) \sqsubset$ Val$^\sigma(a_{i,m-2}^j)$.

(2) Next we consider a vertex $a_{i,l}^j$ with $l < d(i) - 1$. Here we can use the same reasoning as we used in Item 2 in the proof of Lemma E.3 to argue that Val$^\sigma(a_{i,l}^j) \sqsubset$ Val$^\sigma(a_{i,d(i)-1}^j)$.

(3) Next we consider a vertex $a_{i,l}^j$ with $l > d(i)$. Here we can use the same reasoning as we used in Item 3 in the proof of Lemma E.3 to argue that Val$^\sigma(a_{i,l}^j) \sqsubset$ Val$^\sigma(a_{i,d(i)-1}^j)$.

(4) Finally, we consider the vertex $a_{i,d(i)}^j$. Here we can use the same reasoning as we used in Item 4 in the proof of Lemma E.3 to argue that Val$^\sigma(a_{i,d(i)}^j) \sqsubset$ Val$^\sigma(a_{i,d(i)-1}^j)$, although this time we will use case two of Lemma B.8.

So, we have shown that every edge other than the one to e_i^j is less appealing than the edge to $a_{i,d(i)-1}^j$.

Now we will show that the edge to e_i^j is more appealing than the edge to $a_{i,d(i)-1}^j$. There are two cases to consider.

- If $m = d(i)+2$, then $\sigma(d_i^j) = a_{i,d(i)-1}^j$. Since Lemma B.4 implies that $\mathrm{Br}(\sigma)(e_i^j) = d_i^j$, we have that the path that starts at e_i^j and follows σ passes through d_i^j and then arrives at $a_{i,d(i)-1}^j$. The largest priority on this path is $\mathrm{P}(4,i,1,j,0)$, and since this is even, we can conclude that $\mathrm{Val}^\sigma(a_{i,d(i)-1}^j) \sqsubset \mathrm{Val}^\sigma(e_i^j)$.

- If $m > d(i)+2$, then $\sigma(d_i^j) = e_i^j$. In this case Lemma B.4 implies that $\mathrm{Br}(\sigma)(e_i^j) = h_i^j$. The path that starts at e_i^j and follows σ passes through h_i^j and then moves directly to r_i^j. The largest priority on this path is $\mathrm{P}(6,i,1,j0)$, which is bigger than $\mathrm{P}(5,i,d(i),j0)$. Therefore, $\mathrm{MaxDiff}^\sigma(e_i^j, a_{i,d(i)-1}^j) = \mathrm{P}(6,i,1,j0)$, and since this is even, we can conclude that $\mathrm{Val}^\sigma(a_{i,d(i)-1}^j) \sqsubset \mathrm{Val}^\sigma(e_i^j)$.

Hence, we have shown that the edge to e_i^j is the most appealing edge at d_i^j. $\qquad\square$

Lemma E.6. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and for $m = \mathrm{Delay}(j,K) - 1$. For each NOT-gate i greedy all-switches strategy improvement will switch d_i^{1-j} to $\chi_{m+1}^{B,K,j}(d_i^{1-j})$.*

Proof. We must show that the edge to s^{1-j} is the most appealing edge at d_i^{1-j}. It can be verified that all paths starting at the vertices $a_{i,l}^{1-j}$ and e_i^j reach one of r^{1-j}, s^{1-j}, or r^j. Furthermore, the largest possible priority on all of these paths is strictly smaller than $\mathrm{P}(7,0,0,0,0)$. Hence, we can apply part 1 of Lemma A.1 and part 1 of Lemma A.2 to conclude that the edge to s^{1-j} is the most appealing outgoing edge at d_i^{1-j}. $\qquad\square$

APPENDIX F. THE VERTICES z^l

In this section we show that the states z^l correctly switch to the outgoing edge specified by $\chi_{m+1}^{B,K,j}$. The first lemma considers the case where $l = j$, while the second lemma considers the case where $l = 1 - j$.

Lemma F.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and m in the range $1 \leq m \leq \mathrm{Delay}(j,K) - 1$. The greedy all-switches strategy improvement algorithm will switch z^j to $\chi_{m+1}^{B,K,j}(z^j)$.*

Proof. We must show that the edge to r^j is the most appealing edge at z^j. This follows immediately from part 2 of Lemma A.1. $\qquad\square$

Lemma F.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and some m in the range $1 \leq m \leq \mathrm{Delay}(j,K) - 1$. The greedy all-switches strategy improvement algorithm will switch z^{1-j} to $\chi_{m+1}^{B,K,j}(z^{1-j})$.*

Proof. There are two cases to consider.

