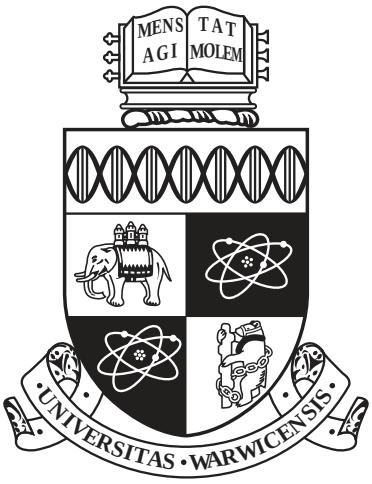

Strategy Iteration Algorithms for Games and Markov Decision Processes

by

John Fearnley

Thesis

Submitted to the University of Warwick

in partial fulfilment of the requirements

for the degree of

Doctor of Philosophy

Department of Computer Science

August 2010

THE UNIVERSITY OF
WARWICK

Contents

List of Figures

List of Tables

Acknowledgments

First, and foremost, I am indebted to my supervisor, Marcin Jurdziński. His insistence on mathematical rigour and clear thinking have been an inspiration to me, and the progress that I have made as a researcher can be attributed entirely to him. I do not think that any of the work in this thesis could have been completed without his influence.

I am also indebted to Ranko Lazić for acting as my advisor during my studies. He has never failed to provide sound advice and guidance during my time here. I would also like to thank Ranko and Kousha Etessami for acting as my examiners, and for their helpful comments that have greatly improved the final version of this thesis.

I am thankful to the Department of Computer Science at Warwick for hosting my six year journey from leaving high-school student to the submission of this thesis. In particular, I would like to thank the members of the DIMAP centre, the algorithms group, and the formal methods group, for providing a stimulating environment in which to do research.

Finally, I would like to thank my colleagues and friends: Haris Aziz, Peter Krusche, Michał Rutkowski, and Ashutosh Trivedi. My time at Warwick would have been a lot less enjoyable without their influence.

Declarations

The research presented in Chapter 4 is a joint work with Marcin Jurdziński and Rahul Savani. My co-authors contributed in the high-level ideas behind the algorithms, but the technical details, along with the lower bounds are my own work. All other work presented in this thesis has been produced by myself unless explicitly stated otherwise in the text.

None of the work presented in this thesis has been submitted for a previous degree, or a degree at another university. All of the work has been conducted during my period of study at Warwick. Preliminary versions of the work presented in this thesis have been published, or have been accepted for publication: Chapter 4 has been published in SOFSEM 2010 [FJS10], Chapter 5 has been published in ICALP 2010 [Fea10b], and Chapter 6 has been accepted for publication at LPAR-16 [Fea10a].

Abstract

In this thesis, we consider the problem of solving two player infinite games, such as parity games, mean-payoff games, and discounted games, the problem of solving Markov decision processes. We study a specific type of algorithm for solving these problems that we call strategy iteration algorithms. Strategy improvement algorithms are an example of a type of algorithm that falls under this classification.

We also study Lemke's algorithm and the Cottle-Dantzig algorithm, which are classical pivoting algorithms for solving the linear complementarity problem. The reduction of Jurdziński and Savani from discounted games to LCPs allows these algorithms to be applied to infinite games [JS08]. We show that, when they are applied to games, these algorithms can be viewed as strategy iteration algorithms. We also resolve the question of their running time on these games by providing a family of examples upon which these algorithm take exponential time.

Greedy strategy improvement is a natural variation of strategy improvement, and Friedmann has recently shown an exponential lower bound for this algorithm when it is applied to infinite games [Fri09]. However, these lower bounds do not apply for Markov decision processes. We extend Friedmann's work in order to prove an exponential lower bound for greedy strategy improvement in the MDP setting.

We also study variations on strategy improvement for infinite games. We show that there are structures in these games that current strategy improvement algorithms do not take advantage of. We also show that lower bounds given by Friedmann [Fri09], and those that are based on his work [FHZ10], work because they exploit this ignorance. We use our insight to design strategy improvement algorithms that avoid poor performance caused by the structures that these examples use.

Chapter 1

Introduction

In this thesis, we study the problem of solving Markov decision processes and two player infinite games played on finite graphs. In particular, we study parity, mean-payoff, and discounted games. We are interested in strategy iteration algorithms, which are a specific type of algorithm that can be used to solve these problems. In this chapter, we give an overview of the problems that we are considering, and the results that will be obtained in this thesis. A formal description of these problems will be given in Chapter 2, and a formal description of the algorithms that we study will be given in Chapter 3.

1.1 Markov Decision Processes

Markov decision processes were originally formulated in order to solve inventory management problems. In keeping with this tradition, we will illustrate this model with a simple inventory management problem. The following example is largely taken from the exposition of Puterman [Put05].

A manager owns a store that sells exactly one good. At the start of each month the manager must order stock, which arrives the following day. During the course of the month customers place orders, which are all shipped on the last

day of the month. Naturally, the manager cannot know how many orders will arrive during a given month, but the manager can use prior experience to give a probability distribution for this. The manager's problem is to decide how much stock should be ordered. Since storing goods is expensive, if too much stock is ordered then the profits on the goods that are sold may be wiped out. On the other hand, if not enough stock is ordered, then the store may not make as much profit as it could have done.

This problem can naturally be modelled as a Markov decision process. At the start of each month the inventory has some state, which is the number of goods that are currently in stock. The manager then takes an action, by deciding how many goods should be ordered. The inventory then moves to a new state, which is decided by a combination of the amount of goods that have been ordered, and the amount of orders that arrive during the month. Since the number of orders is modelled by a probability distribution, the state that the inventory will move to in the next month will be determined by this probability distribution. Each action has a reward, which is the expected profit from the goods that are sold minus the cost of storing the current inventory. If this reward is positive, then a profit is made during that month, and if it is negative, then a loss is made.

A strategy (also known as a policy in the Markov decision process literature) is a rule that the manager can follow to determine how much stock should be ordered. This strategy will obviously depend on the current state of the inventory: if the inventory is almost empty then the manager will want to make a larger order than if the inventory has plenty of stock.

The problem that must be solved is to find an *optimal* strategy. This is a strategy that maximizes the amount of profit that the store makes. The optimality criterion that is used depends on the setting. If the store is only going to be open for a fixed number of months, then we are looking for an optimal strategy in the total-reward criterion. This is a strategy that maximizes the total amount of profit

that is made while the store is open. On the other hand, if the store will remain open indefinitely, then we are looking for an optimal strategy in the average-reward criterion. This is a strategy that maximizes the long-term average profit that is made by the store. For example, the strategy may make a loss for the first few months in order to move the inventory into a state where a better long-term average profit can be obtained. Finally, there is the discounted-reward criterion, in which immediate rewards are worth more than future rewards. In our example, this could be interpreted as the price of the good slowly falling over time. This means that the profit in the first month will be larger than the profits in subsequent months.

The study of Markov decision processes began in the 1950s. The first work on this subject was by Shapley, who studied two player stochastic games [Sha53], and Markov decision processes can be seen as a stochastic game with only one player. The model was then developed in its own right by the works of Bellman [Bel57], Howard [How60], and others. Puterman's book provides a comprehensive modern exposition of the basic results for Markov decision processes [Put05].

Markov decision processes have since been applied in a wide variety of areas. They have become a standard tool for modelling stochastic processes in operations research and engineering. They have also found applications in computer science. For example, they are a fundamental tool used in reinforcement learning, which is a sub-area of artificial intelligence research [SB98].

These applications often produce Markov decision processes that have very large state spaces, and so finding fast algorithms that solve Markov decision processes is an important problem. It has long been known that the problem can be formulated as a linear program [d'E63, Man60], and therefore it can be solved in polynomial time [Kha80, Kar84]. However, this gives only a weakly polynomial time algorithm, which means that its running time is polynomial in the bit length of the numbers occurring in its inputs. There is no known strongly polynomial time algorithm for solving Markov decision processes.

1.2 Two Player Games

The defining text on game theory was written by von Neumann and Morgenstern in 1944 [vM44]. Part of this work was the study of two player zero-sum games, which are games where the goals of the two players are directly opposed. The zero-sum property means that if one player wins some amount, then the opposing player loses that amount. A classical example of this would be a poker game, where if one player wins some amount of money in a round, then his opponents lose that amount of money. These are the type of game that will be studied in this thesis.

Game theory can be applied to to produce a different type of model than those models that arise from Markov decision processes. In many systems it is known that there can be more than one outcome after taking some action, but there may not be a known probability distribution that models this uncertainty about the environment. Game theory can be used to model this, by assuming that the decisions taken by the environment are controlled by an adversary whose goal is to minimize our objective function.

For example, let us return to our inventory control problem. To model this as a Markov decision process, we required a probability distribution that specified the number of orders that will be placed by customers in each month. Now suppose that we do not have such a probability distribution. Instead, we are looking for a strategy for inventory control which maximizes the profit no matter how many orders arrive during each month. To do this, we can assume that the number of orders that are placed is controlled by an adversary, whose objective is to minimize the amount of profit that we make. If we can devise a strategy that ensures a certain amount of profit when playing against the adversary, then this strategy guarantees that we will make at least this amount of profit when exposed to real customers. From this, we can see that playing a game against the adversary allows us to devise a strategy that works in the worst case.

We will study two player infinite games that are played on finite graphs. In particular, we will study mean-payoff games and discounted games. These are similar to Markov decision processes with the average-reward and discounted-reward optimality criteria, but where the randomness used in the Markov decision process model is replaced with an opposing player. The objective in these games is again to compute an optimal strategy, but in this case an optimal strategy is one that guarantees a certain payoff no matter how the opponent plays against that strategy. These games have applications in, for example, online scheduling and online string comparison problems [ZP96].

Whereas the problem of solving a Markov decision process can be solved in polynomial time, there is no known algorithm that solves mean-payoff or discounted games in polynomial time. However, it is known that these games lie in the complexity class NP $\cap$ co-NP [KL93]. This implies that the problems are highly unlikely to be NP-complete or co-NP-complete, since a proof of either of these properties would imply that NP is equal to co-NP.

In fact, it is known that these three problems lie in a more restricted class of problems. A common formulation for the complexity class NP is that it contains decision problems for which, if the answer is yes, then then there is a witness of this fact that can be verified in polynomial time. The complexity class UP contains decision problems for which, if the answer is yes, then there is a *unique* witness of this fact that can be verified in polynomial time. The complexity class co-UP is defined analogously. Jurdziński has shown that the problem of solving these games lies in UP $\cap$ co-UP [Jur98].

The complexity status of these games is rather unusual, because they are one of the few natural combinatorial problems that lie in UP $\cap$ co-UP for which no polynomial time algorithm is known. This complexity class is also inhabited by various number theoretic problems such as deciding whether an integer has a factor that is larger than a given bound, which is assumed to be hard problem. However,

there are few combinatorial problems that share this complexity.

Membership of NP ∩ co-NP does not imply that a problem is hard. There have been problems in this class for which polynomial time algorithms have been devised. For example, the problem of primality testing was known to lie in NP ∩ co-NP, and was considered to be a hard problem. Nevertheless, Agrawal, Kayal, and Saxena discovered a polynomial time algorithm that solves this problem [AKS04]. Another example is linear programming, which was shown to be solvable in polynomial time by Khachiyan [Kha80]. Finding a polynomial time algorithm that solves a mean-payoff game or a discounted game is a major open problem.

1.3 Model Checking And Parity Games

The field of formal verification studies methods that can be used to check whether a computer program is correct. The applications of this are obvious, because real world programs often contain bugs that can cause the system to behave in unpredictable ways. On the other hand, software is often entrusted with safety critical tasks, such as flying an aeroplane or controlling a nuclear power plant. The goal of verification is to provide tools that can be used to validate that these systems are correct, and that they do not contain bugs.

Model checking is one technique that can be used to verify systems [CGP99]. The problem takes two inputs. The first is a representation of a computer program, usually in the form of a Kripke structure. This structure represents the states that the program can be in, and the transitions between these states that the program can make. The second input is a formula, which is written in some kind of logic. The model checking problem is to decide whether the system satisfies the formula.

This problem can be rephrased as a problem about a zero-sum game between two players. This game is played on a graph, which is constructed from the system and the formula. One player is trying to prove that the systems satisfies the property

described by the formula, and the other player is trying to prove the opposite. Therefore, a model checking problem can be solved by deciding which player wins the corresponding game.

The type of game that is played depends on the logic that is used to specify the formula. The modal μ-calculus is a logic that subsumes other commonly used temporal logics, such as LTL and CTL* [Koz82]. When the input formula is written in this logic, the corresponding model checking game will be a parity game [EJS93, Sti95]. Therefore, fast algorithms for solving parity games lead to faster model checkers for the modal μ-calculus. Parity games also have other applications in, for example, checking non-emptiness of non-deterministic parity tree automata [GTW02].

Parity games are strongly related to mean-payoff games and discounted games. There is a polynomial time reduction from parity games to mean-payoff games, and there is a polynomial time reduction from mean-payoff games to discounted games. Parity games are also known to be contained in UP $\cap$ co-UP, and no polynomial time algorithm is known that solves parity games. Finding a polynomial time algorithm that solves parity games is a major open problem, and this also further motivates the study of mean-payoff and discounted games.

1.4 Strategy Improvement

Strategy improvement is a technique that can be applied to solve Markov decision processes and infinite games. Strategy improvement algorithms have been devised for all three optimality criteria in MDPs [Put05], and for all of the games that we have introduced [Pur95, BV07, VJ00]. Strategy improvement algorithms fall under the umbrella of strategy iteration algorithms, because they solve a game or MDP by iteratively trying to improve a strategy. We will introduce the ideas behind strategy improvement by drawing an analogy with simplex methods for linear programming.

The problem of solving a linear program can naturally be seen as the problem of finding a vertex of a convex polytope that maximizes an objective function. The simplex method, given by Dantzig [WD49, Dan49], takes the following approach to finding this vertex. Each vertex of the polytope has a set of neighbours, and since the polytope is convex, we know that a vertex optimizes the objective function if and only if it has no neighbour with a higher value. Therefore, every vertex that is not optimal has at least one neighbour with a higher value. The simplex method begins by considering an arbitrary vertex of the polytope. In each iteration, it chooses some neighbouring vertex with a higher value, and moves to that vertex. This process continues until an optimal vertex is found.

Since there can be multiple neighbours which offer an improvement in the objective function, the simplex method must use a pivot rule to decide which neighbour should be chosen in each iteration. Dantzig's original pivot rule was shown to require exponential time in the worst case by a result of Klee and Minty [KM70]. It is now known that many other pivot rules have exponential running times in the worst case, and there is no pivot rule that has a proof of polynomial time termination. On the other hand, the simplex method has been found to work very well in practice because it almost always terminates in polynomial time on real world examples. For this reason, the simplex method is still frequently used to solve practical problems.

Strategy improvement uses similar techniques to those of the simplex algorithm in order to solve MDPs and games. For each strategy, the set of neighbouring strategies that secure a better value can be computed in polynomial time, and it can be shown that a strategy is optimal if and only if this set is empty. Therefore, strategy improvement algorithms begin with an arbitrary strategy, and in each iteration the algorithm picks some neighbouring strategy that secures a better value.

Strategy improvement algorithms also need a pivot rule to decide which neighbouring strategy should be chosen. In the context of strategy improvement algorithms, pivot rules are called *switching policies*. As with linear programming,

it has been shown that using an unsophisticated switching policy can cause the algorithm to take exponential time [MC94]. However, no exponential lower bounds were known for more sophisticated switching policies. In particular, the greedy policy is a natural switching policy for strategy improvement, and it was considered to be a strong candidate for polynomial time termination for quite some time. This was because, as in the case of linear programming, the greedy switching policy was found to work very well in practice. However, these hopes were dashed by a recent result of Friedmann [Fri09], in which he constructed a family of parity games upon which the strategy improvement algorithm of Vöge and Jurdziński equipped with the greedy switching policy takes exponential time. These results have been adapted to show exponential lower bounds for greedy strategy improvement algorithms for mean-payoff and discounted games [And09].

1.5 The Linear Complementarity Problem

The linear complementarity problem is a fundamental problem in mathematical programming [CPS92], which naturally captures the complementary slackness conditions in linear programming and the Karush-Kuhn-Tucker conditions of quadratic programs. The linear complementarity problem takes a matrix as an input. In general, the problem of finding a solution to a linear complementarity problem is NP-complete [Chu89]. However, when the input matrix is a P-matrix, the problem is known to belong to the more restrictive class PPAD [Pap94]. This class is known to have complete problems, such as the problem of finding a Nash equilibrium in a bimatrix game [DGP06, CDT09], but it is not known whether the P-matrix linear complementarity problem is PPAD-complete. On the other hand, there is no known polynomial time algorithm that solves P-matrix LCPs, and finding such an algorithm is a major open problem.

The simplex method for linear programming can be called a pivoting algo-

rithm, because it uses pivotal algebra to move from one vertex of the polytope to the next. Pivoting algorithms have also been developed to solve the linear complementarity problem. Prominent examples of this are Lemke's algorithm [Lem65], and the Cottle-Dantzig algorithm [DC67]. Although these algorithms may fail to solve an arbitrary LCP, both of these algorithms are guaranteed to terminate with a solution when they are applied to a P-matrix LCP.

Recently, the infinite games that we are studying have been linked to the linear complementarity problem. Polynomial time reductions from simple stochastic games [Con93] to the linear complementarity problem have been proposed [GR05, SV07]. Simple stochastic games are a type of game that also incorporate random moves, and it has been shown that the problem of solving a discounted game can be reduced to the problem of solving a simple stochastic game [ZP96]. Also, a direct polynomial time reduction from discounted games to the P-matrix linear complementarity problem has been devised by Jurdziński and Savani [JS08]. These reductions mean that the classical pivoting algorithm from the LCP literature can now be applied to solve parity, mean-payoff, and discounted games.

1.6 Contribution

In Chapter 4, we study how the pivoting algorithms for the linear complementarity problem behave when they are applied to the LCPs that arise from the reduction of Jurdziński and Savani. We show that Lemke's algorithm and the Cottle-Dantzig algorithm can be viewed as strategy iteration algorithms, and we present versions of these algorithms that work directly on discounted games, rather than on the LCP reductions of these games. This allows researchers who do not have a background in mathematical optimization, but who are interested in solving games, to understand how these algorithms work.

Although there are exponential lower bounds for these algorithm when they

are applied to P-matrix LCPs [Mur78, Fat79], it was not known whether these bounds hold for the LCPs that arise from games. It is possible that the class of LCPs generated by games may form a subclass of P-matrix LCPs that is easier to solve. We show that this is not the case, by providing a family of parity games upon which both Lemke's algorithm and the Cottle-Dantzig algorithm take exponential time. Since parity games lie at the top of the chain of reductions from games to the linear complementarity problem, this lower bound also holds for mean-payoff and discounted games. The work in this chapter is joint work with Marcin Jurdziński and Rahul Savani, and it is a revised and extended version of a paper that was published in the proceedings of SOFSEM 2010 [FJS10].

In Chapter 5, we study strategy improvement algorithms for Markov decision processes. The greedy switching policy was a promising candidate for polynomial time termination in both game and MDP settings. For parity games, the result of Friedmann [Fri09] proved that this was not the case. However, while Friedmann's examples have been adapted to cover mean-payoff and discounted games, no exponential lower bounds have been found for greedy strategy improvement for Markov decision processes. We resolve this situation by adapting Friedmann's examples to provide a family of Markov decision processes upon which greedy strategy improvement takes an exponential number of steps. This lower bound holds for the average-reward criterion. The work presented in this chapter is a revised and extended version of a paper that was published in the proceedings of ICALP 2010 [Fea10a].

In Chapter 6, we study strategy improvement algorithms for parity games and mean-payoff games. With the recent result of Friedmann, doubts have been cast about whether strategy improvement equipped with any switching policy could terminate in polynomial time. This view is reinforced by a recent result of Friedmann, Hansen, and Zwick [FHZ10], who showed that random facet, another prominent switching policy, is not polynomial.

