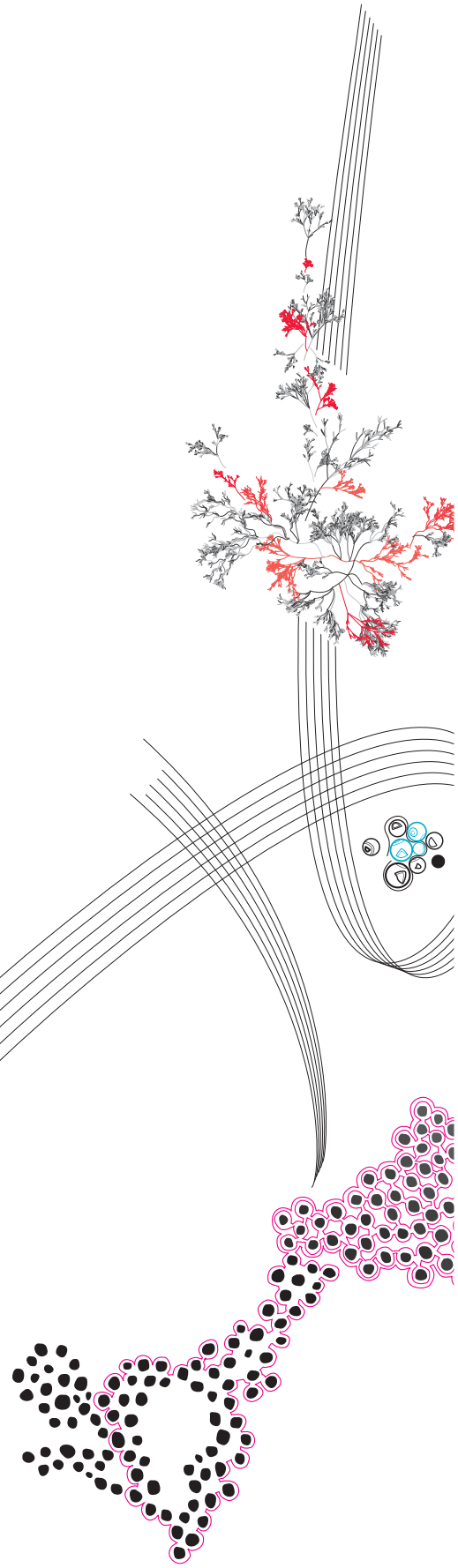


MSc Thesis Applied Mathematics

Instances with exponential running time for strategy iteration

M.T. Maat

Daily supervisor: dr. G. Loho
Graduation supervisor: prof. dr. M. J. Uetz
Committee member: dr. K. Proksch

August, 2022

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science
chair of Discrete Mathematics and Mathematical Programming

UNIVERSITY OF TWENTE.

Acknowledgements

I would like to thank God for giving me this talent. I would like to thank my supervisor for the input and the many discussions about the various subjects in my thesis. I would like to thank Alexander Hopp for the discussion. I would like to thank Marnix for proofreading the thesis. And of course, I want to thank my parents and also my friends for the *gezelligheid* while working together in the educafé.

Instances with exponential running time for strategy iteration

Matthew Maat

July 2022

Abstract

A parity game is a game played by two players on the nodes of a directed graph. Solving a parity game consists of finding out who can win and what the winning strategy is. Solving parity games is linked to many other mathematical problems, and so far there is no polynomial-time algorithm for this problem. A currently often used algorithm is strategy iteration, which iteratively improves a strategy, where decisions are taken by a so-called improvement rule. It has been shown previously that in some cases improvement rules for strategy iteration can be linked to pivot rules for the simplex algorithm. This reduction is via mean payoff games and Markov decision processes and only works for specific cases. In this thesis, an alternative relation between strategy iteration and the simplex algorithm is presented. This relation works for a broader class of parity games. Moreover, a framework for constructing examples with exponential running time for strategy iteration is presented and illustrated with some examples. Finally, a family of graphs with exponential running time for the symmetric strategy iteration algorithm is presented. For this algorithm no class of graphs was known before with superpolynomial running time.

Contents

1 Introduction

A parity game is an infinite game played between two players on the nodes of a directed graph. For various reasons, these games are interesting mathematical objects.

First of all, they are interesting from a complexity perspective. The task of solving a parity game consists of finding which player can win the parity game and what the winning strategy is. Solving parity games is a problem that lies in both complexity classes NP and coNP, and also in UP and coUP[17]. However, it is not known yet whether there exists a polynomial-time algorithm to solve these games.

Second of all, solving parity games is related to many other mathematical problems, and techniques used to solve parity games can transfer to other mathematical problems as well. Solving parity games can be reduced to mean payoff games and in some cases to Markov decision processes, and solving parity games can also be reduced to linear programming. Moreover, they play an important role in the field of formal verification. For example, solving parity games is equivalent to model checking for the modal μ-calculus[8].

1.1 History and motivation

One of the algorithms to solve parity games is the discrete strategy iteration algorithm [18]. This algorithm applies the commonly used technique of strategy iteration, or sometimes called policy iteration, to parity games. It iteratively improves a strategy, using values that are assigned to game states, until optimality is reached. When there are multiple options to improve a strategy, this choice is made according to some *improvement rule*. A number of different improvement rules have been suggested. Despite its efficiency in practice, for none of these rules it has been shown that it makes strategy iteration a polynomial-time algorithm.

For the most well-known improvement rules for discrete strategy iteration, the contrary has been shown. It was shown that for certain families of parity games, the discrete strategy iteration algorithm (with these improvement rules) requires a(n) (sub)exponential number of iterations [11, 15]. The main idea behind these constructions was to create a parity game that simulates a binary counter: certain strategies correspond to a certain binary number, and strategy iteration cycles through all the strategies corresponding to all possible binary numbers with a certain number of bits.

Linear programming is a commonly used mathematical optimization method. It involves optimizing a linear objective function subject to a number of linear inequalities. A well-known open question regarding linear programming is whether there exists a strongly polynomial algorithm for solving linear programs. This is the 9th question on Smales list of problems[27]. Probably the most promising candidate for being a strongly polynomial algorithm, and also the oldest method to solve linear programs (LPs) efficiently is the simplex method. This method maintains a so-called basic feasible solution, and iteratively improves this until optimality is reached. When there are multiple options to improve, the next basic feasible solution to move to is decided by a so-called *pivot rule*. Like in the case of discrete strategy iteration and improvement rules, many pivot rules have been suggested. Moreover, for the most well-known pivot rules, families of linear programs have been constructed for which the different pivot rules need a(n) (sub)exponential number of iterations. For most of these families, the main idea is to use an n-dimensional perturbed hypercube, often called Klee-Minty cube after the first use of such a cube [19]. The idea is then that the simplex algorithm with a pivot rule traverses all vertices of the hypercube.

So there is a similarity between the current complexity status of discrete strategy iteration and the simplex algorithm. Moreover, it would become apparent that this similarity

can be exploited. For a certain class of parity games called sink parity games, it became apparent that they can easily be related to another class of games called mean payoff games (MPGs). Specifically, this means that if discrete strategy iteration behaves a certain way on a sink parity game, then we can construct a mean payoff game where another version of strategy iteration behaves the same way. Moreover, in some cases, this mean payoff game can also be related to a Markov decision process (MDP). In that case, if strategy iteration behaves a certain way on the MPG, then the commonly used policy iteration algorithm on the MDP also behaves in the same way. Finally, there is also a relation between MDPs and linear programming, where one can construct a linear program from an MDP, where the simplex algorithm behaves similar to policy iteration on the MDP.

So, following this chain of reductions:

parity game → mean payoff game → Markov decision process → linear program

it was shown [11, 15] that the random-edge and random-facet pivot rules for the simplex method need a (sub)exponential number of iterations for some family of linear programs. This family was constructed by starting with a family of parity games for which discrete strategy iteration behaves like a binary counter, and then using these reductions to create a linear program from these.

Of course, this provides a novel way to analyze both simplex pivot rules and improvement rules for discrete strategy iteration. The first and the third reduction, from a parity game to an MPG, and from an MDP to a linear program, are simple and well-defined. However, the second reduction from and MPG to an MDP is very tailor-made and only works in specific cases. This motivates the question whether there exists a more general reduction from parity games to linear programs that maintains the similarity between strategy iteration and the simplex algorithm.

1.2 Outline of the thesis

The main research question of this thesis is the following:

> Is there a structured way to construct examples with exponential running time for strategy iteration in parity games, and, related to that, for pivot rules in the simplex algorithm for linear programming?

Indeed, during this master thesis project, a reduction was found that works for a broader class of parity games than the existing reduction. Also, some structures were found that can make it easier to come up with families of parity games on which discrete strategy iteration needs exponential running time. Now, we move to the outline of this report.

First, we discuss two structures for parity games. These are called the 'binary counter' and the 'reverse binary counter.' The idea is that for a graph with such a structure the discrete strategy improvement algorithm behaves like a binary counter for a certain improvement rule, under some conditions. Both structures consist of a number of unspecified gadgets, which can then be different for different improvement rules. The effectiveness of these structures is illustrated by constructing gadgets for a number of logical improvement rules. Using this, we show that there are families of parity games with this structure that indeed show exponential running times for these improvement rules.

Next, we look at a new algorithm for solving parity games, called subgame iteration. This algorithm has some similarities with discrete strategy iteration and the longest shortest path problem[4]. The valuations of these algorithms are partly the same. The interesting part about this algorithm is that we can show that the iterations of this algorithm

on any parity game can be related to the iterations of the simplex algorithm on a certain linear program. This similarity lies in relating values of paths in the graph to values of the inverse of the basis of the LP. So this means that we have a reduction from solving a parity game to solving a linear program with the simplex algorithm. Although the size of the coefficients can be large (doubly exponential), this reduction works for any parity game.

To illustrate this reduction, we look at a number of classical pivot rules for the simplex algorithm. We show how this new reduction can be used to construct families of linear programs for which the simplex algorithm with these pivot rules needs an exponential number of iterations.

Next, we look at a variant of strategy iteration called symmetric strategy iteration, for which it was previously not known whether it solves parity games in polynomial time. We construct a family of parity games for which this algorithm requires an exponential number of iterations. The main idea is that the algorithm behaves like a counter in Gray code, which is a form of binary representation of numbers.

Finally, some things can be found in the appendices. There is an implementation of the counterexamples from Section 6; a table of the most commonly used symbols in this thesis; an example of a parity game that is playable; and a section that describes a separate idea, which is a generalized version of improvement rules in strategy iteration.

1.3 Related work

Klee-Minty cubes

Many pivot rules have been constructed for the simplex algorithm. The first example of linear programs where the simplex algorithm takes an exponential number of iterations for a pivot rule was given by Klee et al. [19] for the largest coefficient pivot rule. Their example is a slightly perturbed version of the hypercube in n dimensions, also called the Klee-Minty cube. This idea formed the basis of many similar proofs for most of the commonly used pivot rules. Some of the more well-known are the proof by Jeroslow [16] for the maximum improvement rule; the proof by Avis and Chvátal [3] for the least index rule; and the proof by Goldfarb and Sit [13] for the steepest edge pivot rule.

Strategy iteration in parity games

A commonly used algorithm for solving parity games (finding winning strategies and determining the winner) is discrete strategy iteration. This algorithm was presented by Jurdzinski and Vöge [18], and it works as follows: if we have some player 0 strategy, then each node receives a valuation from that strategy. If then there are some player 0 nodes where we can choose another edge such that the successor of the node has a higher valuation than the current successor, we call it an *improving edge*. Strategy iteration then, similar to the simplex algorithm, requires an *improvement rule* to decide which set of improving edges we augment the current strategy by. If there are no more improving edges after a number of iterations, the strategy is optimal. An obvious option is to choose all improving edges at once, this we call the *switch-all* improvement rule. Later, Schewe [23] presented an efficient method to find among all possible subsets of improving edges the subset that increases the valuation of all nodes the most. This we call the *switch-best* improvement rule. Two more recent variants of strategy iteration are the snare-memorizing algorithm by Fearnley [10], which memorizes certain actions for a player that 'trap' the other player, and symmetric strategy improvement by Schewe et al. [25], which maintains strategies for both players and improves them simultaneously.

Polynomial-time reductions

There are many reductions between solving parity games and other mathematical problems. First of all, parity games can be reduced to infinitary payoff games as shown by Puri [22]. Using this reduction to infinitary payoff games, Friedmann [11] and Hansen [15] showed for a few specific classes of parity games that they can be reduced to linear programs, where pivot rules for linear programs correspond to improvement rules for strategy iteration. Also, Schewe [24] showed a reduction from any parity game to a linear program. These reductions also imply that strategy iteration algorithms for infinitary payoff games (see Puri [22], Ludwig [20] and Björklund et al. [4]) also apply to parity games.

Finally, many problems in theoretical computer science are dependent on solving parity games. For example, solving parity games is equivalent to model checking for the modal μ-calculus, see Emerson et al. [8].

Worst-case examples

There are some worst-case examples known for discrete strategy iteration in parity games. Two examples were given by Friedmann [11] where the switch-all and switch-best improvement rules yield an exponential number of iterations. This was done by constructing families of parity games that behave like a binary counter for their respective improvement rules. With a similar technique, other parity games were constructed with (sub)exponential numbers of iterations for various randomized improvement rules. This also implies a (sub)exponential number of iterations for the simplex algorithm with the *random-edge* and *random-facet* pivot rules. Moreover, Disser et al. [7] constructed a parity game which acts as a binary counter for strategy iteration with the *least-entered* improvement rule. This rule applies an improving switch that has been done the least amount of times. This also implies exponential running time for the least-entered pivot rule of the simplex algorithm. Björklund et al. [4] presented an example with exponential running time for a variant of strategy iteration in mean payoff games. This example has a somewhat similar structure to the examples presented in this thesis. Finally, van Dijk [28] showed exponential lower bounds on the running time of a range of different algorithms to solve parity games. This was done using a family of games that simulate two binary counters. Many results in this thesis are aimed at generalizing the before mentioned worst-case examples and at creating more widely applicable counterexamples.

Combinatorial simplex algorithms

A technique that is related to the techniques used in this thesis is tropical linear programming. This is described by Allamigeon et al. [2]. Moreover, this technique can be used to solve mean payoff games, which is described by Allamigeon et al. [1].

2 Preliminaries

2.1 Graphs

A directed graph $G = (V, E)$ consists of two sets. A set of elements called *vertices*, which is V, and a set of ordered pairs of vertices, which is called the *edge set* E. For an edge (v_1, v_2) with $v_1, v_2 \in V$, we call v_2 a *successor* of v_1. Also, for edge (v_1, v_2), we call v_1 the *tail* and v_2 the *head* of the edge. We call an edge of the form (v_1, v_1) a *loop*. Sometimes multi-edges (the same edge occurring multiple times in E) are allowed, but in this thesis we assume there are no multi-edges. We say a directed graph is *complete* if every edge in the graph exists, so $(v, w) \in E$ for every possible $v, w \in V$.

We call a sequence of unique vertices $P = (v_1, v_2, v_3, \ldots, v_k)$ a *path* in G if the edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ are all in E. We call v_1 the starting vertex and v_k the end vertex of the path. In this thesis specifically, we call a vertex v *in* path P if $v \in \{v_1, v_2, \ldots, v_{k-1}\}$; so we do not count the end vertex as being in the path. A *closed path* or cycle is the same as a path, except that $v_1 = v_k$, but all other vertices of the path are still unique. A *walk* is the same as a path, except that we do not require the vertices to be unique.

2.2 Multisets

A multiset is the same as a set, except that elements of a multiset may occur multiple times in the multiset. By the *multiplicity* of an element we mean the number of times an element occurs in the multiset. We denote the multiplicity of element a in multiset A by $m_A(a)$. Two possible operations on multisets are the following:

- Multiset addition: if $C = A \uplus B$, then $m_C(a) = m_A(a) + m_B(a)$ for every element a.
- Multiset difference: if $C = A \backslash B$, then $m_C(a) = \max(m_A(a) - m_B(a), 0)$ for every element a.

2.3 Parity games

An *infinitary payoff game* is a game that is played on a directed graph $G = (V, E)$. Two players, called player 0 and player 1 (or player Odd and player Even), each control part of the nodes of the graph. We say $V = V_0 \cup V_1$ with $V_0 \cap V_1 = \emptyset$ and player i controlling V_i. Furthermore, each node v is associated with a priority $p(v) \in \mathbb{N}$, and there is one starting node v_0, which has a pebble on it. At the start of a turn, the pebble is always on a node $v \in V_i$, and player i then chooses one of the outgoing edges of node v. After that, the pebble moves to the head $w \in V_j$ of the chosen edge, after which the turn is passed to player j. We assume that each node has at least one outgoing edge, so that the game continues indefinitely. The winner of the game is determined according to some reward function r that assigns a value or a winner to the sequence of vertices encountered in the game. A *parity game* is a class of infinitary payoff games. The winner of the parity game is determined by the highest priority among the vertices that the pebble moves onto an infinite number of times. If that priority is even, player 0 wins, and otherwise player 1 wins.

The following results are known about parity games:

Lemma 2.1. *For any starting node v, either player 0 or player 1 has a winning strategy. Hence we can partition V into W_0 and W_1, where W_i is the set of nodes from where player i has a winning strategy.*

Lemma 2.2. *There is a memoryless strategy for player 0 (i.e. the choice of edge is only determined by the current location of the pebble) such that player 0 wins from every starting position in W_0. The same holds for player 1 and W_1.*

Considering Lemma 2.2, we can see that the winner of a parity game on some graph G is the same as the winner of the following game played on the same graph:

Instead of choosing an edge every turn, we let both players choose exactly one outgoing edge for each of the vertices that they control at the start of the game. We then let the pebble move along the chosen edges, and end the game when the pebble enters a node for the second time. The trajectory of the pebble the consists of a cycle and possibly some path leading up to the cycle. We then decide the winner according to the parity of the largest priority occurring in the cycle: player 0 winning if it is even, and player 1 winning otherwise. For simplicity, we only consider this finite version of the game and refer to it as a parity game for the rest of this thesis. We can then define a strategy σ for player 0 to be a function $\sigma : V_0 \to V$ with $(v, \sigma(v)) \in E \; \forall v \in V_0$, and a similar definition for player 1 strategies.

Furthermore, we may without loss of generality assume that the priorities of the nodes are unique, as otherwise, if some nodes have the same priority p, we could let one of them have priority p and add 2 to the priorities of all other nodes with priority $\geq p$. One can easily verify that whatever strategies are played by the players, this modification does not change which player wins. Moreover, we can assume that all priorities are between 1 and $2n$, if n is the number of nodes in the graph. That is because if all priorities are unique and the lowest priority is more than 2, we can subtract 2 from all priorities, and if the lowest priority is 1 or 2, and the highest priority is more than $2n$, then there must be two priorities p, q with $p \leq q - 3$ and no priorities in between p and q, hence we can decrease all priorities of q and larger by 2 without changing the outcome of the game.

In the rest of this thesis, we use circles for nodes controlled by player 0, and squares for nodes controlled by player 1. An example of a parity game is shown in Figure 1. The reader is also invited to play the game included in Appendix C.

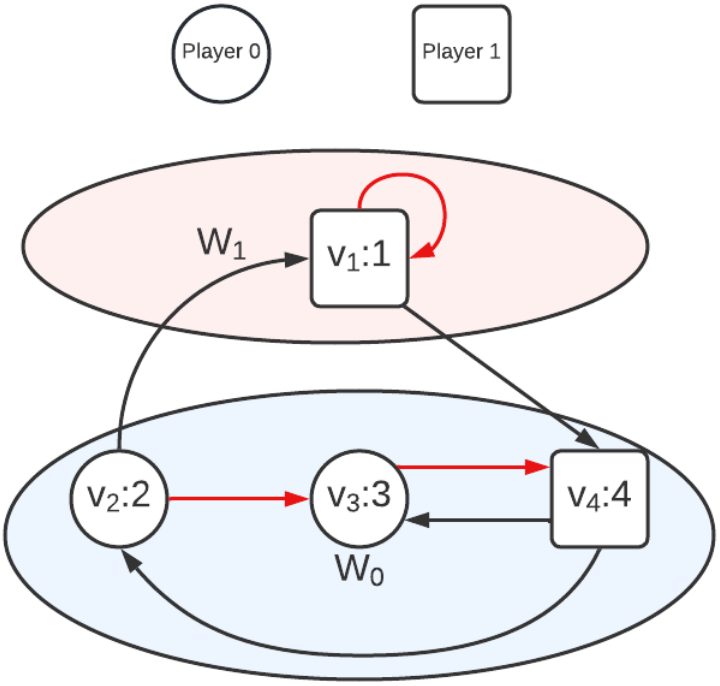


FIGURE 1: A parity game, with encircled the sets W_0 and W_1, that are winning for player 0 and player 1, respectively. The winning strategies for player 0 in W_0 and for player 1 in W_1 are marked in red.

2.4 Discrete strategy iteration

The technique of *strategy iteration* or *policy iteration* is commonly used for finding optimal strategies/policies in infinitary payoff games or Markov decision processes. The main idea is as follows: start with some initial strategy/policy σ_0. We associate with any strategy σ some valuation $\Xi_\sigma(v)$ for every vertex v. Then in every iteration, we check if there are any nodes v for which we could choose another edge such that the valuation of its successor w is higher than that of the current choice $\sigma(v)$. We call such an edge an *improving edge*. Then we construct strategy σ_{i+1} from strategy σ_i by augmenting the strategy with (part of) the improving edges. If the valuation function is chosen well, this yields a sequence of strategies of which the valuation at the vertices is nondecreasing, and hence the sequence is finite (since there are only finitely many strategies possible). Furthermore, the valuation function should have the property that the current strategy is optimal if and only if there are no more improving edges, hence the iterations end at an optimal strategy.

We define a valuation of a player 0 strategy σ for a parity game according to Jurdzinski and Vöge [18]. Let Q be the set of priorities occurring in G. The valuation is a function $\Xi_\sigma : V \to Q \times \mathcal{P}(Q) \times \mathbb{Z}_{\geq 0}$.

The idea of this valuation is to keep track of three things, in decreasing importance: the highest priority of the nodes in the cycle we end up in, the priorities we meet when going to the cycle, and the length of the path towards the cycle. Player 0 will first of all try to get to a cycle with even highest priority, and the higher that priority the better. This is because of course he wants to win, and make it difficult for player 1 to counter his strategy. As a secondary goal, he tries to enter nodes with high even priority on the way to the cycle, and avoid nodes with high odd priority. As a third goal, he tries to make the

path towards the cycle long if he is losing (trying to create other cycles) and short if he is winning.

To compute the valuation, we find an optimal counterstrategy $\bar{\sigma}$ for player 1 (which we define in a moment). Note that, with the choice of any player 0 strategy σ and player 1 strategy τ, the course of the game is determined, and we get a *play* $P = (v_1, v_2, v_3, \ldots, v_l)$, where v_l is the only node that occurs twice in the play, since we stop if a node is encountered twice. Suppose v_k is the node with the highest priority in the cycle part of P. Let $\lambda(P) = p(v_k)$ be the highest priority occurring in the cycle part of P. We define $Prefix(P) = (v_1, v_2, \ldots, v_{k-1}, v_k)$, the (possibly closed) path from the starting node towards the node with highest priority. We define

$$\pi(P) = \{p(v_i) |\ v_i \in Prefix(P) \wedge p(v_i) > \lambda(P)\},$$

the set of high priorities on the path towards the cycle, and $\#(P) = k - 1 = |Prefix(P)|$, the length of the path towards v_k. We can then assign a value $\Theta_{\sigma\tau}(v)$ to the strategy pair σ, τ. This is given by $\Theta_{\sigma\tau}(v) = (\lambda(P), \pi(P), \#(P))$, where the play P results from σ and τ when the game starts in node v. The valuation Ξ_σ will be given by $\Theta_{\sigma\bar{\sigma}}$, but first we need to define what an optimal counterstrategy is.

To compare strategies, we want to compare different values of $\pi(P)$ by how good they are for player 0. To do so, we define a linear order $\preceq$ on $\mathcal{P}(Q)$, the powerset of the priorities. If $B_1 = \pi(P_1)$ and $B_2 = \pi(P_2)$, then let p be the largest element of $B_1 \triangle B_2 = (B_1 \backslash B_2) \cup (B_2 \backslash B_1)$. We then say $B_1 \preceq B_2$ if one of the following holds:

- p is even and $p \in B_2 \backslash B_1$
- p is odd and $p \in B_1 \backslash B_2$
- p does not exist (hence $B_1 = B_2$)

Moreover, we can extend this linear order to $\mathcal{M}(Q)$, the set of multisets of Q. In that case, if the elements of B_1 and B_2 have multiplicity at most 2, we can say $B_1 \preceq B_2$ if and only if $\sum_{p \in B_1} (-3)^p \leq \sum_{p \in B_2} (-3)^p$. This equivalence will be important in Section 4.

We now define a linear order $\trianglelefteq$ on the valuations of the nodes, following the three previously mentioned goals of player 0. We say that $(\lambda_1, B_1, n_1) \trianglelefteq (\lambda_2, B_2, n_2)$ if the following holds:

- λ_2 is better for player 0, i.e. $\lambda_1 \cdot (-1)^{\lambda_1} < \lambda_2 \cdot (-1)^{\lambda_2}$. This happens if λ_1 is odd and λ_2 even, or λ_1 and λ_2 are both odd and λ_1 is larger, or they are both even and λ_2 is larger.
- $\lambda_1 = \lambda_2$ and $B_1 \prec B_2$
- $\lambda_1 = \lambda_2$, $B_1 = B_2$, and one of the following holds:
  - $\lambda_1 (= \lambda_2)$ is odd and $n_2 \geq n_1$
  - λ_1 is even and $n_1 \geq n_2$

Now we define our valuation of strategies such that the resulting value Θ of v is minimal (according to the $\trianglelefteq$-ordering):

$$\Xi_\sigma(v) = \min_{\trianglelefteq}(\Theta_{\sigma\tau}(v) :\ \tau \text{ player 1 strategy})$$

We call the strategy τ an *optimal counterstrategy* to σ for the game starting in v if it is a strategy for which the minimum in the above equation is attained. We have the following result from Jurdzinski and Vöge [18]:

Lemma 2.3. *For any strategy σ, there exists a single strategy $\bar{\sigma}$ for player 1 that is an optimal counterstrategy to σ for the parity game starting in any node $v \in V$. In other words, there is a player 1 strategy $\bar{\sigma}$ such that $\Theta_{\sigma\bar{\sigma}}(v) = \Xi_\sigma(v) \forall v \in V$. This counterstrategy can be computed in polynomial time.*

We call such a strategy $\bar{\sigma}$ an *optimal counterstrategy* to σ. Now we can formulate the strategy iteration algorithm more precisely. For any player 0 strategy σ we can calculate the optimal counterstrategy $\bar{\sigma}$, and from the resulting play P we get our valuation $(\lambda(P), \pi(P), \#(P))$. We can then construct the set of improving edges:

$$I_\sigma = \{(v, w) \in E |\ v \in V_0, \Xi_\sigma(w) \triangleright \Xi_\sigma(\sigma(v))\}$$

Then we view an improvement rule as a rule that selects some subset $I \subseteq I_\sigma$ with no more than 1 outgoing edge per vertex and nonempty if possible. We define the operation $\text{Switch}_I(\sigma)$ as the strategy resulting from σ when the strategy in the vertices that have an outgoing edge in I is changed to the edge that is in I. Strategy iteration can then be formulated as

Algorithm 1 Strategy iteration

```
start with some strategy σ and find I with improvement rule
while I ≠ ∅ do
    σ ← Switch_I(σ)
    find new I with improvement rule
end while
```

An example of an iteration of strategy iteration is given in Figure 2. There, the optimal counterstrategy to the red strategy is given by $\bar{\sigma}(v_1) = v_1$ and $\bar{\sigma}(v_4) = v_2$, since this is the only way in which player 1 can win from every starting node. The valuation of v_1 is $(1, \emptyset, 0)$, since the player starting from v_1 with σ and $\bar{\sigma}$ ends in the cycle with v_1 as the highest priority. The path from v_1 to v_1 is the trivial path (v_1), and recall we do not count the endpoint of a path as being in the path. Hence the path length is 0 for $\Xi_\sigma(v_1)$. For the other nodes, the cycle is also the one with $p(v_1)$ as its highest priority, so the first component of the valuation is 1. The path and the path length are given by the path towards v_1. Note that $\Xi_\sigma(v_3) \triangleright \Xi_\sigma(v_1)$, as the first component is the same and $\{2, 3, 4\} \succ \emptyset$. Hence edge (v_2, v_3) is improving. So $I_\sigma = \{(v_2, v_3)\}$. In the next iteration, we will have strategy σ' with $\sigma'(v_2) = v_3$ and $\sigma'(v_3) = v_4$.

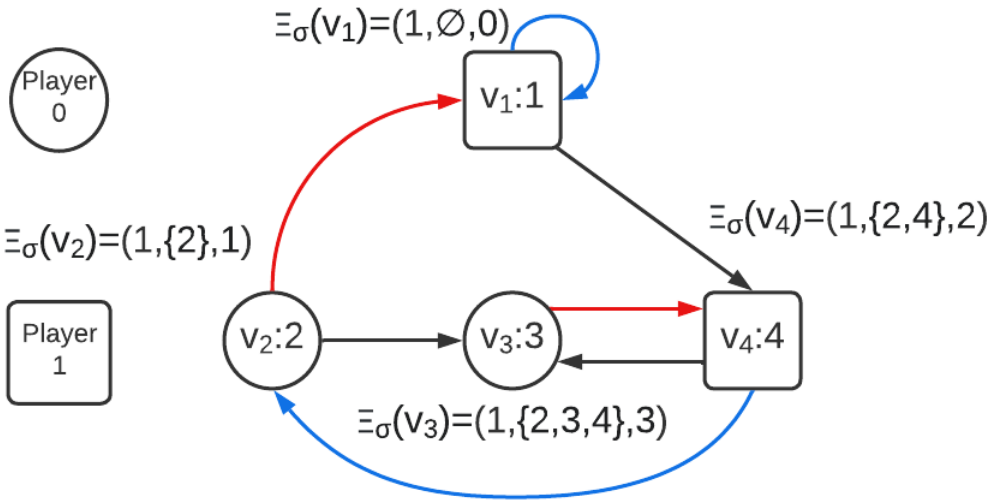


FIGURE 2: An iteration in strategy iteration. In red: current strategy σ. In blue: optimal counterstrategy $\bar{\sigma}$.

The following lemmas can be derived from Jurdzinski and Vöge [18], which imply correctness of the algorithm:

Lemma 2.4. *For any valid nonempty $I \subseteq I_\sigma$, we have $\Xi_\sigma(v) \trianglelefteq \Xi_{\mathrm{Switch}_I(\sigma)}(v)$ for all $v \in V$ and $\Xi_\sigma(v) \triangleleft \Xi_{\mathrm{Switch}_I(\sigma)}(v)$ in at least one node.*

Lemma 2.5. *If $I_\sigma = \emptyset$, then σ is an optimal strategy for player 0 (hence winning on every node in W_0).*

2.5 Sink parity games and improvement rules

A special class of parity games that was introduced by Friedmann [11] is the class of sink parity games.

Definition 2.6. A *sink parity game* is a parity game that has the following properties:

1. (sink existence) There exists a sink node $\top$ such that $\top$ has the smallest priority in the game and $\top$ only has one outgoing edge, to itself. We assume its priority equals 1.

2. (sink seeking) If σ is an optimal (w.r.t $\trianglelefteq$ for all nodes) strategy for player 0, and $\bar{\sigma}$ an optimal counterstrategy for player 1, then the resulting play ends up in $\top$ (i.e. the first element of $\Xi_\sigma(v)$ is $p(\top)$ for all $v \in V$).

The reason that sink parity games are interesting for strategy iteration is because the valuations are simpler than for general parity games. Recall that the valuation $\Xi_\sigma(v)$ for parity game strategy iteration was $(\lambda(P), \pi(P), \#(P))$ for the play P resulting from σ and optimal counterstrategy $\bar{\sigma}$ starting from v. Suppose the initial strategy σ_0 already 'ends up' in the sink node $\top$, which means that the first component of $\Xi_{\sigma_0}(v)$, which is $\lambda(P)$, equals $p(\top) = 1$ for every node $v \in V$. Then, because the optimal player 0 strategy also cannot create better cycles than going to $\top$, every strategy encountered in strategy iteration will have $p(\top)$ as the first component in the valuation of every node. So we can ignore the

first element of the valuation. Moreover, since $\top$ has the lowest priority in the game, every element in the path towards $\top$ ends up in the second component of the valuation. Therefore, if two nodes v and w have the second component $\pi(P)$ of the valuations $\Xi_\sigma(v)$ and $\Xi_\sigma(w)$ the same, then also the third component, the path length $\#(P)$, is the same. So also the third component of the valuation is not relevant for these games.

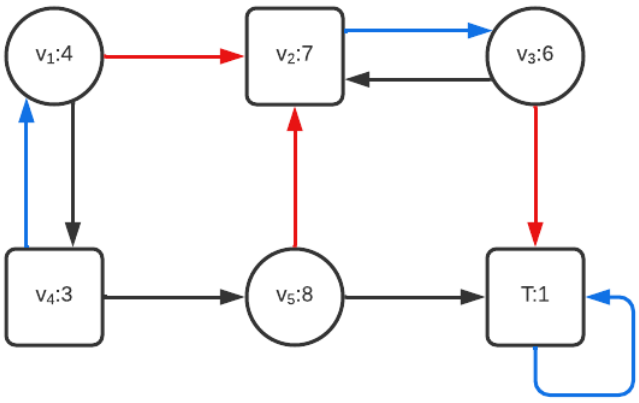


FIGURE 3: Example of a sink parity game. An initial strategy σ_0 for strategy iteration is highlighted in red, and optimal counterstrategy $\bar{\sigma_0}$ in blue. Even though there is a play possible that has a better cycle for player 0, namely $v_1 - v_4$ and also a play possible with the better cycle for player 1, namely $v_2 - v_3$, still if both players play optimally, the play ends in $\top$.