(1) If $m < \mathrm{Delay}(j,K) - 1$, then we must show that the edge to r^{1-j} is the most appealing edge at z^{1-j}. This follows immediately by applying part 2 of Lemma A.1.

(2) If $m = \text{Delay}(j, K) - 1$, then since $\text{Delay}(j, K) + \text{Delay}(1 - j, K) = \text{Length}(K)$, we must show that the edge to s^{1-j} is the most appealing edge at z^{1-j}. This follows immediately from part 1 of Lemma A.1.

$\square$

Appendix G. The vertices y^l

In this section we show that the states y^l correctly switch to the outgoing edge specified by $\chi_{m+1}^{B,K,j}$. The first lemma considers the case where $l = j$, while the second lemma considers the case where $l = 1 - j$.

Lemma G.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m \leq \text{Delay}(j, K) - 1$. The greedy all-switches strategy improvement algorithm will switch y^j to $\chi_{m+1}^{B,K,j}(y^j)$.*

Proof. The definition of $\chi_{m+1}^{B,K,j}(y_i^j)$ specifies that the edge chosen at y^j is defined by $\sigma_{m+\text{Delay}(j,K)}(y^j)$, which is given in Equation (4.7). According to this definition, we must show that the edge to r^j is the most appealing outgoing edge at y^j. This follows immediately from parts 1 and 2 of Lemma A.2. $\square$

Lemma G.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m \leq \text{Delay}(j, K) - 1$. The greedy all-switches strategy improvement algorithm will switch y^{1-j} to $\chi_{m+1}^{B,K,j}(y^{1-j})$.*

Proof. According to Equation (4.7), we must show that the edge to r^j is the most appealing edge at y^{1-j}. This follows immediately from Lemma A.2. $\square$

Appendix H. The vertices p_i^l

In this section we show that the vertices p_i^l correctly switch to the outgoing edge specified by $\chi_{m+1}^{B,K,j}$. The first lemma considers the case where $l = j$, while the second lemma considers the case where $l = 1 - j$.

Lemma H.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m \leq \text{Delay}(j, K) - 1$. For each $i \in$ Input/Output, the greedy all-switches strategy improvement algorithm will switch p_i^j to $\chi_{m+1}^{B,K,j}(p_i^j)$.*

Proof. Since m is in the range $1 \leq m \leq \text{Delay}(j, K) - 1$, the definition given in Equation (4.8) specifies that $o_{I(i)}^j$ should be the most appealing outgoing edge at p_i^j. There are two cases to consider.

(1) If $m = 1$, then observe that the path that starts at $o_{I(i)}^j$ and follows σ will eventually reach s^j, no matter whether $I(i)$ is a Not-gate or an Or-gate. Furthermore, the largest priority on this path is strictly smaller than $P(7, 0, 0, 0, 0)$. On the other hand, the path that starts at $p_{i,1}^j$ moves immediately to r^{1-j}, and the largest priority on this path is strictly smaller than $P(7, 0, 0, 0, 0)$. Hence, we can apply part 2 of Lemma A.2 to argue that $\text{Val}^\sigma(p_{i,1}^j) \sqsubset \text{Val}^\sigma(o_{I(i)}^j)$, as required.

(2) If $m \geq 1$, then then observe that the path that starts at $o^j_{I(i)}$ and follows σ will eventually reach r^j, and again the largest priority on this path is strictly smaller than $\mathrm{P}(7, 0, 0, 0, 0)$. Hence, we can apply part 2 of Lemma A.2 to argue that $\mathrm{Val}^\sigma(p^j_{i,1}) \sqsubset \mathrm{Val}^\sigma(o^j_{I(i)})$, as required.