Most previous switching policies that have been proposed for strategy im-

provement follow simple rules, and do not take into account the structure of the game that they are solving. In this chapter, we show that switching policies can use the structure of the game to make better decisions. We show that certain structures in parity and mean-payoff games are strongly related with the behaviour of strategy improvement algorithms. Moreover, we show that switching policies can exploit this link to make better decisions. We propose an augmentation scheme, which allows traditional switching policies, such as the greedy policy, to take advantage of this knowledge. Finally, we show that these ideas are also exploited by the recent super-polynomial lower bounds, by showing that the augmented version of the greedy policy is polynomial on Friedmann's examples, and that the augmented version of random facet is polynomial on the examples of Friedmann, Hansen, and Zwick. The work in this chapter is a revised and extended version of a paper that will appear in the proceedings of LPAR-16 [Fea10b].

Chapter 2

Problem Statements

In this section we introduce and formally define the problems that will be studied in this thesis. There are three types of problem that we will introduce: two player infinite games, Markov decision processes, and the linear complementarity problem. We will then see how these problems are related to each other, by the know polynomial time reductions between them.

2.1 Infinite Games

In this section we will introduce three different types of infinite game: parity games, mean-payoff games, and discounted games. All of these games are played on finite graphs, in which the set of vertices has been partitioned between two players. The game is played by placing a token on a starting vertex. In each step of the game, the player that owns the vertex upon which the token is placed must move the token along one of the outgoing edges from that vertex. In this fashion, the two players construct an infinite path, and the winner of the game can be determined from the properties of this path.

Parity games are an example of a *qualitative* game, where one player wins and the other player loses. In the setting of infinite games on finite graphs, a qualitative

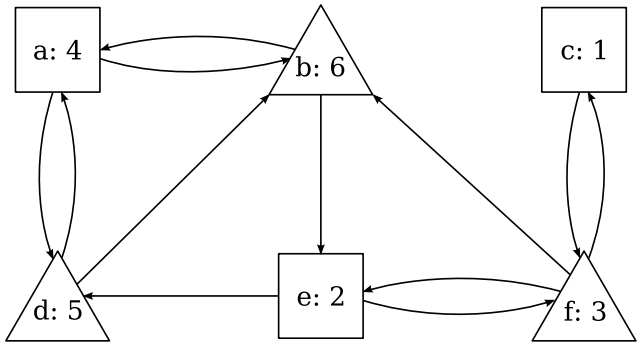

Figure 2.1: An example of a parity game.

winning condition means that one player wins the game if infinite path satisfies some property, while the other player wins if the infinite path does not satisfy that property.

Mean-payoff and discounted games are examples of *quantitative* games. In these games, every infinite path is assigned a payoff, which is a real number. The two players are usually referred to as Max and Min, and Min pays the payoff of the infinite path to Max. Therefore, the two players have opposite goals: it is in the interest of Max to maximize the payoff of the infinite path, and it is in the interest of Min to minimize the payoff of the infinite path.

2.1.1 Parity Games

A parity game is played by two players, called Even and Odd, on a finite graph. Each vertex in the graph is assigned a natural number, which is called the *priority* of the vertex. Formally, a parity game is defined by a tuple $(V, V_{\text{Even}}, V_{\text{Odd}}, E, \text{pri})$, where V is a set of vertices and E is a set of edges, which together form a finite graph. We assume that every vertex in the graph has at least one outgoing edge. The sets V_{Even} and V_{Odd} partition V into vertices belonging to player Even and vertices belonging to player Odd, respectively. The function $\text{pri} : V \to \mathbb{N}$ assigns a priority to each vertex.

Figure 2.1 gives an example of a parity game. Whenever we draw a parity game, we will represent the vertices belonging to player Even as boxes, and the vertices belonging to player Odd as triangles. Each vertex is labeled by a name followed by the priority that is assigned to the vertex.

The game begins by a token being placed on a starting vertex v_0. In each step, the player that owns the vertex upon which the token is placed must choose one outgoing edge from that vertex and move the token along it. In this fashion, the two players form an infinite path $\pi = \langle v_0, v_1, v_2, \dots \rangle$, where $(v_i, v_{i+1}) \in E$ for every $i \in \mathbb{N}$. To determine the winner of the game, we consider the set of priorities that occur *infinitely often* along the path. This is defined to be:

$$\mathrm{Inf}(\pi) = \{d \in \mathbb{N} \ : \ \text{For all } j \in \mathbb{N} \text{ there is an } i > j \text{ such that } \mathrm{pri}(v_i) = d\}.$$

Player Even wins the game if the largest priority occurring infinitely often is even, and player Odd wins the game if the largest priority occurring infinitely often is odd. In other words, player Even wins the game if and only if $\max(\mathrm{Inf}(\pi))$ is even.

In Figure 2.1, the players could construct the path $\langle a, b, e \rangle$ followed by $\langle f, c \rangle^\omega$. The set of priorities that occur infinitely often along this path is $\{1, 3\}$. Since 3 is odd, we know that player Odd would win the game if the two players constructed this path.

A strategy is a function that a player uses to make decisions while playing a parity game. Suppose that the token has just arrived at a vertex $v \in V_{\mathrm{Even}}$. Player Even would use his strategy to decide which outgoing edge the token should be moved along. Player Even's strategy could make its decisions using the entire history of the token, which is the path between the starting vertex and v that the players have formed so far. The strategy could also use randomization to assign a probability distribution over which outgoing edge should be picked. However, we will focus on the class of *positional strategies*. Positional strategies do not use the

history of the token, and they do make probabilistic decisions.

A positional strategy for Even is a function that chooses one outgoing edge for every vertex in V_{Even}. A strategy is denoted by $\sigma : V_{\text{Even}} \to V$, with the condition that $(v, \sigma(v))$ is in E, for every Even vertex v. Positional strategies for player Odd are defined analogously. The sets of positional strategies for Even and Odd are denoted by Π_{Even} and Π_{Odd}, respectively. Given two positional strategies σ and τ, for Even and Odd respectively, and a starting vertex v_0, there is a unique path $\langle v_0, v_1, v_2 \dots \rangle$, where $v_{i+1} = \sigma(v_i)$ if v_i is owned by Even and $v_{i+1} = \tau(v_i)$ if v_i is owned by Odd. This path is known as the *play* induced by the two strategies σ and τ, and will be denoted by $\text{Play}(v_0, \sigma, \tau)$.

An Even strategy is called a *winning strategy* from a given starting vertex if player Even can use the strategy to ensure a win when the game is started at that vertex, no matter how player Odd plays in response. We define $\text{Paths}_{\text{Even}} : V \times \Pi_{\text{Even}} \to 2^{V^\omega}$ to be a function that gives every path that starts at a given vertex and is consistent with some Even strategy. If σ is a positional strategy for Even, and v_0 is a starting vertex then:

$$\text{Paths}_{\text{Even}}(v_0, \sigma) = \{\langle v_0, v_1, \dots \rangle \in V^\omega \ : \ \text{for all } i \in \mathbb{N}, \text{ if } v_i \in V_{\text{Even}} \text{ then } v_{i+1} = \sigma(v_i), \text{ and if } v_i \in V_{\text{Odd}} \text{ then } (v_i, v_{i+1}) \in E)\}.$$

A strategy σ is a winning strategy for player Even from the starting vertex v_0 if $\max(\text{Inf}(\pi))$ is even for every path $\pi \in \text{Paths}_{\text{Even}}(v_0, \sigma)$. The strategy σ is said to be winning for a set of vertices $W \subseteq V$ if it is winning for every vertex $v \in W$. Winning strategies for player Odd are defined analogously.

A game is said to be *determined* if one of the two players has a winning strategy. We now give a fundamental theorem, which states that parity games are determined with positional strategies.

Theorem 2.1 ([EJ91, Mos91])**.** *In every parity game, the set of vertices V can*

be partitioned into two sets (W_{Even}, W_{Odd})*, where Even has a positional winning strategy for* W_{Even}*, and Odd has a positional winning strategy for* W_{Odd}*.*

There are two problems that we are interested in solving for parity games. Firstly, there is the problem of computing the *winning sets* $(W_{\text{Even}}, W_{\text{Odd}})$ whose existence is implied by Theorem 2.1. Secondly, there is the more difficult problem of computing the partition into winning sets, and to provide a strategy for Even that is winning for W_{Even}, and a strategy for Odd that is winning for W_{Odd}.

In the example shown in Figure 2.1, the Even strategy $\{a \mapsto b, e \mapsto d, c \mapsto f\}$ is a winning strategy for the set of vertices $\{a, b, d, e\}$. This is because player Odd cannot avoid seeing the priority 6 infinitely often when Even plays this strategy. On the other hand, the strategy $\{d \mapsto a, b \mapsto a, f \mapsto c\}$ is a winning strategy for Odd for the set of vertices $\{c, f\}$. Therefore, the partition into winning sets for this game is $(\{a, b, d, e\}, \{c, f\})$.

2.1.2 Mean-Payoff Games

A mean-payoff game is similar in structure to a parity game. The game is played by player Max and player Min, who move a token around a finite graph. Instead of priorities, each vertex in this graph is assigned an integer reward. Formally, a mean-payoff game is defined by a tuple $(V, V_{\text{Max}}, V_{\text{Min}}, E, \mathrm{r})$ where V and E form a finite graph. Once again, we assume that every vertex must have at least one outgoing edge. The sets V_{Max} and V_{Min} partition V into vertices belonging to player Max and vertices belonging to player Min, respectively. The function $\mathrm{r} : V \to \mathbb{Z}$ assigns an integer reward to every vertex.

Once again, the game begins at a starting vertex v_0, and the two players construct an infinite path $\langle v_0, v_1, v_2, \dots \rangle$. The payoff of this infinite path is the average reward that is achieved in each step. To capture this, we define $\mathcal{M}(\pi) = \liminf_{n\to\infty} (1/n) \sum_{i=0}^{n} \mathrm{r}(v_i)$. The objective of player Max is to maximize the value of $\mathcal{M}(\pi)$, and the objective of player Min is to minimize it.

The definitions of positional strategies and plays carry over directly from the definitions that were given for parity games. The functions $\text{Paths}_{\text{Max}}$ and $\text{Paths}_{\text{Min}}$ can be defined in a similar way to the function $\text{Paths}_{\text{Even}}$ that was used for parity games. We now define two important concepts, which are known as the *lower* and the *upper* values. These will be denoted as Value_* and Value^*, respectively. For every vertex v we define:

$$\text{Value}_*(v) = \max_{\sigma \in \Pi_{\text{Max}}} \min_{\pi \in \text{Paths}_{\text{Max}}(v,\sigma)} \mathcal{M}(\pi),$$

$$\text{Value}^*(v) = \min_{\tau \in \Pi_{\text{Min}}} \max_{\pi \in \text{Paths}_{\text{Min}}(v,\tau)} \mathcal{M}(\pi).$$

The lower value of a vertex v is the largest payoff that Max can obtain with a positional strategy when the game is started at v, and the upper value of v gives the smallest payoff that Min can obtain with a positional strategy when the game is started at v. It is not difficult to prove that, for every vertex v, the inequality $\text{Value}_*(v) \leq \text{Value}^*(v)$ always holds. However, for mean-payoff games we have that the two quantities are equal, which implies that the games are determined.

Theorem 2.2 (**[LL69]**). *For every starting vertex v in every mean-payoff game we have* $\text{Value}_*(v) = \text{Value}^*(v)$.

This theorem implies that if a player can secure a certain payoff from a given vertex, then that player can also secure that payoff using a positional strategy. Once again, this implies that we only need to consider positional strategies when working with mean-payoff games.

We denote the *value* of the game starting at the vertex v as $\text{Value}(v)$, and we define it to be equal to both $\text{Value}_*(v)$ and $\text{Value}^*(v)$. A Max strategy σ is *optimal* for a vertex v if, when that strategy is used for a game starting at v, it ensures a payoff that is at least the value of the game at v. In other words, as strategy σ is optimal for a vertex v if $\min_{\pi \in \text{Paths}_{\text{Min}}(v,\sigma)} \mathcal{M}(\pi) = \text{Value}(v)$. A Max strategy σ is optimal for the game if it is optimal for every vertex in that game. Optimal

strategies for Min are defined analogously.

There are several problems that we are interested in for mean-payoff games. Firstly, we have the problem of computing the value of the game for every vertex. Another problem is to compute an optimal strategy for the game for both players. We can also solve mean-payoff games qualitatively. In this setting, Max wins if and only if the payoff of the infinite path is strictly greater than 0. In this setting, a Max strategy σ is winning from a vertex v if $\min_{\pi \in \text{Paths}_{\text{Min}}(v,\sigma)} \mathcal{M}(\pi) > 0$, and a Min strategy τ is winning from the vertex v if $\max_{\pi \in \text{Paths}_{\text{Max}}(v,\tau)} \mathcal{M}(\pi) \leq 0$. A strategy is said to be a winning strategy for a set of vertices W if it is a winning strategy for every vertex $v \in W$.

The computational problem associated with the qualitative version of mean-payoff games is sometimes called the *zero-mean partition* problem. To solve this we must compute the partition $(W_{\text{Max}}, W_{\text{Min}})$ of the set V, where Max has a winning strategy for W_{Max}, and where Min has a winning strategy for W_{Min}. An efficient algorithm for the zero-mean partition problem can be used to solve the quantitative version of the mean-payoff game efficiently: Björklund and Vorobyov have shown that only a polynomial number of calls to an algorithm for finding the zero-mean partition are needed to find the value for every vertex in a mean-payoff game [BV07].

Figure 2.2 shows an example of a mean-payoff game. Every time that we draw a mean-payoff game, we represent the vertices belonging to player Max as boxes, and the vertices belonging to player Min as triangles. One possible play in this game would be $\langle a, b, c, d \rangle^\omega$, and the payoff of this play is -6.25. An optimal strategy for player Max is $\{a \mapsto b, c \mapsto e, e \mapsto c\}$, and an optimal strategy for player Min is $\{b \mapsto a, d \mapsto a\}$. The value of the game for the vertices in the set $\{a, b, d\}$ is the average weight on the cycle formed by a and b, which is -3.5, and the value of the game for the vertices in the set $\{c, e\}$ is the average weight on the cycle formed by c and e, which is 0.5. The zero-mean partition for this mean-payoff game is therefore $(\{c, e\}, \{a, b, d\})$.

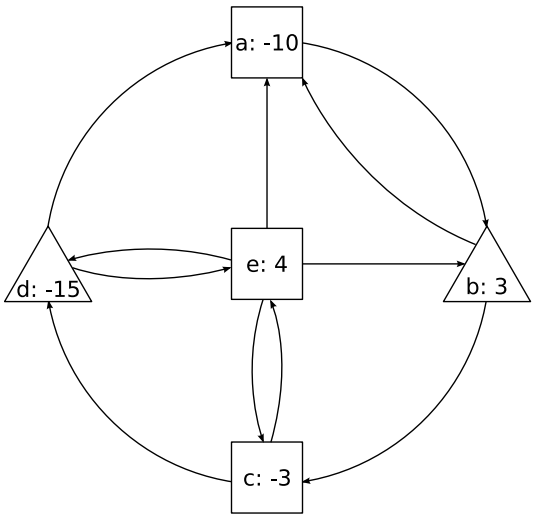

Figure 2.2: An example of a mean-payoff game.

2.1.3 Discounted Games

Discounted games are very similar to mean-payoff games, but with a different definition of the payoff of an infinite path. Formally, a discounted game is defined by a tuple $(V, V_{\text{Max}}, V_{\text{Min}}, E, \mathrm{r}, \beta)$, where the first five components are exactly the same as the definitions given for mean-payoff games. In addition to these, there is also a *discount factor* β, which is a rational number chosen so that $0 \leq \beta < 1$.

As usual, the game begins at a starting vertex v_0, and the two players construct an infinite path $\langle v_0, v_1, v_2, \dots \rangle$. In a discounted game, the payoff of an infinite path is the sum of the rewards on that path. However, for each step that the path takes, the rewards in the game are discounted by a factor of β. Formally, the payoff of an infinite path is defined to be $\mathcal{D}(\pi) = \sum_{i=0}^{\infty} \beta^i \, \mathrm{r}(v_i)$. For example, if we took the game shown in Figure 2.2 to be a discounted game with discount factor 0.5, then the payoff of the path $\langle e, c \rangle^\omega$ would be:

$$4 + (0.5 \times -3) + (0.5^2 \times 4) + (0.5^3 \times -3) + \cdots = \frac{10}{3}.$$

With this definition in hand, we can again define the lower and upper values:

$$\text{Value}_*(v) = \max_{\sigma \in \Pi_{\text{Max}}} \min_{\pi \in \text{Paths}_{\text{Max}}(v,\sigma)} \mathcal{D}(\pi),$$

$$\text{Value}^*(v) = \min_{\tau \in \Pi_{\text{Min}}} \max_{\pi \in \text{Paths}_{\text{Min}}(v,\tau)} \mathcal{D}(\pi).$$

The analogue of Theorem 2.2 was shown by Shapely.

Theorem 2.3 ([**Sha53**])**.** *For every starting vertex v in every discounted game we have* $\text{Value}_*(v) = \text{Value}^*(v)$.

Therefore, we define $\text{Value}(v)$ to be equal to both $\text{Value}_*(v)$ and $\text{Value}^*(v)$, and the definitions of optimal strategies carry over from those that were given for mean-payoff games.

Some results in the literature, such as the reduction from discounted games to the linear complementarity problem presented in Section 2.5.2, require a different definition of a discounted game. In particular, the reduction considers *binary* discounted games that have rewards placed on edges. A binary discounted game is a discounted game in which every vertex has exactly two outgoing edges.

We introduce notation for these games. A binary discounted game with rewards placed on edges is a tuple $G = (V, V_{\text{Max}}, V_{\text{Min}}, \lambda, \rho, r^\lambda, r^\rho, \beta)$, where the set V is the set of vertices, and V_{Max} and V_{Min} partition V into the set of vertices of player Max and the set of vertices of player Min, respectively. Each vertex has exactly two outgoing edges which are given by the left and right successor functions $\lambda, \rho : V \to V$. Each edge has a reward associated with it, which is given by the functions $r^\lambda, r^\rho : V \to \mathbb{Z}$. Finally, the discount factor β is such that $0 \leq \beta < 1$.

All of the concepts that we have described for games with rewards assigned to vertices carry over for games with rewards assigned to edges. The only difference that must be accounted for is the definition of the payoff function. When the rewards are placed on edges, the two players construct an infinite path $\pi = \langle v_0, v_1, v_2, \dots \rangle$ where v_{i+1} is equal to either $\lambda(v_i)$ or $\rho(v_i)$. This path yields the infinite sequence of

rewards $\langle r_0, r_1, r_2, \dots \rangle$, where $r_i = r^\lambda(v_i)$ if $\lambda(v_i) = v_{i+1}$, and $r_i = r^\rho(v_i)$ otherwise. The payoff of the infinite path is then $\mathcal{D}(\pi) = \sum_{i=0}^{\infty} \beta^i r^i$.

At first sight, it is not clear how the two models relate to each other. Allowing rewards to be placed on edges seems to be less restrictive that placing them on vertices, but forcing each vertex to have exactly two outgoing edges seems more restrictive. We will resolve this by showing how a traditional discounted game can be reduced to a binary discounted game with rewards placed on edges.

We will first show that every discounted game can be reduced to a discounted game in which every vertex has exactly two outgoing edges. This will be accomplished by replacing each vertex with a binary tree. Let k be the largest out degree for a vertex in the discounted game. Every vertex in the game will be replaced with a full binary tree of depth $\lceil \log_2 k \rceil - 1$, where each vertex in the tree is owned by the player that owns v, and each leaf of the tree has two outgoing edges. The root of the tree is assigned the reward $\mathrm{r}(v)$, and every other vertex in the tree is assigned reward 0.

For each incoming edge (u, v) in the original game, we add an edge from u to the root of the binary tree in the reduced game. For each outgoing edge (v, u) in the original game, we set the successor of some leaf to be u. If, after this procedure, there is a leaf w that has less than two outgoing edges, then we select some edge (v, u) from the original game, and set u as the destination for the remaining outgoing edges of w. It is not a problem if w has two outgoing edges to u after this procedure, because we use left and right successor functions in our definition of a binary discounted game, rather than an edge relation. We then set the discount factor for the binary game to be $\sqrt[l]{\beta}$, where $l = \lceil \log_2 k \rceil$.