Sink parity games play a huge role in showing exponential lower bounds for the complexity of strategy iteration with certain improvement rules. Some improvement rules that are proposed in literature are the following:

- *switch-all* [18]: switch all improving edges.
- *switch-best* [23]: switch the subset of improving edges that has the best resulting valuation.
- *random-edge*: switch an improving edge at random.
- *least-entered* [29]: apply, with some tiebreaker, the improving switch that has been performed the least.

Other variants of strategy iteration (one could debate whether these are just improvement rules or completely new algorithms) are the following:

- *random-facet* [21]: exclude a random edge that is not in the current strategy and recursively solve the resulting game.
- *snare-memorizing* [9]: remember some winning subgames for player 0 and solve these to choose new edges.
- *symmetric strategy iteration* [25]: maintain strategies σ and τ for player 0 and 1, respectively. Switch, for both players simultaneously, the edges that are both improving and in an optimal counterstrategy to the other player's strategy.

For all of these rules, except the last one, (sub)exponential lower bounds for the (expected) number of iterations have been shown (see [11, 12]). The main idea behind those is constructing a sink parity game that behaves like a binary counter when applying strategy iteration.

2.6 Linear programming and the simplex algorithm

A *linear program* (LP) is an optimization problem that can be written in the following form:

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

where $x, c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $m < n$ and $b \in \mathbb{R}^m$. We assume A has full row rank. We call the set $\{x \in \mathbb{R}^n : Ax = b \wedge x \geq 0\}$ the *feasible set*. So the optimization problem is to find the x in the feasible set with minimal cost $c^T x$. The feasible set is a so-called polyhedron, and can be seen as a convex set in $\mathbb{R}^n$ bounded by a number of hyperplanes.

We call an $m \times m$ matrix B a *basis* if it consists of m linearly independent columns of A. We can then split a vector x into two parts: $x_B \in \mathbb{R}^m$, containing the m elements of the vector corresponding to the m columns of B, and $x_N \in \mathbb{R}^{n-m}$, containing the remaining elements of x. Then a vector x is called the corresponding *basic variable* of B if $x_N = 0$ and $Bx_B = b$. Also, x is called basic feasible if it is basic and $x_B \geq 0$. A basic feasible solution can be seen as a vertex of the feasible polyhedron.

The simplex algorithm now makes use of the fact that if an optimum exists (and if there exist basic feasible solutions) then the optimum is attained in some basic feasible solution. The idea is then to start with some basic feasible solution x and its associated basis B. So we start in a vertex of the polytope. We then repeat the following: we move over an edge of the polytope in a direction that improves the objective value, until we arrive at another vertex. In terms of the variables, this means we choose some index i such that A_i, the i-th column of A is not contained in the basis, and such that $c_i - c_B^T B^{-1} A_i < 0$.

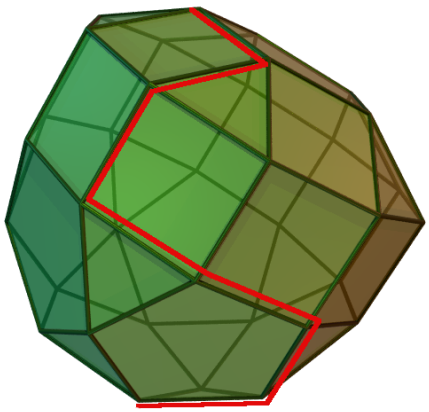


FIGURE 4: Visualization of the simplex method, when vertices on the top have a better objective value than on the bottom. The simplex method keeps jumping from a vertex to an adjacent vertex, increasing the objective value in every step. From: [26].

If there are multiple options to improve, we choose i according to some *pivot rule*. The pivot rule may be seen as a function with as input the matrix A and the basis B, and as output the index of the chosen improving variable to add to the basis. We then add A_i to the basis and remove another column such that the resulting basic variable is still feasible. We call the variable x_i that corresponds to the column that is added to the basis the *entering variable*, and the variable corresponding to the column that is removed the *leaving variable*. Then we have another basic feasible solution with a better objective value. We do this by choosing the index j with the lowest nonnegative value of $\frac{(B^{-1}b)_j}{(B^{-1}A)_{ji}}$, and then the j-th variable of the basis will leave the basis. The newly obtained solution improves the objective function. We continue this process until there is no improvement possible. If $c_i - c_B^T B^{-1} A_i \geq 0$ for all indices i, we cannot improve and the current solution is optimal.

How the simplex algorithm performs is strongly dependent on the pivot rule that is used. Of course, one would like to use a pivot rule that leads to the optimal solution as quickly as possible. Some natural choices are for example the steepest descent rule, that chooses the index with the lowest (most negative) reduced cost, or the steepest edge rule, that chooses the edge that is the 'steepest' in $\mathbb{R}^n$ (see [6]). However, for many pivot rules, it has been shown that they have an exponential worst-case performance. Moreover, it is not known whether there even exists a pivot rule that leads to a polynomially bounded number of iterations.

3 Improvement rules for strategy iteration in parity games

This section discusses the first result, which is the two structures for creating examples with exponential running time for discrete strategy iteration. As mentioned in the previous section, the strategy iteration algorithm requires an improvement rule to decide which improving moves to apply. Of course, a good improvement rule leads to a small number of iterations. However, deciding which rule is best is not trivial. With regard to worst-case behavior, the question whether there exists an efficient improvement rule, i.e. one that leads to a polynomial-time worst case complexity, is still open. This section adds a number of improvement rules to the 'list' of rules that are known to have exponential worst-case running time. It also provides a framework that can make proving exponential worst-case complexity easier. For now, we focus only on deterministic memoryless single-switch improvement rules are considered. Here deterministic means that the rule always makes the same choice given the current strategy, memoryless means it only takes the current strategy and game into account and not the previous iterations, and single-switch means it only chooses to make one switch per iteration. This is for simplicity, but also because there is often a nice connection to memoryless pivot rules in the simplex algorithm. Moreover, we look at a number of examples of deterministic single-switch memoryless improvement rules, and apply the frameworks presented in this section to provide lower bounds on the complexity of strategy iteration with these rules.

We consider two structures, which we call the binary counter and the reverse binary counter. Both these structures consist of smaller subgraphs which we call gadgets. We do not specify these gadgets initially, and what these gadgets look like should depend on the improvement rule. The task for someone who wants to prove exponential running time for a certain improvement rule is then to devise a gadget that has the right properties for that improvement rule. Initially, we define a condition under which strategy iteration needs exponentially many iterations on graphs with such a (reverse) binary counter structure. We first present the binary counter structure, and then use some specific gadget designs to prove lower bounds for some specific improvement rules. As the name suggests, the strategies that strategy iteration passes through simulate a binary counter, where the bits are formed by the choices player 0 makes in the gadgets. Next, we look at the reverse binary counter structure, which is similar to the binary counter, except that it 'counts' in reverse. We also look at some lower bound proofs for improvement rules using the reverse binary counter. The improvement rules in this section are of interest because most of them are simple and logical choices, but also also because we show a connection with pivot rules for the simplex algorithm in Section 5.

3.1 Structure for counterexamples: binary counter

The main theme of the framework of the (reverse) binary counter is, as the name suggests, simulating a binary counter. The idea is that there are parts of the graph that represent n bits of a binary counter. We then encode the binary representation of a number between 0 and $2^n - 1$ in some strategy σ. The goal is to let the strategies encountered by strategy iteration simulate a binary counter; so we start with a strategy that encodes 0 in binary. Then, we want strategy iteration to go to a strategy that encodes 1 in binary, then 2, and so on, until $2^n - 1$. This, of course, implies exponential running time for strategy iteration (given, of course, that the number of nodes and edges of the graph is linear in n).

The binary counter structure for a parity game G_n is given in Figure 5. It is a so-called sink parity game (see Definition 2.6). It consists of n gadgets $A_1, A_2, \ldots, A_n$, of n gadgets $D_1, D_2, \ldots, D_n$ and of nodes a_{n+1}, d_{n+1}, x and y. The choices of the strategy σ in gadgets

A_i and D_i together form the i-th bit in the binary counter. Gadget A_i contains node a_i and D_i contains d_i for $i = 1, 2, \ldots, n$. These nodes a_i and d_i are the nodes with the highest priority in their respective gadgets, and therefore they determine whether it is good or bad to move towards a gadget. The number N used in the priorities is assumed to be an integer larger than the number of nodes in G. The node x is the sink node, where the play will end if both players play optimally. The function of node y is to make the path second component of the valuation better than the empty set, of which the function will become apparent in Section 5.

Suppose we start the game from a_i, player 0 plays strategy σ and player 1 plays optimal counterstrategy $\bar{\sigma}$, and the resulting play passes through the edge from gadget A_i to gadget A_{i+1}. To ease notation, we denote this by $\sigma(A_i) = a_{i+1}$ (or , if $i = n$ and we pass the edge from A_n to a_{n+1}, we also say $\sigma(A_n) = a_{n+1}$). Likewise, we say $\sigma(A_i) = d_{i+1}$ if the play passes the edge from A_i to D_{i+1}, or from A_n to d_{n+1} if $i = n$. We define $\sigma(D_n)$ in a similar manner. In short, the encoding of numbers in binary by strategies happens by making $\sigma(A_i) = \sigma(D_i)$ if the i-th bit is 1, and $\sigma(A_i) \neq \sigma(D_i)$ if the i-th bit is 0. A strategy representing 0 is shown in Figure 5. The encoding of binary numbers is made more precise later. Moreover, there are some specific properties that we want the game to have, which are listed below:

- The priorities $p(a_i)$ and $p(d_i)$ are the largest priorities out of the nodes in A_i and D_i, respectively ($i = 1, 2, \ldots, n$).
- Player 0 cannot win the game (if player 1 plays optimally). Hence the best player 0 can do is try to force the play to pass through vertex y[1], since this strategy will yield values of Ξ_σ that contain $N(4n + 6)$, a large even number, in its path component. We assume that player 0 is able to force the play to enter y.
- If a play enters gadget A_i, say the first node of A_i encountered in the play is v, then player 0 can force player 1 to enter a_i, and cannot do better than that. The same for D_i and d_i. We assume that both incoming edges enter A_i or D_i in the same node v.
- If the game starts from a_i, then player 0 can force the play to leave A_i through either of the two outgoing edges of A_i. Same for d_i and D_i. Again, he cannot do better since he cannot win the game by force.
- The above properties imply that mainly the priorities of a_i and d_i for $i = 1, 2, \ldots, n+1$ that are encountered throughout the game are relevant for what strategy to choose. Clearly the optimal strategy for player 0 (the end result of strategy iteration) is to always force the play from A_i or D_i to pass the edge going to D_{i+1} (or to d_{n+1} from A_n or D_n). This is since it then passes the nodes d_i, which are better for him than a_i.
- We start strategy iteration with $\sigma_0(A_i) = a_{i+1}$ and $\sigma_0(D_i) = d_{i+1}$ for $i = 1, 2, \ldots, n-1$ and with $\sigma_0(A_n) = d_{n+1}$ and $\sigma_0(D_n) = a_{n+1}$. This is clearly not a great strategy since many choices let the play go to a_i for different i, which is never optimal.

[1]With forcing, we mean that player 0's strategy choice, player 1 will have to go to y, in the sense that trying to avoid y is either impossible or will result in player 0 winning the game.

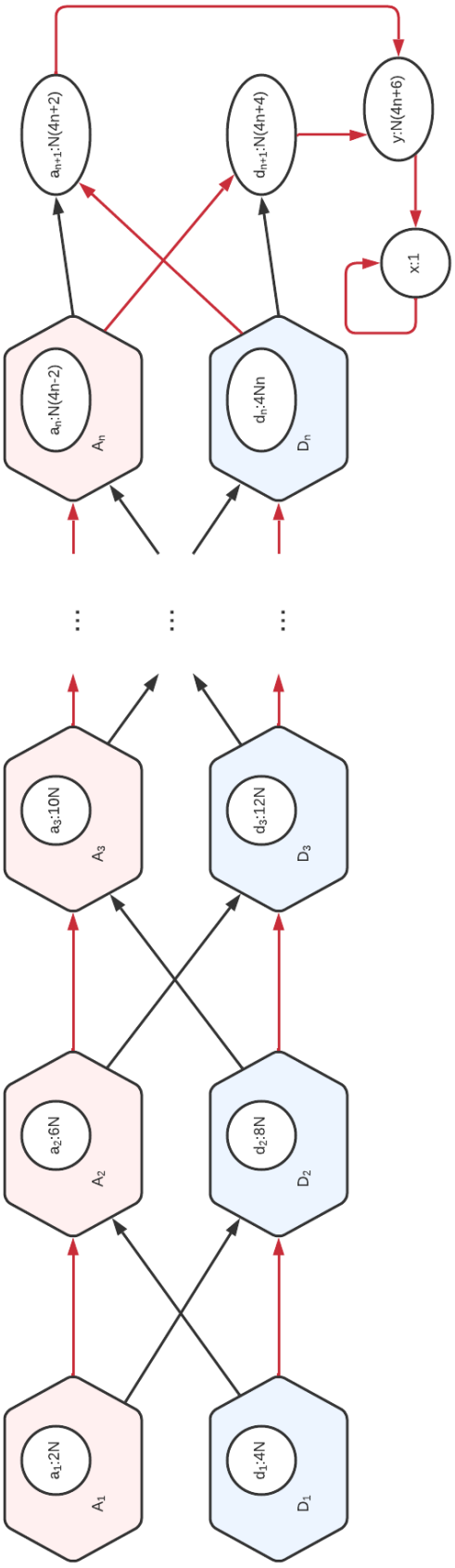


FIGURE 5: Structure for a class of parity games. The red and blue shaded areas represent some subgraph. Each of the subgraphs A_i or D_i has exactly two outgoing edge and exactly two incoming edges (except A_1 and D_1). In red are the edges that the resulting play will pass through if player 0 plays initial strategy σ_0 and player 1 plays its optimal counterstrategy. This encodes the number 0 in binary.

Note that, because of the increasing priorities of a_i and d_i with increasing i, the right-hand side of the game is the most important. Hence, if player 0 makes better choices in A_i than in D_i, then the play starting from a_i passes through different nodes with large priority than d_i. Hence the valuation of a_i will then be better than that of d_i, regardless of what smaller priorities the play may pass through in A_i or D_i itself.

Now our goal is, for a given improvement rule of strategy iteration, to construct a class of parity games $(G_n)_{n\in\mathbb{N}}$, for which this improvement rule has exponential running time. To do so, we formulate a condition for the combination of a parity game G_n and an improvement rule for strategy iteration, and show that this is enough to imply exponential running time.

Definition 3.1. Suppose we have an improvement rule for strategy iteration that has the following preference for improving moves in a game G_n with the structure as shown in Figure 5:

- Any improving move in D_i is preferred over any improving move in D_j if $i < j$.
- Any improving move in A_i is preferred over any improving move in D_i for $i = 1, 2, \ldots, n$.

Then we call this improvement rule *conservative* for G_n.

Now we want to prove that conservative improvement rules have exponential running time. To do so, we show that strategy improvement simulates a binary counter. To make this more formal, we introduce the following notion:

Definition 3.2. Consider a player 0 strategy σ for G_n and a number $B \in \{0, 1, \ldots, 2^n-1\}$. Let $B_nB_{n-1}B_{n-2}\ldots B_1$ be the binary representation of B, with $B_n \in \{0, 1\}$ the most significant bit and $B_1 \in \{0, 1\}$ the least significant bit. Then we say σ is in *bit state* B if the following hold:

- For $i = 1, 2, \ldots, n$, we have that $\sigma(A_i) = \sigma(D_i)$ if and only if $B_i = 1$.
- For $i = 1, 2, \ldots, n$, we have that σ forces the right choice in A_i regarding the edge through which A_i is left in the play resulting from σ and $\bar{\sigma}$. In other words,

$$\Xi_\sigma(\sigma(A_i)) = \max_{\trianglelefteq}(\Xi_\sigma(a_{i+1}), \Xi_\sigma(d_{i+1}))$$

.

Now we prove our result on the running time of conservative improvement rules in the following theorem:

Theorem 3.3. *Suppose we have a class of parity games $(G_n)_{n\in\mathbb{N}}$ that has the binary counter structure as in Figure 5. Suppose that an improvement rule for strategy iteration is conservative for all the games in this class. Then strategy iteration on G_n for any n passes through all bit states in $\{0, 1, \ldots, 2^n - 1\}$, hence it has worst-case running time that is at least exponential in n.*

Proof: We prove this by induction. First of all, as an induction basis, we show that the initial strategy σ_0 is in bit state 0. We have $B_i = 0$ and $\sigma_0(A_i) \neq \sigma_0(D_i)$ for $i = 1, 2, \ldots, n$. Also, because for $i = 1, 2, \ldots, n$ the play starting from a_i passes through d_{n+1}, and the play from d_i passes through a_{n+1}, we see that $\Xi_{\sigma_0}(a_i) \triangleright \Xi_{\sigma_0}(d_i)$ for $i = 1, 2, \ldots, n$. Therefore

$$\Xi_{\sigma_0}(\sigma_0(A_i)) = \Xi_{\sigma_0}(a_{i+1}) = \max_{\trianglelefteq}(\Xi_{\sigma_0}(a_{i+1}), \Xi_{\sigma_0}(d_{i+1}))$$

for $i = 1, 2, \ldots, n-1$, and

$$\Xi_{\sigma_0}(\sigma_0(A_n)) = \Xi_{\sigma_0}(d_{n+1}) = \max_{\trianglelefteq}(\Xi_{\sigma_0}(a_{n+1}), \Xi_{\sigma_0}(d_{n+1}))$$

so all conditions of being in bit state 0 are fulfilled. Next, as an induction step, we prove that if we have a strategy σ at some point during strategy iteration with our conservative improvement rule, and σ is in bit state B with $B < 2^n - 1$, that then we encounter another strategy in bit state $B+1$ at some later point in the algorithm. This clearly implies what we need to prove for this theorem.

To do so, we look at strategy σ that is in bit state B. We know that there are no significant improving moves in A_i for any i (i.e. no moves that would change $\sigma(A_i)$), since player 0 already forces the play to leave A_i in the 'right' way, as we are in bit state B. We just ignore from now these non-significant improving moves that do not change any $\sigma(A_i)$ or $\sigma(D_i)$. Moreover, looking at any D_i, we see that if $B_i = 1$, then $\sigma(D_i) = \sigma(A_i)$, hence player 0 also makes the 'right' choice in D_i. Hence the only gadgets in which there are significant improving moves possible are in D_i where $B_i = 0$. Because we have a conservative improvement rule, we see that we prefer to make improvements in the D_i with the lowest index i. Hence if i_0 is the smallest index for which $B_{i_0} = 0$, then the next significant improvement is when $\sigma(D_{i_0})$ changes, after which it will equal $\sigma(A_{i_0})$.

To deduce what happens after that, we first take a closer look at $\sigma(A_i)$ and $\sigma(D_i)$ for $i < i_0$. We know that $\sigma(A_{i_0-1})$ (if $i_0 > 1$) is the 'right' choice, and we also know that $\sigma(A_{i_0})$ is 'right' and $\sigma(D_{i_0})$ is not, hence $\Xi_\sigma(a_{i_0}) \triangleright \Xi_\sigma(d_{i_0})$. Therefore, since $B_{i_0-1} = 1$ (if $i_0 > 1$), we have $\sigma(A_{i_0-1}) = \sigma(D_{i_0-1}) = a_{i_0}$. Because d_{i_0-1} has higher even priority than a_{i_0-1}, and the rest of the plays determining their values is almost equal, we get $\Xi_\sigma(a_{i_0-1}) \triangleleft \Xi_\sigma(d_{i_0-1})$. Hence with the same logic we get $\sigma(A_{i_0-2}) = \sigma(D_{i_0-2}) = d_{i_0-1}$ (if $i_0 > 2$), and continuing this chain of logic, we get $\sigma(A_i) = \sigma(D_i) = d_{i+1}$ for $i \leq i_0 - 2$. See also Figure 14

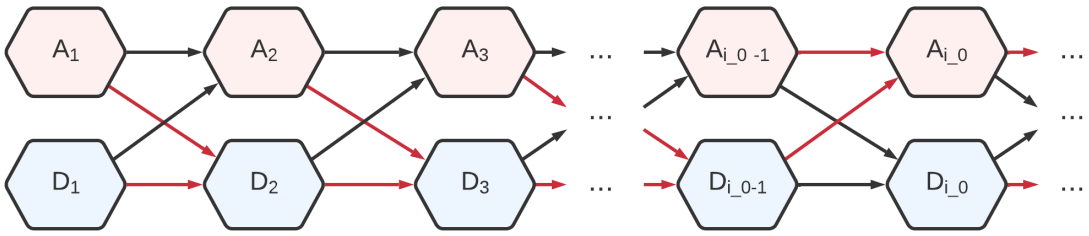


FIGURE 6: The first part of graph G_n and in red the edges that will be used in the plays resulting from strategy σ (that is in bit state B) and optimal counterstrategy $\bar{\sigma}$.

Now say the resulting strategy from making a switch in D_{i_0} that changes $\sigma(D_{i_0})$ is σ'. We see that the valuation $\Xi_{\sigma'}$ of d_{i_0} must be higher than than of a_{i_0}, as the priority of d_{i_0} is higher and the rest of the plays from these vertices are similar. Hence there will now be significant improving moves possible in A_{i_0-1} and in D_{i_0-1}. Because we have a conservative improvement rule, we prefer switches in A_{i_0-1}. Hence the next significant improving switch occurs in A_{i_0-1}, yielding strategy σ''. But after that, a_{i_0-1} has a higher valuation $\Xi_{\sigma''}$ than d_{i_0-1}, since player 0 makes the 'right' choice in A_{i_0-1} but not in D_{i_0-1}. Hence there will be an improving move possible in A_{i_0-2} and in D_{i_0-2}. Again, A_{i_0-2} is preferred, so the next significant improving move occurs in A_{i_0-2}, and, with the same argument, after that

in $A_{i_0-3}, A_{i_0-4}, \ldots, A_1$. Let the resulting strategy after all those improvements be σ'''. We claim that σ''' is in bit state $B+1$. Note that σ''' makes the 'right' choice in A_i for all i, since an improving move was just made in $A_{i_0-1}, A_{i_0-2}, \ldots, A_1$, and since nothing changed compared to σ for A_i for $i \geq i_0$. Also, we have $\sigma'''(A_i) \neq \sigma'''(D_i)$ for $i < i_0$ and $\sigma'''(A_i) = \sigma'''(D_i)$, and $\sigma'''(A_i) = \sigma'''(D_i) \Leftrightarrow B_i = 1$ for $i > i_0$. From that, we conclude that σ''' is in bit state $B+1$ (note that the binary representation of $B+1$ has a 1 in its i_0-position, 0's in the positions $i < i_0$ and the same as B in higher positions). This concludes the proof of the theorem. □

3.2 Lower bounds from binary counter

Now we illustrate the result of Theorem 3.3 by providing lower bounds on the complexity of strategy iteration with various improvement rules. We look at the following improvement rules:

- *highest-priority*: apply the switch on the node with the highest priority.
- *highest-valuation*: Perform the best improving switch from the node v that currently has the highest valuation $\Xi_\sigma(v)$.
- *highest-difference*: We only define this rule for a sink parity game (see Definition 2.6). Choose the improving switch that has the highest difference in valuations. Here highest difference is determined according to the following: if we have an improving move that changes $\sigma(v)$ from a node with valuation (a, A, n_a) to one with valuation (b, B, n_b) and one that changes $\sigma(v')$ from a node with valuation (c, C, n_c) to one with (d, D, n_d), we call the first difference better if $(1, B \uplus C, 1) \triangleright (1, A \uplus D, 1)$.
- *lowest-difference* The opposite of highest-difference. Choose the improving switch with the lowest difference.

There seems to be no mention of any of these in the literature, although they are logical options. For each of these improvement rules, we describe the gadgets of G_n and show that the improvement rule is conservative for G_n. Then it follows from Theorem 3.3 that strategy iteration needs an exponential number of iterations.

Theorem 3.4. *Strategy iteration with the highest-priority improvement rule has at least exponential worst-case running time in the number of nodes and edges.*
Proof: We can use quite simple gadgets for this:

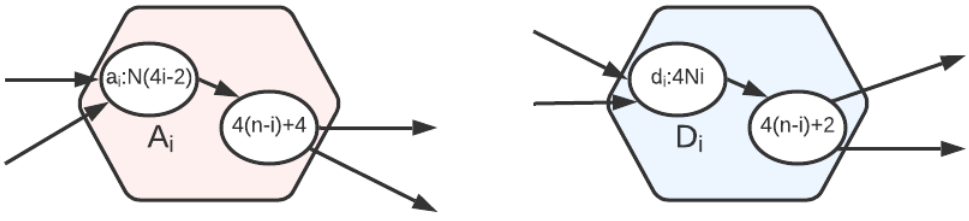


FIGURE 7: Gadgets for G_n for the highest priority improvement rule.

Note that the only choice possible within A_i and D_i are in choosing the outgoing edges of the unnamed nodes. Hence the preference of the highest-priority improvement rule is determined by the priorities of these unnamed nodes. It immediately follows that this

improvement rule prefers to switch in D_i with the lowest i, and that it prefers switching in A_i over D_i, hence it is conservative. From Theorem 3.3 it now follows that strategy iteration has running time exponential in n. Because the number of nodes and edges is linear in n, the running time is also exponential in the number of nodes and edges. □

Theorem 3.5. *Strategy iteration with the highest-valuation improvement rule has worst-case running time that is at least exponential in the number of nodes and edges.*
Proof: We use the following gadgets:

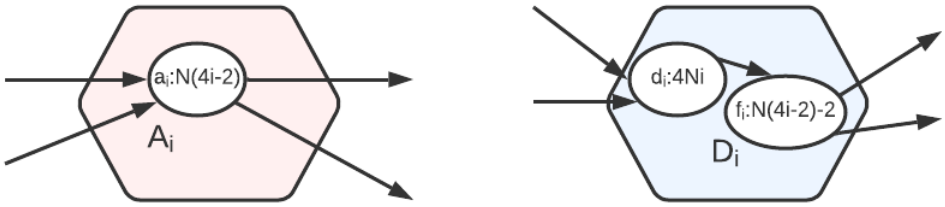


FIGURE 8: Gadgets for G_n for the highest-value improvement rule.

In principle, the idea is that the node f_i will have a lower valuation than a_i, so the improvement rule is tricked into preferring switches in a_i. To prove the theorem, we need to show that in the game G_n, we prefer to switch in D_i with lower index i, and that we prefer to switch in A_i over D_i with these improvement rules. Note that there is only one node in each gadget A_i or D_i where there is a choice, namely a_i in A_i and f_i in D_i.

Suppose we have an improving move possible in both A_i and D_i. This can only be the case if player 0 currently makes the 'wrong' choice with σ in both A_i and D_i. In particular, $\sigma(A_i) = \sigma(D_i)$ (hence $\sigma(a_i) = \sigma(f_i)$). This implies that Ξ_σ is higher for a_i than for f_i. That is because the only difference between $\Xi_\sigma(a_i)$ and $\Xi_\sigma(f_i)$ is that the first contains $p(a_i)$ in its path and the second $p(f_i)$. Therefore switching in A_i is preferred over switching in D_i by the highest-value improvement rule.

Next suppose we have a strategy σ, and we have an improving move possible in D_i and one in D_j, with $i < j$. Since there is an improving move possible in D_j, we have $\Xi_\sigma(\sigma(D_j)) \trianglelefteq \Xi_\sigma(\sigma(A_j))$, and therefore $\Xi_\sigma(f_j) \triangleleft \Xi_\sigma(a_j)$. We know that the play starting at f_i resulting from σ and $\bar{\sigma}$ must pass through either A_j or D_j, since it will end in x. If the play passes through D_j, then clearly the valuation Ξ_σ of f_i is higher than that of f_j, since the valuation of f_i contains a number of extra even priorities in its path compared to $\Xi_\sigma(f_j)$. On the other hand, if the play from f_i passes through A_j, then the valuation of f_i is also higher than that of f_j since its valuation is that of a_j with some extra even numbers in its path component, and we saw before that $\Xi_\sigma(f_j) \triangleleft \Xi_\sigma(a_j)$. Hence we now showed that improving moves in D_i are preferred over improving in D_j for $i < j$.

We conclude that the highest-value rule is conservative for $(G_n)_{n\in\mathbb{N}}$. Now from Theorem 3.3 it follows that strategy iteration takes at least time exponential in n for $(G_n)_{n\in\mathbb{N}}$. Because the number of nodes and edges is linear in n, the running time is also exponential in the number of nodes and edges. □

Theorem 3.6. *Strategy iteration with the highest-difference improvement rule has worst-case running time that is at least exponential in the number of nodes and edges.*
Proof: We use the following gadgets:

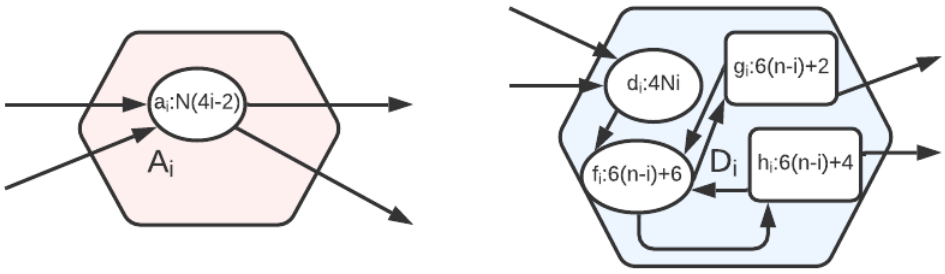


FIGURE 9: Gadgets for G_n for the highest-difference improvement rule.

In short, the idea of these gadgets is that in D_i, the player 1 nodes g_i and h_i 'hide' to player 0 how much he can improve by switching. We first look in more detail at how the gadget D_i works. Note that in gadget D_i, player 0 has a choice in node f_i to move to either g_i or h_i. Irrespective of the vertex player 0 chooses, player 1 will be forced to move out of D_i in one of his nodes in order to avoid creating a cycle that wins for player 0. In this manner, player 0 can force the play starting from d_i to either go to A_{i+1} or D_{i+1} (or, if $i = n$, to a_{n+1} or d_{n+1}). We see that $\sigma(f_i) = g_i$ corresponds to $\sigma(D_i) = a_{i+1}$ and $\sigma(f_i) = h_i$ corresponds to $\sigma(D_i) = d_{i+1}$.

Now we look at the valuation Ξ_σ in D_i. We assume first that going to a_{i+1} is the 'right' choice, so $\Xi_\sigma(a_{i+1}) \triangleright \Xi_\sigma(d_{i+1})$. Suppose that $\sigma(f_i) = g_i$, so then we also have $\sigma(D_i) = a_{i+1}$. Then player 1 is forced to move out of D_i from g_i. Player 1 will also move out of D_i from h_i as this goes to d_{i+1}, which has a lower valuation than f_i. This is shown on the left of Figure 10. Suppose $\Xi_\sigma(v) = (1, P(v), n_v)$, so $P(v)$ denotes the priorities on the path from v. Then we have in our case $P(g_i) = P(a_{i+1}) \cup \{p(g_i)\}$ and $P(h_i) = P(d_{i+1}) \cup \{p(h_i)\}$. We assumed a_{i+1} has a better valuation than d_{i+1}, and therefore a_{i+1} has a much better valuation than d_{i+1} since the large priorities are on the right of the game G_n. Therefore we have $\Xi_\sigma(g_i) \triangleright \Xi_\sigma(h_i)$. So there are no improving moves possible in D_i.

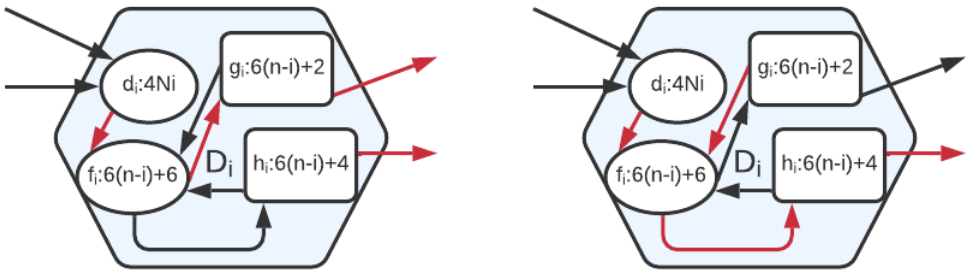


FIGURE 10: Left: red edges are edges chosen either by σ or $\bar{\sigma}$ if $\sigma(f_i) = g_i$. Right: red edges are edges chosen either by σ or $\bar{\sigma}$ if $\sigma(f_i) = h_i$. Assumed is $\Xi_\sigma(a_{i+1}) \triangleright \Xi_\sigma(d_{i+1})$.