$\square$

Lemma H.2. *Let σ be a strategy that agrees with $\chi^{B,K,j}_m$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m \leq \mathrm{Delay}(j, K) - 1$. For each $i \in \textsc{Input}/\textsc{Output}$, the greedy all-switches strategy improvement algorithm will switch p^{1-j}_i to $\chi^{B,K,j}_{m+1}(p^{1-j}_i)$.*

Proof. The definition given in Equation (4.8) specifies that the edge to p^{1-j}_i should be the most appealing edge at p^{1-j}_i. The path that starts at $o^{1-j}_{I(i)}$ and follows σ must eventually arrive at either s^{1-j} or r^{1-j}. In particular, observe that r^j cannot be reached due to the vertices $q^j_{i,0}$, which by Lemma B.2 selects the edge towards $q^j_{i,1}$. On the other hand, the path that starts at $p^{1-j}_{i,1}$ moves directly to r^j. Moreover the largest priorities on both of these paths are strictly smaller than $\mathrm{P}(7, 0, 0, 0, 0)$. Hence, we can apply Lemmas A.1 and A.2 to argue that $\mathrm{Val}^\sigma(o^j_{I(i)}) \sqsubset \mathrm{Val}^\sigma(r^j)$. $\square$

Appendix I. The vertices $h^l_{i,0}$

In this section we show that the vertices $h^l_{i,0}$ correctly switch to the outgoing edge specified by $\chi^{B,K,j}_{m+1}$. The first lemma considers the case where $l = j$, while the second lemma considers the case where $l = 1 - j$.

Lemma I.1. *Let σ be a strategy that agrees with $\chi^{B,K,j}_m$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m \leq \mathrm{Delay}(j, K) - 1$. For each $i \in \textsc{Input}/\textsc{Output}$, the greedy all-switches strategy improvement algorithm will switch $h^j_{i,0}$ to $\chi^{B,K,j}_{m+1}(h^j_{i,0})$.*

Proof. We must show that the most appealing edge at $h^j_{i,0}$ should be $h^j_{i,1}$. Observe that both $h^j_{i,1}$ and $h^j_{i,2}$ move directly to r^j and r^{1-j}, respectively. Moreover, the priorities assigned to these vertices are strictly smaller than $\mathrm{P}(7, 0, 0, 0, 0)$. Since $h^j_{i,1}$ moves to r^j, we can apply part 2 of Lemma A.2 to prove that $h^j_{i,1}$ is the most appealing edge at $h^j_{i,0}$. $\square$

Lemma I.2. *Let σ be a strategy that agrees with $\chi^{B,K,j}_m$ for some $K, B \in \{0, 1\}^n$, some $j \in \{0, 1\}^n$, and some m in the range $1 \leq m \leq \mathrm{Delay}(j, K) - 1$. The greedy all-switches strategy improvement algorithm will switch $h^{1-j}_{i,0}$ to $\chi^{B,K,j}_{m+1}(h^{1-j}_{i,0})$.*

Proof. We must show that the most appealing edge at $h^j_{i,0}$ should be $h^j_{i,1}$. Observe that both $h^{1-j}_{i,1}$ and $h^{1-j}_{i,2}$ move directly to r^{1-j} and r^j, respectively. Moreover, the priorities assigned to these vertices are strictly smaller than $\mathrm{P}(7, 0, 0, 0, 0)$. We will use these fact in order to apply Lemma A.2 in the following case analysis. According to Equation (4.9), there are two cases to consider. Since $h^j_{i,1}$ moves to r^j, we can apply part 2 of Lemma A.2 to prove that $h^j_{i,1}$ is the most appealing edge at $h^j_{i,0}$. $\square$

Appendix J. Input/output gates

In this section we show that the vertices in the input/output gadgets correctly switch to the outgoing edge specified by $\chi_{m+1}^{B,K,j}$. The first two lemmas deal with the case where the input/output gadget for circuit j resets. Note that this occurs one iteration later than the rest of the vertices in circuit j, which is why we prove separate lemmas for this case. Otherwise, the input/output gadgets behave as if they are Not gates, so the proofs that we have already given for the Not gates can be applied with only minor changes. These is formalized in the final two lemmas of this section. The first of these lemmas considers the input/output gadgets in circuit j and the second considers the input/output gadgets in circuit $1 - j$.