We now argue that this construction is correct. Clearly, a player can move from a vertex v to a vertex u in the discounted game if and only if that player can move from v to u in our binary version of that game. Therefore, the players can construct an infinite path $\langle v_0, v_1, \dots \rangle$ in the original game if and only if they can

construct an infinite path $\langle v_0, v_{0,1}, v_{0,2}, \ldots, v_{0,\lceil \log_2 k \rceil - 1}, v_1, \ldots \rangle$ in the binary version of that game, where each vertex v_i is the root of a binary tree, and each vertex $v_{i,j}$ is an internal vertex in the binary tree. To see that the construction is correct, it suffices to notice that these two paths have the same payoff. The payoff of the first path is $\mathrm{r}(v_0) + \beta \cdot \mathrm{r}(v_1) + \ldots$, and the payoff of the second path is:

$$\mathrm{r}(v_0) + \sqrt[l]{\beta} \cdot \mathrm{r}(v_{0,1}) + \cdots + (\sqrt[l]{\beta})^{\lceil \log_2 k \rceil - 1} \cdot v_{0,\lceil \log_2 k \rceil - 1} \cdot \mathrm{r}(v_{0,2}) + (\sqrt[l]{\beta})^l \, \mathrm{r}(v_1) + \ldots$$

Since $\mathrm{r}(v_{0,i}) = 0$ for all i, and $(\sqrt[l]{\beta})^l = \beta$, the two paths must have the same payoff. From this, it follows that the two games must have the same value, and therefore solving the binary game gives a solution for the original game.

We now argue that a discounted game with rewards placed on vertices can be transformed into an equivalent game in which the rewards are placed on edges. The reduction in this case is simple: if a vertex v has reward r in the original game, then we set the reward of every outgoing edge of v to be r in the transformed game. This reduction obviously gives an equivalent game, because in both games you must see a reward of r if you pass through v.

In summary, an efficient algorithm that solves binary discounted games with rewards placed on edges can also be used to efficiently solve discounted games specified with the standard formulation. We will use this formulation when we describe the reduction from discounted games to the linear complementarity problem in Section 2.5.2, and in the work based on this reduction presented in Chapter 4.

2.2 Markov Decision Processes

Markov decision processes are similar to two player games, but with two important differences. Firstly, a Markov decision process does not have two players. This could be thought of as a game in which there is only one player. The second difference is that the moves made in a Markov decision process are not necessarily deterministic.

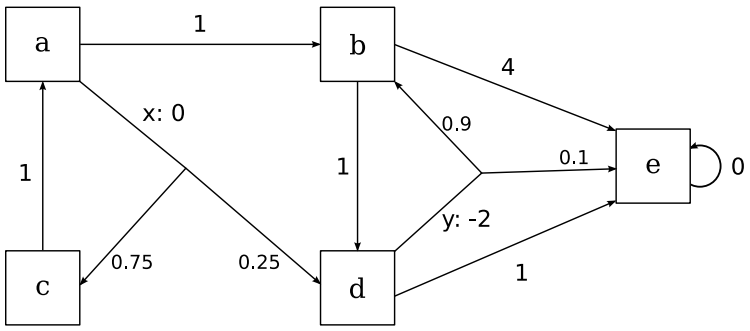

Figure 2.3: An example of a Markov decision process.

In the game setting, the vertices are connected by edges, and when the token is at some vertex, then the owner of that vertex can choose one of the edges and move the token along that edge. In the Markov decision process setting, the vertices are connected by *actions*, and when the token is at some vertex, an action must be chosen. The outcome of picking an action is not deterministic. Instead, each action has a probability distribution over the vertices in the game. The vertex that the token is moved to is determined by this probability distribution.

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

The MDPs that we will study actually contain a large number of *deterministic* actions. An action $a \in A_v$ is deterministic if there is some vertex v' such that $\mathrm{p}(v'|v, a) = 1$. For the sake of convenience, we introduce notation that makes working with these actions easier. We will denote a deterministic action from the vertex v to the vertex v' as (v, v'), and the function $\mathrm{r}(v, v')$ gives the reward of this action.

Figure 2.3 shows an example of a Markov decision process. When we draw

a Markov decision process, the vertices will be drawn as boxes, and the name of a vertex will be displayed on that vertex. Actions are drawn as arrows: deterministic actions are drawn as an arrow between vertices, and probabilistic actions are drawn as arrows that split, and end at multiple vertices. Each deterministic action is labelled with its reward. For probabilistic actions, the probability distribution is marked after the arrow has split, and both the name of the action and the reward of the action are marked before the arrow has split.

Strategies and runs for MDPs are defined in the same way as they are for games. A deterministic memoryless strategy $\sigma : V \to A_s$ is a function that selects one action at each vertex. If an action a is a deterministic action (v, u), then we adopt the notation $\sigma(v) = u$ for when σ chooses the action a. For a given starting vertex v_0, a run that is consistent with a strategy σ is an infinite sequence of vertices $\langle v_0, v_1, v_2, \dots \rangle$ such that $p(v_{i+1}|v_i, \sigma(v_i)) > 0$ for all i.

There could be uncountably many runs that are consistent with a given strategy. We must define a probability space over these runs. The set $\Omega^{\chi}_{v_0}$ contains every run that starts at the vertex v_0 and that is consistent with the strategy χ. In order to define a probability space over this set, we must provide a σ-algebra of measurable events. We define the cylinder set of a finite path to be the set that contains every infinite path which has the finite path as a prefix. Standard results from measure theory imply that there is a unique smallest σ-algebra $\mathcal{F}^{\chi}_{v_0}$ over $\Omega^{\chi}_{v_0}$ that contains every cylinder set. Furthermore, if we define the probability of the cylinder set for a finite path $\langle v_0, v_1, v_2, \dots v_k \rangle$ to be $\prod_{i=0}^{k-1} \mathrm{p}(v_{i+1}|v_i, \sigma(v_i))$, then this uniquely defines a probability measure $\mathbb{P}^{\chi}_{v_0}$ over the σ-algebra $\mathcal{F}^{\chi}_{v_0}$ [ADD00]. Therefore, our probability space will be $(\Omega^{\chi}_{v_0}, \mathcal{F}^{\chi}_{v_0}, \mathbb{P}^{\chi}_{v_0})$. Given a measurable function that assigns a value to each consistent run $f : \Omega \to \mathbb{R}$, we define $\mathbb{E}^{\chi}_{v_0}\{f\}$ to be the expectation of this function in the probability space.

A reward criterion assigns a payoff to each run. In the total-reward criterion, the payoff of a run is the sum of the rewards along that run. In the average-reward

criterion, the payoff of a run is the average reward that is obtained in each step along the infinite run. The *value* of a vertex under some strategy is the expectation of the payoff over the runs that are consistent with that strategy. Formally, the value of a vertex v in the strategy σ under the total-reward criterion is defined to be:

$$\text{Value}^{\sigma}(v) = \mathbb{E}_v^{\sigma}\{\sum_{i=0}^{\infty} \text{r}(v_i, v_{i+1})\}.$$

Under the average-reward criterion the value of each vertex is defined to be:

$$\text{Value}_{\mathcal{A}}^{\sigma}(v) = \mathbb{E}_v^{\sigma}\{\liminf_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N} \text{r}(v_i, v_{i+1})\}.$$

Finally, for the discounted reward criterion a discount factor β is chosen so that $0 \leq \beta < 1$, and the value of each vertex is defined to be:

$$\text{Value}_{\mathcal{D}}^{\sigma}(s) = \mathbb{E}_s^{\sigma}\{\sum_{i=0}^{\infty} \beta^i \cdot \text{r}(s_i, s_{i+1})\}.$$

For a given MDP, the computational objective is to find an optimal strategy σ^*, which is the strategy that maximizes the optimality criterion for every starting vertex: an optimal strategy will satisfy $\text{Value}^{\sigma}(v) \leq \text{Value}^{\sigma^*}(v)$ for every vertex v and every strategy σ. We define the value of a vertex to be the value that is obtained by an optimal strategy from that vertex. That is, we define $\text{Value}(v) = \text{Value}^{\sigma*}(v)$, we define $\text{Value}_{\mathcal{A}}(v) = \text{Value}_{\mathcal{A}}^{\sigma*}(v)$, and we define $\text{Value}_{\mathcal{D}}(v) = \text{Value}_{\mathcal{D}}^{\sigma*}(v)$, for every vertex v.

In the example shown in Figure 2.3 the strategy $\{a \mapsto x, b \mapsto e, c \mapsto a, d \mapsto y, e \mapsto e\}$ is an optimal strategy under the total reward criterion. The value of the vertex a under this strategy is 4.6. Under the average reward criterion, every vertex will have value 0, no matter what strategy is used. This is because we cannot avoid reaching the sink vertex e, and therefore the long term average-reward will always be 0.

2.3 The Linear Complementarity Problem

The linear complementarity problem is a fundamental problem in mathematical optimisation. Given an $n \times n$ matrix M and an n-dimensional vector q, the linear complementarity problem (LCP) is to find two n-dimensional vectors w and z that satisfy the following:

$$w = Mz + q, \tag{2.1}$$

$$w, z \geq 0, \tag{2.2}$$

$$z_i \cdot w_i = 0 \qquad \text{for } i = 1, 2, \ldots, n. \tag{2.3}$$

We will denote this problem as LCP(M, q). The condition given by (2.3) is called the *complementarity condition*, and it insists that for all i, either the i-th component of w is equal to 0 or the i-th component of z is equal to 0. It is for this reason that we refer to w_i and z_i as being complements of each other. From the complementarity condition combined with the non-negativity constraint given by (2.2) it is clear that every solution to the LCP contains exactly n non-negative variables and exactly n variables whose value is forced to be 0. In LCP terminology, the non-negative variables are called *basic* variables and the complements of the basic variables are called *non-basic*.

Before continuing, it must be noted that the system given by (2.1)-(2.3) may not have a solution or it may have many solutions. However, we are interested in a special case of the linear complementarity problem, where the matrix M is a P-matrix. For every set $\alpha \subseteq \{1, 2, \ldots n\}$ the *principal sub-matrix* of M associated with α is obtained by removing every column and every row whose index is not in α. A *principal minor* of M is the determinant of a principal sub-matrix of M.

Definition 2.4 (P-matrix)**.** *A matrix M is a P-matrix if and only if every principal minor of M is positive.*

In our work, we will always consider LCPs in which the matrix M is a P-matrix. LCPs of this form have been shown to have the following special property.

Theorem 2.5 ([CPS92]). *If the matrix M is a P-matrix then, for every n dimensional vector q, we have that* LCP(M, q) *has a unique solution.*

A fundamental operation that can be applied to an LCP is a *pivot* operation. This operation takes two variables w_i and z_j and produces an equivalent LCP in which the roles of these two variables are swapped. In this new LCP, the variable z_j will be the i-th variable in the vector w, and the variable w_i will be the j-th variable in the vector z. To perform a pivot step, we construct a tableaux $A = [I, M, q]$, that is, a matrix whose first n columns are the columns of the n-dimensional identity matrix, whose following n columns are the columns of M, and whose final column is q. Now, suppose that the variable w_i is to be swapped with z_j. The variable w_i is associated with the i-th column of A, and the variable z_j is associated with the $(n + j)$-th column of A. To perform this operation we swap the columns that are associated with w_i and z_j. We then perform Gauss-Jordan elimination on the tableaux A, which transforms the first n columns of A into an identity matrix and the remaining columns into an n by n matrix M' and an n dimensional vector q'.

A matrix M_α and a vector q_α, where $\alpha \subseteq \{1, 2, \ldots, n\}$, are called a *principal pivot transform* of M and q, if they are obtained by performing $|\alpha|$ pivot operations, which exchange the variables w_i and z_i, for every $i \in \alpha$. Each solution of LCP(M_α, q_α) corresponds to a solution of LCP(M, q). If there is a solution of LCP(M_α, q_α) in which the variables $w_i = 0$ for every $i \in \beta$, and the variables $z_i = 0$ for every $i \notin \beta$, then there is a solution to LCP(M, q) with $w_i = 0$ for every $i \in \gamma$ and $z_i = 0$ for every $i \notin \gamma$, where $\gamma = (\alpha \setminus \beta) \cup (\beta \setminus \alpha)$. Once we know which variables should be set to 0, the system given by (2.1)-(2.3) becomes a system of n simultaneous equations over n variables, which can easily be solved to obtain the precise values of w and z. Therefore, to solve an LCP it is sufficient to find a solution to some principal pivot transform of that LCP.

This fact is useful, because some LCPs are easier to solve than others. For example, a problem LCP(M, q) has a *trivial solution* if $q \geq 0$. Indeed, if this is the case then we can set $z = 0$, and the system given by (2.1)-(2.3) becomes:

$$\begin{aligned} w &= q, \\ w, z &\geq 0, \\ z_i \cdot w_i &= 0 \qquad\qquad \text{for } i = 1, 2 \ldots \text{n}. \end{aligned}$$

Since $q \geq 0$ and $z = 0$, this can obviously be satisfied by setting $w = q$. This gives us a naive algorithm for solving the problem LCP(M, q), which checks, for each $\alpha \subseteq \{1, 2, \ldots, n\}$, whether LCP$(M_\alpha, q_\alpha)$ has a trivial solution. In our setting, where M is a P-matrix, there is guaranteed to be exactly one principal pivot transform of M and q for which there is a trivial solution.

2.4 Optimality Equations

Optimality equations are a fundamental tool that is used to study Markov decision processes and infinite games. The idea is that the value of each vertex can be characterised as the solution of a system of optimality equations. Therefore, to find the value for each vertex, it is sufficient to compute a solution to the optimality equations. Optimality equations will play an important role in this thesis, because each of the algorithms that we study can be seen as a process that attempts to find a solution of these equations.

In this section we introduce optimality equations for the average-reward criterion in MDPs, and for mean-payoff and discounted games. We also show, for each of these models, how the optimality equations can be used to decide if a strategy is optimal. The key concept that we introduce is whether an action or edge is *switchable* for a given strategy. A strategy is optimal if and only if it has no switchable edges or actions. This concept will be heavily used by the algorithms that are

studied in this thesis.

2.4.1 Optimality Equations For Markov Decision Processes

We will begin by describing the optimality equations for the average-reward criterion in MDPs. In the literature, there are two models of average-reward MDP that are commonly considered. In the *uni-chain* setting, the structure of the MDP guarantees that every vertex has the same value. This allows a simplified optimality equation to be used. However, the MDPs that will be considered in this thesis do not satisfy these structural properties. Therefore, we will introduce the *multi-chain* optimality equations, which account for the fact that vertices may have different values. In the multi-chain setting, we have two types of optimality equation, which must be solved simultaneously. The first of these are called the gain equations. For each vertex v there is a gain equation, which is defined as follows:

$$G(v) = \max_{a \in A_v} \sum_{v' \in V} p(v'|v,a) \cdot G(v'). \tag{2.4}$$

Secondly we have the bias equations. We define M_v to be the set of actions that achieve the maximum in the gain equation at the vertex v:

$$M_v = \{a \in A_v \ : \ G(v) = \sum_{v' \in V} p(v'|v,a) \cdot G(v')\}.$$

Then, for every vertex v, we have a bias equation that is defined as follows:

$$B(v) = \max_{a \in M_v} \left(\mathrm{r}(v,a) - G(v) + \sum_{v' \in V} p(v'|v,a) \cdot B(v') \right). \tag{2.5}$$

For this system of optimality equations, the solution is not necessarily unique. In every solution the gain will be the same, but the bias may vary. It has been shown that the gain of each vertex in a solution is in fact the expected average reward from that vertex under an optimal strategy.

Theorem 2.6 ([Put05, Theorem 9.1.3]). *For every solution to the optimality equations we have* $\text{Value}_{\mathcal{A}}(v) = G(v)$*, for every vertex* v*.*

We will now describe how these optimality equations can be used to check whether a strategy σ is optimal. This is achieved by computing the gain and bias of the strategy σ, which can be obtained by solving the following system of equations.

$$G^{\sigma}(v) = \sum_{v' \in V} p(v'|v, \sigma(v)) \cdot G^{\sigma}(v') \tag{2.6}$$

$$B^{\sigma}(v) = \mathrm{r}(v, \sigma(v)) - G^{\sigma}(v) + \sum_{v' \in V} p(v'|v, \sigma(v)) \cdot B^{\sigma}(v') \tag{2.7}$$

It is clear that a strategy σ is optimal in the average-reward criterion if and only if G^{σ} and B^{σ} are a solution to the optimality equations.

It should be noted that the system of equations given in (2.6)-(2.7) may not have a unique solution, and this will cause problems in our subsequent definitions. As with the optimality equations, for each strategy σ and each vertex v, there will be a unique value g such that $G^{\sigma}(v) = g$ for every solution to these equations, but $B^{\sigma}(v)$ may vary between solutions. To ensure that there is a unique solution, we add the following constraint, for every vertex $v \in V$:

$$W^{\sigma}(v) = \sum_{v' \in V} p(v'|v, \sigma(v)) \cdot W^{\sigma}(v') - B^{\sigma}(v). \tag{2.8}$$

Adding Equation 2.8 to the system of equations ensures that the solutions of the gain and bias equations will be unique [Put05, Corollary 8.2.9]. In other words, for every vertex v there is a constant g such that every solution of Equations (2.6)-(2.8) has $G^{\sigma}(v) = g$, and there is a constant b such that every solution of Equations (2.6)-(2.8) has $B^{\sigma}(v) = b$. The solution of Equation (2.8) itself may not be unique.

Now suppose that the strategy σ is not optimal. We will define the *appeal* of an action a under a strategy σ. To do this, we find the unique solution of the system specified by (2.6)-(2.8). We then define the gain of a to be $G^{\sigma}(a) = \sum_{v' \in V} p(v'|v, a) \cdot$

$G^\sigma(v')$, and we define the bias of a to be $B^\sigma(a) = \mathrm{r}(v,a) - G(v) + \sum_{v' \in V} p(v'|v,a) \cdot B(v')$. We then define the appeal of the action a to be $\mathrm{Appeal}^\sigma(a) = (G^\sigma(a), B^\sigma(a))$.

We can now define the concept of a switchable action. To decide if an action is switchable, the appeal of the action is compared lexicographically with the gain and bias of the vertex from which it originates. This means that an action $a \in A_v$ is switchable in a strategy σ if either $G^\sigma(a) > G^\sigma(v)$, or if $G^\sigma(a) = G^\sigma(v)$ and $B^\sigma(a) > B^\sigma(v)$.

Clearly, if σ has a switchable action then G^σ and B^σ do not satisfy the optimality equations, which implies that σ is not an optimal strategy. Conversely, if σ does not have a switchable action then G^σ and B^σ must be a solution to the optimality equations. Therefore, we have the following corollary of Theorem 2.6.

Corollary 2.7. *A strategy σ is optimal if and only if it has no switchable actions.*

2.4.2 Optimality Equations For Games

We now introduce the optimality equations for two player games. We begin by describing a system of optimality equations for discounted games. For each vertex v, the optimality equation for v is defined to be:

$$V(v) = \min_{(v,u) \in E} (\mathrm{r}(v) + \beta \cdot V(u)) \qquad \text{if } v \in V_{\mathrm{Min}},$$

$$V(v) = \max_{(v,u) \in E} (\mathrm{r}(v) + \beta \cdot V(u)) \qquad \text{if } v \in V_{\mathrm{Max}}.$$

Shapley has shown the following theorem.

Theorem 2.8 ([Sha53])**.** *The optimality equations have a unique solution, and for every vertex v, we have* $\mathrm{Value}(v) = V(v)$.

If σ is a strategy for Max and τ is a strategy for Min, then we define $\mathrm{Value}^{\sigma,\tau}(v) = \mathcal{D}(\mathrm{Play}(v,\sigma,\tau))$ to be the payoff that is obtained when Max plays σ against τ. The optimality equations imply that to solve a discounted game, it is

sufficient to find a pair of strategies σ and τ such that $\text{Value}^{\sigma,\tau}$ is a solution to the optimality equations.

Given a positional strategy σ for Max and a positional strategy τ for Min, we define the appeal of an edge (v, u) to be $\text{Appeal}^{\sigma,\tau}(v, u) = \text{r}(v) + \beta \cdot \text{Value}^{\sigma,\tau}(u)$. For each Max vertex v, we say that the edge (v, u) is switchable in σ if $\text{Appeal}^{\sigma,\tau}(v, u) > \text{Value}^{\sigma,\tau}(v)$. For each Min vertex v, we say that the edge (v, u) is switchable in τ if $\text{Appeal}^{\sigma,\tau}(v, u) < \text{Value}^{\sigma,\tau}(v)$.