On the other hand, now suppose $\sigma(D_i) = d_{i+1}$, so player 0 makes the 'wrong' choice. Then we have the situation on the right of Figure 10. In particular, player 1 will decide to move to f_i from g_i, since this will lead to node d_{i+1}, which is better for him. This means that $P(g_i) = P(h_i) \cup \{p(f_i), p(g_i)\}$, and therefore $\Xi_\sigma(g_i) \triangleright \Xi_\sigma(h_i)$. This means there is an

improving move in f_i. However, the 'difference' that the highest-difference improvement rule looks at seems very small, as the 'difference' between $\Xi_\sigma(g_i)$ and $\Xi_\sigma(g_i)$ consists only of the priorities of f_i and g_i. Likewise, we can show that for the reversed case, if $\Xi_\sigma(a_{i+1}) \triangleleft \Xi_\sigma(d_{i+1})$ and $\sigma(f_i) = g_i$, then the 'difference' of an improving move consists only of the priorities of f_i and h_i. Since the priorities of f_i, g_i, h_i decrease with i increasing, we see that the highest 'difference' occurs in the D_i with the lowest i, so the highest-difference rule prefers to switch in these D_i. Furthermore, the 'difference' for an improving move in A_i is the difference in valuation between a_{i+1} and d_{i+1}, which is very large compared to the small priorities of the f_i, g_i, h_i. So we also prefer to switch in A_i over D_i for any i. Hence by Theorem 3.3, the highest-difference improvement rule is conservative for $(G_n)_{n\in\mathbb{N}}$, and therefore has running time at least exponential in n. Because the number of nodes and edges is linear in n, the running time is also exponential in the number of nodes and edges. □

Theorem 3.7. *Strategy iteration with the lowest-difference improvement rule has worst-case running time that is at least exponential in the number of nodes and edges.*
Proof: We use the following gadgets:

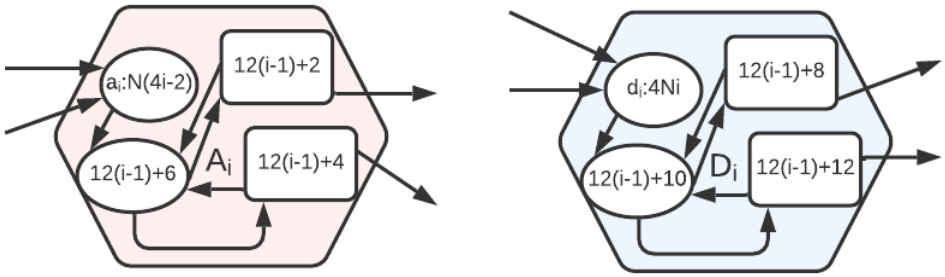


FIGURE 11: Gadgets for G_n with the lowest-difference improvement rule.

Note that the structure of these gadgets is exactly the same as those of D_i from the highest-difference gadget. Hence if there is an improving move in A_i or D_i, then its 'difference' is determined by the priorities of the unnamed nodes in Figure 15. Therefore, it is easy to see that A_i is preferred over D_i, since the priorities of the unnamed nodes are smaller, and by the same argument, D_i is preferred over D_j if $i < j$. Hence by Theorem 3.3, the lowest-difference improvement rule yields running time exponential in n for $(G_n)_{n\in\mathbb{N}}$. Because the number of nodes and edges is linear in n, the running time is also exponential in the number of nodes and edges. □

3.3 Reverse binary counter

Next, we look at a second structure which can also be used to prove exponential running time for improvement rules for strategy iteration. We call this a *reverse binary counter*. This because the underlying graph is the same as the binary counter, but the way that it counts is reversed compared to the regular binary counter. The highest priorities are again multiples of some number N, but unlike before we require N to be odd. As a result, the highest priorities in the gadgets are odd instead of even. The structure can be found in Figure 12.

Similar to before, we want the parity game and strategy iteration to have these properties:

- The priorities of the nodes in A_i and D_i $(i = 1, 2, \ldots, n)$ that are not a_i or d_i are much smaller than $p(a_i)$ and $p(d_i)$, respectively.

- Player 0 cannot win the game. Hence the best player 0 can do is try to force the play to pass through vertex y, since this strategy will yield values of Ξ_σ that contain $p(y) = N(4n + 6)$. We assume that player 0 is able to force the play to enter y.

- If a play enters gadget A_i, say the first node of A_i encountered in the play is v, then player 0 can force player 1 to enter a_i, and cannot do better than that. The same holds for D_i and d_i. We assume that both incoming edges enter A_i or D_i in the same node v.

- If the game starts from a_i, then player 0 can force the play to leave A_i through either of the two outgoing edges of A_i. Same for d_i and D_i. Again, he cannot do better since he cannot win the game by force.

- The above properties imply that only the priorities of a_i and d_i for $i = 1, 2, \ldots, n+1$ that are encountered throughout the game are relevant for what strategy to choose. Here, unlike the normal binary counter, the optimal strategy for player 0 (the end result of strategy iteration) is to always force the play from A_i or D_i to pass the edge going to A_{i+1} (or to a_{n+1} from A_n or D_n). This is of course because the priority of a_i is odd and lower than $p(d_i)$.

- We start strategy iteration with $\sigma_0(A_i) = \sigma_0(D_i) = d_{i+1}$ for $i = 1, 2, \ldots, n$. This is the worst possible strategy for player 0 that still goes to x, since it always passes the largest odd numbers.

- Note that, because of the increasing priorities of a_i and d_i with increasing i, the right-hand side of the game is the most important. For example, if player 0 makes better choices in A_i than in D_i, the valuation of a_i will be much better than that of d_i, since the path element of its valuation will differ with some large priorities from the right of the graph.

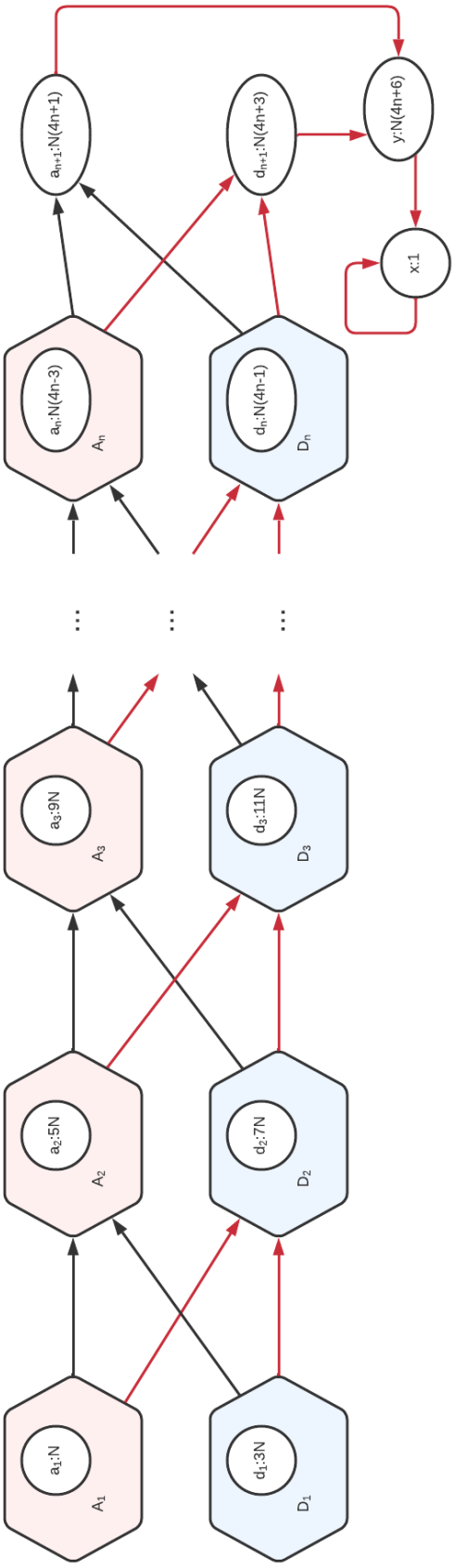


FIGURE 12: The structure of the reverse binary counter. The red and blue shaded areas represent some subgraph. Each of the subgraphs A_i or D_i has exactly two outgoing edges and exactly two incoming edges (except A_1 and D_1). In red are the edges that the resulting play will pass through if player 0 plays initial strategy σ_0 and player 1 plays its optimal counter-strategy $\bar{\sigma}_0$. This encodes the number $2^n - 1$ in binary. In this case, N is odd.

Similar to the binary counter, we formulate a condition for an improvement rule that will imply exponential running time on $(G_n)_{n\in\mathbb{N}}$.

Definition 3.8. Suppose we have an improvement rule for strategy iteration and a game G_n with the structure as shown in Figure 12. Then we call this improvement rule *counter-conservative* for G_n, if the improvement rule has the following preferences:

- Making any switch in A_i is preferred over any switch in A_j if $\Xi_\sigma(a_i) \lhd \Xi_\sigma(a_j)$.
- Making any switch in D_i is preferred over any switch in D_j if $\Xi_\sigma(d_i) \lhd \Xi_\sigma(d_j)$.
- Making any switch in A_i is preferred over any switch in D_j if $\Xi_\sigma(a_i) \lhd \Xi_\sigma(d_j)$.

We show that counter-conservative improvement rules have exponential running time. Similar to the binary counter, we have the following notion:

Definition 3.9. let σ be a player 0 strategy and let $B \in \{0, 1, \ldots, 2^n - 1\}$. Let $B_nB_{n-1}B_{n-2}\ldots B_1$ be the binary representation of B, with $B_n \in \{0, 1\}$ the most significant bit and $B_1 \in \{0, 1\}$ the least significant bit. We say σ is in *reverse bit state* B if the following hold:

- For $i = 1, 2, \ldots, n$, we have that $\sigma(A_i) = \sigma(D_i)$ if and only if $B_i = 1$.
- For $i = 1, 2, \ldots, n$, we have that σ forces the *wrong* choice in A_i w. r. t. through which edge A_i is left. In other words,

$$\Xi_\sigma(\sigma(A_i)) = \min_{\trianglelefteq}(\Xi_\sigma(a_{i+1}), \Xi_\sigma(d_{i+1}))$$

.

Now we are ready to formulate the following theorem:

Theorem 3.10. *Suppose we have a class of parity games $(G_n)_{n\in\mathbb{N}}$ that has the reverse binary counter structure as described in this section. Suppose that an improvement rule for strategy iteration is counter-conservative for all the games in this class. Then strategy iteration on G_n for any n passes through all reverse bit states in $\{0, 1, \ldots, 2^n - 1\}$, hence it has worst-case running time that is at least exponential in n.*

Proof: We use induction to show that all the bit states are traversed from high to low. As an induction basis, we show that our starting strategy σ_0 is in reverse bit state $2^n - 1$. Note that if $B = 2^n - 1$, then $B_i = 1$ for $i = 1, 2, \ldots, n$. We also have $\sigma_0(A_i) = \sigma_0(B_i) = d_{i+1}$ for $i = 1, 2, \ldots, n$, and this implies the first condition for being in this reverse bit state. Also, note that for $i = 1, 2, \ldots, n$, we have $\Xi_{\sigma_0}(a_{i+1}) \rhd \Xi_{\sigma_0}(d_{i+1})$, as a_{i+1} has a lower odd priority than d_{i+1} and plays from a_{i+1} and d_{i+1} pass through the same nodes after leaving A_{i+1} or D_{i+1}. Therefore player 0 is making the 'wrong' choice in A_i for $i = 1, 2, \ldots, n$. Therefore σ_0 is in bit state $2^n - 1$.

Now we show as an induction step the following: if at some point in strategy iteration with our counter-conservative improvement rule, we have strategy σ in bit state $B > 0$, then at some later iteration we get another strategy in bit state $B - 1$. Our goal is to show that the next improving moves occur at $a_1, a_2, \ldots, a_{i_0-1}, d_{i_0}$ in that order, and that the resulting strategy is in bit state $B - 1$.

First, we show that the a_i at the left side of G have very low valuation Ξ_σ. Note that, since we assume that $p(a_i)$ and $p(d_i)$ are much larger than the other priorities in A_i and D_i, respectively, we get $\Xi_\sigma(a_i) \lhd \Xi_\sigma(\sigma(A_i))$. This is because the term on the left-hand side

is the same as the right-hand side, except that its path also contains $p(a_i)$, which is odd, and some small priorities. Also, since σ is in reverse bit state B, we have

$$\Xi_\sigma(\sigma(A_i)) = \min_{\trianglelefteq}(\Xi_\sigma(a_{i+1}), \Xi_\sigma(d_{i+1}))$$

it follows that $\Xi_\sigma(a_i) \triangleleft \Xi_\sigma(\sigma(A_i)) \triangleleft \Xi_\sigma(a_i), \Xi_\sigma(a_i)$. If we apply this argument multiple times, we see that

$$\Xi_\sigma(a_i) \triangleleft \Xi_\sigma(a_j), \Xi_\sigma(d_j) \tag{1}$$

for $j > i$. In particular, a_1 has smaller valuation than all the other nodes, except maybe b_1.

Now we take a closer look at what strategy σ looks like. Let i_0 be the smallest index for which $B_{i_0} = 1$. We have $\sigma(A_{i_0}) = \sigma(D_{i_0})$, and therefore $\Xi_\sigma(d_{i_0}) \triangleleft \Xi_\sigma(a_{i_0})$, since their only signifiacant difference is that one has $p(a_{i_0})$ and one $p(d_{i_0})$ in its path, and as $p(d_i) > p(a_i)$ and both priorities are odd. If we look at $\sigma(A_{i_0-1})$ (if $i_0 > 1$), then we know player 0 makes the 'wrong' choice in A_{i_0-1}, because we are in bit state B. Therefore $\sigma(A_{i_0-1}) = d_{i_0}$. Moreover, as $B_{i_0-1} = 0$, we know $\sigma(D_{i_0-1}) = a_{i_0}$, so player 0 makes the 'right' choice in gadget D_{i_0-1}. Therefore d_{i_0-1} is better valued than a_{i_0-1}, i.e. $\Xi_\sigma(d_{i_0-1}) \triangleright \Xi_\sigma(a_{i_0-1})$. With the same logic, then, we derive that $\sigma(A_{i_0-2}) = a_{i_0-1}$ and $\sigma(D_{i_0-2}) = d_{i_0-1}$, and so on. Therefore σ looks like the strategy in Figure 13.

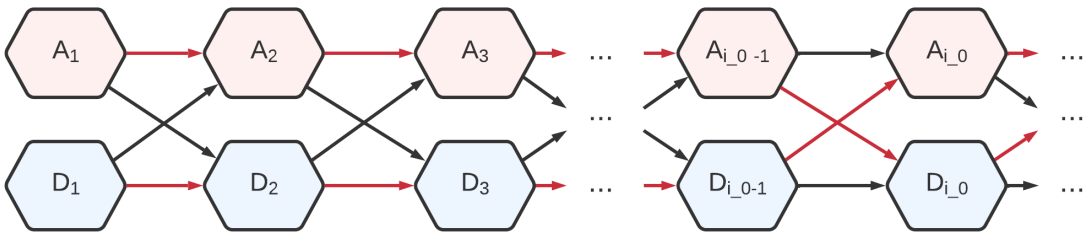


FIGURE 13: The first part of reverse binary counter graph G_n. Marked in red are the edges that are used in the plays resulting from strategy σ (that is in bit state B) and optimal counterstrategy $\bar{\sigma}$.

Now because the most important priorities are on the right of the graph, d_{i_0} has a 'much' lower valuation than a_{i_0} (as in, adding any of the priorities of the part shown in Figure 13 will not change which valuation is lower). It is then easy to see that Ξ_σ is lower for any of the nodes $a_1, a_2, \ldots, a_{i_0-1}, d_{i_0}$ than for any of the nodes $d_1, d_2, \ldots, d_{i_0-1}, a_{i_0}$, since plays from the first set of nodes end up in d_{i_0} and from the second end at a_{i_0}. Now we look at possible improving moves. Like in the proof of Theorem 3.3, we ignore all improving moves that do not change any $\sigma(A_i)$ or $\sigma(D_i)$. Clearly the lowest valuation of the game is at a node of A_1 (unless $i_0 = 1$), which we know because of (1). So the next significant improving move happens when $\sigma(A_1)$ becomes d_2 (or a_2 if $i_0 = 2$). This connects a_1 in the resulting play to a_{i_0}, hence the next lowest valuation after the switch is in A_2 (unless $i_0 \leq 2$). Likewise, this connects a_2 to a_{i_0}. After that, following the same logic, we see that we switch in $A_3, A_4, \ldots, A_{i_0-1}$. Then, all a_i and d_i with $i < i_0$ are connected to a_{i_0} (i.e. plays starting there pass through a_{i_0}). Hence d_{i_0} has a lower valuation than all nodes in A_i, D_i with $i < i_0$. Therefore the next significant improving switch is in D_{i_0}. In summary, we made improving switches in $a_1, a_2, \ldots, a_{i_0-1}, d_{i_0}$ in that order. Let the

strategy that we get after the significant improving switch in D_{i_0} be called σ'. This is shown in Figure 14

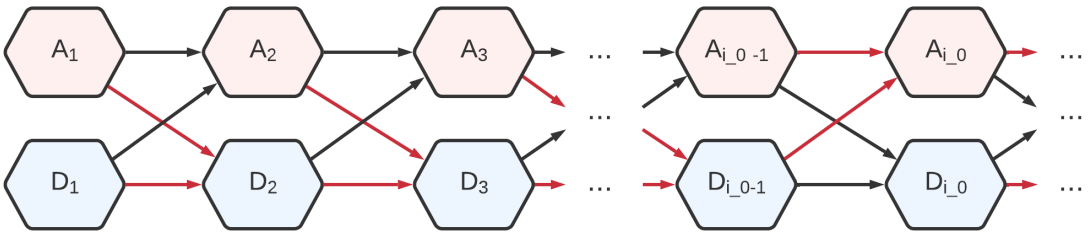


FIGURE 14: The first part of graph G_n and in red the edges that will be used in the plays resulting from strategy σ' and optimal counterstrategy $\bar{\sigma}'$.

We show that σ' is in bit state $B-1$. Note that for $i \leq i_0 - 2$, we can improve $\sigma'(A_i)$ because it is better to pass through a_{i+1}. Also, we can improve $\sigma'(A_{i_0-1})$ (if $i_0 > 1$), because d_{i_0} has a better value than a_{i_0} (since we just improved in d_{i_0}). Also, $\sigma'(A_i)$ for $i \geq i_0$ is still the 'wrong' choice since the right part of the strategy did not change from σ, and σ was in bit state B. Therefore σ' fulfills the second condition of being in some reverse bit state, so it must be in some reverse bit state B'. Moreover, $B'_i = 1$ and $B_i = 0$ for $i < i_0$, $B_{i_0} = 1$, $B'_{i_0} = 0$ and $B'_i = B_i$ for $i > i_0$. Therefore σ' is in bit state $B-1$. This completes the induction proof. □

3.4 Lower bounds from the reverse binary counter

We use Theorem 3.10 to prove lower bounds for some improvement rules. We do this for the following rules:

- *lowest-valuation*: We prefer to make improving switches on the vertices with the currently lowest valuation.
- *highest-lexicographic*: Let $VAL = Q \times \mathcal{P}(Q) \times \mathbb{Z}_{\geq 0}$ be the space of valuations. For each possible improving switch e, let σ_e be the strategy that results from σ if improving move e is applied. We then associate a vector $\xi_e \in VAL^{|V|}$ to the strategy σ_e containing all values of $\sigma_e(v)$ for $v \in V$, and where ξ_e is $\trianglelefteq$-sorted from low to high. We then choose the improving switch e with the lexicographically highest value of ξ_e. In this case, we say $a < b$ lexicographically if for the least index i with $a_i \neq b_i$ we have $a_i < b_i$.
- *lowest-valuation-shortest-path*: Suppose we have an improving move from v for strategy σ that switches from edge (v, v'') to (v, v'). Say the new strategy after this switch is σ' Let $paths(v, v')$ be the set of paths from v to any $w \in V$, where we require all the edges to be either in $E_1 = \{(v_1, v_2) | v_1 \in V_1\}$ or to be part of either σ or σ'. Recall that we do not count the end vertex of a path as being *in* the path. Then, we define $M_\sigma(v, v') = \min_{\preceq} paths(v, v')$, which will be the lowest-valued path from v to any other node. (recall from Section 2.4 that whether $B_1 \preceq B_2$ can be determined from $B_1 \triangle B_2$ or from whether $\sum_{p \in B_1} (-3)^p \leq \sum_{p \in B_2} (-3)^p$). The improvement rule prefers to switch in nodes with a low valuation and with a high value of $M(v, v')$. More precisely, our improvement rule prefers a switch in v to v' over a switch in w to w' if $\Xi_\sigma(v) \uplus M_\sigma(w, w') \trianglelefteq \Xi_\sigma(w) \uplus M_\sigma(v, v')$ (here we use simplified notation; with $(\lambda, \pi, n) \uplus B$ we mean $(\lambda, \pi \uplus B, n)$).

Again, there seems to be no mention of these in the literature. The first rule is a logical rule, and the latter two are used in section 5 because of their connection to the simplex algorithm. Like for the binary counter, we show for these rules that they are counter-conservative, and then use Theorem 3.10 to show that strategy iteration requires an exponential number of iterations. A lower bound for the first rule follows trivially:

Theorem 3.11. *Strategy iteration with the lowest-valuation improvement rule has worst-case running time that is at least exponential in the number of nodes and edges.*
Proof: We use the following trivial gadgets:

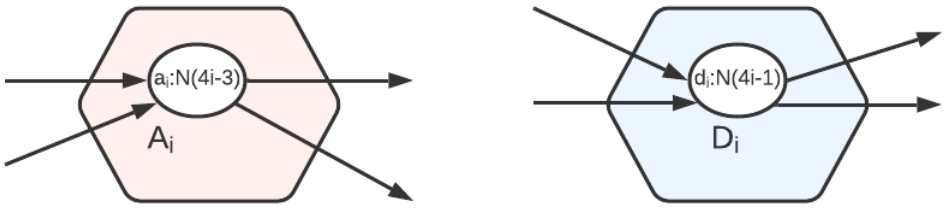


FIGURE 15: Gadgets for G_n with the lowest-valuation improvement rule.

the result follows from Theorem 3.10. □

A lower bound for the second improvement rule is a bit more involved:

Theorem 3.12. *Strategy iteration with the highest-lexicographic improvement rule has worst-case running time that is at least exponential in the number of nodes and edges.*
Proof: We use the following gadgets:

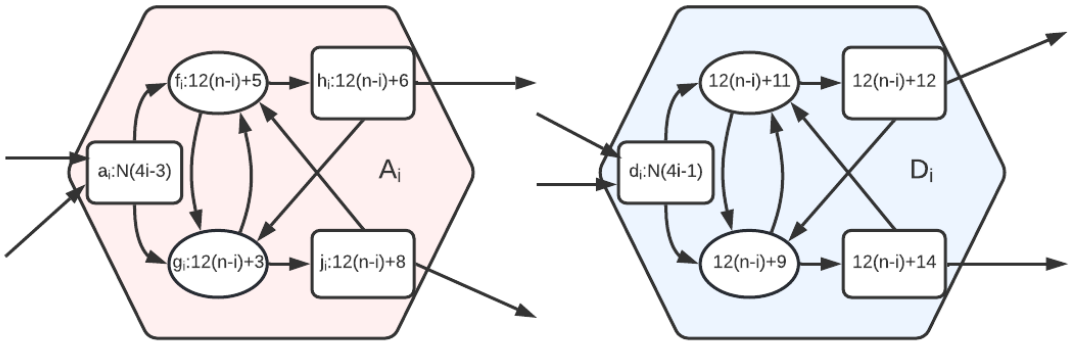


FIGURE 16: Gadgets for G_n with the lowest-difference improvement rule.

Recall that we want to make the improving switch e with this rule that results in the lexicographically highest value of ξ_e. Note that with this improvement rule, we care most about what happens with the nodes that have the lowest valuation; we want their valuations to improve as much as possible. We show that this improvement rule is counter-conservative, but we only do so for the strategies encountered in strategy iteration. Theorem 3.10 will still hold in this case.

But first, we look at how an individual gadget A_i works. Suppose that a_{i+1} has a better valuation than d_{i+1} for some strategy σ. Note that player 0 can force the play starting from a_i to go to a_{i+1} by choosing $\sigma(f_i) = h_i$ and $\sigma(g_i) = f_i$. See left of Figure 17. Player 1 must move out of A_i in h_i to avoid creating a cycle with even highest priority. Player 1 also moves out of A_i in j_i as d_{i+1} has lower valuation than a_{i+1}

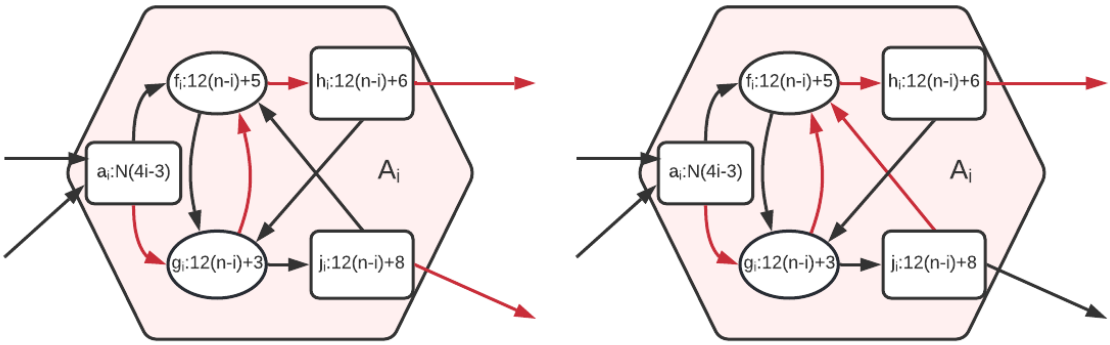


FIGURE 17: Left: red edges show a strategy σ with optimal counterstrategy $\bar{\sigma}$ that forces the play to go to A_{i+1}, if it is the 'good' option. Right: strategy σ with counterstrategy $\bar{\sigma}$ that forces the play to go to A_{i+1}, if it is the 'wrong' option.

Now suppose on the other hand that d_{i+1} has higher valuation than a_{i+1}, but player 0 still makes the same choices as before, then we get the situation on the right of Figure 17, where player 1 now chooses $\bar{\sigma}(j_i) = f_i$. This implies $\Xi_\sigma(j_i) \triangleright \Xi_\sigma(f_i)$ as $\Xi_\sigma(j_i)$ has an extra priority of $12(n-i)+8$ in its path. So there is an improving move possible in g_i, and this is the only improving move in A_i. This results in the situation shown on the left in Figure 18.

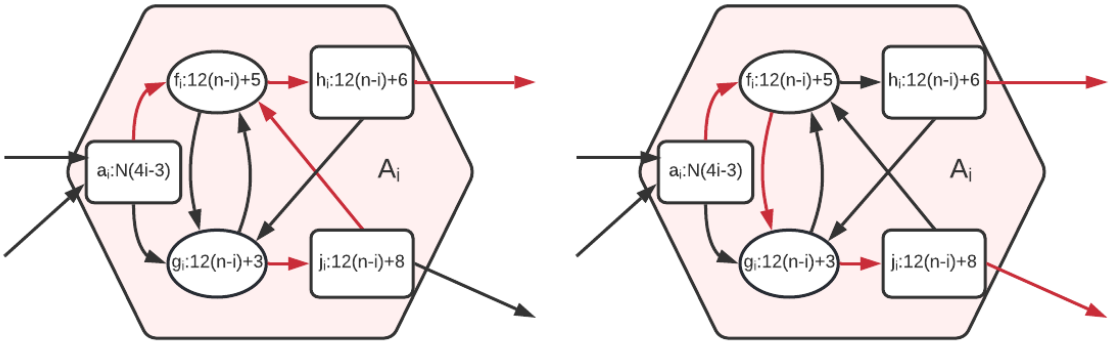


FIGURE 18: Left: red edges show a strategy σ with optimal counterstrategy $\bar{\sigma}$ after the first improving move. Right: strategy σ with counterstrategy $\bar{\sigma}$ after the second improving move.

After the switch, g_i becomes higher valued than f_i, hence player 1 changes his choice in a_i and now picks $\bar{\sigma}(a_i) = f_i$. Therefore, the valuation of a_i slightly increases (priority $12(n-i)+3$ is removed from the path in $\Xi_\sigma(a_i)$). After the first improving move we have $\Xi_\sigma(g_i) \triangleright \Xi_\sigma(h_i)$, since it has three extra priorities, namely $12(n-i)+3$, $12(n-i)+5$ and $12(n-i)+8$ in its path. Therefore f_i has an improving move by switching to g_i. Again, this is the only possible improving move. This new strategy forces player 1 to move out of A_i in j_i to prevent creating a cycle with even highest priority. This results in the situation on the right of Figure 18. Note that the priority of a_i now increases by a lot because of this move because the play now ends up in d_{i+1} instead of a_{i+1}. In summary, changing $\sigma(A_i)$ by making improving moves occurs in two moves: the first increases $\Xi_\sigma(a_i)$ very slightly, and the second one is significant, changing $\sigma(A_i)$ by a lot. Clearly it is similar for improving $\sigma(D_i)$. Moreover, after these two moves the strategy is again one that forces

player 1 to move out of A_i or D_i towards either the 'right' or the 'wrong' next gadget, so one of the situations like in Figure 17.

Now we are ready to find which improving moves the highest-lexicographic rule prefers. We want to show that it performs an improving move in the gadget with the lowest valuation of a_i or d_i. We specify the initia strategy to be the strategy σ_0 where player 0 forces player 1 to go to $\sigma_0(A_i)$ or $\sigma_0(D_i)$. This means that e.g. $\sigma_0(A_i) = a_{i+1}$ implies $\sigma_0(f_i) = h_i$ and $\sigma_0(g_i) = f_i$ and $\sigma_0(A_i) = d_{i+1}$ implies $\sigma_0(f_i) = g_i$ and $\sigma_0(g_i) = j_i$.

Clearly for any strategy we see the node with the lowest valuation in the graph is x, which has valuation $(1, \emptyset, 0)$, while the other nodes have $N(4n + 6)$ in their path. So the first element of ξ_e is always the same. Hence we mainly look at the second element. This is the valuation of the node with the lowest valuation besides x. This must be a_i or d_i for some i, as these are the nodes with large odd priority (and f_i, g_i and such nodes pass through nodes with slightly larger even priority before leaving A_i, D_i). For strategy σ, call the second lowest valuation in the graph ζ_σ. Then our improvement rule prefers the improving move(s) that increase ζ_σ the most.

Recall that improving moves keep all valuations either the same or increase them. Suppose a_i has the lowest valuation after x. Then ζ_σ stays the same after an improving move, unless an improving switch improves the valuation of a_i. The valuation of a_i is improved if the switch happens in any of the A_j or D_j that the play passes through that starts from a_i and results from σ and $\bar{\sigma}$. Hence our improving move prefers a switch in one of the gadgets that are in this play. Moreover, if the improving switch is in A_j or D_j is the first of the two switches, then this only slightly improves the valuation a_j or d_j as we saw before, as only one priority is removed from the path in the valuation, namely some $12(n - j) + k$ with $k \in \{3, 5, 9, 11\}$. And no $\sigma(A_j)$ or $\sigma(D_j)$ change. This results in the same change in the valuation of a_i, so one priority $12(n-j)+k$ is removed from the path. Hence, the improving move that increases $\sigma(a_i)$ the most is the one in the gadget where the highest odd priority would be removed, and that is the one with the lowest j in the play starting from a_i. Obviously, that is a switch in A_i, at the start of the play and that is exactly the gadget with the lowest valuation a_i.

Moreover, after the first improving move, all valuations barely change, so a_i is still the node with the lowest valuation. Now making a second improving move in A_i will improve $\Xi_\sigma(a_i)$ by a lot since $\sigma(A_i)$ changes, and any other improving move is not significant. Therefore, for the second improving move, also A_i is preferred, which is still the gadget with the lowest valuation a_i. Proof for when d_i has the lowest valuation is similar.

From this we conclude that the highest-lexicographic rule prefers an improving move in the gadget with the lowest valued a_i and d_i. Therefore, the rule is counter-conservative for the strategies encountered in strategy iteration with our starting strategy. Hence it follows from Theorem 3.10 that strategy iteration with this rule requires at least a number of iterations exponential in n. Because the number of nodes and edges is linear in n, the running time is also exponential in the number of nodes and edges. □

We conclude this section by showing an exponential lower bound for running time of the last improvement rule.

Theorem 3.13. *Strategy iteration with the lowest-valuation-shortest-path improvement rule has worst-case running time that is at least exponential in the number of nodes and edges.*

Proof: Recall that this rule prefers a switch in v to v' over a switch in w to w' if $\Xi_\sigma(v) \uplus M_\sigma(w, w') \trianglelefteq \Xi_\sigma(w) \uplus M_\sigma(v, v')$. Here $M_\sigma(v, v')$ is the shortest possible path from v to any other node in the graph $G_\sigma \cup \{(v, v')\}$. Recall that G_σ is the subgraph of G that contains edges from σ and the edges of E_1, which come from nodes in V_1. We define again gadgets

for the reverse binary counter structure, which are shown in Figure 19. However, there are some slight changes from the 'standard' reverse binary counter. We split node a_i into two nodes of very similar priority, namely a_i and a'_i. The play is forced to go through either a_i or a'_i, like in the binary counter it is forced to go through a_i. Likewise, we split d_i into d_i and d'_i. Moreover, we add nodes z_1 and z_2 to the graph, where z_1 only has an outgoing edge towards z_2 and z_2 only towards x. Node z_1 has two incoming edges from every gadget.