Lemma J.1. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and for $m = 1$. For each* Input/Output*-gate i, and each l int the range $1 \le l < 2k + 4n + 6$, greedy all-switches strategy improvement will switch $t_{i,l}^j$ to $\chi_{m+1}^{B,K,j}(t_{i,l}^j)$.*

Proof. We must show that the edge to z^j is the most appealing edge at $t_{i,l}^j$. All paths that start at $t_{i,l}^j$ and follow σ will eventually arrive at r^{1-j}, either via y^j, or via p_i^j. On the other hand, the path that starts at z^j moves directly to s^j. Therefore, part 2 of Lemma A.2 implies that the edge to z^j is the most appealing edge at $t_{i,l}^j$. $\qquad\square$

Lemma J.2. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and for $m = 1$. For each* Input/Output*-gate i greedy all-switches strategy improvement will switch d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. This proof uses the same argument as the proof of Lemma J.1, because all paths that start at d_i^j will eventually arrive at r^{1-j}.

$\qquad\square$

Lemma J.3. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and for m in the range $2 \le m \le \mathrm{Delay}(j,K) - 1$. For each* Input/Output*-gate i, and each l int the range $1 \le l < 2k + 4n + 6$, greedy all-switches strategy improvement will switch $t_{i,l}^j$ to $\chi_{m+1}^{B,K,j}(t_{i,l}^j)$ and d_i^j to $\chi_{m+1}^{B,K,j}(d_i^j)$.*

Proof. Since $m \ge 2$, we have that $\sigma(y^j) = r^j$ and $\sigma(z^j) = s^j$. Hence, both $t_{i,l}^j$ and d_i^j behave in exactly the same way as the states $t_{i',l}^j$ and $d_{i',l}^j$ for $i' \in$ Not, with the exception that these states one step behind the Not-gate vertices, due to the delay introduced by y^j. Note, however, that this is account for by placing the edge to p_i^j on $t_{i,d(C)}^j$, rather than $t_{i,d(C)+1}^j$, as would be expected for a Not-gate with depth $d(C) + 1$. Therefore, to prove this lemma, we can use exactly the reasoning as gave for Lemmas D.1, D.2, E.1, E.2, E.3, E.5, and E.4. This is because all of the reasoning used there is done relative to r^j and s^j, and since y^j and z^j have insignificant priorities, none of this reasoning changes. $\qquad\square$

Lemma J.4. *Let σ be a strategy that agrees with $\chi_m^{B,K,j}$ for some $K, B \in \{0,1\}^n$, some $j \in \{0,1\}^n$, and for m in the range $2 \le m < \mathrm{Delay}(j,K) - 1$. For each* Input/Output*-gate i, and each l int the range $1 \le l < 2k + 4n + 6$, greedy all-switches strategy improvement will switch $t_{i,l}^{1-j}$ to $\chi_{m+1}^{B,K,j}(t_{i,l}^{1-j})$ and d_i^{1-j} to $\chi_{m+1}^{B,K,j}(d_i^{1-j})$.*

Proof. Much like the proof of Lemma J.3, this claim can be proved using essentially the same reasoning given as was given for Lemmas D.1, D.2, E.1, E.2, E.3, E.5, and E.4, because an INPUT/OUTPUT-gate behaves exactly like a NOT-gate.

In particular, note that since we have defined $\chi_{\text{Delay}(0,K)}^{B,K,0} = \chi_1^{B,K,1}$ and $\chi_{\text{Delay}(1,K)}^{B,K,1} = \chi_1^{B,K+1,0}$, the gate gadgets in circuit $1-j$ continue to have well-defined strategies, so the d_i^{1-j} will continue to act like a NOT-gate between iterations 1 and 2.

The one point that we must pay attention to is that, in the transition between $\chi_1^{B,K,j}$ and $\chi_2^{B,K,j}$, the state y^{1-j} switches from r^{1-j} to r^j. Note, however, that both $h_{i,0}^j$ and p_i^j both switch to vertex that eventually leads to r^j at exactly the same time, so all paths that exit the gadget will switch from r^{1-j} to r^j. Since almost all of the reasoning in the above lemmas is done relative to r^j, the relative orders over the edge appeals cannot change.

The only arguments that must be changed are the ones that depend on Lemma B.8. Here, the fact that the priority $P(5, i, 2k + 4n + 5, j, 0)$ is assigned to $p_{i,1}^{1-j}$ is sufficient to ensure that $\text{Val}^\sigma(r^j) \sqsubset \text{Val}^\sigma(t_{i,d(C)}^{1-j})$, so the deceleration lane will continue switching. On the other hand, since $P(5, i, 2k + 4n + 5, j, 0) < P(7, 0, 0, 0, 0)$, this priority is not large enough to cause d_i^j to switch away from e_i^j, if $B_i = 1$. $\qquad\square$