It is clear that $\text{Value}^{\sigma,\tau}$ satisfies the optimality equations if and only if both σ and τ have no switchable edges. Therefore, we have the following corollary of Theorem 2.8.

Corollary 2.9. *Let σ be a strategy for Max and let τ be a strategy for Min. These strategies are optimal if and only if neither of them have a switchable edge.*

We give a system of optimality equations that characterise optimal values in a mean-payoff game. These optimality equations are a generalisation of the optimality equations for the average-reward criterion in the Markov decision process setting. For each vertex v we have one of the following gain equations:

$$G(v) = \max_{(v,u)\in E} G(u) \qquad \text{if } v \in V_{\text{Max}},$$

$$G(v) = \min_{(v,u)\in E} G(u) \qquad \text{if } v \in V_{\text{Min}}.$$

We define $M = \{(v, u) \in E \,:\, G(v) = G(u)\}$ to be the set of edges that satisfy the gain equation. For every vertex v we have one of the following bias equations:

$$B(v) = \max_{(v,u)\in M} (\text{r}(v) - G(v) + B(u)) \qquad \text{if } v \in V_{\text{Max}},$$

$$B(v) = \min_{(v,u)\in M} (\text{r}(v) - G(v) + B(u)) \qquad \text{if } v \in V_{\text{Min}}.$$

As with the MDP setting, we have that the solutions to this system of equations correspond to the value of the game at each vertex.

Theorem 2.10 ([FV97]). *For every solution to the system of equations we have* $\text{Value}(v) = G(v)$*, for every vertex* v*.*

For each pair of strategies σ and τ, we define the gain and bias of these strategies to be the solution of the following system of equations. For each vertex v, let $\chi : V \to V$ be a function such that $\chi(v) = \sigma(v)$ when $v \in V_{\text{Max}}$, and $\chi(v) = \tau(v)$ when $v \in V_{\text{Min}}$. We use this to define, for every vertex v:

$$G^{\sigma,\tau}(v) = G^{\sigma,\tau}(\chi(v)), \tag{2.9}$$

$$B^{\sigma,\tau}(v) = \text{r}(v) - G^{\sigma,\tau}(v) + B^{\sigma,\tau}(\chi(v)). \tag{2.10}$$

Once again, it will be useful to ensure that (2.9)-(2.10) have a unique solution. As with the MDP setting, this could be achieved by adding an additional equation for each vertex. However, in the game setting there is a much simpler method for achieving this. Since σ and τ are both positional strategies, we know that the infinite path $\text{Play}(v, \sigma, \tau)$ consists of a finite initial path followed by an infinitely repeated cycle, for every starting vertex v. For each cycle formed by σ and τ, we select one vertex u that lies on the cycle, and set $B^{\sigma,\tau}(u) = 0$. It turns out that with this modification in place, the system given by (2.9)-(2.10) has a unique solution for each pair of strategies σ and τ.

A switchable edge can now be defined in the same way as a switchable action was defined for average-reward MDPs. For each edge, we find the unique solution to the system given by (2.9)-(2.10), and we define $\text{Appeal}^{\sigma,\tau}(v, u) = (G^{\sigma,\tau}(v), B^{\sigma,\tau}(v))$. The edge (v, u) is switchable if $\text{Appeal}^{\sigma,\tau}(v, u)$ is greater than $(G^{\sigma,\tau}(v), G^{\sigma,\tau}(u))$ when compared lexicographically.

2.5 Reductions

In this section we present a chain of reductions from infinite games to the linear complementarity problem. We will show that the problem of solving a parity game can be reduced in polynomial time to the problem of solving a mean-payoff game, and that the problem of solving a mean payoff game can be reduced in polynomial time to the problem of solving a discounted game. This shows that the three types of game are very strongly related.

We go on to describe a polynomial time reduction from the problem of solving a discounted game to the linear complementarity problem. This shows that the linear complementarity problem subsumes all of the games that we have introduced, and motivates the inclusion of this problem in our definitions.

2.5.1 Reductions Between Games

In this section we describe a reduction from parity games to mean-payoff games that was given by Puri [Pur95], and a reduction from mean-payoff games to discounted games that was given by Zwick and Paterson [ZP96]. We begin with Puri's reduction from parity games to mean-payoff games. The reduction does not change the structure of the graph, instead it replaces each priority with a reward. More precisely, if a vertex has priority d in the parity game then it will be assigned a reward of $(-|V|)^d$ in the mean-payoff game. This means that vertices with even priorities will get positive rewards, and vertices with odd priorities will get negative rewards. For example, if we applied the reduction to the parity game shown in Figure 2.1, then we would obtain the mean-payoff game shown in Figure 2.4.

The reduction works because it ensures the following property: the sum of the rewards on a simple cycle in the mean-payoff game is positive if and only if the highest priority on that cycle in the parity game is even. Using this property, Puri showed the following theorem.

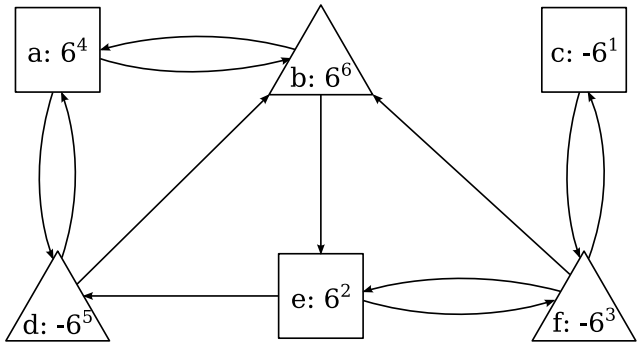

Figure 2.4: The result of reducing the parity game shown in Figure 2.1 to a mean-payoff game.

Theorem 2.11 ([Pur95])**.** *Let G be a parity game, and let G' be the reduction of G to a mean-payoff game. Player Even has a winning strategy from a vertex v in G if and only if* $\text{Value}(v) > 0$ *in G'.*

Theorem 2.11 implies that we only need to solve the qualitative zero-mean partition problem for the mean-payoff game in order to solve the parity game. This is because the partition $(W_{\text{Even}}, W_{\text{Odd}})$ in the parity game must be the same as the partition $(W_{\text{Max}}, W_{\text{Min}})$ in the mean-payoff game. Moreover, a winning strategy for Max on the set W_{Max} is also a winning strategy for Even on the set W_{Even}. The same property holds for the winning strategies of Min and Odd.

We now turn our attention to the reduction from mean-payoff games to discounted games given by Zwick and Paterson [ZP96]. The reduction does not change the structure of the game, or the rewards assigned to the vertices. Instead, it relies on a smart choice of discount factor for the discounted game. In particular, they choose the discount factor to be $1 - 1/(4 \cdot |V|^3 \cdot W)$, where W is $\max_{v \in V}(|\,\mathrm{r}(v)|)$, which is the reward with the largest magnitude. For example, if we were to reduce the mean-payoff game shown in Figure 2.4 to a discounted game, we would choose the discount factor to be $1 - 1/(4 \cdot 6^3 \cdot 6^6)$.

To explain why this choice of discount factor is correct, we will use $\text{Value}_{\mathcal{M}}(v)$ to denote the value of a vertex in the mean-payoff game, and $\text{Value}_{\mathcal{D}}(v)$ to denote

the value of the corresponding vertex in the discounted game. Zwick and Paterson showed bounds on how far away $(1-\beta)\cdot \text{Value}_{\mathcal{D}}(v)$ can be from $\text{Value}_{\mathcal{M}}(v)$ as β approaches 1, and they also showed that the two values converge as β approaches 1. They observed that positional determinacy of mean-payoff games implies that $\text{Value}_{\mathcal{M}}(v)$ must be a rational number whose denominator is at most $|V|$. Their choice for the discount factor ensures that there can only be one such rational number that is close enough to $(1-\beta)\cdot \text{Value}_{\mathcal{D}}(v)$, and therefore $\text{Value}_{\mathcal{M}}(v)$ can be determined from a solution of the discounted game. This gives a polynomial time reduction from mean-payoff games to discounted games.

2.5.2 Reducing Discounted Games To LCPs

In this section we describe the reduction from discounted games to the P-matrix linear complementarity problem, given by Jurdziński and Savani [JS08]. Many of the notations used in this section are taken directly from their exposition. This reduction assumes that the discounted game is a binary game with rewards placed on edges.

The reduction will make heavy use of the optimality equations. As we have seen, these equations consider a pair of strategies, with one strategy for each player. In order to simplify our definitions, we introduce the concept of a *joint strategy*, which specifies the strategy decisions for both players. Formally, a joint strategy is a function $\sigma : V \to V$ that picks one outgoing edge from each vertex in the game. The notation $\sigma \restriction \text{Max}$ and $\sigma \restriction \text{Min}$ will be used to refer to the individual strategies of Max and Min that constitute the joint strategy.

We introduce some notation for binary discounted games with rewards placed on edges that will ease our exposition. For a vertex v, the function $\overline{\sigma}(v)$ gives the successor of v not chosen by the joint strategy σ. In other words, we have $\overline{\sigma}(v) = \lambda(v)$ if and only if $\sigma(v) = \rho(v)$. The functions r^{σ} and $r^{\overline{\sigma}}$ both have the form $V \to \mathbb{Q}$, and they give, for each vertex, the reward on the edge chosen by σ and the reward on

the edge not chosen by σ, respectively.

The concepts that were presented in Section 2.1.3 can easily be adapted for joint strategies. We define $\text{Play}(\sigma, v)$ to be equal to the path $\text{Play}(\sigma \restriction \text{Max}, \sigma \restriction \text{Min}, v)$. For a given joint strategy σ, the value of a vertex v when σ is played is then defined to be $\text{Value}^\sigma(v) = \mathcal{D}(\text{Play}(\sigma, v))$. A joint strategy is optimal only if both $\sigma \restriction \text{Max}$ and $\sigma \restriction \text{Min}$ are optimal, and our objective is to compute an optimal joint strategy.

The reduction begins with the optimality equations for discounted games. For binary discounted game with rewards placed on edges, these equations are, for every vertex v:

$$V(v) = \begin{cases} \min\{r^\lambda(v) + \beta \cdot V(\lambda(v)), r^\rho(v) + \beta \cdot V(\rho(v))\} & \text{if } v \in V_{\text{Min}}, \\ \max\{r^\lambda(v) + \beta \cdot V(\lambda(v)), r^\rho(v) + \beta \cdot V(\rho(v))\} & \text{if } v \in V_{\text{Max}}. \end{cases} \tag{2.11}$$

As we know, the optimality equations can be used to decide whether a strategy is optimal. Given a joint strategy σ and a vertex v, we define the *balance* of v to be the difference between the value of v and the value of the play that starts at v, moves to $\overline{\sigma}(v)$ in the first step, and then follows σ:

$$\text{Bal}^\sigma(v) = \begin{cases} \text{Value}^\sigma(v) - (r^{\overline{\sigma}}(v) + \beta \cdot \text{Value}^\sigma(\overline{\sigma}(v))) & \text{if } v \in V_{\text{Max}}, \\ (r^{\overline{\sigma}}(v) + \beta \cdot \text{Value}^\sigma(\overline{\sigma}(v))) - \text{Value}^\sigma(v) & \text{if } v \in V_{\text{Min}}. \end{cases} \tag{2.12}$$

Clearly, there is a switchable edge at a vertex v if and only if $\text{Bal}^\sigma(v) < 0$. Since each vertex can have only one edge that is not chosen by σ, we will say that a vertex is switchable whenever that vertex has a switchable edge. If $\text{Bal}^\sigma(v) = 0$ for some vertex then that vertex is said to be indifferent.

The reduction modifies the optimality equations for discounted games by introducing *slack variables* $z, w : V \to \mathbb{Q}$. For each vertex v, the optimality equa-

tion (2.11) is replaced by the following set of equations:

$$V(v) - w(v) = r_\lambda(v) + \beta \cdot V(\lambda(s)) \quad \text{if } v \in V_{\text{Max}}, \tag{2.13}$$

$$V(v) - z(v) = r_\rho(v) + \beta \cdot V(\rho(s)) \quad \text{if } v \in V_{\text{Max}}, \tag{2.14}$$

$$V(v) + w(v) = r_\lambda(v) + \beta \cdot V(\lambda(s)) \quad \text{if } v \in V_{\text{Min}}, \tag{2.15}$$

$$V(v) + z(v) = r_\rho(v) + \beta \cdot V(\rho(s)) \quad \text{if } v \in V_{\text{Min}}, \tag{2.16}$$

$$w(v),\ z(v) \geq 0, \tag{2.17}$$

$$w(v) \cdot z(v) = 0. \tag{2.18}$$

We argue that an optimal strategy for the discounted game can be derived from a solution to the system given by (2.13)-(2.18). The complementarity condition given in (2.18) insists that one of the two slack variables must be 0 for every vertex. Let $\alpha \subseteq V$, and suppose that there is a solution to this system of equations such that $w(v) = 0$ for every $v \in \alpha$, and $z(v) = 0$ for every $v \notin \alpha$. Furthermore, suppose that $v \in V_{\text{Max}}$. Applying Equations (2.13) and (2.14) to this solution gives:

$$V(v) = \begin{cases} r_\lambda(v) + \beta \cdot V(\lambda(s)) & \text{if } v \in \alpha, \\ r_\rho(v) + \beta \cdot V(\rho(s)) & \text{if } v \notin \alpha. \end{cases} \tag{2.19}$$

An identical equation can be derived for vertices $v \in V_{\text{Min}}$. Therefore, for each solution of the system of equations, there is a corresponding joint strategy σ for the discounted game, which is defined to be, for every $v \in V$:

$$\sigma(v) = \begin{cases} \lambda(v) & \text{if } v \in \alpha, \\ \rho(v) & \text{if } v \notin \alpha. \end{cases}$$

We can then rewrite Equation 2.19 to obtain, for every vertex v:

$$V(v) = r^\sigma(v) + \beta \cdot V(\sigma(s)).$$

This implies that $V(v) = \text{Value}^{\sigma}(v)$, for every vertex v. In summary, for every solution of the system given by (2.13)-(2.18), there is a joint strategy σ such that $V(v) = \text{Value}^{\sigma}(v)$.

For the reduction to be correct, we must argue that the strategy σ is an optimal strategy in the discounted game. Let v be a vertex such that $v \in \alpha$. We can rewrite Equations (2.14) and (2.16) as:

$$z(v) = V(v) - (r_{\rho}(v) + \beta \cdot V(\rho(s))) \text{ if } v \in V_{\text{Max}},$$
$$z(v) = (r_{\rho}(v) + \beta \cdot V(\rho(s))) - V(v) \text{ if } v \in V_{\text{Min}}.$$

By substituting in our knowledge about the strategy σ into the previous equation, we obtain:

$$z(v) = \begin{cases} \text{Value}^{\sigma}(v) - (r_{\sigma}(v) + \beta \cdot \text{Value}^{\sigma}(\sigma(s))) & \text{if } v \in V_{\text{Max}}, \\ (r_{\sigma}(v) + \beta \cdot \text{Value}^{\sigma}(\sigma(s))) - \text{Value}^{\sigma}(v) & \text{if } v \in V_{\text{Min}}. \end{cases}$$

An identical derivation can be made for vertices $v \notin \alpha$ to obtain the same expression for the variable $w(v)$. Therefore, we have $z(v) = \text{Bal}^{\sigma}(v)$ if $v \in \alpha$, and $w(v) = \text{Bal}^{\sigma}(v)$ if $v \notin \alpha$. The non-negativity constraint (2.17) insists that these variables should not be negative, which implies that $\text{Bal}^{\sigma}(v) \geq 0$ for every vertex v. By Corollary 2.9, we have that σ is an optimal strategy for the discounted game.

We now argue that the system (2.13)-(2.18) can be rewritten as an LCP. We will assume that every vertex in V is assigned a unique index in the range $\{1, 2, \ldots |V|\}$, which can be used to identify it. For each $|V|$ by $|V|$ matrix A, we define $\widehat{A}$ to be A with every element in every column whose index $v \in V_{\text{Min}}$ negated.

$$(\widehat{A})_{st} = \begin{cases} A_{st} & \text{if } t \in V_{\text{Max}}, \\ -A_{st} & \text{if } t \in V_{\text{Min}}. \end{cases}$$

For every joint strategy σ we define a matrix T_σ to be the adjacency matrix of the sub-graph defined by σ. Therefore, we define:

$$(T_\sigma)_{st} = \begin{cases} 1 & \text{if } \sigma(s) = t, \\ 0 & \text{otherwise.} \end{cases}$$

If r_λ and r_ρ are the vectors of rewards assigned to each vertex, and I is the identity matrix, then Equations (2.13)-(2.16) can be rewritten in matrix form as:

$$\widehat{I}V = w + \widehat{I}r_\lambda + \beta\widehat{T_\lambda}V,$$
$$\widehat{I}V = z + \widehat{I}r_\rho + \beta\widehat{T_\rho}V.$$

By eliminating V from these equations and simplifying, we obtain:

$$w = (\widehat{I} - \beta\widehat{T_\lambda})(\widehat{I} - \beta\widehat{T_\rho})^{-1}(z + \widehat{I}r_\rho) - \widehat{I}r_\lambda.$$

Therefore, we can define an LCP with the following inputs:

$$M = (\widehat{I} - \beta\widehat{T_\lambda})(\widehat{I} - \beta\widehat{T_\rho})^{-1},$$
$$q = M\widehat{I}r_\rho - \widehat{I}r_\lambda.$$

Jurdziński and Savani proved the following Theorem.

Theorem 2.12 ([JS08])**.** *The matrix M is a P-matrix, and the unique solution of* $\mathrm{LCP}(M, q)$ *can be used to find an optimal strategy for the discounted game.*

Chapter 3

Algorithms

In this section we describe the algorithms that have been proposed to solve our problems. The first half of this chapter surveys the known results on these topics. In the second half of this chapter, we describe in detail the algorithms that will be studied in this thesis.

3.1 A Survey Of Known Algorithms

There are three prominent techniques that are used to solve Markov decision processes. The first is to compute a solution of the optimality equations using value iteration, which was proposed by Bellman [Bel57]. Secondly, there are reductions from the problem of solving a Markov decision process to linear programming [d'E63, Man60]. These reductions allow the use of standard linear programming techniques, such as the simplex method [WD49, Dan49], the ellipsoid method [Kha80], and the interior point method [Kar84]. Finally, there is strategy improvement, also known as policy iteration, which was proposed by Howard [How60]. Strategy improvement algorithms will be described in detail in Section 3.2.

Most of the complexity results have been shown for the discounted reward criterion. Tseng has shown that value iteration terminates, and that it is a weakly

polynomial time algorithm, under the assumption that the discount factor is constant [Tse90]. Puterman has replicated this result for strategy improvement [Put05]. There are two problems with these results, which are the assumption that the discount factor is constant, and the fact that they do not show strongly polynomial time upper bounds. The linear programming formulation avoids the first problem, since these linear programs can be solved in weak polynomial time irrespective of the choice of the discount factor. In attacking the second problem, Ye has shown a strongly-polynomial time interior point-algorithm for solving these linear programs [Ye05], and he has shown that strategy improvement can be made to terminate in strongly polynomial time [Ye10]. On the other hand, both of Ye's results assume that the discount factor is constant. The problem of finding a strongly polynomial time algorithm that takes the discount factor as part of the input is still open.

Strategy improvement algorithms for two player games were first devised by Hoffman and Karp [HK66], who studied a variation of Shapley's stochastic games [Sha53]. Their algorithm is a generalisation of Howard's strategy improvement algorithm for Markov decision processes [How60]. The techniques used by Hoffman and Karp have been adapted by Puri [Pur95], to produce a strategy improvement algorithm that solves discounted games. This algorithm will be described in detail in Section 3.2.2.

Another possible way of solving a discounted game is to apply Zwick and Paterson's reduction [ZP96] from the problem of solving a discounted game to the problem of solving a *simple stochastic game* [Con92]. Simple stochastic games combine the traits found in Markov decision processes and two player games: the games have two players, and the outcome when a player makes a move is determined by a probability distribution over successor vertices. The two players play a reachability game: one player is trying to move the token to a target vertex, and the opponent is trying to prevent the token from arriving at the target vertex. There are various

algorithms available to solve simple stochastic games [Con93].