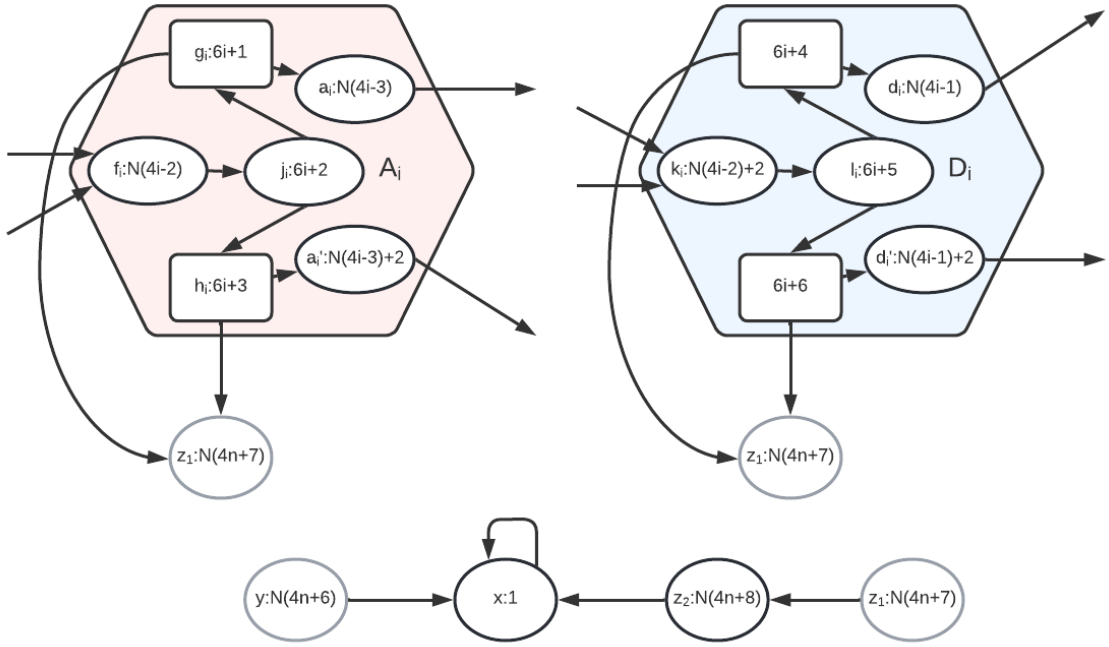


FIGURE 19: Gadgets for the lowest-valuation-shortest-path rule, and the nodes z_1 and z_2. Nodes that do not have all their outgoing or incoming edges shown are outlined in grey

Note that only player 1 has the choice of going to z_1. However, if he goes to z_1, then the play will pass z_2, which has the largest even priority in the game. Therefore, player 1 will never choose to go to z_1. Moreover, the optimal strategies, improving moves and such are still the same as in the reverse binary counter structure. For the rest, the only difference is that we define a counter-conservative improvement rule in this context as a rule that prefers switches in the gadget where either a_i or a'_i (or d_i or d'_i) has the lowest valuation instead of just a_i (or d_i). Note that if there is an improving move possible in j_i, then j_i (or l_i) points towards the lowest-valued of a_i and a'_i (or d_i and d'_i). Then we could also call an improvement rule counter-conservative if it prefers the gadget where j_i (or l_i) has the lowest valuation out of all other j_i and l_i where there are improving moves. Then we could prove analogous to Theorem 3.10 that if this improvement rule prefers to switch in the gadget with the lowest valued j_i or l_i like in a counter-conservative improvement rule, then strategy iteration takes an exponential number of iterations.

Now we look at the choices of the lowest-valuation-shortest-path rule. Say we currently have player 0 strategy σ and the improving move would yield σ'. Note that the only nodes where player 0 has a choice are j_i and l_i for $i = 1, 2, \ldots, n$, so these are the nodes that can have improving moves. First, we want to find $M_\sigma(v, v')$ for an arbitrary node v with an improving move towards v'. Clearly the shortest path that determines $M_\sigma(v, v')$ is a path

towards z_2, since only then does it contain a node with priority $N(4n+7)$ (and not with $N(4n+8)$). Suppose $v = j_i$ for some i. Then we are looking for the shortest j_i-z_2-path that uses edges from σ, σ' or E_1. Note that two such paths are (j_i, g_i, z_1, z_2) and (j_i, h_i, z_1, z_2).

The other option to reach z_2 is to go to A_m or D_m for some $m > i$ (whichever one is reachable, depending on σ) and go from there to z_1. However, then the path passes either f_m or k_m, which have quite large even priority. In particular, this means that the resulting path is $\preceq$-longer than (j_i, g_i, z_1, z_2) and (j_i, h_i, z_1, z_2). We conclude that the shortest j_i-z_2-path is (j_i, h_i, z_1, z_2), hence $M_\sigma(v, v') = \{j_i, h_i, z_1\}$. We get a similar result for $v = l_i$.

Now we see that $M_\sigma(v, v')$ contains only some nodes with priority at most $6(n+6)$, and z_1. So the values of $M_\sigma(j_i)$ and $M_\sigma(l_i)$ are very 'close' to each other with respect to $\preceq$. Their differences are in particular irrelevant compared to the priorities of a_i, a'_i, d_i, d'_i, f_i and k_i for any i. Hence if we compare two improving moves, one from v towards node v', and one from w towards w', then we need to decide whether $\Xi_\sigma(v) \uplus M_\sigma(w, w') \trianglelefteq \Xi_\sigma(w) \uplus M_\sigma(v, v')$. This is then the case if and only if $\Xi_\sigma(v) \trianglelefteq \Xi_\sigma(w)$, since the difference between Ξ_σ for two different nodes that have a choice is much larger than the difference between M_σ (since they differ in which a_i, a'_i, d_i or d'_i the paths contain). Note that if there is an improving move in j_i, then $\sigma(j_i)$ is currently the lowest valued of a_i and a'_i. Therefore $\Xi_\sigma(j_i)$ is almost the same as the lowest valuation of a_i and a'_i. So in particular, if a_i or a'_i has the lowest valuation of all a_i, a'_i, d_i, d'_i where there is an improving move in the same gadget, then j_i is the preferred improving move by our improvement rule, so we prefer to switch in A_i.

Likewise, if d_i or d'_i has the lowest valuation, then we prefer to switch from l_i in D_i. From this, we conclude that this improvement rule behaves like a counter-conservative improvement rule, hence strategy iteration takes a number of iterations exponential in n. Because the number of nodes and edges is linear in n, the running time is also exponential in the number of nodes and edges. □

4 An alternative lower bound proof technique

In this section we explore a link between parity games and linear programming. First, we look at a new algorithm for solving parity games, called subgame iteration. This algorithm is somewhat similar to strategy iteration, but it considers partial strategies instead of strategies. We also look at a specific linear program that can be constructed for any parity game, and we show at the end that this LP can be used to solve the parity game. The main purpose of this section is then to show that the subgame algorithm behaves the same as the simplex algorithm on this related linear program. We show that the intermediate solutions of both algorithms have a one-to-one correspondence and that both algorithms need the same amount of iterations. This may show to be very useful, as this provides a way of translating lower bound proofs for the complexity of subgame iteration into linear programs that exhibit the same behavior for the simplex algorithm.

First some technical assumptions. We assume in this section that all priorities occurring are unique, and that there are n nodes in the game G, and that the priorities are at least 1 and at most $2n$.

Moreover, some notation is used throughout this section. First of all we extend the linear order $\preceq$ from Section 2.4 to the space $\mathcal{M}(Q)$ of multisets of Q (recall that Q is the set of priorities that occur in the graph). We say $B_1 \prec B_2$ if and only if:

- $m := \max((B_1 \backslash B_2) \cup (B_2 \backslash B_1)) \in B_1$ and m is odd
- $m := \max((B_1 \backslash B_2) \cup (B_2 \backslash B_1)) \in B_2$ and m is even

. For ease of notation we also compare multisets of nodes (instead of priorities) with $\preceq$. We say that, if C_1 and C_2 are multisets of nodes, and B_1 and B_2 are the multisets of priorities of the nodes in C_1 and C_2, respectively, then $C_1 \preceq C_2$ if and only if $B_1 \preceq B_2$. We even allow comparison between multisets of nodes and of priorities, like $B_1 \preceq C_2$ (which means $B_1 \preceq B_2$).

4.1 The subgame improvement algorithm

Now we define the *subgame improvement algorithm*. The algorithm maintains two sets: S and T, where both are empty at the start of the algorithm. Throughout the algorithm, we require the following of the sets:

- $S \subseteq E$, and if $(v, w) \in S$, then $v \in V_0$, and there is no other edge (v, z) in S.
- $T \subseteq V_1$.

The set S can be seen as a partial strategy for player 0, and set T can be seen as a set of player 1 controlled nodes that are somewhat good for player 0. Associated with a pair of sets S, T is the valuation function $\Psi_{S,T} : V \to \mathcal{P}(Q \cup \{2n+2\})$. This function is quite similar to the second component (path component) of Ξ_σ for strategy iteration. Moreover, it is also comparable to the distances in algorithm to solve the longest shortest path problem by Björklund et al. [4]. The difference is that, as we show, the distances here are always better than 'zero', which is represented by the empty set for subgame iteration. The space of valuations contains only subsets of Q, so we can compare valuations with the linear order $\preceq$. The main idea of the algorithm is that it iteratively improves the valuations of S and T by making small changes. Note that this linear order is equivalent to saying $B_1 \preceq B_2$ if and only if $(1, B_1, 0) \trianglelefteq (1, B_2, 0)$.

The valuation $\Psi_{S,T}$ is calculated as follows. Let $G_{S,T}$ be the subgraph of G with only edges that are either in S or come from a node in T. We then have the following:

- $\Psi_{S,T}(v) = \emptyset$ if v has no outgoing edges in $G_{S,T}$.
- $\Psi_{S,T}(v) = \{2n+2\}$ if from v we cannot reach any nodes that do not have outgoing edges in $G_{S,T}$.
- Else, $\Psi_{S,T}(v)$ is equal to the value of the shortest path in $G_{S,T}$ towards a node without outgoing edges. The value of a path $(v, v_1, v_2, \ldots, v_k)$, where v_k has no outgoing edges in $G_{S,T}$, is given by the set $\{p(v), p(v_1), p(v_2), \ldots p(v_{k-1})\}$. The shortest path is the path with the $\preceq$-smallest set. Existence of such a shortest path is proved later.

We now call S', T' an *improvement* of S, T if S', T' are created from S, T by one of the following operations:

- For a vertex $v \in V_0$ that has no outgoing edges in $G_{S,T}$, add an edge (v, w) to S with $\Psi_{S,T}(w) \cup \{p(v)\} \succ \emptyset$.
- Replace an edge $(v, w) \in S$ - if $v \in V_0$ and $\Psi_{S,T}(v) \neq \{2n+2\}$ - by another edge (v, z) such that $\Psi_{S,T}(z) \succ \Psi_{S,T}(w)$.
- Take a node $v \in V_1$ that has the property that $\Psi_{S,T}(w) \cup \{p(v)\} \succ \emptyset$ for all edges $(v, w) \in E$, and add it to T.

Now the algorithm works as follows:

Algorithm 2 Subgame improvement

```
start with S = T = ∅
repeat
    (S, T) ← (S', T')
    if an improvement to S, T exists then
        Let S', T' be an improvement to S, T
    else
        (S', T') = (S, T)
    end if
until (S, T) = (S', T')
return S, T
```

Example

In the following graph G (Figure 20) with 4 nodes, suppose we currently have $S = \{(v_2, v_1)\}$ and $T = \{v_4\}$. The graph $G_{S,T}$ then contains the edge that is in S and the two outgoing edges from v_4. We can find the valuations $\Psi_{S,T}$ of the nodes of G using the subgraph $G_{S,T}$. Nodes v_1 and v_3 have no outgoing edges in $G_{S,T}$, so $\Psi_{S,T}(v_1) = \Psi_{S,T}(v_3) = \emptyset$. Node v_2 has exactly one path towards a node without outgoing edges, which is the path (v_2, v_1). hence its valuation is $\{p(v_2)\} = \{2\}$. Node v_4 has two possible paths towards nodes without outgoing edges: towards v_1 or v_3. The shortest of the two is the one towards v_3, hence $\Psi_{S,T}(v_4) = \{4\}$.

Now we can look at improving moves for this S, T. Adding node v_1 to T can only be an improving move if $\Psi_{S,T}(v_1) \cup \{p(v_1)\} \succ \emptyset$, since (v_1, v_1) is an outgoing edge from v_1. However, $\emptyset \cup \{1\} \prec \emptyset$, so adding v_1 is not an improving move. Replacing edge (v_2, v_1) by edge (v_2, v_3) is improving if $\Psi_{S,T}(v_3) \succ \Psi_{S,T}(v_1)$, which is not true, as $\Psi_{S,T}(v_3) = \Psi_{S,T}(v_1) = \emptyset$. Finally, adding edge (v_3, v_4) to S is improving if $\Psi_{S,T}(v_4) \cup \{p(v_3)\} \succ \emptyset$. In this case, $\Psi_{S,T}(v_4) \cup \{p(v_3)\} = \{3, 4\} \succ \emptyset$, so this is an

improving move. So in this case, there is only one improving move possible, and in the next iteration we get $S = \{(v_2, v_1), (v_3, v_4)\}$ and $T = \{v_4\}$.

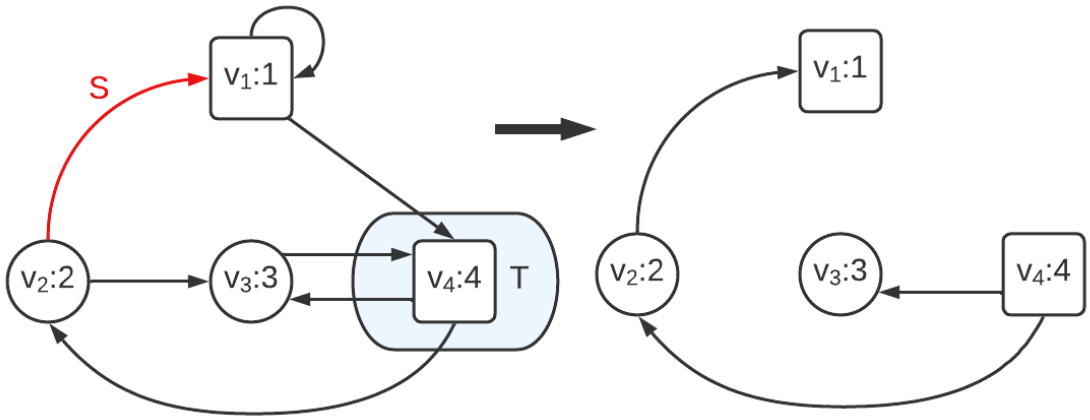


FIGURE 20: The graph G with node priorities, and the resulting subgraph $G_{S,T}$.

The subgame iteration algorithm has some similiarities to strategy improvement as in [18]. Its valuation is almost the same as the second component of the valuation in strategy improvement, except that we always have $\Psi \succeq \emptyset$ (we prove this later) and that there is some 'infinity'-element $\{2n+2\}$. Moreover, we see in Section 5 that under some conditions, subgame iteration and strategy iteration behave exactly the same on sink parity games. In the remainder of this section, we show how subgame iteration can be linked to an LP for any parity game.

4.2 Linear programming formulation

We now introduce a linear program, on which we show later that the simplex algorithm behaves similar to the subgame iteration algorithm on a parity game. This linear program is constructed from an arbitrary parity game on the graph $G = (V, E)$ where, of course $V = V_0 \cup V_1$. Let $E_i = \{(v, w) \in E | v \in V_i\}$ for $i \in \{0, 1\}$. The LP contains the following variables:

- For every node v in V_1, a variable t_v. The idea is that in the simplex algorithm, t_v being in the basis corresponds to v being in T for subgame iteration.
- For every edge $e \in E_0 = \{(v, w) \in E : v \in V_0\}$, we have a variable s_e. The idea is that in the simplex algorithm, s_e being in the basis corresponds to edge e being in S for subgame iteration.
- For every node $v \in V$, we have a variable z_v. The idea is that z_v being in the basis in the simplex algorithm corresponds to v not having any outgoing edges in graph $G_{S,T}$ in the subgame algorithm (so $v \in V_1 \backslash T$ or $v \in V_0$ and there is no edge (v, w) in S).

The LP has an equation for every node in V. In matrix form, it is as follows:

$$\begin{aligned}
\min \quad & \mathbf{1}^T \boldsymbol{z} \\
s.t. \quad & A \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{s} \\ \boldsymbol{t} \end{bmatrix} = \mathbf{1} \\
& \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{s} \\ \boldsymbol{t} \end{bmatrix} \geq \mathbf{0}
\end{aligned}$$

$$\text{where} \quad A_{ij} = \begin{cases} 1 & j \text{ corresponds to } z_i,\ s_{(i,i')} \text{ or } t_i \\ -b^{-(-3)^{p(i')}} & j \text{ corresponds to } s_{(i',i)} \text{ or } t_{i'} \text{ with } (i',i) \in E_1 \\ 0 & \text{otherwise} \end{cases}$$

Where $b = n^{3n}$, and where the variables are vectors $\boldsymbol{s} \in \mathbb{R}^{E_0}$, $\boldsymbol{t} \in \mathbb{R}^{V_1}$ and $\boldsymbol{z} \in \mathbb{R}^V$. In equation form, the LP is as follows:

$$\begin{aligned}
\min \quad & \sum_{v \in V} z_v \\
s.t. \quad & \sum_{v':(v',v)\in E_0} -b^{-(-3)^{p(v')}} s_{(v',v)} + \sum_{v':(v',v)\in E_1} -b^{-(-3)^{p(v')}} t_{v'} \\
& \qquad + \sum_{v':(v,v')\in E_0} s_{(v,v')} + z_v = 1 && \forall v \in V_0 \\
& \sum_{v':(v',v)\in E_0} -b^{-(-3)^{p(v')}} s_{(v',v)} + \sum_{v':(v',v)\in E_1} -b^{-(-3)^{p(v')}} t_{v'} \\
& \qquad + t_v + z_v = 1 && \forall v \in V_1 \\
& \qquad s_e, t_v, z_{v'} \geq 0 && \forall e \in E_0, v \in V_1, v' \in V
\end{aligned} \tag{2}$$

This LP (2) can be seen as the dual of the LP described by Schewe [24][2]. As an initial solution for the simplex algorithm, we take $\boldsymbol{z}$ as a basis, leading to the basic feasible solution $z_v = 1\ \forall v \in V$.

[2]If player 0 and player 1 are switched, and the variables in the LP described by [24] are free variables instead of just nonnegative, as the optimum is attained for nonnegative variables, then taking the dual yields the LP (2).

4.3 Illustration of subgame iteration and simplex algorithm

We use the same graph G as before (see Figure 21).

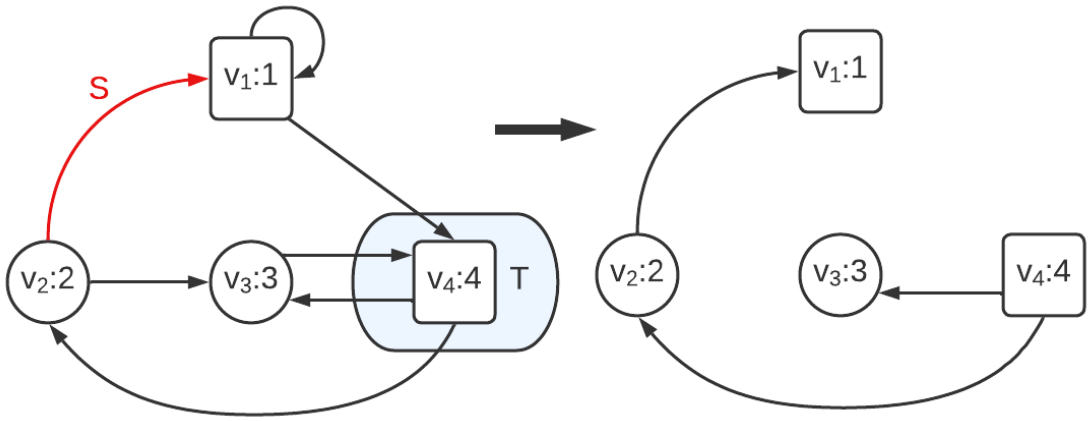


FIGURE 21: The graph G with node priorities, and the graph $G_{S,T}$.

For this graph, we have the following linear program:

$$\begin{array}{lrl} \min & z_{v_1} + z_{v_2} + z_{v_3} + z_{v_4} & \\ s.t. & -b^{-9}s_{(v_2,v_1)} - b^3 t_{v_1} + t_{v_1} + z_{v_1} & = 1 \\ & -b^{-81}t_{v_4} + s_{(v_2,v_1)} + s_{(v_2,v_3)} + z_{v_2} & = 1 \\ & -b^{-9}s_{(v_2,v_3)} - b^{-81}t_{v_4} + s_{(v_3,v_4)} + z_{v_3} & = 1 \\ & -b^{27}s_{(v_3,v_4)} - b^3 t_{v_1} + t_{v_4} + z_{v_4} & = 1 \\ & \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{z} & \geq 0 \end{array} \tag{3}$$

Suppose we currently have the basis $s_{(v_2,v_1)}, t_{v_4}, z_{v_1}, z_{v_3}$ in the simplex algorithm. The corresponding basic (feasible) solution has $t_{v_4} = 1$, $s_{(v_2,v_1)} = z_{v_3} = 1 + b^{-81}$ and $z_{v_1} = 1 + b^{-9} + b^{-90}$. This basis corresponds to $S = \{(v_2, v_1)\}$ and $T = \{v_4\}$ in subgame iteration (we formalize what this correspondence exactly means later). Now we want to see what improving pivoting moves there are. If adding variable x to the basis is improving, that is the same as saying that if we increase x by a little bit (and compensate for it with the basic variables), then the objective function slightly decreases.

So, is adding variable t_{v_1} to the basis improving? If we increase it slightly, then we can decrease z_{v_1} by that amount in the first equation of 3, but we also must compensate by increasing z_{v_1} by b^3 times as much to correct in the first equation of the LP. We also must increase t_{v_4} in the fourth equation, because of which we must increase z_{v_3} in the third equation and some more variables as a result, but we ignore that part for now. From the first equation we know that the objective function (in particular z_{v_1}) increases by b^3 as much as it decreases, so adding t_1 is not improving. In particular, because of the loop on v_1 and its odd priority. In subgame iteration, adding v_1 was also not improving because of this loop.

We may also wonder if adding $s_{(v_2,v_3)}$ to the basis is improving. If we increase $s_{(v_2,v_3)}$ by a little amount ϵ, then $s_{(v_2,v_1)}$ decreases by ϵ to make the second equation of 3 correct. Then, z_{v_1} decreases by $b^{-9}\epsilon$, while z_{v_3} increases by $b^{-9}\epsilon$ because of the first and third equation. So the objective function stays the same, hence adding $s_{(v_2,v_3)}$ is not an improving pivot. Note

that the reason for this is that both z_{v_1} and z_{v_3} are basic, or in terms of subgame iteration, we have that both v_1 and v_3 have no outgoing edges, so $\Psi_{S,T}(v_1) = \Psi_{S,T}(v_3) = \emptyset$. Recall that also switching edge (v_2, v_1) for edge (v_2, v_3) was not improving in subgame iteration.

Finally, we may wonder if adding variable $s_{(v_3,v_4)}$ to the basis is improving. If we increase $s_{(v_3,v_4)}$ by a small amount ϵ, then z_{v_3} decreases by ϵ to make the third equation of (3) correct. Because of the fourth equation, t_{v_4} must increase by $b^{27}\epsilon$. Therefore in the third equation, z_{v_3} must increase by $b^{27-81}\epsilon$. Also, in the second equation, we see that because t_{v_4} increases by $b^{27}\epsilon$, the variable $s_{(v_2,v_1)}$ must increase by $b^{27-81}\epsilon$. Finally, for that reason in the first equation, we must increase z_1 by $b^{27-81-9}$. In summary, z_3 decreases by ϵ, and z_1 and z_3 increase by the small amounts $b^{27-81-9}\epsilon$ and $b^{27-81}\epsilon$. Hence $s_{(v_3,v_4)}$ is an improving pivot. We could say this in a more general way: for every path P from v_4 to a node w without outgoing edges, following the train of logic along the paths we found that z_w must increase by an amount

$$b^{-\sum_{v \in P \cup \{v_3\}} (-3)^{p(v)}} \epsilon$$

In particular, if $\Psi_{S,T}(v_4) \cup p(v_3) \succ 0$, then $P \cup v_3 \succ 0$ for every path from v_4 to a node without outgoing edges. Then the exponent in the above equation is negative, so the increase in z_w is negligible compared to the decrease in z_{v_3}. This is the main idea of the link between the simplex algorithm and subgame iteration: we link the question whether a variable is improving (negative reduced cost) to the question whether a certain set of priorities is $\preceq$-better than the empty set or another valuation Ψ. The remainder of this section proves this rigorously.

4.4 Proof outline

Our goal is to show that the subgame iteration algorithm on a game on G and the simplex algorithm on (2) can be considered the same algorithm. In each step, their options for choosing an improving move and choosing a variable to pivot over have a one-to-one correspondence. This is very useful, as this allows us to construct an example with exponential running time for some improvement rule in the subgame iteration algorithm, which then immediately implies that the simplex algorithm with a related pivot rule has exponential running time. In Section 5, we construct this relation explicitly for some well-known pivot rules.

The largest part of this section is devoted to proving that the two algorithms are 'the same'. To do so, we first find estimates for the elements of B^{-1}, where B is a basis that can be encountered in the simplex algorithm. We show that there is a close relation between elements of B^{-1} and $\preceq$-shortest paths in the graph $G_{S,T}$. This is shown in Lemma 4.5. But before that, we first prove a few lemmas that help us get to this estimate.

If we then have estimated the elements of B^{-1}, we can explicitly calculate the reduced costs of the linear program, the values of the basic feasible solution and the elements of $B^{-1}A$, which determine which variable would leave the basis. We can then check all the cases one by one to see that the options for pivoting are the same as the options for improving moves, hence showing the two algorithms are 'the same'.

Finally, we show that the subgame iteration algorithm and the simplex algorithm on (2) both solve the parity game. This means that they find a winning strategy for player 0 and find the winning set for player 0.

Recall that $G_{S,T}$ is the subgraph of G which contains only the edges of S and outgoing edges from T. Also, recall that we assumed that all priorities in the graph are unique. Also, whenever we mention a path P, that starts at a vertex i and ends on a vertex j, we do <u>not</u> count the endpoint of the path as being *in* the path.

Moreover, we use the notation

$$b(P) := b^{-\sum_{v \in P}(-3)^{p(v)}}$$

where P is a path, walk or multiset of nodes. We also use the same notation for multisets of priorities, e.g.

$$b(M) := b^{-\sum_{p \in M}(-3)^{p}}$$

where $M \in \mathcal{M}(Q)$ is a multiset of priorities. Note that, if X and Y are both multisets or walks where every node or priority occurs at most twice, then

$$b(X) \leq b(Y) \Leftrightarrow X \succeq Y$$

we often use this fact to show connections between values of matrices and values of paths.

4.5 Uniqueness of valuation and values in reverse basis

We show that valuations are unique in subgame iteration and we estimate the values of B^{-1}. First, we prove a statement about matrices, which helps to bound error terms in calculating B^{-1}.

Lemma 4.1. *Let $A \in \mathbb{R}_{\geq 0}^{n \times n}$ be a square nonnegative matrix. We view A as the matrix of edge weights in a complete directed graph G (recall that this graph contains all possible edges). This means A_{ij} represents the weight of edge (j, i). Suppose the product of all the edges in any cycle is bounded from above by some constant number c, with $0 < c < 1$. More formally, this means the following: let $1 \leq l \leq n$, and let $q_1, q_2, \ldots, q_{l+1}$ be a sequence of distinct elements of $[n]$, except that $q_1 = q_{l+1}$. Then for any such sequence we have*

$$\prod_{m=1}^{l} A_{q_m q_{m+1}} \leq c$$

Suppose also that M_{ij} is an upper bound on the product of the edge weights on any path from j to i for all pairs i, j in $[n]$. Formally, this means for any sequence $q_1, q_2, \ldots, q_{l+1}$ with $0 \leq l \leq n-1$ of distinct elements of $[n]$ (except maybe $q_1 = q_{l+1}$), that satisfies $q_1 = i$, $q_{l+1} = j$, we have

$$\prod_{m=1}^{l} A_{q_m q_{m+1}} \leq M_{ij}$$

(note that $l = 0$ is allowed here so $M_{ii} \geq 1$). Then we can bound the elements of powers of A as follows:

$$A_{ij}^{p} \leq n^{p-1} c^{\lfloor \frac{p}{n} \rfloor} M_{ij} \; \forall i, j \in [n], p \in \mathbb{N}$$

Proof: We write $p = kn + k' - 1$, with $k = \lfloor \frac{p}{n} \rfloor$, and $k' \in \mathbb{N}$. Consider the matrix $A^{kn+k'-1}$. Note that we can explicitly write the elements of $A^{kn+k'-1}$ as the following sum:

$$A_{i,j}^{kn+k'-1} = \sum_{\substack{q_1, q_2, \ldots, q_{kn+k'} \in [n] \\ q_1 = i,\, q_{kn+k'} = j}} \prod_{m=1}^{kn+k'-1} A_{q_m q_{m+1}} \tag{4}$$

We prove the lemma by bounding this sum from above. Consider a term $\prod_{m=1}^{kn+k'-1} A_{q_m q_{m+1}}$ of this sum. We could view this as the product of the edges on a $kn+k'-1$-edge long walk from node $q_{kn+k'}$ to q_1. Because the q_m can only have n different values and because $kn+k' \geq n+1$, we can find m_1, m_2 in $\{1, 2, \ldots, kn+k'\}$ with $q_{m_1} = q_{m_2}$ and $0 < m_2 - m_1 \leq n$ by the pigeon hole principle, and the q_m in between q_{m_1} and q_{m_2} unique. Because of the assumption of the lemma, we know $\prod_{m=m_1}^{m_2-1} |A_{q_m q_{m+1}}| \leq c$. Then we can create a sequence $q'_1, q'_2, \ldots, q'_{kn+1-m_2+m_1}$ by removing $q_{m_1}, q_{m_1+1}, \ldots, q_{m_2-1}$ from the sequence $q_1, \ldots, q_{kn+k'}$. We could view this as removing a cycle from the $kn+k'-1$-edge long path. This results in the following inequality:

$$\prod_{m=1}^{kn+k'-1} A_{q_m q_{m+1}} \leq c \cdot \prod_{m=1}^{kn+k'-1-m_2+m_1} A_{q'_m q'_{m+1}}$$

We can, moreover, repeat this process of 'cutting out cycles' from the sequence and bounding the term until we cannot go any further, and then the resulting sequence has at most n terms. Say we have as a result the sequence $q''_1, q''_2, \ldots, q''_{l+1}$ with $0 \leq l \leq n-1$. Then we have

$$\prod_{m=1}^{kn+k'-1} A_{q_m q_{m+1}} \leq c^f \cdot \prod_{m=1}^{l} A_{q''_m q''_{m+1}}$$

where $f \in \mathbb{Z} \geq 0$ and $0 \leq l \leq n-1$, and we know $f \geq k$, since in each step at most n elements are removed from the sequence and we started with $kn+k'$ elements. Note that still $q_1 = q''_1$ and that $q_{kn+k'} = q''_{l+1}$, and also the elements in the sequence $q''_1 \ldots, q''_{l+1}$ are unique. Then, we know

$$\prod_{m=1}^{kn+k'-1} A_{q_m q_{m+1}} \leq c^f \cdot \prod_{m=1}^{l} A_{q''_m q''_{m+1}} \leq c^k \cdot M_{ij} \tag{5}$$

Finally, note that there are $n^{kn+k'-2}$ sequences of length $kn+k'$ where the first element is i and the last is j if $kn+k' \geq 2$, so the sum (4) has $n^{kn+k'-2}$ terms. Then (4) yields

$$A_{i,j}^{kn+k'-1} \overset{(4)}{\leq} \sum_{\substack{q_1, q_2, \ldots, q_{kn+k'} \in [n] \\ q_1 = i,\, q_{kn+k'} = j}} \prod_{m=1}^{kn+k'-1} A_{q_m q_{m+1}} \overset{(5)}{\leq} \sum_{\substack{q_1, q_2, \ldots, q_{kn+k'} \in [n] \\ q_1 = i,\, q_{kn+k'} = j}} c^k \cdot M_{ij} \leq n^{kn+k'-2} c^k M_{ij} \tag{6}$$

and this is exactly what we needed to prove. ■

Before we start to estimate the elements of B^{-1}, we first need to restrict ourselves to bases that we can actually encounter in the simplex algorithm. We introduce the notion of *nice* pairs of sets and a *nice* basis, and we show later that all the bases that we encounter with the simplex algorithm are nice. Thus we will only need to calculate B^{-1} for a nice basis.

Definition 4.2. We call a pair of sets (S, T) *nice* if they satisfy the following conditions:

- $S \subseteq E$, and if $(v, w) \in S$, then $v \in V_0$, and there is no other edge (v, z) in S.
- $T \subseteq V_1$.

- In the subgraph $G_{S,T}$, there is no cycle with as its highest priority an odd number.
- In the subgraph $G_{S,T}$, for any path P that ends on a node without outgoing edges, we have $P \succeq \emptyset$.

We call a basis of the LP (2) a *nice S,T-basis* if (S,T) is nice and the basic variables consist exactly of the $s_{(v,v')}$ for the edges $(v,v') \in S$, the t_v for $v \in T$ and the z_v of the vertices v that have no outgoing edges in $G_{S,T}$. We also require the related basic solution to be feasible for (2). We let the basis be the submatrix B of A, where the i-th column of B corresponds to either z_i, the edge in S coming from v_i or t_i (we can assume so since we could just reorder the columns of A and the variables corresponding to these columns).

Next, we show that shortest paths are unique in $G_{S,T}$, and that these shortest paths are related to values of $\prod_{m=1}^{l-1} |B_{q_m q_{m+1}}|$. Also, we show that the values of the valuation Ψ are unique if they are not equal to $\emptyset$ or $\{2n+2\}$.

Lemma 4.3. *(uniqueness of shortest path) Suppose we have a nice (S,T) and the corresponding submatrix B of A (with those columns as defined in Definition 4.2). Then the following statements hold:*

1. *For any i,j, if a path exists from i to j in $G_{S,T}$, then the $\preceq$-shortest walk P from i to j in the graph $G_{S,T}$ exists and is unique.*
2. *For any i,j, if a sequence $q_1, q_2, \ldots, q_{l+1} \in [n]$ exists such that $\prod_{m=1}^{l} |B_{q_m q_{m+1}}|$ is nonzero, then the largest possible value of $\prod_{m=1}^{l} |B_{q_m q_{m+1}}|$ is attained for one unique sequence $q_1, q_2, \ldots, q_{l+1} \in [n]$. Moreover, the largest possible value of $\prod_{m=1}^{l} |B_{q_m q_{m+1}}|$ is equal to*

$$b(P) = b^{-\sum_{v \in P} (-3)^{p(v)}}$$

Proof: First, we prove the first statement. Since there are no cycles with an odd number as the highest priority, any walk with cycles can be made $\preceq$-shorter by cutting out cycles. Hence it is clear that if there is a path from i to j, any $\preceq$-shortest walk must be a path, and as there are finitely many paths, a shortest path exists. Suppose there are two shortest paths from i to j with the same value, say P_1 and P_2. Since the priorities of all vertices are unique, the two paths must contain the exact same vertices, just in a different order. In particular, we can say $P_1 = (P_1^1, w, P_1^2)$ and $P_2 = (P_2^1, w, P_2^2)$ with $\{p | p \in P_1^1\} \neq \{p | p \in P_2^1\}$ for some vertex w.[3] W. l. o. g. $P_1^1 \prec P_2^1$. This implies $P_1^2 \succ P_2^2$, as the value of P_1 and P_2 is equal. But then (P_1^1, w, P_2^2) is an i,j-path as well, with a shorter distance than P_1 and P_2, and that is a contradiction. For the second part, we see that for two i,j-paths P_1 and P_2, we have $P_1 \preceq P_2$ if and only if

$$b(P_1) = b^{-\sum_{v \in P_1} (-3)^{p(v)}} \geq b^{-\sum_{v \in P_2} (-3)^{p(v)}} = b(P_2)$$

and suppose $P_k = (q_{l+1}^k = i, q_l^k, q_{l-1}^k, \ldots, q_1^k)$ with $q_1^k = j$ (recall we do not count the endpoint of a path) for $k = 1, 2$. Also, recall that $|B_{k_1 k_2}|$ for $k_1 \neq k_2$ equals $-b^{(-3)^{p(k_2)}}$ if $(k_2, k_1) \in G_{S,T}$ and 0 otherwise. Then we also have $P_1 \preceq P_2$ if and only if

$$\prod_{m=1}^{l} |B_{q_m^1 q_{m+1}^1}| = b(P_1) \geq b(P_2) = \prod_{m=1}^{l} |B_{q_m^2 q_{m+1}^2}|$$

[3] For P_1^1 and P_2^1 we make an exception and do count the last vertex as being in the path to keep notation simple.

hence because the $\preceq$-shortest i,j-path (if it exists) is unique, we also see that the sequence that has the largest $\prod_{m=1}^{l} |B_{q_m q_{m+1}}|$ is unique. Also, if there is no i,j-path, then there is also no sequence that yields a positive $\prod_{m=1}^{l} |B_{q_m q_{m+1}}|$, so its largest value is 0. ■

Lemma 4.4. *(uniqueness of valuation) If (S,T) is nice and we have two distinct vertices v,w with $\Psi_{S,T}(v), \Psi_{S,T}(w) \neq \emptyset$, $\Psi_{S,T}(v), \Psi_{S,T}(w) \neq \{2n+2\}$, then $\Psi_{S,T}(v) \neq \Psi_{S,T}(w)$.*
Proof: Suppose $\Psi_{S,T}(v) = \Psi_{S,T}(w)$. Then let P_1 be the shortest path from v to a node without outgoing edges in $G_{S,T}$, and let P_2 be defined likewise for node w. We see that P_1 and P_2 contain the same priorities, hence by uniqueness they contain the exact same nodes, only in a different order. We know P_1 starts with v, and P_2 with w, and therefore both paths contain at least two nodes. Then we can say $P_1 = (P_1^1, x, P_1^2)$ and $P_2 = (P_2^1, x, P_2^2)$, for some vertex x and with $\{p | p \in P_1^1\} \neq \{p | p \in P_2^1\}$.[4] W. l. o. g. $P_1^1 \succ P_2^1$, and that implies $P_1^2 \prec P_2^2$. Hence (P_2^1, x, P_1^2) is a path from w to a node without outgoing edges in $G_{S,T}$ that is $\preceq$-shorter than P_2, and that is a contradiction. ■

Now that we proved these results, we are ready to estimate the elements of B^{-1}.

Lemma 4.5. *If (S,T) is nice, then there is a corresponding nice S,T-basis B for (2). Moreover, the inverse of B can be approximated in the following way:*

1. *$1 \leq B_{ii}^{-1} \leq 1 + \frac{2}{n^n}$ for $i \in [n]$.*
2. *If there is no cycle containing i in $G_{S,T}$, then $B_{ii}^{-1} = 1$.*
3. *If $i \neq j$ and there is no path from i to j in $G_{S,T}$, then $B_{ji}^{-1} = 0$.*
4. *If $i \neq j$ and there is a path from i to j in $G_{S,T}$, then let P be the $\preceq$-shortest path from i to j. then*

$$b(P) = b^{-\sum_{v \in P}(-3)^{p(v)}} \leq B_{ji}^{-1} \leq b(P)(1 + \frac{2}{n^n})$$

Proof: The proof is outlined as follows. We first show that there is a nice S,T-basis. After that, we estimate the elements of B^{-1} using the terms of an infinite series.

First, we need to show that there is a nice S,T-basis. Suppose B is the submatrix of A consisting of the columns that correspond to the s,t,z that are required to be in the basis according to Definition 4.2. To prove that B is nice, it suffices to show that B is invertible and the related basic solution is feasible. We assume w.l.o.g. that the i-th column of B corresponds to either z_i, $s_{(i,j)}$ or t_i (as we could just change the order of the columns in A and the order of the elements of $\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{z}$).

We introduce the matrix $\tilde{B} \in \mathbb{R}^{n \times n}$, which we define as follows:

$$\tilde{B}_{ij} = \begin{cases} b^{-(-3)^{p(j)}} & (j,i) \in S \text{ or } (j,i) \in E_1 \text{ and } j \in T \\ 0 & \text{otherwise} \end{cases}$$

Note that $B = I - \tilde{B}$. Also note that $\tilde{B}$ can be seen as a matrix containing priority-dependent positive edge weights for all edges in the graph G. Now we look at the series $I + \tilde{B} + \tilde{B}^2 + \tilde{B}^3 + \ldots$ and we claim that this sequence converges elementwise to the inverse of B. We want to use Lemma 4.1 to bound the terms of the series. Note that, if $i \neq j$, then $|B_{ij}| = |\tilde{B}_{ij}|$. Then from Lemma 4.3 it follows that the largest value of $\prod_{m=1}^{l} |\tilde{B}_{q_m q_{m+1}}|$ (where the sequence $q_1, q_2, \ldots, q_{l+1}$ is defined as in Lemma 4.3) is unique if it is nonzero, and its value equals $b(P_{ji})$ for the shortest j,i-path P_{ji} if such

[4] Again we count the last vertex in P_1^1 and P_2^1.

a path exists. It is 0 otherwise. Moreover, since (S,T) is nice, every cycle in $G_{S,T}$ has an even number as a highest priority. Therefore if $q_1, q_2, \ldots, q_{l+1}$ corresponds to any nontrivial i,i-walk P, we have $\prod_{m=1}^{l} |\tilde{B}_{q_m q_{m+1}}| \leq \frac{1}{b}$, since the term on the left is given by $b^{-\sum_{v\in P}(-3)^{p(v)}}$ and $-\sum_{v\in P}(-3)^{p(v)} < 0$. So now we have an upper bound of $\frac{1}{b}$ on the values of $\prod_{m=1}^{l} |\tilde{B}_{q_m q_{m+1}}| \leq \frac{1}{b}$ corresponding to cycles and an upper bound of $b(P_{ij})$ or 0 on the values of $\prod_{m=1}^{l} |\tilde{B}_{q_m q_{m+1}}| \leq \frac{1}{b}$ corresponding to paths. Then Lemma 4.1 tells us that

$$|\tilde{B}^k_{ij}| \leq n^{k-1} \frac{1}{b^{\lfloor \frac{k}{n} \rfloor}} M_{ij} \tag{7}$$

where $M_{ij} = b(P_{ji})$ if a j,i-path exists and $M_{ij} = 0$ otherwise. Since $b = n^{3n}$, we see that the term on the right of (7) decreases exponentially. Also, we can drop the absolute value notation since $\tilde{B}$ is a nonnegative matrix. Therefore the infinite sum

$$\sum_{k=0}^{\infty} \tilde{B}^k_{ij}$$

converges for any $i,j \in [n]$. We conclude that the series $I + \tilde{B} + \tilde{B}^2 + \tilde{B}^3 + \ldots$ converges elementwise. From that, we see that

$$B(I+\tilde{B}+\tilde{B}^2+\ldots) = (I-\tilde{B})(I+\tilde{B}+\tilde{B}^2+\ldots) = (I+\tilde{B}+\tilde{B}^2)-(\tilde{B}+\tilde{B}^2+\tilde{B}^3) = I \tag{8}$$

hence B is invertible and $I + \tilde{B} + \tilde{B}^2 + \tilde{B}^3 + \ldots$ is the inverse of B. Then, as we know all elements of $\tilde{B}$ are nonnegative, we know B^{-1} has only nonnegative elements, and the corresponding basic solution is given by $B^{-1}\mathbf{1}$, which is nonnegative hence feasible for (2). We conclude that B is indeed nice, as it is invertible and it yields a basic feasible solution. Now we estimate the elements of B^{-1}.

We can immediately see that, if there is no nontrivial i,i-walk in $G_{S,T}$, then $\prod_{m=1}^{l} \tilde{B}_{q_m q_{m+1}} = 0$ for any sequence $i = q_1, q_2, \ldots, q_{l+1} = i$. Therefore $\tilde{B}^l_{ii} = 0$ for all $l \in \mathbb{N}$, since

$$\tilde{B}^l_{ii} = \sum_{q_1=i, q_2, \ldots, q_{l+1}=i} \prod_{m=1}^{l} \tilde{B}_{q_m q_{m+1}} = 0$$

This implies $B^{-1}_{ii} = 1$ by equation (11), so we proved the second statement of the lemma. Equation (7) also tells us that for any i,j, we have

$$\begin{aligned}
\sum_{k'=n}^{\infty} \tilde{B}^{k'}_{i,j} &= \sum_{k=1}^{\infty} \sum_{k'=1}^{n} \tilde{B}^{kn+k'-1}_{i,j} \\
&\leq \sum_{k=1}^{\infty} \sum_{k'=1}^{n} n^{kn+k'-2} \frac{1}{b^k} M_{ij} \\
&\leq \sum_{k=1}^{\infty} \sum_{k'=1}^{n} n^{kn+n-2} \frac{1}{n^{3nk}} M_{ij} \\
&= \sum_{k=1}^{\infty} n^{-2kn+n-1} M_{ij} \leq n^{-n} M_{ij}
\end{aligned} \tag{9}$$

Furthermore, Lemma 4.3 tells us that the largest value of $\prod_{m=1}^{l} |\tilde{B}_{q_m q_{m+1}}|$ over all sequences that represent i,j-paths is unique if it is nonzero. Since these values are powers

of B, that implies that there is one value of $\prod_{m=1}^{l} |B_{q_m q_{m+1}}|$ that equals M_{ij}, and that all other values are at most $\frac{1}{b} M_{ij}$. Then we get

$$\sum_{k'=0}^{n-1} \tilde{B}_{i,j}^{k'} = \sum_{k'=0}^{n-1} \sum_{\substack{q_1,q_2,\ldots,q_{k'+1} \in [n] \\ q_1=i,\, q_{k'+1}=j}} \prod_{m=1}^{k'} \tilde{B}_{q_m q_{m+1}} \quad \leq$$

$$M_{ij} + (1 + n + n^2 + \ldots + n^{n-1} - 1)\frac{1}{b} M_{ij} \leq M_{ij} + n^{-2n} M_{ij} \tag{10}$$

We conclude from, (8), (9) and (10) that

$$M_{ij} \leq B_{ij}^{-1} = \sum_{k'=0}^{\infty} \tilde{B}_{ij}^{k'} \leq M_{ij} + \frac{2}{n^n} M_{ij} \tag{11}$$

Now, we see that $M_{ii} = 1$ ($M_{ii} = b(P_{ii}) = b(\emptyset)$) for any i, and that $M_{ij} = 0$ if $i \neq j$ and there is no path from j to i in $G_{S,T}$. Also, $M_{ij} = b(P_{ji}) = b^{-\sum_{v \in P_{ji}} (-3)^{p(v)}}$ for the shortest path P_{ji} if it exists and $i \neq j$. Using these values of M_{ij} in (11) gives us statements 1,3 and 4 of the lemma. This completes the proof. ■

4.6 Main results

Now that we estimated B^{-1}, we are able to calculate what happens in the simplex algorithm. This allows us to prove the main result of this section, that the simplex algorithm on (2) and the subgame iteration algorithm are, in some sense, the same algorithm. We also show that both algorithms solve parity games.

Theorem 4.6. *The following statements hold for the subgame iteration algorithm on a parity game on $G = (V, E)$ and for the simplex algorithm on the LP (2):*

1. *All (S, T)-pairs encountered in the subgame iteration are nice and all bases encountered by the simplex algorithm are S, T-nice for some S, T. This holds for any subgame iteration improvement rule and for any simplex pivot rule.*

2. *For a nice S, T and corresponding nice basis, the reduced cost of any variable z_v is always nonnegative.*

3. *For a nice S, T and corresponding nice basis, the reduced cost of $s_{(v,v')}$ is negative if and only if adding (v, v') to S (and possibly removing another edge from S) is an improving move in the subgame iteration algorithm that is at (S, T).*

4. *For a nice S, T and corresponding nice basis, the reduced cost of t_v is negative if and only if adding v to T is an improving move in the subgame iteration algorithm.*

Proof: We prove this by induction on the number of iterations done by the subgame improvement algorithm and the simplex algorithm. As an induction basis, we look at the initial solutions. For the subgame iteration algorithm, the initial solution is $S = T = \emptyset$. Clearly $S \subseteq E$ and every node of V_0 has at most one outgoing edge of S, and also $T \subseteq V_1$. There are no cycles in $G_{S,T}$ with odd highest priority since there are no cycles, and the only possible paths are trivial paths, which have value of $\emptyset$. Hence (S, T) is nice. The initial solution of the simplex algorithm, which has $\boldsymbol{z}$ as an initial basis, is feasible, since $\boldsymbol{z} = \mathbf{1}$, and $B = I$ is invertible, so it is a nice S, T-basis.

As an induction hypothesis, we assume we have some nice (S,T) and a related nice S,T-basis that are encountered in the algorithms. We now need to prove two things as an induction step: that the improving moves of the subgame iteration and the simplex algorithm are related (statements 2,3,4 of the theorem), and that the sets (S,T) and the related S,T-basis are nice in the next iteration for any pivoting/improvement choice. We prove first statements 2,3,4 from the theorem for the current iteration, and then statement 1 for the next iteration. To ease notation, we denote P_{ij} for the unique $\preceq$ shortest path from i to j (where we still do not count the j at the end) in the graph $G_{S,T}$. If there is no i,j-path, then we say $P_{ij} = \{2n+2\}$. We let $b(P_{ij}) = b^{-\sum_{v \in P_{ij}} (-3)^{p(v)}}$ if an i,j-path exists, and $b(P_{ij}) = 0$ otherwise. Now we prove the different statements for the current iteration:

2 Recall that the reduced cost of variable i with basis B is given by $c_i - c_B^T B^{-1} A_i$, where c_B contains the costs related to the basic variables, and A_i is the i-column of A. For variable z_i, we have $c_{z_i} = 1$ and $A_{z_i} = e_i$, where e_i is the i-unit vector in $\mathbb{R}^n$. Note that then $B^{-1}A_{z_i}$ is just the i-column of B^{-1}. Note that if z_i is in the basis, then $B^{-1}A_{z_i} = e_i$ and $c_{z_i} - c_B^T e_i = c_{z_i} - c_{z_i} = 0$. If z_i is not in the basis, then recall that $B^{-1}_{ji} = b(P_{ij})(1+\epsilon_{ij})$ for some $0 \leq \epsilon_{ij} \leq n^{-n}$ by Lemma 4.5. Also note that the j-component of c_B is 0 if vertex j is in T or has an outgoing edge in S, and it is equal to 1 otherwise. This yields

$$c_{z_i} - c_B B^{-1} A_{z_i} = 1 - \sum_{j \in V : (c_B)_j = 1} B^{-1}_{ji} = 1 - \sum_{j \in V : \Psi_{S,T}(j) = \emptyset} b(P_{ij})(1+\epsilon_{ij}) \quad (12)$$

In the above equation, we sum over the j without outgoing edges in $G_{S,T}$, so the j with $\Psi_{S,T}(j) = \emptyset$. Now since z_i was not in the basis, we know $\Psi_{S,T}(i) \neq \emptyset$, hence $\Psi_{S,T}(i) \succ \emptyset$ as (S,T) is nice. Therefore, $P_{ij} \succ \emptyset$ for any j without outgoing edges in $G_{S,T}$, and therefore $b(P_{ij}) \leq \frac{1}{b}$ for the terms on the right in (12). Hence the reduced cost of variable z_i is positive if z_i is not in the basis, as $\frac{1}{b}$ is very small. We conclude that the reduced cost $c_{z_i} - c_B B^{-1} A_{z_i}$ is always nonnegative. This completes the induction step for part 2 of the theorem.

3 Now we look at a non-basic variable $s_{(v,v')}$. Again, we look at the reduced cost $c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}}$. We have $c_{s_{(v,v')}} = 0$, and again the i-element of c_B is 1 if $\Psi_{S,T}(i) = \emptyset$ and 0 otherwise. Moreover, we have

$$\left(A_{s_{(v,v')}}\right)_i = \begin{cases} 1 & i = v \\ -b^{-(-3)^{p(v)}} & i = v' \\ 0 & \text{else} \end{cases}$$

Putting these together, we get

$$c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} = \sum_{i \in V : \Psi_{S,T}(i) = \emptyset} \left(b^{-(-3)^{p(v)}} \cdot B^{-1}_{iv'} - B^{-1}_{iv} \right)$$

$$= \sum_{i \in V : \Psi_{S,T}(i) = \emptyset} b(P_{v'i} \uplus \{v\})(1+\epsilon_{v'i}) - \sum_{i \in V : \Psi_{S,T}(i) = \emptyset} b(P_{vi})(1+\epsilon_{vi}) \quad (13)$$

We distinguish two cases:

- Suppose adding (v, v') to S (and possibly removing one edge) is an improving move in the subgame iteration. Note that, whether v had any outgoing edge in S before or not, we must have that $\Psi_{S,T}(v') \uplus \{p(v)\} \succ \Psi_{S,T}(v)$. This implies that $P_{v'i} \uplus \{p(v)\} \succ \Psi_{S,T}(v)$ for all i with $\Psi_{S,T}(i) = \emptyset$. Say the shortest path determining $\Psi_{S,T}(v)$ is P_{vj}, then we have $b(P_{v'i} \uplus \{v\}) \leq \frac{1}{b} b(P_{vj})$ for all i with $\Psi_{S,T}(i) = \emptyset$. But if we look at the right side of (13), then we see that the elements in the first summation are therefore much smaller than the largest element in the second summation, hence we get $c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} < 0$. More precisely, we get

$$c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} = -b(\Psi_{S,T}(v))(k + \epsilon) \tag{14}$$

  where k is an integer with $1 \leq k \leq n$, and $|\epsilon| \leq \frac{2n}{n^n} + \frac{n}{b} + \frac{2n}{bn^n}$. This is because at least one and at most n elements of the summation equal $b(\Psi_{S,T})(1 + \epsilon_{vi})$, and the other terms are at most $\frac{1}{b}$ times that (we use equation (14) later).
- Suppose on the other hand that adding (v, v') to S is not an improving move, which is equivalent to saying $\Psi_{S,T}(v') \uplus \{v\} \preceq \Psi_{S,T}(v)$ or $\Psi_{S,T}(v) = \{2n + 2\}$. We have three cases:
  - If $\Psi_{S,T}(v') \uplus \{v\} \prec \Psi_{S,T}(v)$, then by the same argument as before we get that $c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} > 0$, since the rightmost sum of (13) is much smaller in absolute value than the left sum.
  - If $\Psi_{S,T}(v) = \{2n+2\}$, then $b(P_{vi}) = 0$ for all i with $\Psi_{S,T}(i) = \emptyset$. Therefore the rightmost sum of (13) equals 0, hence the reduced cost is nonnegative.
  - If $\Psi_{S,T}(v') \uplus \{v\} = \Psi_{S,T}(v)$, then clearly $\Psi_{S,T}(v) \neq \{2n + 2\}$ and $\Psi_{S,T}(v) \neq \emptyset$. If v'' is the current successor of v in $G_{S,T}$, then any (nontrivial) path from v passes through v''. Therefore $\Psi_{S,T}(v'') \uplus \{v\} = \Psi_{S,T}(v)$. This implies $\Psi_{S,T}(v'') = \Psi_{S,T}(v')$, hence by Lemma 4.4, $\Psi_{S,T}(v') = \Psi_{S,T}(v'') = \emptyset$ (note that $\Psi_{S,T}(v'') \neq \{2n + 2\}$ as that would imply $\Psi_{S,T}(v) = \{2n + 2\}$). Then, we get by Lemma 4.5 that $B^{-1}_{v'v'} = 1$ and $B^{-1}_{iv'} = 0$ for $i \neq v'$. Also, as there are no paths from v to any other i in $G_{S,T}$ except v and v'', we get $B^{-1}_{iv} = 0$ for these i. Moreover, there is only one walk possible from v to v'', namely the path over the edge (v, v'') hence we can use $B^{-1} = I + \tilde{B} + \tilde{B}^2 + \ldots$ to find that $B^{-1}_{v''v} = b^{-(-3)^{p(v)}}$. Filling all these into (13) yields

$$\begin{aligned} c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} &= \sum_{i \in V : \Psi_{S,T}(i) = \emptyset} \left(b^{-(-3)^{p(v)}} \cdot B^{-1}_{iv'} - B^{-1}_{iv} \right) \\ &= b^{-(-3)^{p(v)}} \cdot B^{-1}_{v'v'} - B^{-1}_{v''v} = b^{-(-3)^{p(v)}} \cdot 1 - b^{-(-3)^{p(v)}} \quad = \quad 0 \end{aligned}$$

  Hence we conclude that adding (v, v') to S is an improving move if and only if the reduced cost of $s_{(v,v')}$ is negative. This completes the induction step for part 3 of the theorem.

4 We look at the reduced cost of non-basic variable t_v, with $v \in V_1$. This is given by $c_{t_v} - c_B B^{-1} A_{t_v}$. We have $c_{t_v} = 0$, and the i-th element of c_B equal to 1 if z_i is in the basis and equal to 0 if z_i is not in the basis. We have

$$(A_{t_v})_i = \begin{cases} 1 & i = v \\ -b^{-(-3)^{p(v)}} & (v, i) \in E \\ 0 & \text{else} \end{cases}$$

Moreover, since v is not in T, v has no outgoing edges in $G_{S,T}$, hence by Lemma 4.5, we get $B^{-1}_{vv} = 1$ and $B^{-1}_{iv} = 0$ for all $i \neq v$. Then we get, similar to the other cases, that

$$c_{t_v} - c_B B^{-1} A_{t_v} = \sum_{i \in V : \Psi_{S,T}(i) = \emptyset} \left(\sum_{v' \in V : (v,v') \in E} b^{-(-3)^{p(v)}} \cdot B^{-1}_{iv'} - B^{-1}_{iv} \right)$$

$$\overset{\text{(Lemma 4.5)}}{=} \sum_{i \in V : \Psi_{S,T}(i) = \emptyset} \left(\sum_{v' \in V : (v,v') \in E} b(P_{v'i} \uplus \{v\})(1 + \epsilon_{v'i}) \right) - 1 \qquad (15)$$

We have two cases:

- If it is an improving move to add v to T, then this means $\Psi_{S,T}(v') \uplus \{v\} \succ \emptyset$ for all successors of v in G, which implies $b(P_{v'i} \uplus \{v\}) \leq \frac{1}{b}$ for all successors v' of v in G and all i with $\Psi_{S,T}(i) = \emptyset$. This clearly implies that the expression on the righthand side of (15) is negative, so the reduced cost is negative.
- If it is not an improving move to add v to T, this means that there is at least one successor of v in G, say v'', for which $\Psi_{S,T}(v'') \uplus \{v\} \preceq \emptyset$. But this implies $b(P_{v''i} \uplus \{v\}) \geq 1$, and hence the righthand side of (15) is nonnegative, so the reduced cost is nonnegative.

We conclude that the reduced cost of t_v is negative if and only if adding v to T is an improving move in the subgame iteration algorithm. This completes the induction step for statement 4 of the theorem.

1 Now we show that the sets and related basis in the next iteration are nice if the current sets (S, T) and related basis are nice. First, we show that the sets (S', T'), resulting from the improving move in subgame iteration, are nice. We do so by contradiction, so suppose that (S', T') is not nice. Clearly the first two conditions of nice sets (Definition 4.2) are satisfied. We use a simplified notation: by $A \uplus B$ for A a (multi)set of priorities and B a set of nodes, we mean $A \uplus B := A \uplus \{p(v) | v \in B\}$. There are three cases left:

- Suppose adding an edge (v, v') to S (and possibly removing another) creates a cycle with an odd number as its highest priority. Then this cycle, say C, must contain v, as (S, T) is nice. Note that $C \backslash \{v\}$ is a path from v' to v in $G_{S,T}$. Then we can put $C \backslash \{v\}$ before any path from v to create a path from v'. In particular, for any path P_{vj} in $G_{S,T}$ where j is a vertex with $\Psi_{S,T}(j) = \emptyset$, we also know that $(C \backslash \{v\}, P_{vj})$ is a path from v' to j in $G_{S,T}$. This implies that $\Psi_{S,T}(v') \preceq \Psi_{S,T}(v) \uplus C \backslash \{v\}$, which yields

  $$\Psi_{S,T}(v) \prec \Psi_{S,T}(v') \uplus \{v\} \preceq \Psi_{S,T}(v) \uplus C$$

  where the first inequality is because adding (v', v) is improving. But that is not possible as we assumed C has odd highest priority, and therefore $\Psi_{S,T}(v) \succ \Psi_{S,T}(v) \uplus C$.
- Suppose adding a vertex v to T creates a cycle C with an odd highest priority. Then this cycle contains v, and say v' is the successor of v in the cycle. Then note that $C \backslash \{v\}$ is a path from v' to v in $G_{S,T}$, and also v has no outgoing edges in subgraph $G_{S,T}$. Therefore $\Psi_{S,T}(v') \preceq C \backslash \{v\}$. This yields, together with

$\Psi_{S,T}(v') \uplus \{v\} \succ \emptyset$ (which we get because we are looking at an improving move) the following:

$$\emptyset \prec \Psi_{S,T}(v') \uplus \{v\} \preceq (C \backslash \{v\}) \uplus \{v\} = C$$

and this is a contradiction as $C \prec \emptyset$.

- Suppose adding (v, v') to S or adding v to T creates a path towards a node with $\Psi_{S,T}$ less than or equal to $\emptyset$. This is clearly not possible, since the value of the shortest path from v to a node without outgoing edges is increased by the improving move (as we require for example $\Psi_{S,T}(v') \uplus \{p(v)\} \succ \Psi_{S,T}(v)$ for adding edge (v, v')).

So we conclude that (S', T') is also nice for any improving move.

Next, we need to prove that the simplex basis resulting from pivoting on $s_{(v,v')}$ or t_v results in a nice corresponding S', T'-basis. We already know that the variable that enters the basis corresponds to an improving move in subgame iteration, and that the sets (S', T') resulting from the improving move are nice. We then know by Lemma 4.5 that there exists a nice basis B' corresponding to (S', T'). We argue that the simplex algorithm removes the 'right' variable from B to arrive at basis B'. Suppose w. l. o. g. we are adding variable $s_{(v,v')}$ to the basis B. We slowly increase the value of $s_{(v,v')}$ and see which of the current basic variables becomes 0 first. This variable is removed by the simplex algorithm. Note that the basic feasible solution corresponding to B' has value equal to $(B')^{-1}\mathbf{1} \geq \mathbf{1} > 0$. Therefore we see that, when we increase the value of the variable that enters the basis, the first basic variable that becomes 0 can only be the one that has to leave the basis to get to B'. (i.e. z_v if S had no outgoing edges from v or we add a vertex to T, or otherwise $s_{(v,v'')}$ when $(v, v'') \in S$). So the basis resulting from the pivoting step is a nice S', T'-basis. This completes the proof of the induction step for statement 1 of the theorem.

Now we proved that statements 2,3,4 of the theorem hold for the current iteration if we have a nice basis and corresponding nice basis, and that statement 1 holds for the next iteration. Now we completed the induction step, hence this completes the proof of the theorem. □

Corollary 4.7. *The statements of Theorem 4.6 also hold if we start subgame iteration with an arbitrary pair of nice sets (S, T), and the simplex algorithm with the basis related to S, T.*

Proof: According to Lemma 4.5, there always exists a corresponding nice basis for S, T. This provides an induction basis. The induction step of the induction proof would be exactly the same as for Theorem 4.6. ■

Now we have a framework for showing that a certain pivot rule on the simplex algorithm has exponential running time: we can show that the improvement rule that makes the same choices has exponential running time on a class of parity games. Finally, although not needed in the context of lower bounds, we show that both these algorithms actually solve the parity game.

Theorem 4.8. *The subgame iteration algorithm terminates. The resulting set S - together with an arbitrary choice for the nodes in V_0 that do not yet have an outgoing edge - form a strategy that is winning for the winning set of player 0 (hence, an optimal basis for (2) forms an optimal player 0 strategy in a similar manner). The winning set is given by the vertices with $\Psi_{S,T}(v) = \{2n + 2\}$.*

Proof: Note that an optimum exists for (2), since the objective value is bounded from below by 0 and there are feasible solutions. From Theorem 4.6 it follows immediately that both algorithms terminate, since the simplex algorithm terminates.

Now we show that S yields an optimal strategy by contradiction. Let W_0 be the winning set of player 0. Suppose that the algorithms terminate, resulting in the sets S and T, and that S cannot be extended to a strategy that wins for player 0 on W_0.

Furthermore, let σ be a strategy that wins for player 0 on W_0, and let

$$S' = (S \cap \{(v, w) : v \in V_0 \backslash W_0\}) \cup \{(v, \sigma(v)) : v \in V_0 \cap W_0\}$$

So S' is the same as S, except in W_0 its edges are determined by σ. Moreover, let $T' = T \cup W_0$. First of all, note that in $G_{S',T'}$, there are no edges moving out of W_0. Otherwise, this would imply that player 0 moves out of the winning set for strategy σ, or that player 1 can move out of W_0. The first contradicts the assumption that σ is winning in W_0 and the second that W_0 is the winning set.

Now the goal is to show that the value of the objective function ($\mathbf{1}^T \boldsymbol{z}$) is lower for the basic feasible solution related to S', T' than for S, T. This would be a contradiction with the assumption that the basic feasible solution related to S, T was optimal. Therefore it would prove that the assumption was wrong and imply that S, T yield a winning strategy for W_0. It is clear that (S', T') is also nice, since

- There are no cycles with odd highest number in W_0 in the graph $G_{S',T'}$ (as σ is winning there)
- There is no path from a node in W_0 to a node without outgoing edges in $G_{S',T'}$, since there are no such nodes in W_0 and no edges leaving W_0.
- The rest of the graph is the same as $G_{S,T}$, which is nice.

.

Let i be a vertex without outgoing edges in $V \backslash W_0$. Let B' be the basis related to (S', T') Recall that

$$\tilde{B}^k_{ij} = \sum_{q_1=j, q_2, \ldots, q_{l+1}=i} \prod_{m=1}^{k} \tilde{B}_{q_m q_{m+1}} = \sum_{r \in R^k_{ji}} b(r)$$

where R^k_{ji} is the set of k-long walks in $G_{S,T}$ from j to i. Supppose i has no outgoing edges in $G_{S',T'}$. Now if we look at $G_{S',T'}$, then we see that any walk from j to i can only contain nodes in $V \backslash W_0$, as i cannot lie in W_0 and W_0 has no outgoing edges. Hence any walk from j to i in $G_{S',T'}$ must also exist in $G_{S,T}$. From this we conclude $\tilde{B}^k_{ij} \geq (\tilde{B}')^k_{ij}$ for all i without outgoing edges in $G_{S',T'}$, and for all $k \geq 0$ and $j \in V$. Therefore, as $B^{-1} = I + \tilde{B} + \tilde{B}^2 + \ldots$, we have

$$B^{-1}_{ij} \geq (B')^{-1}_{ij}$$

for all i without outgoing edges and $j \in V$. Moreover, since S, T do no result in a strategy winning on W_0, there is at least one vertex j in W_0 that has a path towards a node without outgoing edges i in $G_{S,T}$ (since otherwise it would only be able to reach cycles with an even highest priority, because (S, T) is nice). Hence $B^{-1}_{ij} > (B')^{-1}_{ij}$ for these i, j. Moreover,

note that any node without outgoing edges in $G_{S',T'}$ has also no outgoing edges in $G_{S,T}$. Putting all these together, we find

$$\begin{aligned}\mathbf{1}^T \boldsymbol{z}_{B'} = \sum_{\substack{i \in V \\ \text{no outgoing edge in } G_{S',T'}}} z_i &= \sum_{\substack{i \in V \\ \text{no outgoing edge in } G_{S',T'}}} \sum_{j \in V} (B')^{-1}_{ij} \\ &< \sum_{\substack{i \in V \\ \text{no outgoing edge in } G_{S',T'}}} \sum_{j \in V} B^{-1}_{ij} \\ &\leq \sum_{\substack{i \in V \\ \text{no outgoing edge in } G_{S,T}}} \sum_{j \in V} B^{-1}_{ij} = \mathbf{1}^T \boldsymbol{z}_B \end{aligned} \tag{16}$$

where $\boldsymbol{z}_B, \boldsymbol{z}_{B'}$, respectively, are the variables $\boldsymbol{z}$ resulting from basis B and B'. Note that if z_i is in the basis, then its value is given by $(B^{-1}\mathbf{1})_i$. From (16) we see that indeed B' yields a better objective value, so we get a contradiction with our assumption that (S, T) was optimal. This concludes the proof that we get a winning strategy from S, T. With the same argument we can also see that for an optimal solution to the LP (2), all elements of $W_0 \cap V_1$ must be in T and all elements of $W_0 \cap V_0$ must have an outgoing edge in S. Otherwise, the second inequality of (16) would hold strictly as there is a vertex i in W_0 without outgoing edges (and $B^{-1}_{ii} = 1 > 0$). Hence we conclude that for an optimal solution, from any vertex $v \in W_0$, we cannot reach a vertex without outgoing edges in $G_{S,T}$ if the algorithms terminate. Hence $\Psi_{S,T}(v) = \{2n + 2\}$ for $v \in W_0$. Clearly there is no node outside W_0 with $\Psi_{S,T}$ equal to $\{2n + 2\}$, as that would imply that one cannot reach any other cycle with odd highest priority with the induced strategy, which would imply player 0 is winning. So the winning set of player 0 is exactly the set of nodes with $\Psi_{S,T}$ equal to $\{2n + 2\}$ when the algorithms have terminated. This completes the proof of the theorem. □

5 Examples: alternative lower bound proofs for classical pivot rules.

In this section, we make the relation between subgame iteration and the simplex algorithm more concrete. Recall that the simplex algorithm requires a pivot rule, which chooses which improving variable enters the basis. For four simplex pivot rules, we use this relation to construct a lower bound for their complexity, with a relatively short proof. We construct alternative lower bounds for the following pivot rules[5]:

- Least-index rule (Bland's rule) [5]. Of the possible x_i to add to the basis, choose the one with the lowest i.
- Steepest descent rule (Dantzig's largest coefficient rule) [6]. Add the x_i to the basis that has the lowest (most negative) reduced cost $c_i - c_B^T B^{-1} A_i$.
- Steepest edge rule [6]. Add the x_i with the lowest value of $\frac{c_i - c_B^T B^{-1} A_i}{||B^{-1} A_i||}$.
- Largest improvement rule [19]. Add the x_i to the basis that will result in the largest decrease in the objective function $c^T \boldsymbol{x}$.