The most widely known algorithm for solving mean-payoff games is the psuedo-polynomial time algorithm given by Zwick and Paterson [ZP96]. The running time of this algorithm is bounded by $O(|V|^3 \cdot |E| \cdot W)$, where W is the largest magnitude of a reward. Therefore, this algorithm will perform well when all rewards are small. However, this algorithm will not perform well on games with exponentially large rewards, such as those produced by the reduction from parity games to mean-payoff games.

Two strategy improvement algorithms have been considered for mean-payoff games. The first is based on the gain-bias optimality equations. Filar and Vrieze devised a strategy improvement algorithm for a concurrent version of mean-payoff games [FV97]. In these games, the vertices are not divided between the players. Instead, at each vertex, the players play a matrix game to decide where the token is moved. This means that the two players pick their moves concurrently, and the successor is determined by the combination of these two moves. Mean-payoff games can be seen as a special case of these games, and the strategy improvement algorithm of Filar and Vrieze corresponds to a strategy improvement algorithm that uses the gain-bias optimality equations for this special case.

The second strategy improvement algorithm for mean-payoff games is the algorithm given by Björklund and Vorobyov [BV07]. While the strategy improvement algorithm of Filar and Vrieze computes the exact value for each vertex, this algorithm only computes the zero mean partition of the mean-payoff game. On the other hand, it can be shown that the running time of this algorithm is always bounded by $O(|V|^3 \cdot |E| \cdot W \cdot (\log n + \log W))$, which makes this algorithm competitive with the algorithm of Zwick and Paterson.

The classical algorithm for parity games is a recursive algorithm that was originally given by McNaughton [McN93]. A reinterpretation of this algorithm was provided by Zielonka [Zie98]. The best upper bound on the running time of this

algorithm is $O(2^{|V|})$. This result can be improved if the game contains many vertices that are assigned the same priority. In this case, the running time can be bounded by $O(|V|^{d+1})$, where d is the number of *distinct* priorities that are used in the game. On the other hand, it is known that this algorithm is not polynomial, because families of games are known for which this algorithm takes an exponential number of steps [Jur00, Fri10b].

Jurdziński, Paterson, and Zwick present a modified version of the recursive algorithm that achieves a better running time in the worst case [JPZ06]. The recursive algorithm can take exponential time, because the result of each recursive call may allow the algorithm to make only a small amount of progress. They show how a preprocessing step can be added before each recursive call, and how this preprocessing ensures that the result of the recursive call allows a significant amount of progress to be made. By performing a careful balancing act between the amount of time spent preprocessing, and the amount of progress that each recursive call achieves, they obtained an algorithm whose worst case running time is sub-exponential: the running time of their algorithm is bounded by $n^{O(\sqrt{n})}$. This is the best known running time for solving a parity game in which the number of distinct priorities is large.

The small progress measures algorithm given by Jurdziński is faster than the recursive algorithm when the number of distinct priorities is small [Jur00]. A progress measure is a labelling of the vertices that satisfies certain properties. The existence of a progress measure implies that one of the players has a winning strategy for a certain set of vertices in the game. Jurdziński showed that a small progress measure can be computed in $O(m \cdot n^{\lceil d/2 \rceil})$ time, which is better than the recursive algorithm when the number of distinct priorities is smaller than $\sqrt{|V|}$. Jurdziński also provides an exponential lower bound for this algorithm by providing a family of polynomially sized games upon which the small progress measure algorithm takes $(\lceil n/d \rceil)^{\lceil d/2 \rceil}$ many steps.

Schewe has combined the ideas from the small progress measures algorithm

with the sub-exponential recursive algorithm to obtain the state of the art algorithm for parity games with a small number of distinct priorities [Sch07]. The algorithm of Jurdziński, Paterson, and Zwick uses a brute force method to perform preprocessing. Schewe showed that the small progress measures algorithm can be adapted to perform this preprocessing faster than the brute force method. Through careful analysis, he obtained a bound of $O(m \cdot n^{d/3+o(1)})$ for the running time of his algorithm.

Finally, Vöge and Jurdziński have proposed a discrete strategy improvement algorithm for parity games [VJ00]. Most strategy improvement algorithms measure the worth of a strategy using a vector of rational numbers. However, Vöge and Jurdziński give a discrete measure that can be used to rate strategies. This is useful, because the discrete measure provides an attractive platform for analysis of strategy improvement for parity games.

One simple algorithm for solving a linear complementary problem is Murty's least-index algorithm [Mur74]. This is a pivoting algorithm, and it selects the variable that will be pivoted using a least-index rule. This algorithm is guaranteed to terminate when the input matrix is a P-matrix. However, there are families of examples upon which this algorithm takes an exponential number of steps to find a solution to the LCP, even when applied to a P-matrix LCP [Mur78].

The problem of solving a bimatrix game can be reduced to the problem of solving a linear complementarity problem. Lemke and Howson devised an algorithm to solve the LCPs that arise from this reduction [LH64]. This was later generalised by Lemke to provide an algorithm that can be applied to an arbitrary LCP [Lem65]. We will give a detailed description of Lemke's algorithm in Section 3.3.1.

Lemke's algorithm is guaranteed to terminate, but it is not guaranteed to find a solution to the LCP. This is because the methods used by the algorithm may not be able to process a given LCP. When this occurs, Lemke's algorithm terminates in a well defined manner, which is called *ray termination*. The fact that Lemke's

algorithm terminates in this way for a certain LCP does not imply that the LCP has no solution. It only implies that Lemke's algorithm is incapable of finding a solution for that LCP. Fortunately, in the setting that we are interested in, where the input is a P-matrix, it is known that Lemke's algorithm will always terminate with the unique solution to the LCP.

It is known that Lemke's algorithm can take an exponential number of steps to find a solution [Mur78, Fat79], even for P-matrix LCPs. However, the behaviour of the algorithm depends on a user supplied *covering vector*. For example, Adler and Megiddo studied the performance of Lemke's algorithm for the LCPs arising from randomly chosen linear programming problems [AM85]. They show that if a natural choice for the covering vector is used, then the expected number of steps taken by Lemke's algorithm is exponential. On the other hand, a carefully chosen covering vector allows Lemke's algorithm to terminate after an expected quadratic number of steps on these LCPs.

Another pivoting method for solving a linear complementarity problem is the Cottle-Dantzig algorithm [DC67]. This method is also known to always terminate with a solution when it is applied to a P-matrix linear complementarity problem. It is also known that this algorithm can take an exponential number of steps [Mur88]. This algorithm will be discussed in detail in Section 3.3.2.

3.2 Strategy Improvement

In this section we describe the strategy improvement algorithms that have been proposed for Markov decision processes and for two player games. Each model has a specific strategy improvement algorithm, however all of these algorithms have the same basic structure. We will begin by giving an outline of the features that all strategy improvement algorithms share, and we then go on to describe the particular details for each specific strategy improvement algorithm.

The core idea behind strategy improvement is to use a *valuation* function Val^σ, which assigns a valuation to each vertex. Typically, the valuation of a vertex will be a rational number, but there are some algorithms that assign more complex valuations to each vertex. The discrete strategy improvement algorithm for parity games is one example of such an algorithm. For now we will assume that our valuation function is of the form $\mathrm{Val}^\sigma : V \to \mathbb{Q}$. The valuation function measures how good a strategy is, and it therefore can be used to compare different strategies. Given two strategies σ and σ', we we say that $\sigma \prec \sigma'$ if $\mathrm{Val}^\sigma(v) \leq \mathrm{Val}^{\sigma'}(v)$ for every vertex v, and $\mathrm{Val}^\sigma(v) < \mathrm{Val}^{\sigma'}(v)$ for some vertex v. Note that it is possible for two strategies to be incomparable in the $\prec$ ordering, and that therefore, the ordering $\prec$ provides a partial order over strategies.

The principal idea behind strategy improvement is to modify each strategy that it considers in order to create an improved strategy. To do this, it uses the concept of a switchable action or edge that was introduced in Section 2.4. Strategy improvement begins with an arbitrary strategy. In each iteration the algorithm computes the set of actions or edges that are switchable in the strategy that it is currently considering, and it then picks some subset of the switchable actions and *switches* them. This operation creates a new strategy that will be considered in the subsequent iteration. The algorithm only terminates when it reaches a strategy with no switchable actions or edges. Algorithm 1 shows the general structure of a strategy improvement algorithm.

Algorithm 1 A strategy improvement algorithm.

```
σ := an arbitrary strategy
while σ has a switchable edge do
  Compute Val^σ
  P := the set of the switchable edges in σ
  F := some subset of P
  σ := σ with the edges in F switched
end while
return σ
```

The key property is that switching a subset of switchable edges creates an improved strategy. Strategy improvement computes a sequence of strategies that monotonically increase in the $\prec$ ordering. Since there are only a finite number of positional strategies, this process cannot continue forever. This implies that strategy improvement must eventually find a strategy that has no switchable edges or actions. The second property that must hold is that if a strategy has no switchable actions or edges, then it is an optimal strategy for the MDP or game. In other words, strategy improvement must terminate, and it only terminates when a solution to the MDP or game has been found.

An important part of a strategy improvement algorithm is the *switching policy*. This is a procedure that decides which subset of the switchable edges or actions should be switched in each iteration. The choice of switching policy has an effect on the running time of the strategy improvement algorithm. We know that all strategy improvement algorithms cannot take more iterations than the total number of positional strategies that are available. However, strategy improvement algorithms can be shown to have better worst case running bounds when an appropriate switching policy is chosen.

The proofs of upper bounds on the worst case running time for switching policies are often independent from the specific strategy improvement algorithms that use them. This is because all strategy improvement algorithms have an underlying combinatorial structure called a *unique sink orientation* [Wil88]. Many proofs of worst case running time are in fact proofs that deal with these structures, rather than with a specific strategy improvement algorithm for a game or an MDP. On the other hand, proofs of lower bounds for the running time of a switching policy are often specific to a particular strategy improvement algorithm.

This section begins by giving strategy improvement algorithms for the various types of MDP, and game, that we are interested in. The second half of this section gives a survey of the switching policies that have been proposed, and the running

time bounds that have been proved for these switching policies.

3.2.1 Strategy Improvement For Markov Decision Processes

We will describe the strategy improvement algorithm for the average-reward criterion that was given by Howard [How60]. This algorithm attempts to find a solution of the gain-bias optimality equations. For each strategy σ, the algorithm computes the gain and bias of σ by solving the system of equations given in (2.6)–(2.8), and for every vertex v we define $\mathrm{Val}^\sigma(v) = (G^\sigma(v), B^\sigma(v))$. These valuations will be compared lexicographically: we say that $(g, b) < (g', b')$ if $g < g'$, or if $g = g'$ and $b < b'$.

The algorithm first checks whether there is an action a at a vertex v such that $G^\sigma(a) > G^\sigma(v)$. We will call such an action *gain-switchable*. If there is such an action, then the algorithm picks some subset of the gain-switchable actions and switches them. We denote the operation of switching the action a at the vertex w in the strategy σ as $\sigma[a]$, and the strategy $\sigma[a]$ is defined to be, for every vertex v:

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

If S is some subset of the switchable actions in σ that contains at most one action for each vertex, then we define $\sigma[S]$ to be σ with every action in S switched. It can be shown that switching a subset of the gain-switchable actions will yield a strategy with an improved gain. The original proof of this theorem, as shown in the book of Puterman [Put05, Theorem 9.2.6], only deals with the case where every gain-switchable action is switched. This can easily be adapted to prove our claim, by removing the gain-switchable actions that we do not intend to switch, and then applying the original theorem to the resulting MDP.

Theorem 3.1 ([Put05, Theorem 9.2.6])**.** *In the average-reward criterion, if σ is*

a strategy and σ' is a strategy that is obtained by switching some subset of gain-switchable actions in σ then $G^{\sigma}(v) \leq G^{\sigma'}(v)$ for every vertex v, and there is at least one vertex for which this inequality is strict.

If there are no gain-switchable actions, then the algorithm checks whether there are any *bias-switchable* actions. An action is bias-switchable if it is switchable but not gain-switchable. If there is at least one bias-switchable action, then the algorithm selects some subset of the bias-switchable actions and switches them. Once again, it can be shown that switching a subset of the bias-switchable actions produces an improved strategy. The following theorem can be proved using the Theorem shown in the book of Puterman [Put05, Theorem 9.2.6], in the same manner as Theorem 3.1.

Theorem 3.2 ([Put05, Theorem 9.2.6])**.** *In the average-reward criterion, if σ is a strategy and σ' is a strategy that is obtained by switching some subset of bias-switchable actions in σ then $G^{\sigma}(v) \leq G^{\sigma'}(v)$ for every vertex v, and if $G^{\sigma}(v) = G^{\sigma'}(v)$, then $B^{\sigma}(v) \leq B^{\sigma'}(v)$. Moreover, there is at least one vertex v such that either $G^{\sigma}(v) < G^{\sigma'}(v)$, or $G^{\sigma}(v) = G^{\sigma'}(v)$ and $B^{\sigma}(v) < B^{\sigma'}(v)$.*

Theorems 3.1 and 3.2 together imply that the valuation function Val^{σ} must monotonically increase when a subset of the switchable actions is switched. This implies the correctness of the strategy improvement algorithm that we have described.

3.2.2 Strategy Improvement For Discounted Games

Strategy improvement for two player games uses the same ideas as strategy improvement for Markov decision processes. In this section we describe a strategy improvement algorithm for discounted games that was originally given by Puri [Pur95]. Strategy improvement algorithms for games choose one player to be the *strategy improver*, and attempt to find an optimal strategy for that player. In our exposition, we will choose player Max to take this role.

The first problem that we have is to define what a valuation should be in this setting. When we considered MDPs, we could directly use the value of a strategy as a valuation. This is not possible in the game setting, because even if we fix a strategy for one of the players, the payoff that is obtained is determined by how the opponent chooses to play against that strategy. In order to define the valuation for a given strategy, we assume that the opponent is playing a *best response* against that strategy. If Max plays a strategy σ, then a Min strategy τ is a best response if it satisfies $\text{Value}^{\sigma,\tau}(v) \leq \text{Value}^{\sigma,\tau'}(v)$ for every Min strategy τ' and every vertex v. Therefore, our valuation of a Max strategy will be the value that it obtains in the worst case, which occurs when Min plays a best response.

A best response can be computed by solving a one player discounted game. Once a Max strategy σ has been fixed, every edge (v, u) such that $v \in V_{\text{Max}}$ and $\sigma(v) \neq u$ can be removed from the game. What remains is a game in which only Player Min has meaningful strategic decisions, and this can be converted into an equivalent one player game by giving all the Max vertices to player Min. The problem of solving a one player discounted game can be formulated as a linear program, and therefore, a best response to a Max strategy σ can be computed in polynomial time. Moreover, there is a strongly-polynomial time algorithm for the linear programs that arise from this reduction [MTZ10].

For a given Max strategy σ, there may be several Min strategies that are best responses to σ. However, strategy improvement will compute only one best response for each strategy σ. Therefore, we define the function $\text{br} : \Pi_{\text{Max}} \to \Pi_{\text{Min}}$, which selects a best response for each Max strategy. We make no assumptions about which best response is chosen. The valuation function for the strategy σ will be $\text{Value}^{\sigma,\text{br}(\sigma)}$. For this reason, we define $\text{Val}^{\sigma}(v)$ to be shorthand for $\text{Value}^{\sigma,\text{br}(\sigma)}(v)$.

It is not difficult to see that our valuation function satisfies the condition that a strategy σ is optimal if and only if it has no switchable edges. The best response is an optimal counter strategy, which implies that it cannot have any switchable edges

when it is played against σ. Therefore, if σ also has no switchable edges, then we know that $\mathrm{Val}^{\sigma}(v)$ is a solution to the optimality equations for discounted games. If this holds, then σ is an optimal strategy for Max, and that $\mathrm{br}(\sigma)$ is an optimal strategy for Min.

We must also have that switching some subset of the switchable edges in a strategy creates an improved strategy. The concept of switching carries over from Markov decision processes: switching an edge (v, u) in a strategy σ creates a strategy $\sigma[v \mapsto u]$ where, for every vertex w:

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

If S is a set of edges that contains at most one edge for each vertex, then we once again define $\sigma[S]$ to be σ with every edge in P switched.

Theorem 3.3 ([Pur95])**.** *Let σ be a strategy for Max, and let S be a subset of the switchable edges in σ that contains at most one edge for each vertex. For every vertex v we have:*

$$\mathrm{Val}^{\sigma}(v) \leq \mathrm{Val}^{\sigma[S]}(v).$$

Moreover, there is some vertex for which this inequality is strict.

3.2.3 Strategy Improvement For Mean-Payoff Games

The methods that we described in the previous section can easily be adapted to produce a strategy improvement algorithm for mean-payoff games that uses gain and bias to measure the quality of each Max strategy. However, Björklund and Vorobyov have given a different strategy improvement algorithm for the zero-mean partition problem that does not use the gain and bias formulation [BV07]. We will first describe this algorithm, and we will then explain how it is related to the gain and bias optimality equations.

As usual, we will pick Max to be the strategy improver. The first thing that this algorithm does is to modify the game. The game is modified by adding a special sink vertex s, and an additional outgoing edge (v, s) will be added from every Max vertex v.

Definition 3.4 (Modified Game)**.** *A mean-payoff game* $(V, V_{Max}, V_{Min}, E, \mathrm{r})$ *will be modified to create* $(V \cup \{s\}, V_{Max}, V_{Min} \cup \{s\}, E', \mathrm{r}')$*, where* $E' = E \cup \{(v, s) : v \in V_{Max}\} \cup \{(s, s)\}$*, and:*

$$\mathrm{r}'(v) = \begin{cases} 0 & \text{if } v = s, \\ \mathrm{r}(v) & \text{otherwise.} \end{cases}$$

The strategy improvement algorithm of Björklund and Vorobyov always works with a modified game. Therefore, whenever we speak about the algorithm we will assume that the game has been modified in this way. The purpose of modifying the game is to ensure that Max always has an *admissible* strategy. A Max strategy σ is admissible if Min cannot form a non-positive cycle when playing against σ. This means that $\mathcal{M}(\mathrm{Play}(v, \sigma, \tau)) > 0$ for every Min strategy τ.

The algorithm must be initialised with some admissible strategy. If the user provides an initial admissible strategy then it can be used. Otherwise, the modification of the game ensures that an admissible strategy can always be found. This strategy will be called σ_0, and it is defined so that $\sigma_0(v) = s$ for every vertex $v \in V_{\mathrm{Max}}$. The strategy σ_0 is admissible unless there is some negative cycle that Min can form without ever passing through a vertex in V_{Max}. However, these cycles, and the vertices from which Min can force the token to these cycles, can be removed and added to W_{Min} in a preprocessing step, using Karp's algorithm for finding the minimum weighted cycle in a directed graph [Kar78]. Therefore, we can assume that σ_0 is an admissible strategy.

We can now describe the valuation function that this algorithm uses. We begin by giving a valuation function for a pair of strategies σ and τ, for players Max

and Min respectively. Suppose that Max plays an admissible positional strategy σ and Min plays a positional strategy τ. Since both these strategies are positional, the play that is consistent with σ and τ from a starting vertex v_0 must either consist of, a possibly empty, initial simple path followed by an infinitely repeated simple cycle, or consist of a finite path that ends at the sink. In the first case, the admissibility of the strategy σ ensures that the sum of the rewards on the cycle is positive, and so we define the valuation of the vertex v when σ and τ are played to be ∞. In the second case, we define the valuation of the vertex v when σ and τ are played to be sum of the rewards on the path to the sink.

Definition 3.5 (Valuation)**.** *Let σ be an admissible positional strategy for Max and let τ be a positional strategy for Min. We define the function* $\mathrm{Val}^{\sigma,\tau} : V \to \mathbb{Z} \cup \{\infty\}$ *as follows. If* $\mathrm{Play}(v_0, \sigma, \tau) = \langle v_0, v_1, \ldots, v_k, \langle c_0, c_1, \ldots, c_l \rangle^\omega \rangle$*, for some vertex v_0, then we define* $\mathrm{Val}^{\sigma,\tau}(v_0) = \infty$*. Alternatively, if* $\mathrm{Play}(v, \sigma, \tau) = \langle v_0, v_1, \ldots, v_k, \langle s \rangle^\omega \rangle$ *then we define* $\mathrm{Val}^{\sigma,\tau}(v_0) = \sum_{i=0}^{k} \mathrm{r}(v_i)$*.*

Given an admissible strategy σ for Max, a best response is then a strategy for Min that minimizes the valuation when σ is played. Formally, an Min strategy τ is a best response to σ if $\mathrm{Val}^{\sigma,\tau}(v) \leq \mathrm{Val}^{\sigma,\tau'}(v)$ for every Min strategy τ' and every vertex v. For an admissible strategy, we can compute a best response by removing, from each Max vertex, every edge that is not chosen by σ, and then computing the shortest path from each vertex to the sink. If there is not path to the sink from some vertex, then the valuation of that vertex must be ∞. We again define $\mathrm{br}(\sigma)$ to be a function that chooses some best response for the strategy σ. The valuation function for each Max strategy σ will be $\mathrm{Val}^{\sigma,\mathrm{br}(\sigma)}$, and we again define $\mathrm{Val}^{\sigma} = \mathrm{Val}^{\sigma,\mathrm{br}(\sigma)}$.