It has already been shown for these rules that, in the worst case, they have exponential running time in the input size of the LP. This was done for the first rule by Avis and Chvátal [3], for the second one by Klee et al. [19], for the third by Goldfarb and Sit [13], and for the last rule by Jeroslow [16]. All of these construction use the Klee-Minty cube - a perturbed n-dimensional hypercube - as a starting point. In this section, we show alternative constructions that prove exponential running time for these rules - at least in the number of variables and equations of the LP. However, using the results from earlier sections, the alternative proofs here will be relatively short.

The structure of the lower bound proofs in this section will be the same for all four rules. First, we take an improvement rule for strategy iteration in parity games from Section 3. We already know that these have exponential worst-case running times. Next, we show that subgame iteration (with a certain improvement rule) behaves similar to strategy iteration for this specific improvement rule. This similarity is made more concrete in Lemma 5.1. So then we know that subgame iteration with some improvement rule has exponential worst-case running time (in terms of number of nodes and edges). Finally, we use the relation from Section 4 which says that subgame iteration and the simplex algorithm are 'the same.' We show that our improvement rule for subgame iteration and the simplex pivot rule make the same choices. This implies that the simplex algorithm has exponential worst-case running time (in terms of variables and equations). This is made more precise in Corollaries 5.2 and 5.3.

We now make the similarity between strategy iteration and subgame iteration more formal. We show that if in a sink parity game (see Definition 2.6) the valuations Ξ_σ are high enough for current strategy σ in discrete strategy iteration, then the strategy σ corresponds to a nice pair of sets (S_σ, T_σ) in subgame iteration (see Definition 4.2).

[5]These rules may need specified tiebreaks, but we ignore that, as in the proofs there will be no ties.

Lemma 5.1. *Suppose we have a sink parity game G, and a player 0 strategy σ, such that $\Xi_\sigma(v) \trianglerighteq (1, \emptyset, 0)$ for all $v \in V$. Let $S_\sigma = \{(v, \sigma(v)) : v \in V_0 \backslash \{x\}\}$ and let $T_\sigma = V_1$. Then the following hold:*

1. *$\Xi_\sigma(v) = (1, \Psi_{S_\sigma,T_\sigma}, |\Psi_{S_\sigma,T_\sigma}|)$ for any $v \in V$.*
2. *Switching edge e_1 for e_2 in strategy iteration with strategy σ is improving if and only if replacing edge $e_1 \in S_\sigma$ by edge e_2 is improving in subgame iteration with S_σ, T_σ.*

Proof: Because $\Xi_\sigma(v) \trianglerighteq (1, \emptyset, 0)$, every path P from v to x has $P \succeq \emptyset$. Because we have a sink parity game and because of our starting strategy, player 1 can only win by going to x if player 0 plays strategy σ. Hence there are also no cycles with odd highest number in subgraph G_{S_σ,T_σ}. For these two reasons the pair of sets (S_σ, T_σ) is nice. Hence the subgame iteration valuation Ψ_{S_σ,T_σ} is defined. Now it is clear that the second component of Ξ_σ equals the $\preceq$-shortest path from v to x. Also, valuation $\Psi_{S_\sigma,T_\sigma}(v)$ is defined as the $\preceq$-shortest path towards a node without outgoing edges in G_{S_σ,T_σ}, and the only node without outgoing edges is x. So the second component of Ξ_σ is equal to Ψ_{S_σ,T_σ}. As a result, since all priorities in a sink parity game are larger than 1, this implies the third component of Ξ_σ equals $|\Psi_{S_\sigma,T_\sigma}|$. This proves the first statement.

Because of Lemma 4.4, the values of Ψ in the graph are unique, as there is only one node with Ψ equal to $\emptyset$ and no nodes with Ψ equal to $\{2n+2\}$. Therefore, the third component of Ξ_σ is irrelevant when comparing valuations Ξ_σ with each other. Therefore, obviously, $\Xi_\sigma(v) \triangleright \Xi_\sigma(w)$ if and only if $\Psi_{S_\sigma,T_\sigma}(v) \succ \Psi_{S_\sigma,T_\sigma}(w)$. Note that exchanging edge (v', w) for (v', v) in strategy iteration is an improving move if $\Xi_\sigma(v) \triangleright \Xi_\sigma(w)$. Also, in subgame iteration, the only possible improving move is exchanging an edge in E_0 for another edge, as we cannot add any more edges to S_σ or nodes to T_σ. Exchanging edge (v', w) for (v', v) is improving if $\Xi_\sigma(v) \triangleright \Xi_\sigma(w)$. This implies that the improving moves of both algorithms are indeed the same. ■

Corollary 5.2. *Suppose we have a sink parity game on graph $G = (V, E)$, and initial strategy σ_0 with $\Xi_{\sigma_0}(v) \trianglerighteq (1, \emptyset, 0)$ for all $v \in V$. Suppose strategy iteration takes an exponential number of iterations for some improvement rule. Then if subgame iteration starts with $S_{\sigma_0}, T_{\sigma_0}$ (as defined in Lemma 5.1), then it also takes an exponential number of iterations, given that it uses an improvement rule that makes the same choices as the beforementioned one for strategy iteration.*

Proof: Recall from Lemma 2.4 that the valuation of any node can only increase with strategy iteration. Hence we can apply Lemma 5.1 to any strategy σ encountered with strategy iteration, hence both algorithms, since they make the same choices, encounter exactly the same strategies in σ and S_σ, so they take the same number of iterations.

Corollary 5.3. *Suppose we have the conditions of Corollary 5.2. Suppose additionally that the simplex algorithm on (2) starts with the nice basis corresponding to nice sets $(S_{\sigma_0}, T_{\sigma_0})$. Suppose also that a simplex pivot rule is used that makes the 'same' choices as the beforementioned improvement rule for subgame iteration and the improvement rule for strategy iteration. Then the simplex algorithm requires the same number of iterations as strategy iteration on the graph.*

Proof: Note that from Corollary 4.7 we know that there is a one-to-one correspondence between subgame iteration and the simpelx algorithm, and that this also holds if we start with the nice sets $(S_{\sigma_0}, T_{\sigma_0})$. It follows that the simplex algorithm needs the same number of iterations as subgame iteration, and we know by Corollary 5.2 that subgame iteration in turn needs the same number of iterations as strategy iteration. ■

5.1 Construction of linear programs from parity games

Now we are ready to make the four lower bound constructions, which we summarize in the following four theorems.

Theorem 5.4. *There is a family of linear programs such that the simplex algorithm with the least index pivot rule needs running time that is at least exponential in the number of variables and equations.*

Proof: Recall that the highest-priority improvement rule for strategy iteration prefers to switch edges coming from the node with the highest priority. In Theorem 3.4, we showed that strategy iteration requires a number of iterations exponential in the number of nodes and edges. Moreover, for the family of counterexamples used there, all the valuations are at least as $\preceq$-large as $(1, \emptyset, 0)$. This is because of the vertex y that has very high even priority. It follows by Corollary 5.2 that subgame iteration with the same choices - which means also preferring edges from nodes with a high priority - requires exponentially many iterations. Moreover, for an arbitrary LP derived from a parity game like in Section 4, we can order the variables corresponding to edges in E_0 (recall $E_0 = \{(v, w) \in E : v \in V_0\}$) by the priority of their starting node, from high to low. Then, the simplex algorithm makes the 'same' choices as the highest-priority rule in subgame iteration. It follows from Corollary 5.3 that the simplex algorithm with the least index pivot rule requires a number of iterations equal to that of strategy iteration, which is exponential in the number of nodes and edges in the original graph. From this follows what we wanted to prove. □

Theorem 5.5. *There is a family of linear programs such that the simplex algorithm with the steepest descent rule needs running time that is at least exponential in the number of variables and equations.*

Proof: First, consider the lowest-valuation improvement rule for strategy iteration on parity games. Recall that this rule prefers to switch edges from nodes that currently have the lowest valuation Ξ_σ. By Theorem 3.11, strategy iteration requires a number of iterations that is exponential in the number of nodes and edges with this improvement rule. Moreover, by Corollary 5.2, this also means that if we apply subgame iteration on the same parity games and it makes the same choices, then it also requires exponentially many iterations. From Lemma 5.1 we know for all $v \in V$ that $\Xi_\sigma(v) = (1, \Psi_{S_\sigma, T_\sigma}, k_v)$ for some $k_v \geq 0$. Subgame iteration makes the same choices as strategy iteration if we choose to switch in the node with the $\preceq$-lowest value of $\Psi_{S,T}$.

Now we show that the steepest descent rule makes the same choices as when we choose to switch on the lowest valued vertex in subgame iteration. That would imply by Corollary 5.3 that steepest descent needs exponentially many iterations. We know from (14) that the reduced cost of $s_{(v,v')}$ equals

$$c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} = -b(\Psi_{S,T}(v))(k + \epsilon)$$

where k is an integer with $1 \leq k \leq n$, and $|\epsilon| \leq \frac{2n}{n^n} + \frac{n}{b} + \frac{2n}{bn^n}$. The valuation for non-sink nodes cannot equal $\emptyset$ or $\{2n+2\}$ as every non-sink node has an outgoing edge and player 1 can always reach the sink node. Then because of Lemma 4.4 we know that the valuations of non-sink nodes are unique. Therefore $b(\Psi_{S,T}(v))$ differs by a factor of at least b for different v. Moreover, the variable with the lowest reduced cost is the one corresponding to an edge (v, v') with the highest possible $b(\Psi_{S,T}(v))$, hence with the $\preceq$-lowest value of $\Psi_{S,T}$. And we see that indeed the improving move from the node with the lowest corresponding valuation Ψ is preferred by the simplex algorithm. It follows that the simplex algorithm requires a number of iterations at least exponential in the number of nodes and edges of the original graph. This completes our proof. □

Theorem 5.6. *There is a family of linear programs such that the simplex algorithm with the largest improvement rule needs running time that is at least exponential in the number of variables and equations.*

Proof: Recall that the highest-lexicographic rule for strategy iteration is as follows:

> Let $VAL = Q \times \mathcal{P}(Q) \times \mathbb{Z}_{\geq 0}$ be the space of valuations Ξ_σ. For each possible improving switch e, let σ_e be the strategy that results from σ if improving move e is applied. We then associate a vector $\xi_e \in VAL^{|V|}$ to the strategy σ_e containing all values of $\sigma_e(v)$ for $v \in V$, and where ξ_e is $\trianglelefteq$-sorted from low to high. We then choose the improving switch e with the lexicographically highest value of ξ_e. In this case, we say $a < b$ lexicographically if for the least index i with $a_i \neq b_i$ we have $a_i < b_i$.

We know from Theorem 3.12 that strategy iteration with the highest-lexicographic improvement rule requires a number of iterations exponential in the number of nodes and edges. Subgame iteration makes the same choices as strategy iteration if we prefer the improving move there that will result in the lexicographically highest sorted vector of values $\Psi_{S,T}$ (similar to the highest-lexicographic rule). So we know by Corollary 5.2 that subgame iteration also has exponential running time for this family of parity games. Now our goal is to show that subgame iteration with this equivalent of the highest-lexicographic rule makes the same choices as the simplex algorithm with the highest improvement rule. But we only show that this holds for the class of games $(G_n)_{n\in\mathbb{N}}$ from Theorem 3.12.

Note that the only basic variable in the corresponding LP (2) that has nonzero cost is z_x. That is because z_v is not in the basis for any other node, since all nodes of V_1 are in T, and all other nodes of V_0 have an outgoing edge in S. Therefore the cost of a basic feasible solution corresponding to (S, T) is

$$c_B B^{-1}\mathbf{1} = \sum_{j\in V} B^{-1}_{xj}$$

Since the only node with $\Psi_{S,T}(v) = \emptyset$ is x, we get from Lemma 4.5 that $B^{-1}_{xx} = 1$ and that $B^{-1}_{xj} = b(\Psi_{S,T}(j))(1 + \epsilon_{jx})$ with $\epsilon_{jx} \leq \frac{2}{n^n}$. Therefore the cost of the basic feasible solution is given by $1+\sum_{j\in V} b(\Psi_{S,T}(j))(1+\epsilon_{jx})$. Note that, as $b(P)$ is larger for $\preceq$-smaller paths P, this cost (ignoring the 1) is dominated by the $b(P)$ with the smallest P. Now from Lemma 4.4 we know that the valuations Ψ of the non-sink nodes are unique, hence in particular the second lowest valuation ($\emptyset$ is the lowest) is unique. Moreover, if the second lowest valuation for sets (S_1, T_1) is $\preceq$-lower than that of sets (S_2, T_2), then clearly the objective value of the basis related to (S_1, T_1) is higher, since it has a term $b(\Psi_{S,T}(j))(1+\epsilon_{jx})$ that is much higher than these occurring in the sum that makes the cost of (S_2, T_2). So the largest improvement rule prefers to pivot the variable corresponding to the improving move that improves the second lowest valuation by the most.

We know that in the proof of Theorem 3.12, highest-lexicographic also prefers to switch such that it improves the second lowest valuation by the most. Hence strategy iteration, subgame iteration and the simplex algorithm make the same choices for this family of parity games. So Corollary 5.3 now implies that the simplex algorithm requires a number of iterations exponential in the number of edges and vertices of the original graph. This completes the proof. □

Theorem 5.7. *There is a family of linear programs such that the simplex algorithm with the steepest edge rule needs running time that is at least exponential in the number of variables and equations.*

Proof: Recall from Section 3.4 that the lowest-valuation-shortest-path rule is as follows:

> Suppose we have an improving move from v for strategy σ that switches from edge (v, v'') to (v, v'). Say the new strategy after this switch is σ' Let $paths(v, v')$ be the set of paths from v to any $w \in V$, where we require all the edges to be either in $E_1 = \{(v_1, v_2) | v_1 \in V_1\}$ or to be part of either σ or σ'. Recall that we do not count the end vertex of a path as being *in* the path. Then, we define $M_\sigma(v, v') = \min_{\preceq} paths(v, v')$, which will be the lowest-valued path from v to any other node. (recall from Section 2.4 that whether $B_1 \preceq B_2$ can be determined from $B_1 \triangle B_2$ or from whether $\sum_{p \in B_1}(-3)^p \leq \sum_{p \in B_2}(-3)^p$). The improvement rule prefers to switch in nodes with a low valuation and with a high value of $M(v, v')$. More precisely, our improvement rule prefers a switch in v to v' over a switch in w to w' if $\Xi_\sigma(v) \uplus M_\sigma(w, w') \trianglelefteq \Xi_\sigma(w) \uplus M_\sigma(v, v')$ (here we use simplified notation; with $(\lambda, \pi, n) \uplus B$ we mean $(\lambda, \pi \uplus B, n)$).

From Theorem 3.13 we know that the lowest-valuation-shortest-path improvement rule for strategy iteration requires running time exponential in the number of nodes and edges of graphs $(G_n)_{n \in \mathbb{N}}$. The equivalent rule for subgame iteration in sink parity games would be to prefer (v, v') over (w, w') if $\Psi_{S_\sigma, T_\sigma}(v) \uplus M_\sigma(w, w') \preceq \Psi_{S_\sigma, T_\sigma}(w) \uplus M_\sigma(v, v')$. Then, from Corollary 5.2, we know that subgame iteration with this equivalent improvement rule takes the same number of iterations on this class of parity games.

Next, we show that subgame iteration with this improvement rule makes the same choices as the simplex algorithm on the related LP with the steepest edge pivot rule, at least on the class of graphs from the proof of Theorem 3.13. Recall that this pivot rule chooses the index with the lowest (negative) value of $\frac{c_i - c_B^T B^{-1} A_i}{||B^{-1} A_i||}$. At any point in the simplex algorithm, the only entering variables i that can be improving are those $s_{(v,v')}$ corresponding to an improving edge (v, v'). Again, from (14) we know that

$$c_{s_{(v,v')}} - c_B B^{-1} A_{s_{(v,v')}} = -b(\Psi_{S,T}(v))(k + \epsilon)$$

where k is an integer with $1 \leq k \leq n$, and $|\epsilon| \leq \frac{2n}{n^n} + \frac{n}{b} + \frac{2n}{bn^n}$. This gives us an estimate for the numerator of $\frac{c_i - c_B^T B^{-1} A_i}{||B^{-1} A_i||}$.

Next, we want to estimate the denominator, so we look at the vector $B^{-1} A_{s_{(v,v')}}$. Recall that $A_{s_{(v,v')}}$ has a 1 in its v-position, $-b^{-(-3)^{p(v)}}$ in its v'-position and 0's everywhere else. Now we can use Lemma 4.5 to estimate:

$$\begin{aligned}(B^{-1} A_{s_{(v,v')}})_j &= B^{-1}_{jv} - b^{-(-3)^{p(v)}} B^{-1}_{jv'} \\ &\overset{\text{(Lemma 4.5)}}{=} b(P_{vj})(1 + \epsilon_{vj}) - b(P_{v'j} \uplus \{v\})(1 + \epsilon_{v'j})\end{aligned}$$

with $\epsilon_{ij} \leq \frac{2}{n^n}$. Now we know that the $\preceq$-shortest possible path from v towards any node in $G_{S,T} \cup \{(v, v')\}$ is $M_\sigma(v, v')$, and suppose that it goes from v to j_0. Assume additionally that every other path is strictly $\preceq$-longer than $M_\sigma(v, v')$. This is the case in the counterexample from Theorem 3.13. Therefore, $b(P) \leq \frac{1}{b} \cdot b(M_\sigma(v, v'))$ for any other path P starting from v. Thus we know that

$$\begin{aligned}(B^{-1} A_{s_{(v,v')}})_{j_0} &= b(M_\sigma(v, v'))(1 + \epsilon) \\ (B^{-1} A_{s_{(v,v')}})_j &< \tfrac{3}{b} b(M_\sigma(v, v')) \qquad \forall j \neq j_0\end{aligned}$$

where $|\epsilon| < \frac{3}{n^n}$. Now we can estimate the value of $||(B^{-1}A_{s_{(v,v')}})||$:

$$||(B^{-1}A_{s_{(v,v')}})|| = \sqrt{\sum_j (B^{-1}A_{s_{(v,v')}})_j^2} = b(M_\sigma(v,v'))(1+\epsilon)$$

where one could bound the small terms to show that $|\epsilon| < \frac{4}{n^n}$.

Now suppose we have two possible improving moves, either adding edge (v,v') or edge (w,w'). Then our pivot rule prefers edge (v,v') if

$$\frac{-b(\Psi_{S,T}(v))(k_1+\epsilon_1)}{b(M_\sigma(v,v'))(1+\epsilon_2)} = \frac{c_i - c_B^T B^{-1} A_{s_{(v,v')}}}{||B^{-1}A_{s_{(v,v')}}||} < \frac{c_i - c_B^T B^{-1} A_{s_{(w,w')}}}{||B^{-1}A_{s_{(w,w')}}||} = \frac{-b(\Psi_{S,T}(w))(k_2+\epsilon_3)}{b(M_\sigma(w,w'))(1+\epsilon_4)}$$

for the appropriate values of $k_1, k_2, \epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4$. This can be rewritten to

$$b(\Psi_{S,T}(v))(k_1+\epsilon_1) \cdot b(M_\sigma(w,w'))(1+\epsilon_4) > b(\Psi_{S,T}(w))(k_2+\epsilon_3) \cdot b(M_\sigma(v,v'))(1+\epsilon_2)$$

which holds (assuming there are no ties for path lengths, which is the case for our counterexamples), if and only if

$$\Psi_{S,T}(v) \uplus M_\sigma(w,w') \prec \Psi_{S,T}(w) \uplus M_\sigma(v,v')$$

and this happens if and only if the lowest-valuation-shortest-path rule prefers switching (v,v') over (w,w'). So we showed that lowest-valuation-shortest-path makes the same choices in subgame iteration on $(G_n)_{n\in\mathbb{N}}$ as the simplex algorithm with the steepest-edge pivot rule That, by Corollary 5.3, completes the proof. □

6 Worst-case complexity of symmetric strategy iteration

In this section we show that the worst-case complexity of symmetric strategy iteration in parity games, which is introduced by Schewe et al. [25], is at least exponential in the number of nodes and edges of the graph. Symmetric strategy iteration is an adaptation of the discrete strategy iteration algorithm that is symmetric for the two players of the parity game. For both players, it switches the edges that are both improving for themselves and also part of an optimal counterstrategy to the other player's strategy. The algorithm is of particular interest because it avoids the traps of existing lower bound constructions. It seems that there is no family of examples with superpolynomial running time for this algorithm known in literature so far.

The main idea of the construction is that we have n pairs of vertices that simulate a variant of a binary counter, that counts in so-called Gray code.

6.1 The algorithm

The symmetric strategy iteration algorithm maintains two strategies: a player 0 strategy σ and a player 1 strategy τ. We have a valuation $\Xi_\sigma : V \to Q \times \mathcal{P}(Q) \times \mathbb{Z}_{\geq 0}$ for player 0, like we have used throughout this whole thesis. This is based on the function $\Theta_{\sigma\tau}(v) = (\lambda(P), \pi(P), \#(P))$. Here $\lambda(P)$ is the highest priority of the cycle part of play P resulting from σ and τ, $\pi(P)$ is the set of large priorities on the path towards the cycle of P and $\#(P)$ is the length of the path towards the node with high priority in the cycle. Recall from Section 2.4 that the valuation Ξ is defined as

$$\Xi_\sigma(v) = \min_{\trianglelefteq}(\Theta_{\sigma\tau}(v) : \ \tau \text{ player 1 strategy})$$

Recall also that we denote an optimal counterstrategy (by player 1) to σ by $\bar{\sigma}$ such that $\Xi_\sigma = \Theta_{\sigma\bar{\sigma}}$. Analogously, we now also consider an optimal counterstrategy by player 0 against τ, which we call $\bar{\tau}$. The play resulting from τ and $\bar{\tau}$ then defines a valuation for player 1, the function $\Xi^\tau : V \to Q \times \mathcal{P}(Q) \times \mathbb{Z}_{\geq 0}$. This is defined similar to Ξ_σ:

$$\Xi^\tau(v) = \max_{\trianglelefteq}(\Theta_{\sigma\tau}(v) : \ \sigma \text{ player 0 strategy}) = \Theta_{\bar{\tau}\tau}(v)$$

Similar to improving moves for player 0, an edge (v, v') with $v \in V_1$ is an improving move for player 1 if $\Xi^\tau(v') \triangleleft \Xi^\tau(\tau(v))$. Let I_σ be the set of improving moves for player 0 for strategy σ, and I_τ be the set of improving moves for player 1 for strategy τ. Now we are ready to define the algorithm:

Algorithm 3 Symmetric strategy iteration

1: start with some pair of strategies σ,τ
2: find an optimal counterstrategy $\bar{\sigma}$ to σ
3: find an optimal counterstrategy $\bar{\tau}$ to τ
4: Let σ' be the strategy resulting from applying all improving moves
5: from $I_\sigma \cap \{(v, \bar{\tau}(v)) | v \in V_0\}$ to σ.
6: Let τ' be the strategy resulting from by applying all improving moves
7: from $I_\tau \cap \{(v, \bar{\sigma}(v)) | v \in V_0\}$ to τ
8: **if** $\sigma = \sigma'$ **and** $\tau = \tau'$ **then return** (σ, τ)
9: **else**
10: $\quad \sigma \leftarrow \sigma'$, $\tau \leftarrow \tau'$
11: $\quad$ **go to** 2
12: **end if**

Of course, the algorithm terminates, since any improving move improves the valuation of the respective player, and there is a finite number of improving moves possible. The following lemma implies that the algorithm only terminates when the resulting pair of strategies (σ, τ) is optimal for the players. The lemma is equivalent to Lemma 3.3 of [25].

Lemma 6.1. *Suppose σ is not an optimal player 0 strategy or τ is not an optimal player 1 strategy. Let $\bar{\sigma}$ and $\bar{\tau}$ be optimal counterstrategies for respectively σ and τ. Then either $I_\sigma \cap \{(v, \bar{\tau}(v)) | v \in V_0\}$ or $I_\tau \cap \{(v, \bar{\sigma}(v)) | v \in V_0\}$ is nonempty.*

Example

Suppose that strategy iteration currently has strategies (σ, τ) as in Figure 22. The first step of the symmetric strategy algorithm is then to compute The optimal counterstrategies $\bar{\sigma}$ and $\bar{\tau}$. These are also shown in the figure.

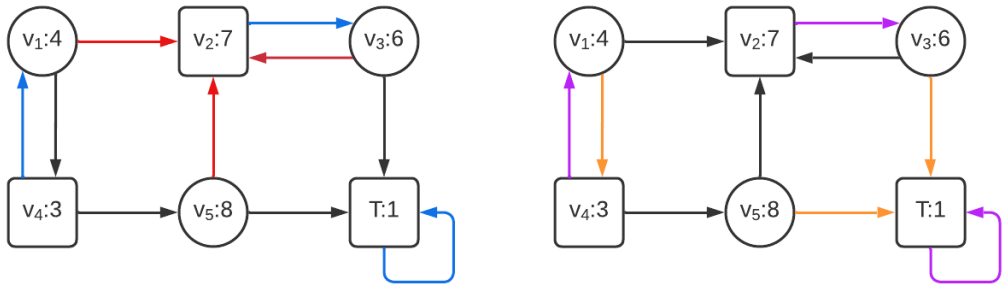


FIGURE 22: Example of strategies in symmetric strategy iteration. Left: strategy σ (red) and counterstrategy $\bar{\sigma}$ (blue). Right: Strategy τ (purple) and counterstrategy $\bar{\tau}$ (orange).

Next, we find the improving moves. For player 0, the edges $(v_3, \top)$ and $(v_5, \top)$ are improving since the first element of valuation Ξ_τ of $\top$ is 1 and of v_2 is 7. The edge (v_1, v_4) is improving because the path component of $\Xi_\sigma(v_4)$ contains an extra 3 and 4 compared to $\Xi_\sigma(v_2)$. As we see, all these edges are also part of the counterstrategy $\bar{\sigma}$. So $I_\sigma \cap \{(v, \bar{\tau}(v)) | v \in V_0\} = \{(v_1, v_4), (v_3, \top), (v_5, \top)\}$. For player 1, the edge (v_4, v_5) is an improving move, since the first component of $\Xi^\tau(v_5)$ is 1 and the first component of $\Xi^\tau(v_1)$ is 4. This is the only improving move for player 1. However, edge (v_4, v_5) is not used by $\bar{\tau}$. Therefore $I_\tau \cap \{(v, \bar{\sigma}(v)) | v \in V_0\} = \emptyset$. The algorithm switches edges $(v_1, v_4), (v_3, \top), (v_5, \top)$, yielding strategies (σ', τ') as shown in Figure 23. These are then the start of the next iteration.

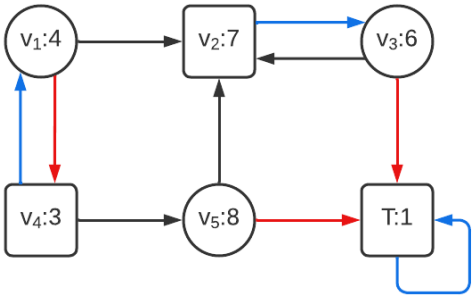


FIGURE 23: Strategies σ' (red), and τ' (blue) after making the switches in an iteration of symmetric strategy iteration.

6.2 Gray code

Now we introduce Gray code, see also [14]. Gray code is a form of binary representation of a number, where consecutive numbers differ in only one digit. For a number $B \in \{0, 1, \ldots, 2^n - 1\}$, we denote its Gray code by $g_n g_{n-1} g_{n-2} \ldots g_1$, where $g_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. Here g_n is the most significant bit and g_1 the least significant bit. The following facts are well-known about Gray code:

- If $B_n B_{n-1} \ldots B_1$ is the binary representation of B (so $B = \sum_{i=1}^{n} 2^{i-1} B_i$), then $g_n = B_n$ and $g_i = B_i + B_{i+1} (\mod 2)$ for $i = 1, 2, \ldots, n-1$. Moreover, we have $B_i = \sum_{j=i}^{n} g_j (\mod 2)$ for $i = 1, 2, \ldots, n$.
- If $g_n g_{n-1} \ldots g_1$ represents B and $g'_n g'_{n-1} \ldots g'_1$ represents $B' = B+1$, then $g_n g_{n-1} \ldots g_1$ and $g'_n g'_{n-1} \ldots g'_1$ differ in exactly one position, say $g_m = 1 - g'_m$. Moreover, m is the smallest index for which $\sum_{i=m}^{n} g_i$ is even.

6.3 Family of parity games

We show that the family of graphs $(G_n)_{n \in \mathbb{Z}_{\geq 2}}$ needs a number of iterations exponential in n with symmetric strategy iteration. Graph G_n is shown in Figure 24. It has the following nodes and edges:

Node	Player	Priority	Successors
a_1	Player 0	3	a_2, d_2
a_i, $i = 2, \ldots, n-1$	Player 0	$2i+1$	a_1, a_{i+1}, d_{i+1}
d_1	Player 1	4	a_2, d_2
d_i, $i = 2, \ldots, n-1$	Player 1	$2i+2$	d_1, a_{i+1}, d_{i+1}
a_{n+1}	Player 0	1	x
d_{n+1}	Player 1	2	d_{n+1}

This family of graphs is in principle the parity game version of the counterexample from Björklund et al. [4], with added edges back to a_1 and d_1. As an initial pair of strategies we have σ_0 and τ_0. We have $\sigma_0(a_i) = a_{i+1}$ for $i = 1, 2, \ldots, n-1$ and $\sigma_0(a_n) = a_{n+1}$, and $\tau_0(d_i) = d_{i+1}$ for $i = 1, 2, \ldots, n-1$ and $\tau_0(d_n) = d_{n+1}$.

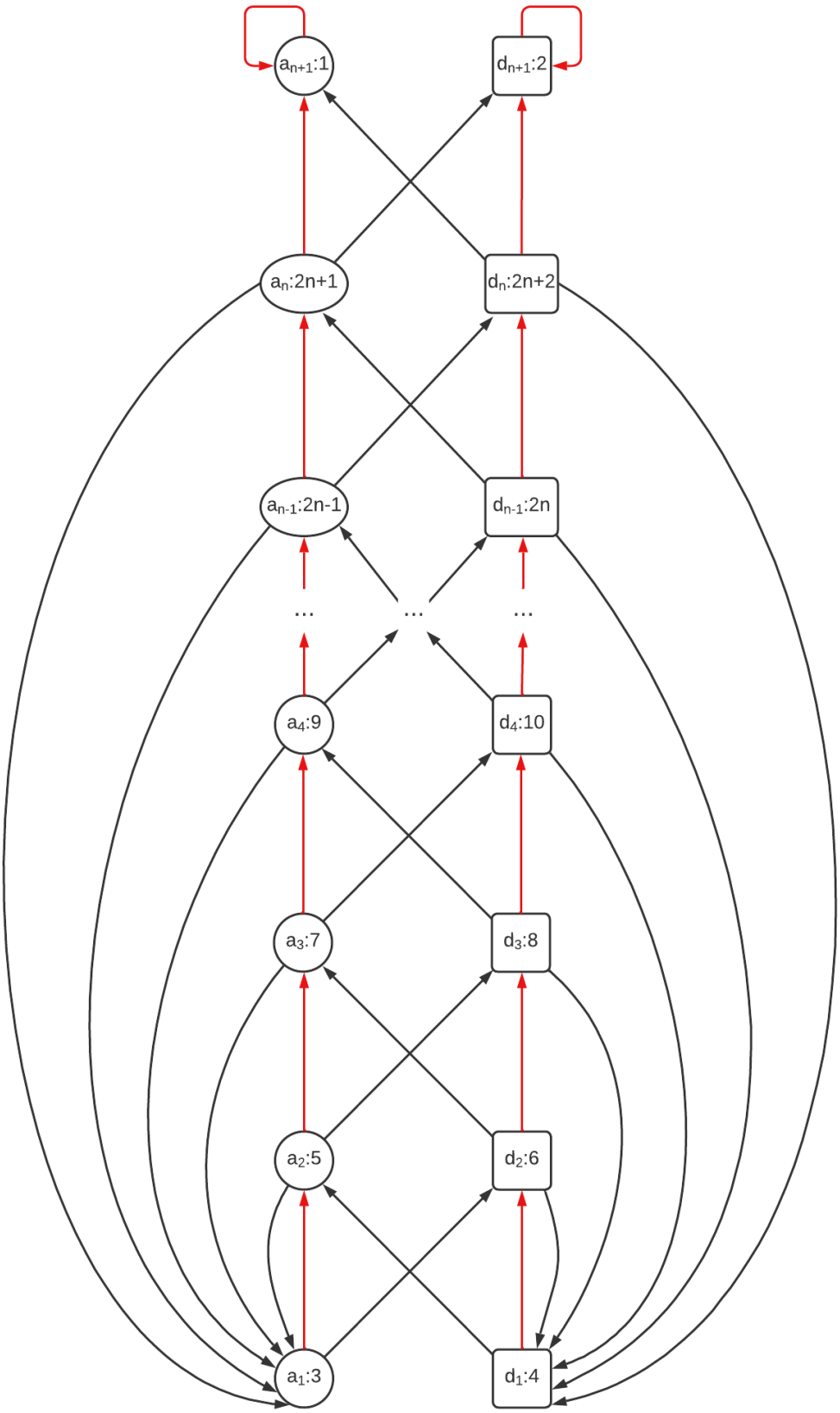


FIGURE 24: The graph G_n with initial strategies σ_0 and τ_0.