We define the appeal of an edge to be $\mathrm{Appeal}^{\sigma}(v, u) = \mathrm{r}(v) + \mathrm{Val}^{\sigma}(u)$. As usual, an edge is switchable in the Max strategy σ if $\mathrm{Appeal}^{\sigma}(v, u) > \mathrm{Val}^{\sigma}(v)$. This is equivalent to the condition $\mathrm{Val}^{\sigma}(\sigma(v)) < \mathrm{Val}^{\sigma}(u)$. As usual, we have that switching a subset of switchable edges will create an improved strategy.

Theorem 3.6 ([BV07, Theorem 5.1]). *Let σ be an admissible strategy and P be the set of edges that are switchable in σ. Let $F \subseteq P$ be a subset of the switchable edges that contains at most one outgoing edge from each vertex. The strategy $\sigma[F]$ is admissible, and we have $\sigma \prec \sigma[F]$.*

Secondly, we have that a strategy with no switchable edges is optimal in the valuation ordering. A strategy σ is optimal in the valuation ordering if there is no strategy σ' with $\mathrm{Val}^{\sigma}(v) < \mathrm{Val}^{\sigma'}(v)$ for some vertex v.

Theorem 3.7 ([BV07, Theorem 5.2]). *A strategy with no switchable edges is optimal in the valuation ordering.*

A solution to the zero mean partition problem can be derived from an optimal strategy σ: the set $W_{\mathrm{Max}} = \{v \in V \ : \ \mathrm{Val}^{\sigma}(v) = \infty\}$ and the set $W_{\mathrm{Min}} = \{v \in V \ : \ \mathrm{Val}^{\sigma}(v) < \infty\}$.

We now argue that this strategy improvement algorithm is strongly related to the gain and bias optimality equations. Suppose that we add the edge (s, s) to the sink vertex s, and consider the mean-payoff game played on this modified game. If σ is a strategy such that $\mathrm{Val}^{\sigma}(v) < \infty$ for some vertex v, then we would have $G^{\sigma}(v) = 0$ in the mean-payoff game, because the path chosen by σ and $\mathrm{br}(\sigma)$ ends at the sink, and the mean-payoff that is obtained from the sink vertex is 0. Now, if we rewrite the bias equation with the assumption that $G^{\sigma}(v) = 0$ we obtain:

$$B^{\sigma}(v) = \begin{cases} \mathrm{r}(v) + B^{\sigma}(\sigma(v)) & \text{if } v \in V_{\mathrm{Max}}, \\ \mathrm{r}(v) + B^{\sigma}(\mathrm{br}(\sigma)(v)) & \text{if } v \in V_{\mathrm{Min}}. \end{cases}$$

Therefore, if $\mathrm{Play}(v, \sigma, \mathrm{br}(\sigma)) = \langle v = v_0, v_1, \ldots, v_k, s, \ldots \rangle$, then we have:

$$B^{\sigma}(v) = B^{\sigma}(s) + \sum_{i=0}^{k} \mathrm{r}(v_i).$$

Since s is a one-vertex cycle, and we insist that some vertex on every cycle should

have the bias set to 0, we must have $B^\sigma(s) = 0$. Therefore, $B^\sigma(v) = \text{Val}^\sigma(v)$ for every vertex with a finite valuation. In summary, the Björklund-Vorobyov strategy improvement algorithm modifies the game so that the gain of every strategy is at least 0, and then uses a simplification of the bias equation for its valuation function.

3.2.4 Switching Policies

The section describes the final component of a strategy improvement algorithm. Strategy improvement allows for any subset of switchable edges to be switched in each iteration of strategy improvement. Clearly, in order to have a complete algorithm, we need a procedure that picks which edges should be switched in each iteration. We will call this procedure a *switching policy*. A simple switching policy can be thought of as a function that picks a subset of the edges or actions that are switchable in the current strategy. In the average-reward MDP setting, we will have a switching policy for picking gain switchable actions, and a switching policy for picking bias switchable actions.

The switching policies that have been studied so far work for both two player games and Markov decision processes, and the upper bounds for the running time of these switching policies that have been found are usually the same across the two models. On the other hand, the lower bounds that have been found are usually specific to a particular type model. We will indicate the scope of the lower and upper bound results as we present them. When we give formal definitions of these switching policies, we will use the game formulation. These definitions can easily be adapted for the MDP setting.

We begin by stating a trivial upper bound on the number of iterations that any strategy improvement algorithm can take to terminate. Since strategy improvement algorithms cannot consider the same strategy twice, the number of iterations is obviously bounded by the total number of positional strategies that can be considered by the algorithm. Let $\text{Degree}(v)$ denote the number of outgoing edges from

the vertex v in a two player game, or the number of outgoing actions from the vertex v in an MDP. Strategy improvement for two player games must terminate after at most $\prod_{v \in V_{\text{Max}}} \text{Degree}(v)$ iterations, and strategy improvement for MDPs must terminate after at most $\prod_{v \in V} \text{Degree}(v)$ iterations. These bounds hold no matter what choices the switching policy makes.

The simplest possible switching policy is to arbitrarily pick a single switchable edge in each iteration. This switching policy can be defined as $\text{Single}(F) = \{(v, u)\}$ where (v, u) is some edge contained in F. It has long been known that strategy improvement equipped with the single-switch policy can take exponential time. For games, the examples were originally found by Lebedev, and Björklund and Vorobyov adapted them to show an exponential lower bound on the number of iterations taken by their strategy improvement algorithm, when it is equipped with the single-switch policy [BV07]. For MDPs, Melekopoglou and Condon have shown an exponential lower bound for a single-switch strategy improvement using a very similar family of examples [MC94].

The most natural class of switching policies are all-switches policies. The idea here is that the strategy should be switched at every vertex that has a switchable edge. This defines a class of switching policies, because a vertex may have more than one switchable edge, and different switching policies may pick different edges to switch at each vertex. The most natural all-switches policy is the *greedy* switching policy, that always picks the edge with maximum appeal.

We formally define the greedy switching policy. We must be aware that there may be more than one edge that maximizes the appeal at a vertex. For the sake of simplicity, we will use an index function to break ties: we will assume that each vertex v is given a unique index in the range $\{1, 2, \ldots, |V|\}$, which we will denote as $\text{Index}(v)$. The set of edges that the greedy policy will pick for the strategy σ can

then be defined as follows:

$$\begin{aligned}\text{Greedy}^{\sigma}(F) = \{(v,u) \ : \ &\text{There is no edge } (v,w) \in F \text{ with} \\ &\text{Appeal}^{\sigma}(v,u) < \text{Appeal}^{\sigma}(v,w) \text{ or with} \\ &\text{Appeal}^{\sigma}(v,u) = \text{Appeal}^{\sigma}(v,w) \text{ and } \text{Index}(u) < \text{Index}(w)\}.\end{aligned}$$

The best upper bound that has been shown for strategy improvement algorithms equipped with the greedy policy is $O(2^n/n)$ iterations [MS99]. This upper bound holds for games and for MDPs. For many years, people were unable to find examples upon which strategy improvement equipped with the greedy policy took significantly more than a linear number of iterations to terminate. It was for this reason that greedy was conjectured to always terminate after a polynomial number of steps. However, in a breakthrough result, Friedmann found a family of parity games upon which the strategy improvement algorithm of Vöge and Jurdziński [VJ00] equipped with the greedy policy takes an exponential number of steps [Fri09, Fri10a]. It was later shown that this result can be generalised to prove an exponential lower bound for the strategy improvement for discounted games [And09]. It is also not difficult to adapt Friedmann's examples to produce a set of input instances upon which the Björklund-Vorobyov strategy improvement algorithm equipped with the greedy policy takes an exponential number of steps.

Perhaps the most intriguing type of switching policies are optimal switching policies. A switching policy is *optimal* if for every strategy σ it selects a subset of switchable edges F that satisfies $\text{Val}^{\sigma[H]}(v) \leq \text{Val}^{\sigma[F]}(v)$ for every set H that is a subset of the switchable edges in σ, and for every vertex v. It is not difficult to show that such a set of edges must always exist, however at first glance it would seem unlikely that such a set could be efficiently computed. Nevertheless, Schewe has given an algorithm that computes such a set in polynomial time for the Björklund-Vorobyov strategy improvement algorithm [Sch08]. Therefore, optimal switching

policies can realistically be implemented for solving parity and mean-payoff games. No analogue of this result is known for discounted games or for MDPs.

Although, using an optimal switching policy would seem likely to produce better results than the greedy policy, Friedmann has shown that his exponential time examples for the greedy policy can be adapted to provide a family of examples upon which an optimal switching policy will take an exponential number of steps [Fri10a]. Therefore, optimal switching policies perform no better than greedy switching policies in the worst case. It should be noted that the word optimal is used to describe the set of edges that an optimal switching policy chooses to switch. It does not imply that a strategy improvement algorithm equipped with an optimal policy will have an optimal running time. There may be switching policies that do not always make the maximal increase in valuations in every iteration, but still perform better in terms of worst case complexity.

Randomized switching policies have been shown to have better worst case complexity bounds. Mansour and Singh considered a switching policy that selects a subset of switchable edges uniformly at random [MS99]. They showed that this switching policy will terminate after expected $O(2^{0.78n})$ number of iterations for binary games, and an expected $O((1 + 2/\log k) \cdot k/2)^n)$ number of iterations for games with out-degree at most k. These upper bounds hold for both games and MDPs.

The switching policy with the best currently-known worst case complexity bound is the random-facet switching policy. This switching policy is based on the randomized simplex methods for linear programming given by Kalai [Kal92] and Matoušek, Sharir, and Welzl [MSW96]. The first switching policy based on this method was given by Ludwig [Lud95], which terminates after an expected $2^{O(\sqrt{n})}$ number of iterations. However, his switching policy only works for binary games. This shortcoming has been rectified by the switching policy described by Björklund and Vorobyov [BV05], which terminates after an expected $2^{O(\sqrt{n \log n})}$ number of

iterations for both games and MDPs.

In a recent result, Friedmann, Hansen, and Zwick have found a family of parity games upon which this bound is almost tight: the random-facet switching policy will take an expected $2^{\Omega(\sqrt{n}/\log n)}$ iterations to terminate [FHZ10]. This lower bound can be extended to cover the strategy improvement algorithm for discounted games, and the Björklund-Vorobyov strategy improvement algorithm for the zero mean partition. This lower bound is not known to hold for strategy improvement for MDPs.

3.3 Algorithms for LCPs

3.3.1 Lemke's Algorithm

Lemke's algorithm modifies the LCP with a positive *covering vector* d and a positive scalar z_0. The scalar z_0 becomes a new variable in the modified problem. The covering vector can be chosen to be any positive vector. The LCP is modified by replacing the matrix equation (2.1) with:

$$w = Mz + q + dz_0. \tag{3.1}$$

We are interested in *almost complementary* solutions to this modified system. A solution is almost complementary if it has n basic variables, and if there is at most one pair of complementary variables that are non-basic in the solution (recall that a variable is basic if it is allowed to be non-negative in the solution). Lemke's algorithm will compute a sequence of almost complementary solutions to the modified system.

An initial almost complementary solution can be found easily. Let $e = \min(q_i/d_i \ : \ 1 \leq i \leq n)$. If e is positive, then every element of q must be positive, so the LCP has a trivial solution, and the algorithm can terminate. Otherwise, if we set $z_0 = -e$, then we have that setting $w = q + dz_0$ and $z = 0$ is an almost

complementary solution. The two vectors clearly satisfy Equation 3.1 and they obviously satisfy the complementarity condition. They satisfy the non-negativity constraint, because for each variable w_i we have $w_i = q - d \cdot e \geq 0$. Moreover, for the index i where $q_i = e$ we have $w_i = q - d \cdot e = 0$ and $z_i = 0$. The algorithm proceeds by performing a pivot operation between the variable z_0 and the variable w_i.

For example, consider the following P-matrix linear complementarity problem:

$$M = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}, \qquad q = \begin{pmatrix} -2 \\ -3 \end{pmatrix}.$$

If we choose the covering vector to be $d_i = 1$ for every i, then we have that $\min(q_i/d_i \ : \ 1 \leq i \leq n)$ is -3. It can easily be verified that if z_0 is set to -3, then $q + d \cdot z_0$ is positive. Moreover, the second component of w is 0. The algorithm would therefore perform a pivot operation between the variable z_0 and the variable w_2.

This pivot operation is implemented as follows. To begin, the algorithm constructs the tableaux $[I, M, d, q]$, which is:

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 1 & -2 \\ 0 & 1 & 1 & 3 & 1 & -3 \end{bmatrix}.$$

The column associated with z_0 is the column that contains the covering vector d, and the column associated with w_2 is the second column of the identity matrix. These two columns are swapped to obtain the following tableaux:

$$\begin{bmatrix} 1 & 1 & 2 & 1 & 0 & -2 \\ 0 & 1 & 1 & 3 & 1 & -3 \end{bmatrix}.$$

The algorithm then performs Gauss-Jordan elimination to transform the first two columns back to an identity matrix. It can easily be seen that subtracting the

second row from the first row will achieve this. Therefore, the algorithm arrives at the following tableaux:

$$\begin{bmatrix} 1 & 0 & 1 & -2 & -1 & 1 \\ 0 & 1 & 1 & 3 & 1 & -3 \end{bmatrix}.$$

The new LCP can be read off from this tableaux, to give:

$$M = \begin{pmatrix} 1 & -2 \\ 1 & 3 \end{pmatrix}, \qquad d = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \qquad q = \begin{pmatrix} 1 \\ -3 \end{pmatrix}.$$

In this LCP, the variable z_0 is the second variable in w, and the variable w_2 is multiplied by the covering vector d. It can easily be verified that setting $w_1 = q_1$, $z_0 = 3$, and all other variables to 0 gives an almost complementary solution to this LCP.

We now discuss how Lemke's algorithm moves from one almost complementary solution to the next. Since each almost complementary solution has exactly n basic variables, one of which will be z_0, the solution must contain a complementary pair of variables w_i and z_i where both $w_i = 0$ and $z_i = 0$. One of these variables will be chosen to be the *driving variable*. In the next almost complementary solution, the driving variable will be basic, and some basic variable in the current solution will be made non-basic. The difficulty is choosing which basic variable in the current solution can be made non-basic while ensuring that we still have an almost complementary solution. In particular, we want to ensure that all basic variables in the next solution are positive.

In each step of the algorithm our almost complementary solution will have the following form:

$$x_B = M' x_N + q'.$$

Where x_B is the n dimensional vector of basic variables, x_N is the $n+1$ dimensional vector of nonbasic variables, M' is an n by $n+1$ matrix which is the result of applying

pivot operations to M and d, and q' is the result of applying pivot operations to q. Suppose that the driving variable n_i is the i-th component of x_N. In the next solution, we will allow n_i to become basic, which means that n_i will rise from 0 to some positive value. The question we are interested in is: how far can n_i be raised before some basic variable becomes negative? If e is the i-th column of M', then for each basic variable b_j we have:

$$b_j = e_j \cdot n_i + q'_j.$$

If e_j is positive then the value of the basic variable b_j will rise as the value of n_i rises. On the other hand, if e_j is negative then n_i can only be increased to $-q'_j/e_j$ before the variable b_j becomes negative.

This gives us the following *minimum ratio test* to decide which variable should be made non-basic:

$$\min\{-q'_j/e_j \ : \ j \in \{1, 2, \ldots n\} \text{ and } e_j < 0\}. \tag{3.2}$$

The minimum ratio test gives us the smallest amount that n_i can be increased before some basic variable becomes negative. The basic variable b_j that achieves this minimum is called the *blocking variable*. Lemke's algorithm will perform a pivot operation between the driving variable and the blocking variable.

After the pivot operation has taken place, we can set the driving variable be basic, and the blocking variable to be non-basic. We can then compute a solution to our new system of equations. By using the minimum ratio test, it is guaranteed that all of our basic variables will be positive in this new solution. Moreover, in the previous iteration there was exactly one complementary pair of variables that were non-basic. Since we chose one of those variables to be the driving variable, it follows that there can be at most one pair of complementary variables that are non-basic in the current iteration. Since z_0 is still a basic variable, we also have that there

is at least one pair of complementary variables that are non-basic. Therefore, we have arrived at an almost complementary solution, which can be taken into the next iteration.

To complete our description of the algorithm, we must specify which variable will be chosen to be the driving variable in each iteration. As we have noted, there will always be some complementary pair of variables w_i and z_i that are non-basic. In the first iteration, the algorithm selects z_i to be the driving variable. In each subsequent iteration, the algorithm selects the complement of the blocking variable from the previous iteration. The algorithm continues until z_0 is the blocking variable. After this occurs, the variable z_0 will become non-basic, and a solution to the original LCP is recovered.

Let us return to our example. In the first iteration the variables w_2 and z_2 are both non-basic. Since w_2 was pivoted with z_0, the algorithm will select z_2 to be the driving variable in this iteration. The column associated with z_2 is the second column of M', which means that the vector e will be $(-2, 3)$. Only the first element of this vector is negative, therefore, the variable w_1 must be chosen as the blocking variable in this iteration. Therefore, the variable z_2 will be exchanged with the variable w_1 in a pivot operation to obtain an LCP in which the first element of w is z_2 and the second element of w is z_0.

In general, Lemke's algorithm may not find a solution to the LCP. This is because there may not be a basic variable b_j such that e_j is negative, and therefore the minimum ratio test given in (3.2) is not well defined. In this situation, no basic variable will ever become negative, no matter how much n_j is increased. If Lemke's algorithm ever encounters such a situation, then it will give up, and terminate without a solution. This is called *ray termination*. Fortunately, this situation can never arise when the input matrix M is a P-matrix.

Proposition 3.8 ([CPS92]). *If M is a P-matrix, then for every q, Lemke's algorithm will terminate with a solution when it is applied to LCP(M, q).*

3.3.2 The Cottle-Dantzig Algorithm

The Cottle-Dantzig algorithm allows the non-negativity constraint to be broken. We say that a *partial solution* to the LCP is a pair of vectors w and z that satisfy the conditions given by (2.1) and (2.3), and may or may not satisfy the condition given by (2.2). The algorithm goes through a sequence of *major iterations*, where it attempts to modify its partial solution to satisfy the non-negativity constraint. Each major iteration begins with a partial solution where there are k indices at which either $w_i < 0$ or $z_i < 0$. The objective of the major iteration is to find a partial solution where there are strictly less than k indices at which either $w_i < 0$ or $z_i < 0$. Therefore, after n major iterations the algorithm will arrive at a partial solution that also satisfies the non-negativity constraint, which is a solution to the LCP.

Finding an initial partial solution is easy. Setting $w = q$ and $z = 0$ obviously satisfies the conditions given by (2.1) and (2.3). If every element of q is non-negative, then this also satisfies the non-negativity constraint, and the LCP has a trivial solution. Otherwise, for each partial solution α we define the set P_α, which contains every index i such that $w_i \geq 0$ and $z_i \geq 0$ in α. The objective of a major iteration is to find a partial solution β such that $P_\alpha \subset P_\beta$. This means that we want the non-negativity constraint to be satisfied at some complementary pair that does not satisfy this constraint in α. Moreover, we do not want the non-negativity constraint to be broken for any complementary pairs that already satisfy this constraint in α.

We now describe how a major iteration is implemented. The major iteration begins with some partial solution α, and it picks some basic variable b_i such that $i \notin P_\alpha$, to be the *distinguished variable*. Our goal is to find a partial solution in which this variable and its complement satisfy the non-negativity constraint. To achieve this, we temporarily violate the complementarity condition, by allowing both b_i and its complement to be basic variables. The idea is to increase the value of the complement of b_i until b_i becomes 0. At this point, the variable b_i becomes non-

basic, and compliance with the complementarity condition is restored. Moreover, since the complement of b_i was increased away from 0, both b_i and its complement satisfy the non-negativity constraint. The only problem is preventing variables in P_α from decreasing below 0.

To achieve this, the algorithm uses the same machinery as Lemke's algorithm. Whereas Lemke's algorithm pivoted between almost complementary solutions to the modified system, this algorithm pivots between almost complementary partial solutions to the original LCP. To be more precise, this algorithm pivots between partial solutions with n basic variables and n non-basic variables. The distinguished variable and its complement are always both basic variables in these solutions, and the solutions contain exactly one pair of complementary variables that are both non-basic. We can use the methods that are used in Lemke's algorithm to pivot between almost complementary partial solutions.

The first driving variable is chosen to be the complement of the distinguished variable. In each iteration we use a modified version of the minimum ratio test to find the blocking variable. As with Lemke's algorithm, each iteration will have a system of the form:

$$Ix_B = M'x_N + q',$$

where M' and q' are obtained through a sequence of pivots to M and q. If the driving variable is n_i, whose corresponding column in M' is e, then we use the following modified minimum ratio test to compute the blocking variable:

$$\min\{-q'_j/e_j \ : \ j \in \{1, 2, \ldots n\},\ e_j < 0 \text{ and } j \in P_\alpha\}.$$

This minimum ratio test is only concerned with basic variables b_j such that $b_j \geq 0$ in α. By using this minimum ratio test, we ensure that none of these variables ever have a negative value in the sequence of almost complementary partial solutions that we compute.

As with Lemke's algorithm, the blocking variable and the driving variable are exchanged in a pivot operation. The major iteration terminates when the distinguished variable is chosen as the blocking variable. After this has occurred, compliance with the complementarity condition will have been restored, and the major iteration iteration can terminate.

Chapter 4

Linear Complementarity Algorithms For Infinite Games

In this chapter, we study how pivoting algorithms for the linear complementarity problem behave on the LCPs that arise from Jurdziński and Savani's reduction from discounted games, which was described in Section 2.5.2. There are other reductions from games to the linear complementarity problem, such as the reductions from simple stochastic games [GR05, SV07], the reduction of Jurdziński and Savani gives particularly natural linear complementarity problems. It is for this reason that this chapter focuses on this reduction.

The reduction allows linear complementarity algorithms to be applied in order to solve discounted games. However, very little is known about their behaviour. Our goal in this chapter is to describe how these algorithms behave when the input is a discounted game. Moreover, we want to prove lower bounds for the running time of these algorithms.

The first part of this chapter describes how Lemke's algorithm and the Cottle-Dantzig algorithm behave when the input is a discounted game. This exposition will be in the form of an algorithm that works directly on the discounted game. We provide proofs of correctness and termination for these algorithms. These proofs do

not rely on the correctness and termination of the corresponding LCP algorithms. This exposition is for the benefit of readers who are interested in infinite games, but who may lack experience in the literature on the linear complementarity problem.

We then argue that the two algorithms are in fact the same as the two algorithms for LCPs. In order to achieve this, we argue that there is a direct correspondence between the steps made by the discounted game algorithm, and the steps made by the corresponding LCP algorithm. Therefore, any lower bound on the running time of the discounted game version of an algorithm also holds for the LCP version of that algorithm.

Finally, we show exponential lower bounds for both Lemke's algorithm and the Cottle-Dantzig algorithm when the input is a discounted game. It is known that both of these algorithms can take an exponential number of steps when the input is an arbitrary LCP, however this lower bound did not necessarily hold for the class of LCPs that arise from the reduction of Jurdziński and Savani. We describe a family of parity games upon which both algorithms take an exponential number of steps. Therefore, we establish lower bounds for parity, mean-payoff, and discounted games.

4.1 Algorithms

4.1.1 Lemke's Algorithm For Discounted Games

The key idea behind Lemke's algorithm is to consider modified versions of the input game. In particular, these games will be modified by adding or subtracting a parameter z_0 from the rewards on the left edges at each vertex. As with the LCP version of the algorithm, the user is allowed to pick a positive covering vector. For each $v \in V$ there will be a positive *covering value* which is denoted as d_v.

Definition 4.1 (Modified Game For Lemke's Algorithm)**.** *For each real number z, we define the game G_z to be the same as G but with a modified left-edge reward*

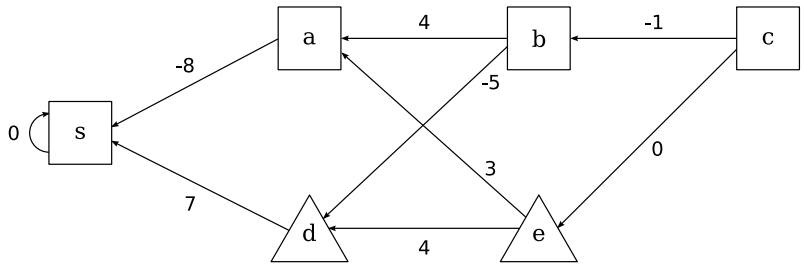

Figure 4.1: The discounted game that will be used to demonstrate Lemke's algorithm.

function, denoted by r_z^λ*, and defined, for every vertex* v*, by:*

$$r_z^\lambda(v) = \begin{cases} r^\lambda(v) - d_v \cdot z & v \in V_{Max}, \\ r^\lambda(v) + d_v \cdot z & v \in V_{Min}. \end{cases} \tag{4.1}$$

For each modified game G_z, the function r_z^σ will give the rewards on the edges chosen by a joint strategy σ. The notations Value_z^σ and Bal_z^σ will give the values and the balances of the vertices in the game G_z, respectively.

Lemke's algorithm begins with an arbitrarily chosen joint strategy σ_0. It then chooses a parameter z_0 so that σ_0 is an optimal strategy in the game G_{z_0}. The idea is to transform the modified game back to the original one, while always maintaining the invariant that the current strategy is optimal in the modified game. Therefore, the algorithm should produce a sequence of pairs $\langle(\sigma_0, G_{z_0}), (\sigma_1, G_{z_1}), \ldots (\sigma_k, G_{z_k})\rangle$, where $z_0 > z_1 > \cdots > z_k$, and $z_k = 0$. For each joint strategy σ_i, the following three properties will hold:

1. The strategy σ_i is optimal for the game G_{z_i}.

2. For every parameter $y < z_i$, the strategy σ_i is not optimal for the game G_y.

3. There is at least one vertex v with $\text{Bal}_{z_i}^{\sigma_i}(v) = 0$.

The first property ensures that the algorithm is correct, because the fact that $G = G_0$ implies that σ_k is an optimal strategy for the original game. The second property en-

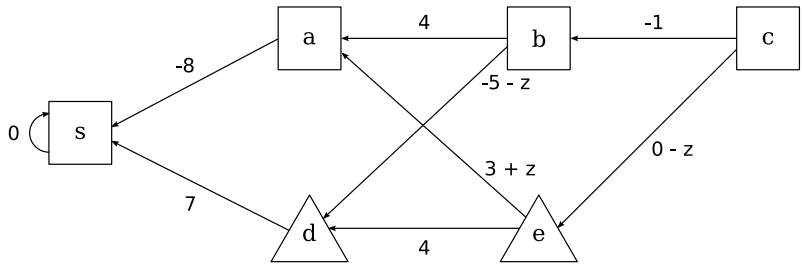

Figure 4.2: The game G_z where G is the game shown in Figure 4.1.

sures that the algorithm terminates, because once a strategy σ_i has been considered, it can never satisfy the first property again.

Throughout this section, we will use the example shown in Figure 4.1 to demonstrate how Lemke's algorithm works. The discount factor of this game is chosen to be $\beta = 0.9$. The right-hand edges are specified as follows: we have $\rho(b) = a$, $\rho(c) = b$, and $\rho(e) = d$. The figure only shows one outgoing edge for the vertices a, d, and s, but this edge is duplicated to ensure that the game is a binary game. For example, we have $\lambda(a) = \rho(a) = s$, and we have $r^{\lambda}(a) = r^{\rho}(a) = -8$. It is important to note that the strategy decision at these three vertices is irrelevant, because the same value is obtained no matter what edge is chosen at these vertices, and the balance will always be 0. Therefore, we are trying to find an optimal strategy for the vertices b, c, and e. We will use the unit covering vector in our exposition, which means that $d_v = 1$ for every vertex v.

We begin by describing how the algorithm finds the initial pair (σ_0, z_0). We will pick the joint strategy $\sigma_0 = \rho$ to be the strategy that selects the right successor for every vertex in the game. This allows the user to pick an arbitrary starting strategy by swapping the left and right edges at each vertex. Let us consider how the parameter z_0 can be found for the game shown in Figure 4.1. The modification of this game is shown in Figure 4.2. We must pick a parameter that makes the joint strategy ρ optimal, and we know that in an optimal strategy no vertex has a negative balance. We can derive the following expression for the balance of the

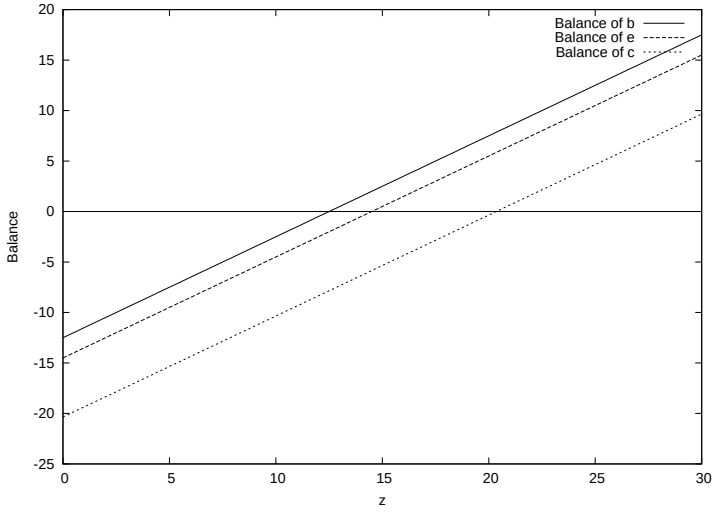

Figure 4.3: The balance of each vertex under the joint strategy ρ in the game G_z.

vertex b in the game G_z:

$$\mathrm{Bal}_z^\rho(b) = -4 - \beta \cdot 8 - (-5 - z + \beta \cdot 7) = -12.5 + z.$$

Therefore, if we set the parameter $z = 12.5$, the balance at the vertex b will not be negative. We can derive similar expressions for the vertices c and e:

$$\mathrm{Bal}_z^\rho(c) = -20.35 + z,$$
$$\mathrm{Bal}_z^\rho(e) = -14.5 + z.$$

Figure 4.3 shows a plot of these three functions. It can clearly be seen that setting $z_0 = 20.35$ will cause the pair (ρ, z_0) to satisfy the three properties that we are after. The joint strategy ρ is optimal in the game G_{z_0} because the balance of all three vertices is non-negative. For every parameter $y < z_0$, we have that ρ is not an optimal strategy in the game G_y, because the balance of c will be negative in the game G_y. Finally, the vertex c is switchable when ρ is played in G_{z_0}, because it has a balance of 0.

In general, the algorithm can initialise itself by selecting z_0 to be the negative balance with the largest magnitude. If σ_0 happens to be an optimal strategy for the original game, then the algorithm can set $z_0 = 0$ and terminate. Otherwise, the algorithm picks $z_0 = \max\{-\operatorname{Bal}^{\sigma_0}(v)/d_v \ : \ v \in V\}$. We can prove that the pair (σ_0, z_0) will satisfy the three properties.

Proposition 4.2. *Let $z_0 = \max\{-\operatorname{Bal}^{\sigma_0}(v)/d_v \ : \ v \in V\}$. The joint strategy σ_0 is optimal in G_{z_0} and the vertex v in V that maximizes $-\operatorname{Bal}^{\sigma_0}(v)$ is indifferent. Moreover, there is no value $y < z_0$ for which σ_0 is optimal in G_y.*

Proof. We begin by arguing that $\operatorname{Value}^{\sigma_0}_{z_0}(v) = \operatorname{Value}^{\sigma_0}(v)$ for every vertex v. This holds because the modifications made in Definition 4.1 only change the rewards on the left edges, and by definition σ_0 only chooses right edges. Therefore, we have $r^{\sigma}_{z_0}(v) = r^{\sigma}(v)$ for every vertex v. It is then easy to see that $\operatorname{Value}^{\sigma_0}_{z_0}(v) = \operatorname{Value}^{\sigma_0}(v)$ by noticing that the characterisation of $\operatorname{Value}^{\sigma}(v)$ given in (4.2) uses only the rewards $r^{\sigma}(v)$.

We now prove that σ_0 is optimal in G_y, by arguing that $\operatorname{Bal}^{\sigma_0}_{z_0}(v) \geq 0$ for every state v. Applying the knowledge that we have gained in the previous paragraph, along with the definition of $r^{\lambda}_{z_0}$ gives, for every Max vertex v:

$$\begin{aligned}
\operatorname{Bal}^{\sigma_0}_{z_0}(v) &= \operatorname{Value}^{\sigma_0}_{z_0}(v) - (r^{\overline{\sigma_0}}_{z_0}(v) + \beta \cdot \operatorname{Value}^{\sigma_0}_{z_0}(\overline{\sigma}(v))) \\
&= \operatorname{Value}^{\sigma_0}(v) - (r^{\lambda}_{z_0}(v) + \beta \cdot \operatorname{Value}^{\sigma_0}(\overline{\sigma}(v))) \\
&= \operatorname{Value}^{\sigma_0}(v) - (r^{\lambda}(v) - d_v \cdot z_0 + \beta \cdot \operatorname{Value}^{\sigma_0}(\overline{\sigma}(v))) \\
&= \operatorname{Bal}^{\sigma_0}(v) + d_v \cdot z_0.
\end{aligned}$$

A similar derivation can be made in order to obtain an identical expression for every Min vertex. Let x be a vertex such that $z_0 = -\operatorname{Bal}^{\sigma_0}(x)/d_v$, and note that by assumption we have that $z_0 > 0$. If $\operatorname{Bal}^{\sigma_0}(v) \geq 0$ then clearly $\operatorname{Bal}^{\sigma_0}_{z_0}(v) > 0$. If $\operatorname{Bal}^{\sigma_0}(v) < 0$, then by choice of z_0, we have that $\operatorname{Bal}^{\sigma_0}(x) \leq \operatorname{Bal}^{\sigma_0}(v)$, and therefore we also have $\operatorname{Bal}^{\sigma_0}_{z_0}(v) > 0$.

To prove that there is at least one vertex that is indifferent, we again consider the vertex x such that $z_0 = -\operatorname{Bal}^{\sigma_0}(x)/d_v$. For this vertex we have:

$$\operatorname{Bal}^{\sigma_0}_{z_0}(x) = \operatorname{Bal}^{\sigma_0}(x) + d_v \cdot z_0 = \operatorname{Bal}^{\sigma_0}(x) - d_v \cdot \frac{\operatorname{Bal}^{\sigma_0}(x)}{d_v} = 0.$$

Therefore, the vertex x is indifferent when σ_0 is played in G_{z_0}. To finish the proof we must argue that σ_0 is not optimal in every game G_y where $y < z_0$. At the vertex x we have:

$$\operatorname{Bal}^{\sigma_0}_{y}(x) = \operatorname{Bal}^{\sigma_0}(x) + d_v \cdot y < \operatorname{Bal}^{\sigma_0}(x) + d_v \cdot z_0 = 0.$$

Therefore, the vertex x will be switchable when σ_0 is played on G_y, and Corollary 2.9 therefore implies that σ_0 is not optimal for G_y. □

We now describe how the algorithm moves from the pair (σ_i, G_{z_i}) to the pair $(\sigma_{i+1}, G_{z_{i+1}})$. We begin by describing how σ_{i+1} is computed. By assumption there is some vertex w that is indifferent when σ_i is played in G_{z_i}. We *switch* this vertex to create σ_{i+1}. For every vertex v we define:

$$\sigma_{i+1}(v) = \begin{cases} \overline{\sigma_i}(v) & \text{if } v = w, \\ \sigma_i(v) & \text{otherwise.} \end{cases}$$

It is easy to see from the optimality equations that switching an indifferent vertex in an optimal strategy will produce another optimal strategy. Therefore, σ_{i+1} is also an optimal strategy for G_{z_i}.

The next task is to find the value z_{i+1} that is the smallest value for which σ_{i+1} is an optimal strategy in $G_{z_{i+1}}$. Let us return to the example shown in Figure 4.1. The vertex c was indifferent when σ_0 was played in G_{z_0}, and so this vertex will be switched to create the joint strategy σ_1. Figure 4.4 shows the function $\operatorname{Bal}^{\sigma_1}_z$ for each of the vertices. Since the vertex c has been switched, the function $\operatorname{Bal}^{\sigma_1}_z(c)$ has been reflected in the axis. This is because the balance of a vertex is computed as

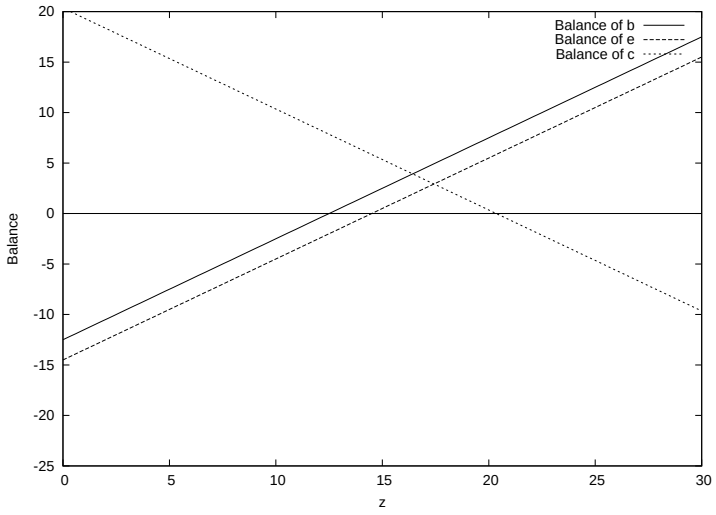

Figure 4.4: The balance of each vertex under the joint strategy σ_1 in the game G_z.

the difference between the value of the vertex and the value of the alternate successor. By switching this vertex, we cause these two quantities to be interchanged. Therefore, we have:

$$\mathrm{Bal}_z^{\sigma_1}(c) = -z + \beta \cdot 4 + \beta^2 \cdot 7 - (-1 + \beta \cdot -4 + \beta^2 \cdot -8) = 20.35 - z.$$

This means that the value of c will now rise as z is decreased instead of falling.

The plot shown in Figure 4.4 gives a good outline of the situation that occurs after a vertex has been switched. There is a range of values $z \in [a, b]$ for which the joint strategy σ_1 is optimal in G_z. We know that the upper bound of this range is z_0, and our task is to compute the lower bound. This occurs when the balance of some other vertex falls below 0. It can also be seen that the joint strategy σ_1 is not optimal in G_z for every value of z that lies outside this range. This is because there is always some vertex that has a negative balance.

To compute z_{i+1} we will need to understand, for every vertex v, the rate at which $\mathrm{Value}_z^{\sigma_{i+1}}(v)$ changes as z is decreased. For a joint joint strategy σ, we denote the rate of change of the value as z decreases as $\partial_{-z} \mathrm{Value}_z^{\sigma}(v)$, which is equal to

$-\partial_z \operatorname{Value}_z^\sigma(v)$. It will turn out that this relationship is linear, this means that if we decrease z by a constant c then:

$$\operatorname{Value}_{z-c}^\sigma(v) - \operatorname{Value}_z^\sigma(v) = \partial_{-z} \operatorname{Value}_z^\sigma(v) \cdot c.$$

In our running example, we would have $\partial_{-z} \operatorname{Value}_z^{\sigma_1}(c) = 1$. This is because $\operatorname{Value}_z^{\sigma_1}(c)$ can be rewritten as

$$\operatorname{Value}_z^{\sigma_1}(c) = -z + \beta \cdot 4 + \beta^2 \cdot 7 = 9.27 - z.$$

Therefore, as z is decreased, the value of the vertex c will increase.