Note that there is only one strategy for player 0 for which he can be sure to reach d_{n+1} from $a_1, a_2, \ldots, a_n$ whatever player 1 does. That is the strategy for which $\sigma(a_i) = a_{i+1}$ for $i = 1, 2, \ldots, n-1$ and $\sigma(a_n) = d_{n+1}$. Similarly, player 1 has one strategy that allows him to reach a_{n+1} from any of the nodes $d_1, d_2, \ldots, d_n$. That is the strategy for which $\tau(d_i) = d_{i+1}$ for $i = 1, 2, \ldots, n-1$ and $\tau(d_n) = a_{n+1}$. Player 0 also cannot create any other cycles that win for him. If he goes back to a_1 from any a_j (i.e. $\sigma(a_j) = a_1$), then either he has to choose $\sigma(a_i) = a_{i+1}$ for $i < j$, which creates a winning cycle for player 1, or choose $\sigma(a_i) = d_{i+1}$ for some $i < j$, which allows player 1 to 'escape' to a_{n+1} with the beforementioned strategy τ. From that, we conclude that the optimal strategy for player 0 is the beforementioned strategy that allows him to go to d_{n+1} from nodes $a_1, a_2, \ldots, a_n$. Likewise, the beforementioned player 1 strategy τ that allows him to go to a_{n+1} from $d_1, d_2, \ldots, d_n$ is optimal for player 1.

In conclusion, both players are initially only one improving move away from their optimal strategy. However, it will still take them exponentially many iterations to reach their optimal strategies. The main idea is that symmetric strategy iteration simulates counting in Gray code, where the choices at a_i and d_i represent the bit g_i for $i = 1, 2, \ldots, n$.

We make the counting in Gray code more precise. We make sure that a bit g_i being 0 corresponds to the strategies from a_i and d_i being 'parallel', and g_i being 1 corresponds to these strategies forming a 'cross.' More precisely, we introduce the following notion:

Definition 6.2. Let σ be a player 0 strategy and τ a player 1 strategy for G_n. Let $B \in \{0, 1, \ldots, 2^n - 1\}$, and let $g_n g_{n-1} \ldots g_1$ be the Gray code representation of B. We say that (σ, τ) is in Gray bit state B if the following hold for $1 \leq i \leq n$:

- If $g_i = 0$, then $\sigma(a_i) = a_{i+1}$ and $\tau(d_i) = d_{i+1}$.
- If $g_i = 1$, then $\sigma(a_i) = d_{i+1}$ and $\tau(d_i) = a_{i+1}$.

Note that our initial strategy is in the bit state corresponding to Gray code $000\ldots0$, so in Gray bit state 0. Also, our pair of optimal strategies is in the bit state corresponding to Gray code $100\ldots0$, which is Gray bit state $2^n - 1$.

We also introduce a slightly less strict definition of being in some bit state B.

Definition 6.3. Let j be an integer with $1 \leq j \leq n$, let σ be a player 0 strategy and τ a player 1 strategy. Then we say that (σ, τ) is *j-bitlike* if for $i = j, j+1, \ldots, n$ the following holds:

- Either $\sigma(a_i) = a_{i+1}$ and $\tau(d_i) = d_{i+1}$
- Or $\sigma(a_i) = d_{i+1}$ and $\tau(d_i) = a_{i+1}$

Less formally, we could say a pair of j-bitlike strategies is 'parallel' or 'crossed' for the bits from j up to n. Note that if a (σ, τ) is 1-bitlike, then all bits are parallel or crossed, so then (σ, τ) is in Gray bit state B for some B. The converse also holds: if (σ, τ) is in bit state B, then the pair of strategies is 1-bitlike.

6.4 Steps of symmetric strategy iteration

Now the goal is to show that symmetric strategy iteration on G_n, which starts with a pair of strategies in Gray bit state 0, passes through all 2^n possible Gray bit states from low to high. The main idea why the algorithm makes such tiny improvements is because the backwards edges are almost always part of the optimal counterstrategies but almost never improving. This prevents the algorithm from making many switches at a time. First, we show that there is actually only one case in which a backwards edge is improving:

Lemma 6.4. *For any $i > 1$ and any pair of strategies (σ, τ) encountered in symmetric strategy iteration, we have the following:*

1. *Edge (a_i, a_1) is improving for player 0 only if $\sigma(a_{i-1}) = d_i$ and $\sigma(a_j) = a_{j+1}$ for all $j < i - 1$.*

2. *Edge (d_i, d_1) is not an improving move for player 1.*

Proof: recall from Section 6.3 that neither player 0 nor player 1 can create winning cycles for themselves (other than nodes a_{n+1} and d_{n+1}) against σ_0 or τ_0. So the first component of Ξ_{σ_0} and Ξ^{τ_0} is 1 or 2. In particular, this implies that throughout the algorithm no improving move of player 0 can allow player 1 to create a cycle with odd highest priority except for node a_{n+1}. Otherwise, the valuation Ξ_σ would decrease, which contradicts Lemma 2.4. Likewise, no player 1 improving move can allow player 0 to create another cycle with even highest priority except for node d_{n+1}. Using this, we prove the two statements.

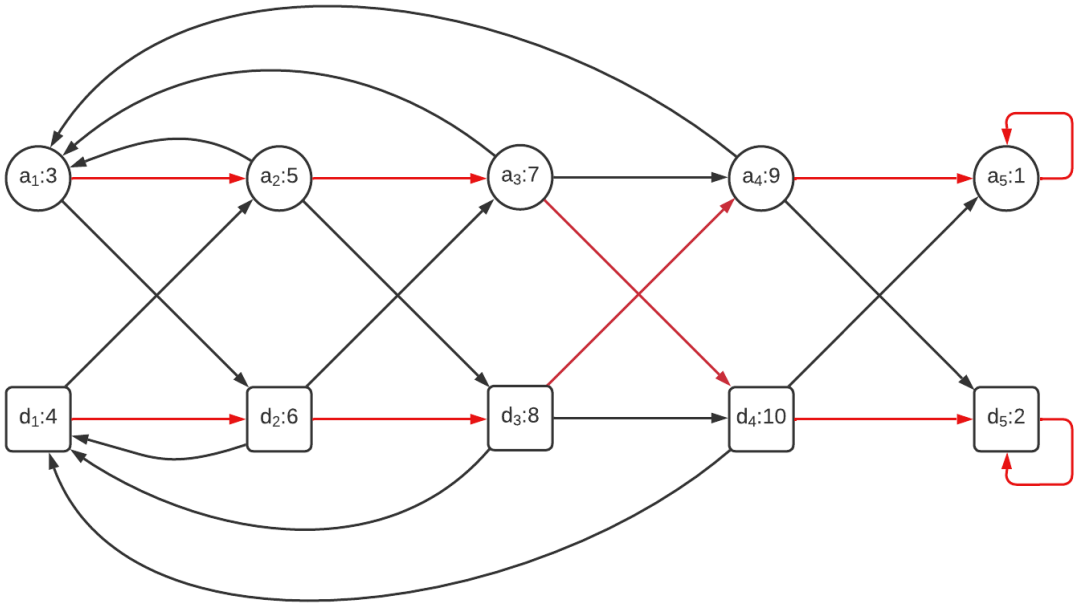


FIGURE 25: One case in symmetric strategy iteration in G_4 where edge (a_4, a_1) is an improving edge. Strategies σ, τ are marked in red.

1. We prove this by contradiction. So suppose (a_i, a_1) is improving but σ is different than described in the lemma. First, suppose $\sigma(a_j) = d_{j+1}$ for some $j < i - 1$. Then player 1 can play $\tau(d_j) = d_{j+1}$ for $j < i - 1$ and $\tau(d_{i-1}) = a_i$. This means that the play starting from a_1 ends up in a_i, so after the improving move there is a cycle with highest priority $p(a_i) = 2i + 1$, and that is not possible. On the other hand, suppose $\sigma(a_j) = a_{j+1}$ for all $j < i$. Then applying the improving move creates the cycle $(a_1, a_2, \ldots, a_i)$, which has again highest priority $2i + 1$. Since both cases lead to a contradiction, the only option left is that $\sigma(a_{i-1}) = d_i$ and $\sigma(a_j) = a_{j+1}$ for all $j < i - 1$.

2. Suppose (d_i, d_1) is an improving move. Note that player 0 can play $\sigma(a_{i-1}) = d_i$, $\sigma(a_i) = a_1$ and $\sigma(a_j) = a_{j+1}$ for all $j < i-1$. If player 1 applies the improving move, then he cannot avoid going back to d_i without creating another winning cycle for player 0. He also cannot escape the set of nodes $\{a_1, a_2, \ldots, a_i, d_1, d_2, \ldots, d_i\}$. In any case player 0 could create a cycle with even highest priority that is larger than 2, which is not possible. ■

Next, we determine where in G_n improving moves are possible.

Lemma 6.5. *Let (σ, τ) be a pair of strategies encountered in strategy iteration. Suppose that (σ, τ) is i-bitlike for some $i \leq n$. Then the following statements are equivalent:*

1. $\Xi_\sigma(a_i) \lhd \Xi_\sigma(d_i)$.
2. $\Xi^\tau(a_i) \lhd \Xi^\tau(d_i)$.
3. *a_i has an improving move towards a_{i+1} or d_{i+1} for player 0.*
4. *d_i has an improving move towards a_{i+1} or d_{i+1} for player 1.*
5. $B_i := \sum_{j=i}^{n} g_j (\mod 2) = 0$.

Proof: Recall that our condition on the strategies σ, τ can be formulated as requiring all the strategies to be either 'parallel' or 'crossed' above a_i and d_i. Also, if (σ, τ) is in some Gray bit state B, then (σ, τ) is i-bitlike for all i. We prove the lemma by (backwards) induction on i. As an induction basis, we show that statements 1,2 and 5 hold for $i = n+1$. Clearly $(1, \emptyset, 0) = \Xi_\sigma(a_{n+1}) = \Xi^\tau(a_{n+1}) \lhd \Xi_\sigma(d_{n+1}) = \Xi^\tau(d_{n+1}) = (2, \emptyset, 0)$. Moreover, $B_{n+1} := \sum_{j=n+1}^{n} g_j (\mod 2) = 0$, so that completes the induction basis. Now as an induction hypothesis, we assume that statements 1,2 and 5 are equivalent for $i+1$, for any pair of $i+1$-bitlike strategies ($i \leq n$). From that, we prove the three statements for i. Note that if a pair of strategies is i-bitlike, then it is also $i+1$-bitlike. We distinguish two cases:

- Suppose $B_{i+1} = 0$. Then from the induction hypothesis we know that $\Xi_\sigma(a_i) \lhd \Xi_\sigma(d_i)$ and $\Xi^\tau(a_i) \lhd \Xi^\tau(d_i)$. Now if $g_i = 0$, so $\sigma(a_i) = a_{i+1}$ and $\sigma(d_i) = d_{i+1}$, then both players have an improving move (edges (a_i, d_{i+1}) and (d_i, a_{i+1})), and also we get $B_i = B_{i+1} + g_i = 0 (\mod 2)$. So statements 3,4 and 5 hold. Moreover, $\sigma(a_i)$ is the worst for player 0 out of a_{i+1} and d_{i+1}, and also the play resulting from σ and $\bar{\sigma}$ from d_i must end in a_{i+1} or d_{i+1}. But d_i has a large even priority, and can only end in the worse of the two successors (since the play has to end at a_{n+1} or d_{n+1}), so the valuation Ξ_σ of d_i can only be higher than that of a_i. So statement 1 holds. With a similar argument, we find that the valuation Ξ_τ of a_i must be lower than that of d_i, implying the second statement. So if $g_i = 0$, then all five statements are true.

  If on the other hand $g_i = 1$, then there are no improving moves for both players towards a_{i+1} or d_{i+1}. The valuations Ξ_σ and Ξ_τ of a_i and d_i must differ in at least some priorities larger than $2i+2$ in the path. That is because for example the plays resulting from σ and $\bar{\sigma}$ starting from a_i and d_i go to a_{n+1} and d_{n+1}, respectively. Therefore there is no use for player 0 to make a switch back to a_1 from a_i, since the play must end back at either a_i, which would create a cycle winning for player 1, or back at d_i, which has a much worse valuation than a_i's current successor. Likewise, there is no use for player 1 to go back to d_1. So there are no improving moves for either player from a_i and d_i, and also $B_i = B_{i+1} + g_i = 1 (\mod 2)$. So in this case

statements 3,4 and 5 all do not hold. Moreover, player 0 is currently choosing the highest valued out of a_{i+1} and d_{i+1}, while in the counterstrategy $\bar{\sigma}$ player 1 chooses the worst out of these two (or something else that yields an even lower valuation). Therefore the valuation Ξ_σ of d_i must be lower than that of a_i. With a similar argument, $\Xi^\tau(a_i) \triangleright \Xi^\tau(d_i)$. Hence all five statements are false. In conclusion, if $B_{i+1} = 0$, then statements 1-5 are equivalent.

- Suppose $B_{i+1} = 1$. Then we know from the induction hypothesis that $\Xi_\sigma(a_i) \triangleright \Xi_\sigma(d_i)$ and $\Xi^\tau(a_i) \triangleright \Xi^\tau(d_i)$. Then if $g_i = 1$, we can argue analogous to the case $B_{i+1} = g_i = 0$ that all five statements are true for i. We could do this just by switching nodes a_{i+1} and d_{i+1} in the proof. If on the other hand $g_i = 0$, we can argue analogous to the case $B_{i+1} = 0$, $g_i = 1$ than all five statements are false for i. Hence also in the case $B_{i+1} = 1$, statements 1-5 are equivalent.

This completes the proof of the induction step and hence of the lemma. ■

Next, we determine what the optimal counterstrategies look like for different nodes in G_n.

Lemma 6.6. *Let i be an integer such that $1 < i \leq n$, and let strategies (σ, τ) be an i-bitlike pair of strategies encountered in symmetric strategy iteration. Suppose that the nodes a_i and d_i both have an improving move towards respectively a_{i+1} and d_{i+1} or respectively d_{i+1} and a_{i+1}. Then $\bar{\tau}(a_i) = a_1$ and $\bar{\sigma}(d_i) = d_1$.*

Proof: Suppose first that $\bar{\tau}(a_i)$ is not equal to $\sigma(a_i)$ but to the 'other' out of a_{i+1} and d_{i+1}, which is $\tau(d_i)$ (as the strategies are i-bitlike). Then player 0 could still improve the play from a_i against τ by choosing edges $(a_i, a_1), (a_1, a_2), (a_2, a_3), \ldots, (a_{i-2}, a_{i-1}), (a_{i-1}, d_i)$, since this also ends up in $\tau(d_i)$ but picks up an extra priority $p(d_i) = 2i + 2$ in the path. Suppose on the other hand that $\bar{\tau}(a_i) = \sigma(a_i)$. Since switching to $\tau(d_i)$ is improving, we have $\Xi_\sigma(\tau(d_i)) \triangleright \Xi_\sigma(\sigma(a_i))$. By Lemma 6.5, this implies $\Xi^\tau(\tau(d_i)) \triangleright \Xi^\tau(\sigma(a_i))$. But then $\bar{\tau}(a_i)$ cannot equal $\sigma(a_i)$, since player 0 could do better against τ by going to $\tau(d_i)$ from a_i. So the only remaining option for $\bar{\tau}(a_i)$ is a_1. This proves the statement for player 0. Proof for player 1 is similar. ■

Combining the results of the previous lemmas, we can determine what happens in certain iterations of the algorithm:

Lemma 6.7. *Suppose $B \in \{0, 1, \ldots, 2^n - 1\}$ is even and (σ, τ) is in Gray bit state B. Then the sets of switches of symmetric strategy iteration are $I_\sigma \cap \{(v, \bar{\tau}(v)) | v \in V_0\} = \{(a_1, \tau(d_1))\}$ and $I_\tau \cap \{(v, \bar{\sigma}(v)) | v \in V_1\} = \{(d_1, \sigma(a_1))\}$. Moreover, after the switches of symmetric strategy iteration, we are in Gray bit state $B + 1$.*

Proof: Note that if B is even, then $B_1 = 0$, so by Lemma 6.5, a_1 and d_1 have an improving move. Because the first two two statements of the same lemma are equivalent, it follows that $\Xi_\sigma(\sigma(a_1)) \triangleleft \Xi_\sigma(\tau(d_1))$ and $\Xi^{tau}(\sigma(a_1)) \triangleleft \Xi^{tau}(\tau(d_1))$. Therefore, these improving moves are also in $\bar{\sigma}$ and $\bar{\tau}$, respectively. Now we need to show there are no other improving moves in $\bar{\sigma}$ and $\bar{\tau}$. From Lemma 6.6, if the other nodes have improving moves that are also part of $\bar{\sigma}$ or $\bar{\tau}$, they must be edges (a_i, a_1) or (d_i, d_1) for some i. If $g_1 = 1$, then these two edges are not improving moves by Lemma 6.4. If $g_1 = 0$, then according to Lemma 6.4, only a_i can have an improving move if $g_1 = g_2 = \ldots = g_{i-2} = 0$ and $g_{i-1} = 1$. But that implies $B_i = 1$, as we assumed $B_1 = 0$. And then Lemma 6.5 implies that a_i and d_i have no improving moves. So indeed, the switches of symmetric strategy iteration are $(d_1, \sigma(a_1))$ and $(a_1, \tau(d_1))$. The Gray code of the strategy pair after the switch is the same, except that g_1 switched, and we had $B_1 = \sum_{j=1}^{n} g_i (\mod 2) = 0$, so the resulting Gray code is that of $B + 1$. This completes the proof of the lemma.

The case where B is odd is a bit more involved than the case where B is even. We introduce a notion to describe the strategies in the bottom part of G_n:

Definition 6.8. Consider a strategy pair (σ, τ). We call the strategy pair j-suboptimal if the following hold:

- $\sigma(a_j) = d_{j+1}$ or $\sigma(a_j) = d_{n+1}$ if $j = n$.
- $\tau(d_j) = a_{j+1}$ or $\tau(d_j) = a_{n+1}$ if $j = n$.
- $\sigma(a_i) = a_{i+1}$ and $\tau(d_i) = d_{i+1}$ for all $i < j$.

Note that if (σ, τ) is in bit state B and it is also j-suboptimal, and $g_n g_{n-1} \dots g_1$ is the Gray code of B, then $g_j = 1$ and $g_i = 0$ for $i < j$. Now we are ready to describe what happens for strategies in Gray bit states corresponding to odd number B:

Lemma 6.9. *Suppose $1 \leq B \leq 2^n - 3$ is odd and (σ, τ) is in Gray bit state B. Suppose also that (σ, τ) is j-suboptimal. Then in the next 3 iterations, the following edges are switched:*

Iteration 1: (a_{j+1}, a_1)

Iteration 2: $(d_{j+1}, \sigma(a_{j+1}))$

Iteration 3: $(a_{j+1}, \tau(d_{j+1}))$

Moreover, the resulting pair of strategies after these three iterations is in bit state $B+1$.

Proof: Note that as $B > 0$, the Gray code of B contains a 1 and hence (σ, τ) is j-suboptimal for some j. Since B is odd, we have $B_1 = 1$ and therefore by Lemma 6.5 there are no improving moves in a_1 and d_1. By Lemma 6.6, the only possible edges to improve in the algorithm are of the form (a_i, a_1) or (d_i, d_1). By Lemma 6.4, only the edge (a_{j+1}, a_1) of these can be improving. By Lemma 6.1 the algorithm always applies some improving move if the strategies are not optimal yet, hence the algorithm applies exactly one improving move, which is (a_{j+1}, a_1), in the first iteration. Let the strategy pair after the first iteration be called (σ^1, τ^1) (see left of Figure 26).

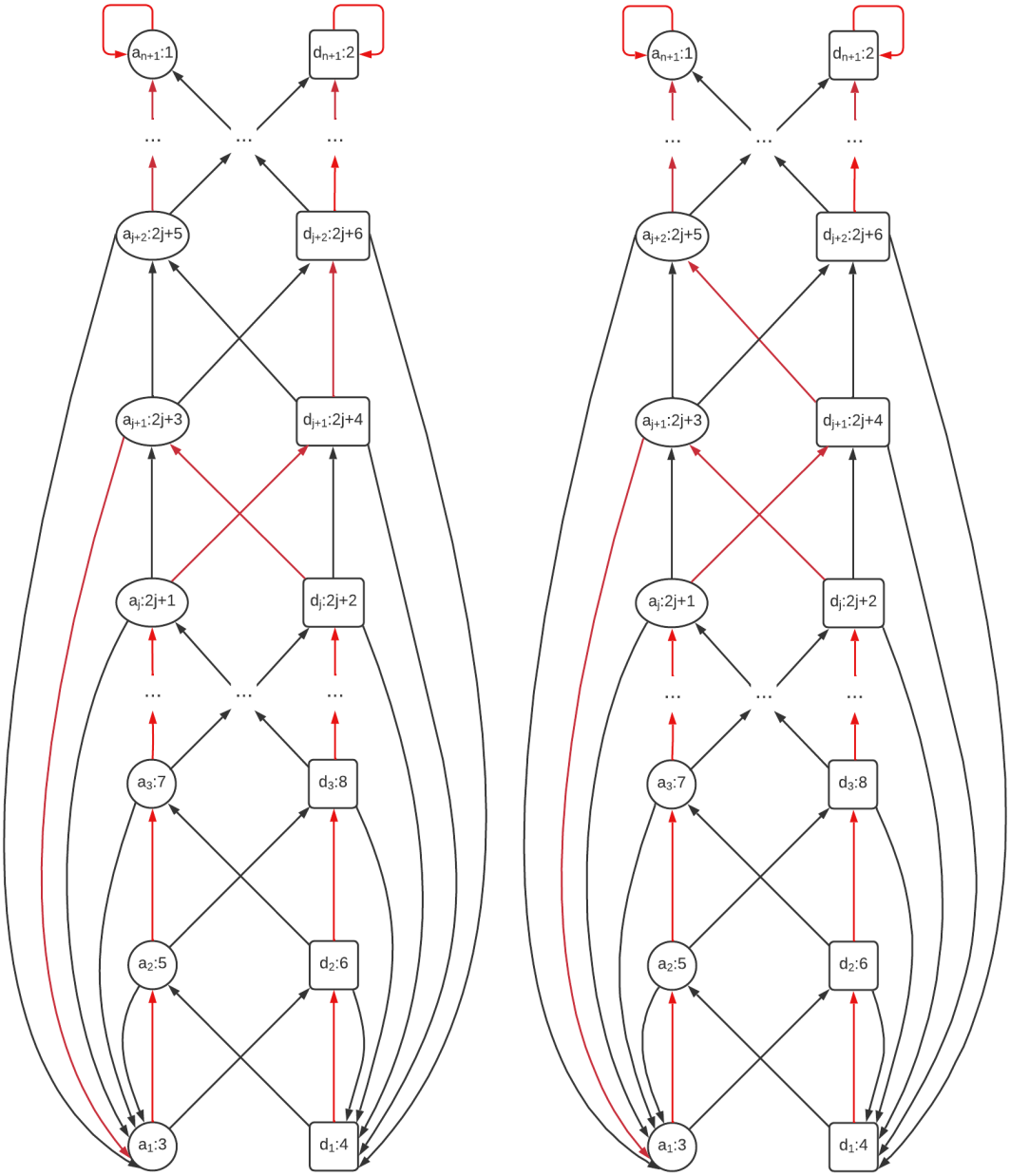


FIGURE 26: Left: in red are strategies after the first iteration from (σ, τ) in bit state B. Right: strategies after the second iteration. Note that there are different options for strategies at the top. The only thing we know about those is that $\tau(d_j)$ has a higher valuation Ξ_σ and Ξ^τ than $\sigma(a_j)$.

Since this pair of strategies is still $j+2$-bitlike, it follows from Lemmas 6.4, 6.5, 6.6 that symmetric strategy iteration makes no switches on a_i and d_i for $i \geq j+2$. Moreover, since in the first iteration there was an improving move at a_{j+1}, Lemma 6.5 tells us that $\Xi_\sigma(\sigma(a_{j+1})) \lhd \Xi_\sigma(\tau(d_{j+1}))$ and hence $\Xi^\tau(\sigma(a_{j+1})) \lhd \Xi_\tau(\tau(d_{j+1}))$. Since nothing changed in the top part, we can derive analogous to the proof of Lemma 6.5 that $\Xi^{\tau^1}(\sigma(a_{j+1})) \lhd \Xi_{\tau^1}(\tau^1(d_{j+1}))$ and $\Xi_{\sigma^1}(\sigma(a_{j+1})) \lhd \Xi_{\sigma^1}(\tau^1(d_{j+1}))$. The first implies that edge $(d_{j+1}, \sigma(a_{j+1}))$

is improving, and the second tells us that $\overline{\sigma^1}(d_{j+1})$ cannot equal $\tau^1(d_{j+1})$ as in player 1's counterstrategy to σ^1 the node $\tau^1(d_{j+1})$ is the higher valued of the nodes a_{j+2} and d_{j+2}. Moreover, edge (d_{j+1}, d_1) is not in the counterstrategy $\overline{\sigma^1}$ anymore, since it would always create a cycle with even highest priority if player 1 used this edge. Therefore $\overline{\sigma^1}(d_{j+1})$ can only be equal to $\sigma(a_{j+1})$, and we saw that this was an improving move, so this edge is switched by the algorithm.

Now we need to prove there are no other switches for nodes $a_1, a_2, \ldots, a_{j+1}, d_1, d_2, \ldots, d_j$. The best player 0 can do against τ^1 is to reach node $\tau^1(d_{j+1})$, since this has the highest valuation Ξ^{τ^1} out of a_{j+2}, d_{j+2}. Moreover, the best path to get there is always through d_{j+1}. The only way to get there from all these player 0 controlled nodes is to use edges $(a_{j+1}, a_1), (a_1, a_2), (a_2, a_3), \ldots, (a_{j-1}, a_j), (a_j, d_{j+1})$, which is exactly the same as player 0 already does with strategy σ^1. The best player 1 can do against σ^1 is to reach $\sigma(a_{j+1})$, since it has the lowest valuation Ξ_{σ^1} out of a_{j+2}, d_{j+2}. This is only possible by passing through d_{j+1} and choosing $\overline{\sigma^1}(d_{j+1}) = \sigma(a_{j+1})$. Moreover, there is one strategy to let all paths from $d_1, d_2, \ldots, d_j$ to d_{j+1} pass through a_{j+1}, which has a high odd priority. That is to use the edges $(d_1, d_2), (d_2, d_3), \ldots, (d_{j-1}, d_j), (d_j, a_{j+1})$. But that is, except for edge $(d_{j+1}, \sigma(a_{j+1}))$, the same as player 1 already plays with strategy τ^1. We conclude that symmetric strategy iteration makes no improving moves in the nodes $a_1, a_2, \ldots, a_{j+1}, d_1, d_2, \ldots, d_j$, so the only improvement that it makes is the edge $(d_{j+1}, \sigma(a_{j+1}))$. We call the strategy pair we get after this switch (σ^2, τ^2) (see right of Figure 26).

Finally, we look at what switches are made in the third iteration on strategies (σ^2, τ^2). We still have a $j+2$-bitlike pair of strategies. Hence we can argue in the same way as before that still no switches will be made for a_i and d_i with $i > j+1$, and also that $\Xi^{\tau^2}(\sigma(a_{j+1})) = \Xi^{\tau^2}(\tau^2(d_{j+1})) \lhd \Xi_{\tau^2}(\tau^1(d_{j+1}))$ and $\Xi_{\sigma^2}(\tau^2(d_{j+1})) \lhd \Xi_{\sigma^2}(\tau^1(d_{j+1}))$. Now we look at the optimal counterstrategies for the nodes $a_1, a_2, \ldots, a_{j+1}, d_1, d_2, \ldots, d_{j+1}$. First we look at the counterstrategy to σ^2. Like before, player 1 would like to end up in $\sigma(a_{j+1}) = \tau^2(d_{j+1})$, and pass a_{j+1} on the way there. The only way to do so is to play the strategy that player 1 is already playing. So symmetric strategy iteration will not find any improving moves that are also part of $\overline{\sigma^2}$. Now to the counterstrategy against τ^2. Player 0 would like to end up in node $\tau^1(d_{j+1})$. It is not possible to get here via d_{j+1} anymore, since player 1 switched in the previous iteration. In fact, the only strategy to get to $\tau^1(d_{j+1})$ from the player 0 nodes is to play the strategy consisting of the edges $(a_1, a_2), (a_2, a_3), \ldots, (a_j, a_{j+1}), (a_{j+1}, \tau^1(d_{j+1}))$. The edges $(a_1, a_2), \ldots, (a_{j-1}, a_j)$ are already used by player 0 in σ^2. The edge (a_j, a_{j+1}) is obviously not an improving move, since switching it creates a cycle with $p(a_{j+1}) = 2j+3$ as highest priority. So the only edge that can be switched by symmetric strategy iteration is $(a_{j+1}, \tau^1(d_{j+1}))$. By Lemma 6.1, there is always at least one edge switched if the strategies are not optimal yet, so exactly this edge is switched by symmetric strategy iteration.

Now we proved that the three iterations indeed work as claimed in the lemma, and we only need to prove that the result is in Gray bit state $B+1$. Our three iterations changed $\sigma(a_{j+1})$ and $\tau(d_{j+1})$ to the other of a_{j+2} and d_{j+2}, so the result must be in some Gray bit state B'. We know B is j-suboptimal, so $g_j = 1$ and $g_i = 0$ for $i < j$. Since B is odd, this also means that $B_1 = \sum_{i=1}^{n} g_i (\mod 2) = 1$. It follows that $B_k = \sum_{i=k}^{n} g_i (\mod 2) = 1$ for $k \leq j$, and $B_{j+1} = 0$.The Gray code of B' differs from that of B in the $j+1$-th bit, and this is indeed the least significant bit k for which $\sum_{i=k} g_i$ is even. So from the properties of Gray code, we know $B' = B+1$, so after three iterations of symmetric strategy iteration, we are in Gray bit state $B+1$. ■

This leads to the main result of this section, which is described in the following theorem.

Theorem 6.10. *The worst case running time of symmetric strategy iteration is exponential in the number of nodes and edges of the graph.*
Proof: From Lemma 6.7 and Lemma 6.9 we know that symmetric strategy iteration runs through all the bit states in $\{0, 1, \ldots, 2^n - 1\}$. From an even bit state, it takes 1 iteration to go to the next one, and from an odd one, it takes 3 iterations. Hence, if $n \geq 2$, then symmetric strategy iteration algorithm reaches the pair of optimal strategies for G_n after $1 \cdot (2^{n-1}) + 3 \cdot (2^{n-1} - 1) = 2^{n+1} - 3$ iterations. Note also that G_n has $2n + 2$ nodes and $6n$ edges. Finally, the worst-case number of iterations cannot be more than exponential as the number of possible strategies is exponentially bounded in the number of nodes and edges. This implies the result of the theorem. □

6.5 Possible improvement

Now we look at how symmetric strategy iteration may be improved such that it does not need an exponential number of iterations on the examples presented in this section. The main reason that symmetric strategy iteration needs an exponential number of iterations is because it only switches the a_i and d_i with the lowest indices. This, in turn, happens because it is only allowed to make switches that are part of the optimal counterstrategies $\bar{\sigma}$ and $\bar{\tau}$. This restriction is there, even though the optimal counterstrategies are not very good strategies during symmetric strategy iteration. So it would make sense to allow the players more freedom to make improving moves while still considering the current other player their strategy. There are probably many ways to do so. Here, we look at one concrete way to do so. We define the following sets:

$$J_\sigma(\tau) = \{(v, w) : v \in V_0 \wedge \Xi^\tau(w) \trianglerighteq \Xi^\tau(\sigma(v))\}$$
$$J_\tau(\sigma) = \{(v, w) : v \in V_1 \wedge \Xi_\sigma(w) \trianglelefteq \Xi_\sigma(\tau(v))\}$$

We could view $J_\sigma(\tau)$ as the set of edges that are 'better' for player 0 against τ than σ. Note that $(v, \bar{\tau}(v)) \in J_\sigma(\tau)\ \forall v \in V_0$ (if one such edge was not in $J_\sigma(\tau)$, this would imply player 0 has an even better strategy against τ than $\bar{\tau}$). Likewise, $J_\tau(\sigma)$ can be viewed as the edges that are 'better' for player 1 than τ against σ. Also $(v, \bar{\sigma}(v)) \in J_\tau(\sigma)\ \forall v \in V_1$. Recall that

$$I_\tau = \{(v, w) : v \in V_1 \wedge \Xi^\tau(w) \triangleleft \Xi^\tau(\sigma(v))\}$$
$$I_\sigma = \{(v, w) : v \in V_0 \wedge \Xi_\sigma(w) \triangleright \Xi_\sigma(\tau(v))\}$$

Then we could, instead of letting player 0 switch the edges from $I_\sigma \cap \{(v, \bar{\tau}(v)) | v \in V_0\}$ to σ, let him switch a number of edges from $I_\sigma \cap J_\sigma(\tau)$ in symmetric strategy iteration. Of course, there can be multiple options per node, so one would need an improvement rule to decide which edges to switch. Likewise, player 1 can switch a number of edges from $I_\tau \cap J_\tau(\sigma)$ instead of $I_\tau \cap \{(v, \bar{\sigma}(v)) | v \in V_0\}$.

Correctness of this algorithm is immediately implied by Lemma 6.1, since we can still switch edges from $\bar{\sigma}$ and $\bar{\tau}$. Moreover, if the switch-all improvement rule is used (switch one edge from each node where there is an improving move), then the algorithm solves graph G_n in n iterations. This is because it would make the right switches in a_n and d_n in the first iteration, and in a_{n-1} and d_{n-1} in the next, and so on. Moreover, it likely still does not fall for the traps from Friedmanns many counterexamples [11, 12]. This is for the same argument as for symmetric strategy iteration: because once player 1 moves out of a 'trap', he will not fall for it again as it is not an improving move. Player 0 will also not be under the illusion that player 1 will fall for the trap again as he always considers strategy τ.

7 Conclusion and discussion

In this thesis, we looked at two structures that can be used to construct families of parity games with exponential running time for discrete strategy iteration: the binary counter and the reverse binary counter. These structures could be used to make such examples for a number of different deterministic single-switch memoryless improvement rules. Of course, this structure would not work for any deterministic single-switch memoryless improvement rule, let alone for any improvement rule. But filling in a gadget makes it relatively easy to make a counterexample. Mostly these structures can serve as a benchmark to help in designing good improvement rules for parity games.

Secondly, we looked at a new algorithm called subgame iteration, and showed that any run of this algorithm can be related to a run of the simplex algorithm. This means that examples with exponential running time in parity games can be translated to a linear program where the simplex algorithm needs an exponential number of iterations. This holds both for discrete strategy iteration on sink parity games, and for subgame iteration. This mostly provides a tool to analyze the combinatorial structure of LP polyhedrons more easily. Coming up with an example with exponential running time (in terms of number of equations and variables) can be reduced to finding the equivalent improvement rules and constructing a gadget with the desired behavior. Although this is definitely not trivial and takes some creativity, it is definitely possible as shown by the four examples of classical pivot rules. A drawback of the reduction is the size of the coefficients of the linear program, as they are doubly exponential in terms of the node priorities. Therefore, the reduction cannot immediately be used to show exponential running time in the input size of the linear program.

Finally, we showed that the symmetric strategy iteration algorithm has exponential worst-case performance. This was by constructing a family of parity games where this algorithm behaves like a counter in Gray code. This example could be used as a benchmark for new algorithms for solving parity games.

The immediate applications of the results from this thesis lie in the new improvement rules and the adaptation of symmetric strategy iteration. In a bit less direct way, the tools for analyzing improvement rules for strategy iteration and pivot rules for the simplex algorithm might lead to better rules.

To answer the main question: Is there a structured way to construct examples with exponential running time for strategy iteration in parity games, and, related to that, for pivot rules in the simplex algorithm for linear programming? There is definitely a structured way to construct such examples for strategy iteration in parity games. There are even multiple ways as shown in this thesis. There is also a structured way to translate this to the simplex algorithm, in terms of the number of variables and equations. So in conclusion, the answer to the main question is yes for many pivot and improvement rules, with some creativity, for one definition of exponential running time.

8 Recommendations for future research

Probably the most useful way in which the results of this thesis can contribute to future research is in the tools to analyze pivot rules and improvement rules for strategy iteration. The question whether there exists an efficient (polynomial-time) simplex pivot rule or strategy iteration improvement rule is still open. It could be that trying to find deterministic single-switch memoryless improvement rules that solve the 'hard' parity games described in this thesis could lead to better improvement rules. In particular the question whether there is an improvement rule that makes the algorithm from Section 6.5 a polynomial-time algorithm is interesting.

Moreover, in designing a potential polynomial-time improvement rule for discrete strategy iteration, one would need to make a rule that switches the gadgets from Section 3 closest to the sink first. Therefore it would be interesting to see if improvement rules can be designed that prefer nodes 'close' to the sink, and if they perform well. Although one should be careful that if the rule naively switches nodes with a low path value, then one may design a parity game in which player 1 can make the most important switches look 'far away.'

For applied purposes, one could investigate the performance in practice of the many suggested improvement rules for discrete strategy iteration, symmetric strategy iteration and the subgame iteration algorithm. One could compare them against currently used algorithms, or try to combine them with these improvement rules.

About the reduction in this thesis, one could still look at the coefficients in the reduction from parity games to linear programs. In particular, one could wonder whether the size of these can be reduced to at most exponential in terms of the node priorities of the graph. Then the technique described in this thesis could yield families of linear programs that need a number of iterations exponential in terms of the input size of the LP.

Finally possible continuation could be to consider more complicated pivot rules for the simplex algorithm, maybe even primal-dual variations of the simplex algorithm. One could look if these rules also have a combinatorial interpretation in strategy iteration, subgame iteration, or maybe a completely new algorithm in parity games. It may be possible to construct a reduction from parity games to linear programs that, similar to the results of this thesis, relates a discrete algorithm in parity games to the simplex algorithm.

A Implementation

A Python implementation of the symmetric strategy iteration algorithm and counterexamples in Section 6 can be found at `https://github.com/MatthewMaat/Master-thesis`.

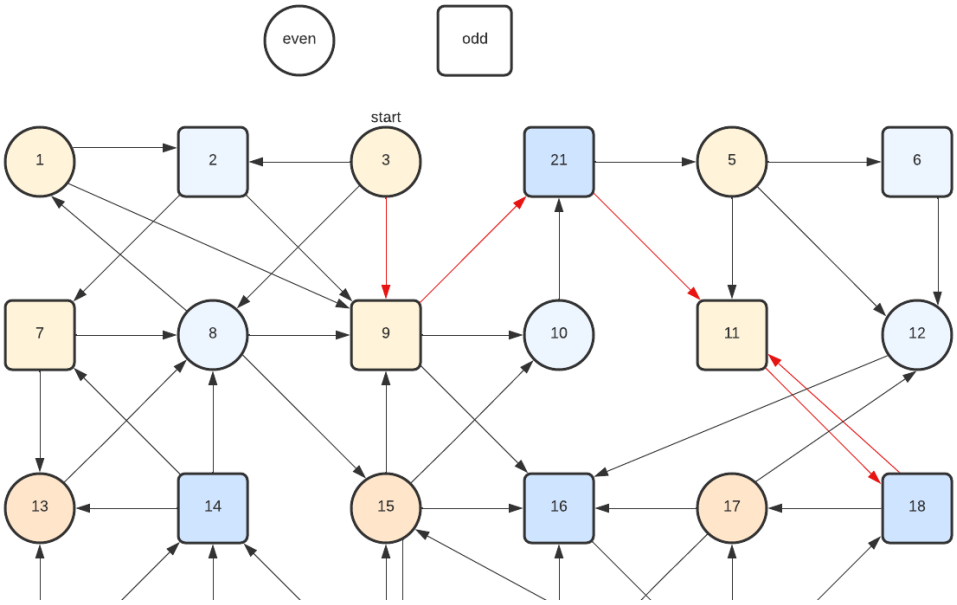
B Table of symbols

Symbol	Meaning(s)	Defined in section
$\preceq, \trianglelefteq$	linear orders on paths and valuations	2.4, 4
$\uplus, \setminus$	Multiset addition, multiset difference	2.2
A	Matrix of coefficients in simplex algorithm	2.6
a_i, A_i	A_i: gadget in G_n. a_i: node in A_i	3
b	Constant vector in linear programming	2.6
	Constant equal to $b = n^{3n}$	4
$b(P)$	Function of the priorities of P	4.4
B	Basis of linear program	2.6
	Number between 0 and $2^n - 1$	3, 6
B_i	i-th bit of binary representation of B	3, 6
c	Cost vector of linear programming	2.6
d_i, D_i	D_i: gadget in G_n. d_i: node in A_i	3
E	Edge set of graph G	2.3
$E_i, i \in \{0, 1\}$	Set of edges that can be used by player i	2.3
g_i	i-th bit of Gray code	6
G	$G = (V, E)$ is a graph	2.3
$(G_n))n \in \mathbb{N}$	Family of graphs	3
$G_{S,T}$	Subgraph of G defined by S and T	4.1
I, I_σ, I_τ	Set of improving moves (for strategy σ or τ)	2.4, 6.1
$\mathcal{M}(Q)$	Set of multisets of Q	2.4
$M_\sigma(v, v')$	Shortest path from v to any other node	3.4
n	Number of nodes or gadgets in G	
N	large (odd) number	3
P_{ij}	(shortest) path from i to j	4
$p(v)$	Priority of vertex v	2.3
$\mathcal{P}(Q)$	Powerset of Q	2.4
Q	Set of priorities occurring in G	2.4
S	Partial player 0 strategy	4.1
T	A set of player 1 controlled nodes	4.1
V	Vertex set of graph G	2.3
$V_i, i \in \{0, 1\}$	Set of vertices controlled by player i	2.3
$W_i, i \in \{0, 1\}$	Set of vertices from which player i has a winning strategy	2.3
ξ_e	Lexicographically sorted vector of valuations	3.4
Ξ_σ	Valuation for strategy iteration for player 0 strategy σ	2.4
Ξ^τ	Valuation for strategy iteration for player 1 strategy τ	6.1
σ	Strategy for player 0, $\sigma : V_0 \to V$	2.3
$\bar{\sigma}, \bar{\tau}$	Optimal counterstrategy to strategy σ, τ	2.4
τ	Strategy for player 1, $\tau : V_1 \to V$	2.3
$\Psi_{S,T}(v)$	Valuation of v for S, T in subgame iteration	4.1

C A playable parity game

The rules of the parity game are as follows:

- The game starts in the node marked with "start"
- If you are at a circle, player "even" marks an outgoing edge of the current node, and if you are at a square, player "odd" marks an outgoing edge from the current node.
- If marking an edge creates a cycle, the game ends. The winner of the game is determined by the highest priority in the cycle. If it is even, the even player wins, and if it is odd, the odd player wins.
- For example, in the game below, the even player marks the edge from the start node. Because the next node is a square, it is player "odd" his turn. Then player "odd" goes to the node with priority 21, then to the one with priority 11, then to the one with 18, and back to the one with 11. The game ends since there is a cycle. The largest number in the cycle is 18, so the even player wins.



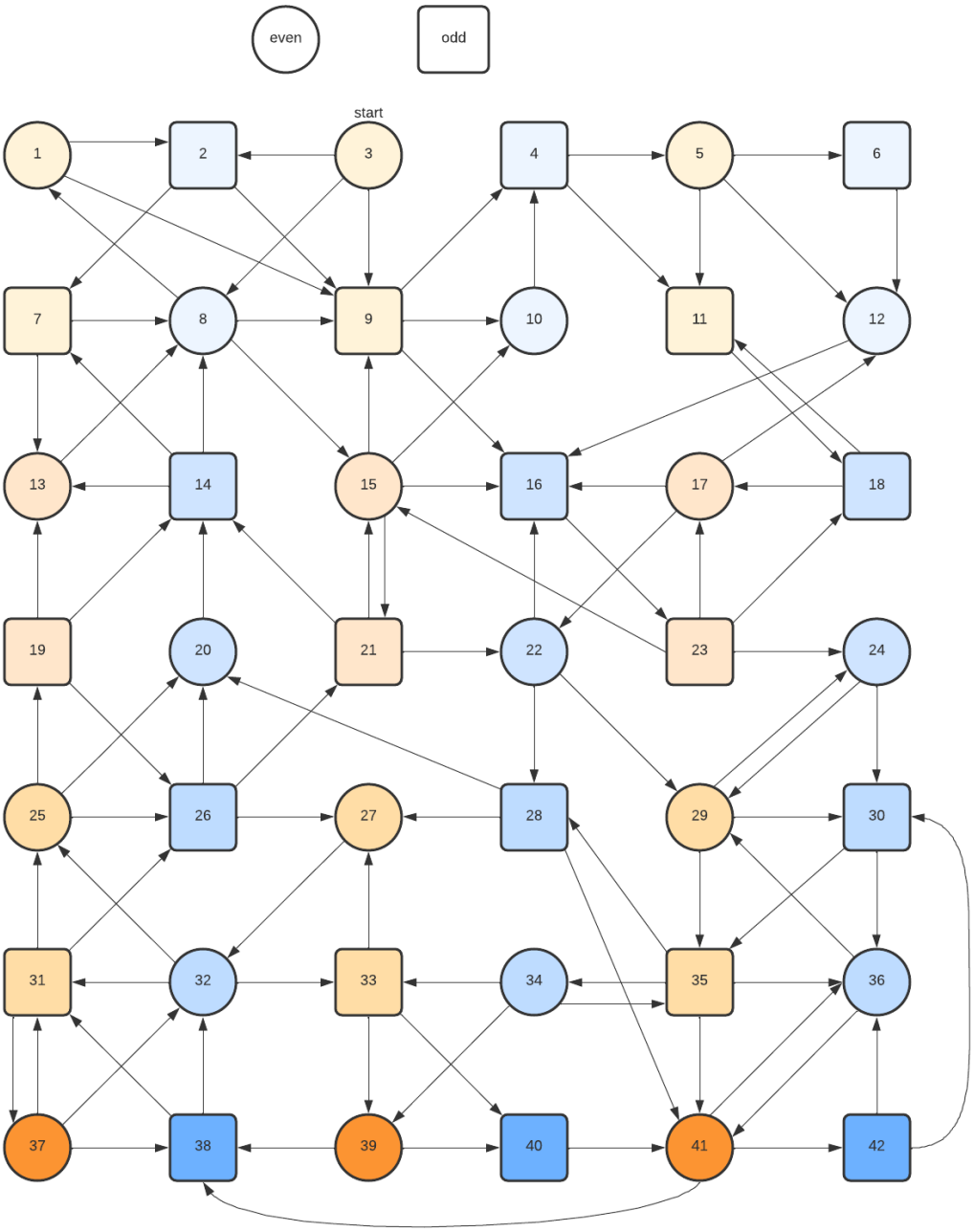


FIGURE 27: Playable parity game

D Generalized improvement rules

This section describes a generalized notion of improving moves for discrete strategy iteration in parity games. Every improving move in the usual strategy iteration sense is also a generalized improving move, but generalized framework could allow moves that would not be considered improving in the usual way. However, no procedure to find these generalized improving moves efficiently was found during the course of the master thesis. The concepts and notation used in this section is very similar to [18].

We let from now on n be the number of nodes in a parity game. Recall that we can assume that all priorities in the game are unique and between 1 and $2n$. Let $E_\sigma = \{(v,w) \in E |\ \sigma(v) = w \vee (v,w) \in E_1\}$ for a player 0 strategy σ and $E_\tau = \{(v,w) \in E |\ \tau(v) = w \vee (v,w) \in E_0\}$ for a player 1 strategy τ.

We introduce a linear order $\preceq$ on Q, given by $l_1 \preceq l_2 \Leftrightarrow (-1)^{l_1} \cdot l_1 \leq (-1)^{l_2} \cdot l_2$. We also have the pre-order $\tilde{\trianglelefteq}$ on the space of valuations $Q \times M(Q) \times \mathbb{Z}_{\geq 0}$ given by $(l_1, B_1, m_1) \tilde{\trianglelefteq} (l_2, B_2, m_2) \Leftrightarrow (l_1, B_1, m_1) \trianglelefteq (l_2, B_2, m_1)$.

We introduce a generalized version of an improvement rule. First, we introduce the $\boxplus$ operator, which behaves similar to the $\boxplus$ operator from [18], but formulated in a way that will simplify the notation a bit. Let VAL be the space of possible valuations $(Q \times M(Q) \times \mathbb{Z}_{\geq 0})$, where $M(Q)$ is the set of multisets of Q. We define the operator $\boxplus : V \times VAL \to VAL$ as follows:

- If $p(v) \leq l$ then $(l, B, m) \boxplus v = (l, B, m+1)$.
- If $p(v) > l$, then $(l, B, m) \boxplus v = (l, B \uplus \{p(v)\}, m+1)$, where $\uplus$ denotes multiset addition.

For simplicity we define $\boxplus$ also for a path $R = (v_1, v_2, \ldots, v_k)$ by

$$(l, B, m) \boxplus R = (\ldots((l, B, m) \boxplus v_1) \boxplus v_2) \boxplus \ldots v_k$$

Next, we introduce the concept of a *low-progressive* valuation, which generalizes the concept of *locally under-progressive* valuations from [18].

Definition D.1. We call a valuation $\Xi = (\Lambda, \Pi) : V \to Q \times (M(Q) \times \mathbb{Z}_{\geq 0})$ low-progressive for a strategy σ if the following holds for the graph $G_\sigma = (V, E_\sigma)$:

- For every vertex $v \in V$, there exists a set $X_v \subseteq V$ such that:
  1. There does not exist a play $P = (v, v_2, \ldots, v_l)$ in G_σ such that $P \cap X_v = \emptyset$.
  2. Paths from v to X_v cannot 'skip' the node with priority $\Lambda(v)$. More precisely, if there is a path $R = (v, v_1, v_2, \ldots, v_k)$ which contains a vertex $w \neq v$ with $p(w) = \Lambda(v)$, and if $(v_1, v_2, \ldots, v_k) \cap X_v = v_k$, then $v_k = w$ and $w \in X_v$.
  3. For every (possibly closed) path $R = (v, v_1, v_2, \ldots, v_k)$ with $v_k \in X_v$, we have the following:
     (a) If $\Lambda(v) \neq p(v)$, then $\Xi(v) \trianglelefteq \Xi(v_k) \boxplus (v, v_1, v_2, \ldots, v_{k-1})$.
     (b) If $\Lambda(v) = p(v)$, then $\Pi(v) = (\emptyset, 0)$ and $\Xi(v) \tilde{\trianglelefteq} \Xi(v_k) \boxplus (v, v_1, v_2, \ldots, v_{k-1})$.

This then allows us to formulate the notion of a general improving switch.

Definition D.2. We call player 0 strategy σ' a *general improvement* for player 0 strategy σ if Ξ_σ is low-progressive for strategy σ', and if there is at least one node v for which all inequalities of the third condition of low-progressiveness are strict.

We prove the following lemma, which shows that performing a general improvement results in a better strategy for player 0, since it implies that $\Xi_{\sigma'}(v) \trianglerighteq \Xi_\sigma(v)$ for all nodes and $\Xi_{\sigma'}(v) \triangleright \Xi_\sigma(v)$ for at least one node.

Lemma D.3. *If $\Xi = (\Lambda, \Pi)$ is low-progressive for some strategy σ, then $\Xi(v) \trianglelefteq \Xi_\sigma(v)$ for all $v \in V$, and $\Xi(v) \triangleleft \Xi_\sigma(v)$ for at least one node v.*

Proof: For the first part, it suffices to show that $\Xi(v) \trianglelefteq (\lambda(P), \pi(P), \#(P))$ for all $v \in V$ and for every play P that starts in v on the game on G_σ. Let $P = (v_1, v_2, v_3, \ldots, v_l)$, and let $p(v_j)$ be the largest priority in the cycle of P.

First, we show that $\Lambda(v_1) \preceq \lambda(P)$. To do so, it sufices to show that $\Lambda(v) \preceq \lambda(P)$ for all v on the cycle of P. This is because then, we could pick a successor v_1^1 of v_1 in P that is in X_{v_1}, and then a successor v_1^2 of v_1^1 that is in $X_{v_1^1}$, until we get a v_1^k that is in the cycle. If then $Path(v, w)$ denotes the path from v to w (not including w) in P, then we get from low-progressiveness

$$\Xi(v_1) \tilde{\trianglelefteq} \Xi(v_1^k) \boxplus Path(v_1^{k-1}, v_1^k) \boxplus \ldots \boxplus Path(v_1, v_1^1)$$

Hence $\Lambda(v_1) \preceq \Lambda(v_1^k) \preceq \lambda(P)$ in that case.

Now suppose, on the contrary, that $\Lambda(v^0) \succ \lambda(P)$ for some v^0 on the cycle of P. Because of low-progressiveness, there must be some vertex $v^1 \in X_v$ on the cycle of P with therefore $\Xi(v) \tilde{\trianglelefteq} \Xi(v^1) \boxplus Path(v, v^1)$. With the same argument, there is some $v^2 \in X_{v^1}$ in the cycle, some $v^3 \in X_{v^2}$ and so on. At some point we must find some vertex twice in the sequence $v^0, v^1, v^2, \ldots$, say $v^s = v^t$. This implies that

$$\Xi(v^s) \tilde{\trianglelefteq} \Xi(v^s) \boxplus Path(v^s, v^{s+1}) \boxplus \ldots \boxplus Path(v^{t-1}, v^s) = \Xi(v^s) \boxplus C \boxplus C \boxplus \ldots \boxplus C \quad (17)$$

where C denotes the cycle of P. Note that we also have $\Lambda(v^s) \succ \lambda(P)$, and also, because of the inequalities along the cycle, we must have $\Lambda(v^s) = \Lambda(v^{s+1}) = \ldots = \Lambda(v^t)$. We distinguish a number of cases:

1. $\Lambda(v^s)$ is even. Then, we have two subcases:

   (a) $\lambda(P) < \Lambda(v^s)$. In that case, $\Lambda(v^s)$ does not occur in the cycle of P, and therefore the inequality in (17) holds for $\trianglelefteq$ as well (because case 3(b) of the low-progressiveness condition does not occur). Also, adding the cycle to a valuation only increases the third component of the valuation. So we get

   $$\Xi(v^s) \trianglelefteq \Xi(v^s) \boxplus C \boxplus C \boxplus \ldots \boxplus C$$

   because of low-progressiveness, where the only difference between the left and right side is that the right side has a larger third component. But then we have a contradiction, because

   $$\Xi(v^s) \triangleright \Xi(v^s) \boxplus C \boxplus C \boxplus \ldots \boxplus C$$

   as $\Lambda(v^s)$ is even, so a larger third component decreases the valuation.

   (b) $\lambda(P) > \Lambda(v^s)$, which implies that $\lambda(P)$ is odd, since we had $\lambda(P) \prec \Lambda(v^s)$. But then

   $$\Xi(v^s) \tilde{\triangleright} \Xi(v^s) \boxplus C \boxplus C \boxplus \ldots \boxplus C$$

   because the largest number in the cycle is odd, which contradicts (17).

2. $\Lambda(v^s)$ is odd. This implies that $\lambda(P) > \Lambda(v^s)$ and $\lambda(P)$ is odd. But then again

$$\Xi(v^s) \tilde{\rhd} \Xi(v^s) \boxplus C \boxplus C \boxplus \ldots \boxplus C$$

because the largest number in the cycle is odd, and this contradicts (17).

Since all cases lead to contradiction, we conclude $\Lambda(v^s) \preceq \lambda(P)$, hence $\Lambda(v^0) \preceq \lambda(P)$ for all v^0 in the cycle, hence $\Lambda(w) \preceq \lambda(P)$ for all nodes w in P from low-progressiveness, in particular for the starting node v_1.

Now we need to prove that $\Xi(v) \trianglelefteq (\lambda(P), \pi(P), \#(P))$ if $\Lambda(v) = \lambda(P)$. We again define $v^1 \in P$ to be the next node in X_v in P, and similarly $v^2 \in X_{v^1}$, and so on. We already proved that $\Lambda(w) \leq \lambda(P)$ for all nodes w in P, and as $\Lambda(v) \preceq \Lambda(v^1) \preceq \Lambda(v^2) \preceq \ldots$, we have $\lambda(P) = \Lambda(v) = \Lambda(v^1) = \Lambda(v^2) = \ldots$.

Now let $\bar{v}$ be the vertex in the cycle of P such that $p(\bar{v}) = \lambda(P)$. By the second condition of low-progressiveness, we see that the sequence $v, v^1, v^2, \ldots$ cannot 'skip' $\bar{v}$, and we can say that r is the first index for which $v^r = \bar{v}$. Then $p(v^i) \neq \lambda(P)$ for $i < r$, and we have by the third condition of low-progressiveness that

$$\begin{aligned}
\Xi(v) &\trianglelefteq \Xi(\bar{v}) \boxplus Path(v, v^1) \boxplus Path(v^1, v^2) \boxplus \ldots \boxplus Path(v^{r-1}, \bar{v}) \\
&= \Xi(\bar{v}) \boxplus Path(v, \bar{v}) \\
&= (\lambda(P), \emptyset, 0) \boxplus Path(v, \bar{v}) \\
&= (\lambda(P), \pi(P), \#(P))
\end{aligned}$$

and this completes the proof that $\Xi(v) \trianglelefteq \Xi_\sigma(v)$ for all $v \in V$. Next, by, low-progressiveness we know that there is a vertex v such that the inequalities of the third condition are strict. Consider the play resulting from strategy σ and optimal counterstrategy $\bar{\sigma}$ when the game is started from v, which we call $P_{\sigma\bar{\sigma}}(v)$. If v^1 is the next node of X_v on $P_{\sigma\bar{\sigma}}(v)$, then we see that $\Xi_\sigma(v) = \Xi_\sigma(v^1) \boxplus Path(v, v^1)$. Because of the assumption on v we also have $\Xi(v) \tilde{\lhd} \Xi(v^1) \boxplus Path(v, v^1)$ (note that if $\tilde{\lhd}$ holds then also $\lhd$ holds). Finally, because of what we just proved about Ξ and Ξ_σ, we have $\Xi(v^1) \trianglelefteq \Xi_\sigma(v^1)$. This yields

$$\Xi(v) \lhd \Xi(v^1) \boxplus Path(v, v^1) \trianglelefteq \Xi_\sigma(v^1) \boxplus Path(v, v^1) = \Xi_\sigma(v)$$

This completes the proof of Lemma D.3 ■

So now we know that applying a general improvement increases the valuation of the nodes in the graph. Now we only need to prove that there is a general improvement if and only if the current strategy is not optimal.

Lemma D.4. *The following statements are equivalent:*

1. *There exists a general improvement σ' for player 0 strategy σ.*
2. *σ is not optimal*
3. *There exists an improving move for σ, i.e. the set of improving switches I is nonempty.*

Proof:
$(1 \Rightarrow 2)$ This follows immediately from Lemma D.3.
$(2 \Rightarrow 3)$ This follows from Lemma 5.8 from [18].
$(3 \Rightarrow 1)$ Suppose there is an improving move from player 0 node v_0, say $\sigma(v_0) = w$, $(v_0, z) \in E_0$ and $\Xi_\sigma(z) \rhd \Xi_\sigma(w)$. Then we claim that the strategy σ', which is the same as σ except that $\sigma'(v_0) = z$, is a general improvement to σ. If σ' is obtained from σ by an improving move, then Proposition 5.3 of [18] says that $\Xi_\sigma = (\Lambda, \Pi)$ is *locally under-progressive* for σ', i.e. for any edge (v, w) in G_σ we have

- If $\Lambda(v) \neq p(v)$, then $\Xi_\sigma(v) \trianglelefteq \Xi_\sigma(w) \boxplus v$
- If $\Lambda(v) = p(v)$, then $\Pi(v) = (\emptyset, 0)$ and $\Xi_\sigma(v) \tilde{\trianglelefteq} \Xi_\sigma(w) \boxplus v$

It is obvious that if Ξ_σ is locally under-progressive for σ', then it is also low-progressive for σ', since we can just choose X_v to be the set of successors of v in $G_{\sigma'}$. In particular, choosing $X_{v_0} = \{z\}$ we have that $\Xi_\sigma(v_0) \triangleleft \Xi_\sigma(w) \boxplus v_0$ so the inequalities of the third condition of low-progressiveness hold strictly for v.[6] Therefore, the improvement is also a general improvement. ■

We conclude that from Lemma D.3 and Lemma D.4 follows that an algorithm that applies general improvements until there are no general improvements possible solves parity games.

[6] If $\Lambda(v) = p(v)$ we should have $\Xi_\sigma(z) \tilde{\triangleright} \Xi_\sigma(w)$

References

[1] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. Combinatorial Simplex Algorithms Can Solve Mean Payoff Games. *http://dx.doi.org/10.1137/140953800*, 24(4):2096–2117, 12 2014. ISSN 10526234. doi:10.1137/140953800. URL https://epubs.siam.org/doi/abs/10.1137/140953800.

[2] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. Tropicalizing the Simplex Algorithm. *http://dx.doi.org/10.1137/130936464*, 29(2):751–795, 4 2015. ISSN 08954801. doi:10.1137/130936464. URL https://epubs.siam.org/doi/abs/10.1137/130936464.

[3] D. Avis and V. Chvátal. Notes on Bland's pivoting rule. pages 24–34, 1978. doi:10.1007/BFB0121192.

[4] Henrik Björklund, Sven Sandberg, and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3153:673–685, 2004. ISSN 16113349. doi:10.1007/978-3-540-28629-5_52/COVER/. URL https://link.springer.com/chapter/10.1007/978-3-540-28629-5_52.

[5] Robert G. Bland. New Finite Pivoting Rules for the Simplex Method. *https://doi.org/10.1287/moor.2.2.103*, 2(2):103–107, 5 1977. ISSN 0364-765X. doi:10.1287/MOOR.2.2.103. URL https://pubsonline.informs.org/doi/abs/10.1287/moor.2.2.103.

[6] George Dantzig. Linear Programming and Extensions. *Linear Programming and Extensions*, 12 1963. doi:10.1515/9781400884179/HTML.

[7] Yann Disser, Oliver Friedmann, and Alexander V Hopp. An Exponential Lower Bound for Zadeh's pivot rule *. Technical report, 2020. URL https://doi.org/10.48550/arXiv.1911.01074.

[8] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 697 LNCS:385–396, 1993. ISSN 16113349. doi:10.1007/3-540-56922-7_32/COVER/. URL https://link.springer.com/chapter/10.1007/3-540-56922-7_32.

[9] John Fearnley. Exponential lower bounds for policy iteration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6199 LNCS(PART 2):551–562, 2010. ISSN 03029743. doi:10.1007/978-3-642-14162-1_46.

[10] John Fearnley. Non-oblivious Strategy Improvement. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6355 LNAI:212–230, 2010. ISSN 03029743. doi:10.1007/978-3-642-17511-4_13. URL https://link.springer.com/chapter/10.1007/978-3-642-17511-4_13.

[11] Oliver Friedmann. Exponential lower bounds for solving infinitary payoff games and linear programs. 2011. URL https://edoc.ub.uni-muenchen.de/13294/.

[12] Oliver Friedmann. A superpolynomial lower bound for strategy iteration based on snare memorization. *Discrete Applied Mathematics*, 161(10-11):1317–1337, 7 2013. ISSN 0166-218X. doi:10.1016/J.DAM.2013.02.007.

[13] Donald Goldfarb and William Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277–285, 12 1979. ISSN 0166-218X. doi:10.1016/0166-218X(79)90004-0.

[14] Frank Gray. Pulse code communication, 1953. URL `https://www.freepatentsonline.com/2632058.pdf`.

[15] Thomas Dueholm Hansen. *Worst-case analysis of strategy iteration and the simplex method*. PhD thesis, 2012. URL `https://pure.au.dk/portal/files/52807524/PhD_dissertation_Thomas_Dueholm_Hansen.pdf`.

[16] R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Elsevier*. URL `https://www.sciencedirect.com/science/article/pii/0012365X73901714`.

[17] Marcin Jurdziński. Deciding the winner in parity games is in UP co-UP. *Information processing letters*, 68(3):119–124, 1998. URL `https://www.sciencedirect.com/science/article/pii/S0020019098001501`.

[18] Marcin Jurdzinski and Jens Vöge. A Discrete Stratety Improvement Algorithm for Solving Parity Games. *BRICS*, 2000. URL `https://www.brics.dk/RS/00/48/BRICS-RS-00-48.pdf`.

[19] Victor Klee, George Minty, János Pach, Bruce Reed, and Yelena Yuditsky. How good is the simplex algorithm. *cgm.cs.mcgill.ca*, 1970. doi:10.4230/LIPIcs.SoCG.2018.68. URL `http://cgm.cs.mcgill.ca/~avis/Kyoto/courses/te/abs_examples.pdf`.

[20] Walter Ludwig. A Subexponential Randomized Algorithm for the Simple Stochastic Game Problem. *Information and Computation*, 117(1):151–155, 2 1995. ISSN 0890-5401. doi:10.1006/INCO.1995.1035.

[21] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica 1996 16:4*, 16(4):498–516, 1996. ISSN 1432-0541. doi:10.1007/BF01940877. URL `https://link.springer.com/article/10.1007/BF01940877`.

[22] Anuj Puri. Theory of hybrid systems and discrete event systems., 1995. URL `https://www.elibrary.ru/item.asp?id=5408583`.

[23] Sven Schewe. An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games. *Computer Science Logic*, pages 369–384, 8 2008. doi:10.1007/978-3-540-87531-4_27. URL `https://link.springer.com/chapter/10.1007/978-3-540-87531-4_27`.

[24] Sven Schewe. From parity and payoff games to linear programming. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5734 LNCS:675–686, 2009. ISSN 03029743. doi:10.1007/978-3-642-03816-7_57/COVER/. URL `https://link.springer.com/chapter/10.1007/978-3-642-03816-7_57`.

[25] Sven Schewe, Ashutosh Trivedi, and Thomas Varghese. Symmetric Strategy Improvement. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9135:388–400, 2015. ISSN 16113349. doi:10.1007/978-3-662-47666-6_31. URL https://link.springer.com/chapter/10.1007/978-3-662-47666-6_31.

[26] Sdo. Polyhedron of simplex algorithm in 3D, 2006. URL https://en.wikipedia.org/wiki/Simplex_algorithm#/media/File:Simplex-method-3-dimensions.png.

[27] Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer 1998 20:2*, 20(2):7–15, 1 2009. ISSN 03436993. doi:10.1007/BF03025291. URL https://link.springer.com/article/10.1007/BF03025291.

[28] Tom van Dijk. A Parity Game Tale of Two Counters. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 305:107–122, 7 2018. doi:10.4204/EPTCS.305.8. URL http://dx.doi.org/10.4204/EPTCS.305.8.

[29] Norman Zadeh. What is the worst case behavior of the simplex algorithm. 1980. URL https://books.google.com/books?hl=en&lr=&id=uuzf-Fy1Cj8C&oi=fnd&pg=PA131&dq=What+is+the+worst+case+behaviour+of+the+simplex+algorithm%3F&ots=XmZF9aFnEO&sig=3IlZmu1B86F53TKANRxKyyqWD-E.