We now explain how a formula for $\operatorname{Value}_z^\sigma(v)$ can be derived for every joint strategy σ. This formula will be based on relationship between the rewards that an infinite path passes through, and the payoff of that path. If a play begins at a vertex v and follows a positional joint strategy σ then we know that the resulting infinite path can be represented by a simple path followed by an infinitely repeated cycle. If $\operatorname{Play}(\sigma, v) = \langle v_0, v_1, \ldots, v_{k-1}, \langle c_0, c_1, \ldots, c_{l-1} \rangle^\omega \rangle$, then it is then easy to see that:

$$\operatorname{Value}^\sigma(v) = \sum_{i=0}^{k-1} \beta^i \cdot r^\sigma(v_i) + \sum_{i=0}^{l-1} \frac{\beta^{k+i}}{1-\beta^l} \cdot r^\sigma(c_i). \tag{4.2}$$

We are interested in how the rewards given by $r^\sigma(v_i)$ and $r^\sigma(c_i)$ affect $\operatorname{Value}^\sigma(v)$. Specifically, we will frequently want to know how much $\operatorname{Value}^\sigma(v)$ changes when we increase or decrease one of these rewards. This will be given by the contribution coefficient.

Definition 4.3 (Contribution Coefficient)**.** *For vertices v and u, and for a positional*

joint strategy σ, we define:

$$\mathrm{D}^v_\sigma(u) = \begin{cases} \beta^i & \text{if } u = v_i \text{ for some } 0 \le i < k, \\ \frac{\beta^{k+i}}{1-\beta^l} & \text{if } u = c_i \text{ for some } 0 \le i < l, \\ 0 & \text{otherwise.} \end{cases}$$

Using this notation, we have that $\mathrm{Value}^\sigma(v) = \sum_{u \in V} \mathrm{D}^v_\sigma(u) \cdot r^\sigma(v_i)$. Therefore, if $r^\sigma(u)$ is increased by δ then $\mathrm{Value}^\sigma(v)$ will be increased by $\mathrm{D}^v_\sigma(u) \cdot \delta$. This fact is used in the next proposition to give a characterisation of $\partial_{-z} \mathrm{Value}^\sigma_z(v)$ for every joint strategy σ. For a proposition p, we define $[p]$ to be equal to 1 if p is true, and 0 otherwise.

Proposition 4.4. *For a vertex v and a joint strategy σ, let L be the set of vertices for which σ picks the left successor, $L = \{v \in V \ : \ \sigma(v) = \lambda(v)\}$. The rate of change of the value of v is:*

$$\partial_{-z} \mathrm{Value}^\sigma_z(v) = \sum_{u \in L} ([u \in V_{Max}] - [u \in V_{Min}]) \cdot d_v \cdot \mathrm{D}^v_\sigma(u).$$

Proof. Using the definition of r^λ_z given in (4.1), gives the following formula for the rewards on the edges chosen by σ. For every vertex v we have:

$$r^\sigma_{z-c}(v) = \begin{cases} r^\sigma_z(v) + d_v \cdot c & \text{if } v \in V_{\mathrm{Max}} \text{ and } \sigma(v) = \lambda(v), \\ r^\sigma_z(v) - d_v \cdot c & \text{if } v \in V_{\mathrm{Min}} \text{ and } \sigma(v) = \lambda(v), \\ r^\sigma_z(v) & \text{otherwise.} \end{cases}$$

We can now apply the characterisation of the value given by (4.2) to obtain:

$$\begin{aligned}
&\mathrm{Value}^{\sigma}_{z-c}(v) - \mathrm{Value}^{\sigma}_{z}(v) \\
&= \sum_{u \in V} \mathrm{D}^{v}_{\sigma}(u) \cdot r^{\sigma}_{z-c}(u) - \mathrm{Value}^{\sigma}_{z}(v) \\
&= \sum_{u \in V} \mathrm{D}^{v}_{\sigma}(u)(r^{\sigma}_{z}(u) + [u \in V_{\mathrm{Max}}] \cdot d_v \cdot c - [u \in V_{\mathrm{Min}}] \cdot d_v \cdot c) - \mathrm{Value}^{\sigma}_{z}(v) \\
&= \mathrm{Value}^{\sigma}_{z}(v) + \sum_{u \in L} ([u \in V_{\mathrm{Max}}] - [u \in V_{\mathrm{Min}}])\, \mathrm{D}^{v}_{\sigma}(u) \cdot d_v \cdot c - \mathrm{Value}^{\sigma}_{z}(v) \\
&= \sum_{u \in L} ([u \in V_{\mathrm{Max}}] - [u \in V_{\mathrm{Min}}])\, \mathrm{D}^{v}_{\sigma}(u) \cdot d_v \cdot c. \qquad \square
\end{aligned}$$

Now that we know how the value changes as z is decreased, we can use this to compute the rate of change of the balance as z is decreased. We will denote this, for each vertex v, as $\partial_{-z} \mathrm{Bal}^{\sigma}_{z}(v)$. As with our notation for the rate of change of the value, we will have:

$$\mathrm{Bal}^{\sigma}_{z-c}(v) - \mathrm{Bal}^{\sigma}_{z}(v) = \partial_{-z} \mathrm{Bal}^{\sigma}_{z}(v) \cdot c. \tag{4.3}$$

In our example, we would have $\partial_{-z} \mathrm{Bal}^{\sigma_1}_{z}(v) = 1$, because we have already shown that $\mathrm{Bal}^{\sigma_1}_{z}(c) = 20.35 - z$. The next proposition gives a characterisation of $\mathrm{Bal}^{\sigma}_{z-c}(v)$ for every vertex v and every joint strategy σ.

Proposition 4.5. *For a vertex v and a joint strategy σ, if $v \in V_{Max}$ then:*

$$\partial_{-z} \mathrm{Bal}^{\sigma}_{z}(v) = \partial_{-z} \mathrm{Value}^{\sigma}_{z}(v) - ([\overline{\sigma}(v) = \lambda(v)] \cdot d_v + \beta \cdot \partial_{-z} \mathrm{Value}^{\sigma}_{z}(\overline{\sigma}(v))),$$

and if $v \in V_{Min}$ then:

$$\partial_{-z} \mathrm{Bal}^{\sigma}_{z}(v) = -[\overline{\sigma}(v) = \lambda(v)] \cdot d_v + \beta \cdot \partial_{-z} \mathrm{Value}^{\sigma}_{z}(\overline{\sigma}(v)) - \partial_{-z} \mathrm{Value}^{\sigma}_{z}(v).$$

Proof. We will prove the claim for a vertex $v \in V_{\mathrm{Max}}$. The proof for vertices $v \in V_{\mathrm{Min}}$

is symmetric. The following is obtained by using the definition of Bal^σ, rearrangement, and by using the definition of $\partial_{-z}\,\mathrm{Value}^\sigma_z$.

$$\begin{aligned}
&\mathrm{Bal}^\sigma_{z-c}(v) - \mathrm{Bal}^\sigma_z(v)\\
=&(\mathrm{Value}^\sigma_{z-c} - (r^{\overline{\sigma}}_{z-c}(v) + \beta \cdot \mathrm{Value}^\sigma_{z-c}(\overline{\sigma}(v)))) - (\mathrm{Value}^\sigma_z - (r^{\overline{\sigma}}_z(v) + \beta \cdot \mathrm{Value}^\sigma_z(\overline{\sigma}(v))))\\
=&(\mathrm{Value}^\sigma_{z-c} - \mathrm{Value}^\sigma_z(v)) - (r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v) + \beta \cdot (\mathrm{Value}^\sigma_{z-c}(\overline{\sigma}(v)) - \mathrm{Value}^\sigma_z(\overline{\sigma}(v)))\\
=&\partial_{-z}\,\mathrm{Value}^\sigma_z(v) \cdot c - (r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v) + \beta \cdot \partial_{-z}\,\mathrm{Value}^\sigma_z(\overline{\sigma}(v)) \cdot c).
\end{aligned}$$

To obtain the required form, we must deal with the expression $r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v)$. If $\overline{\sigma}(v) = \rho(v)$, then we have that $r^{\overline{\sigma}}_{z-c}(v) = r^{\overline{\sigma}}_z(v)$, and therefore $r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v) = 0$. On the other hand, if $\overline{\sigma}(v) = \lambda(v)$, then we have $r^{\overline{\sigma}}_{z-c}(v) = r^{\overline{\sigma}}_z(v) + d_v \cdot c$, and therefore $r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v) = d_v \cdot c$. From these two facts, we can conclude that $r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v) = [\overline{\sigma}(v) = \lambda(v)] \cdot d_v \cdot c$. This characterisation can be used to complete the proof.

$$\begin{aligned}
&\partial_{-z}\,\mathrm{Value}^\sigma_z(v) \cdot c - (r^{\overline{\sigma}}_{z-c}(v) - r^{\overline{\sigma}}_z(v) + \beta \cdot \partial_{-z}\,\mathrm{Value}^\sigma_z(\overline{\sigma}(v)) \cdot c)\\
=&\partial_{-z}\,\mathrm{Value}^\sigma_z(v) \cdot c - ([\overline{\sigma}(v) = \lambda(v)] \cdot d_v \cdot c + \beta \cdot \partial_{-z}\,\mathrm{Value}^\sigma_z(\overline{\sigma}(v)) \cdot c)\\
=&(\partial_{-z}\,\mathrm{Value}^\sigma_z(v) - ([\overline{\sigma}(v) = \lambda(v)] \cdot d_v + \beta \cdot \partial_{-z}\,\mathrm{Value}^\sigma_z(\overline{\sigma}(v)))) \cdot c. \qquad \square
\end{aligned}$$

Now that we have a characterisation of $\partial_{-z}\,\mathrm{Bal}^\sigma_z(v)$, we can explain how it will be used. The following is a simple rearrangement of (4.3).

$$\mathrm{Bal}^\sigma_{z-c}(v) = \mathrm{Bal}^\sigma_z(v) + \partial_{-z}\,\mathrm{Bal}^\sigma_z(v) \cdot c. \tag{4.4}$$

From this it can be seen that if $\partial_{-z}\,\mathrm{Bal}^\sigma_z(v)$ is positive for some vertex v, then the balance of v under the strategy σ will increase as z is decreased. Likewise, if $\partial_{-z}\,\mathrm{Bal}^\sigma_z(v)$ is negative, then the balance of v under the strategy σ will decrease as z is decreased.

To find the first vertex whose balance falls below 0, we only need to consider the vertices v for which $\partial_{-z} \operatorname{Bal}_z^{\sigma}(v)$ is negative. Then, for each vertex v we have that the following ratio gives the amount that z_i can be decreased by in order to for the balance of v to be 0.

$$\frac{\operatorname{Bal}_{z_i}^{\sigma_{i+1}}(v)}{-\partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v)}.$$

Therefore, to find the smallest amount by which z can be decreased, we simply take the minimum over these ratios. The next proposition shows that this minimum ratio test is correct.

Proposition 4.6. *Suppose that σ_{i+1} is an optimal joint strategy for the game G_{z_i}. We define:*

$$z_{i+1} = z_i - \min\{\frac{\operatorname{Bal}_{z_i}^{\sigma_{i+1}}(v)}{-\partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v)} \ : \ v \in V \text{ and } \partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v) < 0\}. \tag{4.5}$$

The following three properties hold.

1. *The strategy σ_{i+1} is optimal for the game $G_{z_{i+1}}$.*
2. *For every parameter $y < z_{i+1}$ the strategy σ_i is not optimal for the game G_y.*
3. *There is at least one vertex v with $\operatorname{Bal}_{z_{i+1}}^{\sigma_{i+1}}(v) = 0$.*

Proof. We begin by proving the first property. We can ignore the vertices v with $\partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v) \geq 0$, because the balance of these vertices will not decrease as we decrease the parameter. Let v be a vertex with $\partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v) < 0$, and let $c_v = \operatorname{Bal}_{z_i}^{\sigma_{i+1}}(v) / -\partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v)$. Substituting this into (4.4) gives:

$$\operatorname{Bal}_{z_i - c_v}^{\sigma^{i+1}}(v) = \operatorname{Bal}_{z_i}^{\sigma^{i+1}}(v) - \operatorname{Bal}_{z_i}^{\sigma^{i+1}}(v) = 0.$$

Clearly, for every value $c < c_v$ we have $\operatorname{Bal}_{z_i - c}^{\sigma^{i+1}}(v) > 0$. Therefore, by setting $c = \min\{\operatorname{Bal}_{z_i}^{\sigma_{i+1}}(v) / -\partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v) \ : \ v \in V \text{ and } \partial_{-z} \operatorname{Bal}_z^{\sigma_{i+1}}(v) < 0\}$ we ensure

that no vertex has a balance below 0 in the game $G_{z_{i+1}}$. Therefore σ_{i+1} is optimal in the game $G_{z_{i+1}}$.

To prove the third property, it suffices to note that there is some vertex x for which $c_x = c$. Therefore, we have $\mathrm{Bal}_{z_{i+1}}^{\sigma^{i+1}}(x) = 0$. This also proves the second property, because for every value $y < z_{i+1}$ we have $\mathrm{Bal}_{y}^{\sigma^{i+1}}(x) < 0$. □

It is possible that there is no value $y < z_i$ for which σ_{i+1} is optimal in G_y. In this case we will have $z_{i+1} = z_i$. This is known as a *degenerate* step. We will postpone discussion about degeneracy until Section 4.1.3, and for now we will assume that no degenerate steps occur during the execution of the algorithm. This implies that for every i we have $z_{i+1} < z_i$.

Algorithm 2 Lemke(G)

$i := 0$; $\sigma_0 := \rho$; $z_0 := \max\{-\mathrm{Bal}^{\sigma_0}(v) \ : \ v \in V\}$
while $z_i > 0$ **do**
  $\sigma_{i+1} := \sigma_i[\overline{\sigma}_i(v)/v]$ for some vertex v with $\mathrm{Bal}_{z_i}^{\sigma_i}(v) = 0$
  $z_{i+1} := z_i - \min\{\frac{\mathrm{Bal}_{z_i}^{\sigma_{i+1}}(v)}{-\partial_{-z}\,\mathrm{Bal}_{z}^{\sigma_{i+1}}(v)} \ : \ v \in V \text{ and } \partial_{-z}\,\mathrm{Bal}_{z}^{\sigma_{i+1}}(v) < 0\}$
  $i := i + 1$
end while

Lemke's algorithm is shown as Algorithm 2. Since in each step we know that there is no value $y < z_i$ for which σ_i is optimal in G_y and $z_{i+1} < z_i$, it follows that we can never visit the same strategy twice without violating the condition that the current strategy should be optimal in the modified game. Therefore the algorithm must terminate after at most $2^{|V|}$ steps, which corresponds to the total number of joint strategies. The algorithm can only terminate when z has reached 0, and G_0 is the same game as G. It follows that whatever strategy the algorithm terminates with must be optimal in the original game. Therefore, we have shown following theorem.

Theorem 4.7. *Algorithm 2 terminates, with a joint strategy σ that is optimal for G after at most $2^{|V|}$ iterations.*

4.1.2 The Cottle-Dantzig Algorithm For Discounted Games

The reader should be aware that in this section we will override many of the notations that were used to describe Lemke's algorithm. In the future, whenever these notations are used the algorithm that is being referred to will always be clear from the context.

The Cottle-Dantzig algorithm begins with an arbitrary joint strategy σ_0, and it produces a sequence of joint strategies $\langle \sigma_0, \sigma_1, \ldots, \sigma_k \rangle$. The process of moving from σ_i to σ_{i+1} is called a *major* iteration. Let $P_i = \{v \in V \ : \ \text{Bal}^{\sigma_i}(v) \geq 0\}$ denote the set of vertices with non-negative balance in σ_i. The key property of a major iteration is that $P_i \subset P_{i+1}$. This means that at least one vertex with a negative balance in σ_i will have a non-negative balance in σ_{i+1}, and that every vertex with a non-negative balance in σ_i still has a non-negative balance in σ_{i+1}. Since $P \subset P_{i+1}$ for every i, the algorithm can go through at most $|V|$ iterations before finding a joint strategy σ_j for which $P_j = V$. By Corollary 2.9, we have that σ_j is an optimal strategy.

The bulk of this section will be dedicated to describing how a major iteration is carried out. A major iteration begins with a joint strategy σ_i. The algorithm then picks a vertex v with $\text{Bal}^{\sigma_i}(v) < 0$ to be the *distinguished vertex*. Throughout this section, we will denote this vertex as d. The distinguished vertex will have the property $\text{Bal}^{\sigma_{i+1}}(d) \geq 0$, and we will therefore have $P_i \cup \{d\} \subseteq P_{i+1}$. Once a distinguished vertex has been chosen, the algorithm then temporarily modifies the game, by adding a bonus to the edge chosen by σ_{i+1} at d. This modification will last only for the duration of the current major iteration.

Definition 4.8 (Modified Game For The Cottle-Dantzig Algorithm)**.** *For a rational number w, a joint strategy σ, and a distinguished vertex d, we define the game G_w to be the same as G but with a different reward on the edge chosen by σ at d. If σ chooses the left successor at d then the left reward function is defined, for every u*

in V, by:

$$r_w^\lambda(u) = \begin{cases} r^\lambda(u) + w & \text{if } u = d \text{ and } u \in V_{Max}, \\ r^\lambda(u) - w & \text{if } u = d \text{ and } u \in V_{Min}, \\ r^\lambda(u) & \text{otherwise.} \end{cases}$$

If σ chooses the right successor at d then r^ρ modified in a similar manner.

The reward on the edge chosen by σ at v in the game G_w is denoted as $r_w^\sigma(v)$. For a joint strategy σ and a vertex v, the value and balance of v for the strategy σ in the game G_w are denoted as $\text{Value}_w^\sigma(v)$ and $\text{Bal}_w^\sigma(v)$, respectively.

To see why the game is modified in this way, it is useful to look at the balance of d in the modified game. The properties that we describe will hold no matter who owns d, but for the purposes of demonstration we assume that $d \in V_{\text{Max}}$. The balance of d for σ_i in G_w is then:

$$\text{Bal}_w^{\sigma_i}(d) = \text{Value}_w^{\sigma_i}(d) - (r_w^{\overline{\sigma_i}}(v) + \beta \cdot \text{Value}_w^{\sigma_i}(\overline{\sigma_i}(v))).$$

Since $r_w^{\sigma_i}(v) = r^{\sigma_i}(v) + w$, we can see that $\text{Value}_w^{\sigma_i}(d)$ must increase as w is increased. It may also be the case that $\text{Value}_w^{\sigma_i}(\overline{\sigma_i}(v))$ increases as w is increased, however due to discounting, it must increase at a slower rate than $\text{Value}_w^{\sigma_i}(d)$. Therefore, as w is increased $\text{Bal}_w^{\sigma_i}(d)$ will also increase.

The algorithm will use machinery that is similar to the methods employed by Lemke's algorithm. In each major iteration the algorithm will produce a sequence of pairs $\langle(\sigma_i = \sigma_{i,0}, w_0), (\sigma_{i,1}, w_1), \ldots, (\sigma_{i,k}, w_k)\rangle$, with $w_0 < w_1 < \cdots < w_k$, which satisfies the following properties.

1. For every vertex $v \in P_i$ we have $\text{Bal}_{w_j}^{\sigma_{i,j}}(v) \geq 0$.
2. For every value $y > w_j$ there is some vertex $v \in P_i$ with $\text{Bal}_{w_j}^{\sigma_{i,j}}(v) < 0$.
3. There is some vertex $v \in P_i$ with $\text{Bal}_{w_j}^{\sigma_{i,j}}(v) = 0$.

Much like in Lemke's algorithm, the first property ensures correctness, by never allowing a vertex in P_i to have a negative balance, and the second property ensures termination, by preventing the algorithm from considering the same joint strategy twice. The third property ensures that there is always some vertex $v \in P_i$ in $\sigma_{i,j}$ that can be switched to produce $\sigma_{i,j+1}$.

Each step of a major iteration begins with a joint strategy σ_i, and value w_{i-1}, for which σ_i satisfies the first property in $G_{w_{i-1}}$. For σ_0, we can use $w_{-1} = 0$ to obtain this property. Much like in Lemke's algorithm, we want to compute the value $w_i = w_{i-1} + c$ that satisfies all of the properties. We therefore need to know the rate at which the balance of a vertex increases as we increase c. For each joint strategy σ, we denote this as $\partial_w \operatorname{Value}^\sigma_w(u)$, with the understanding that:

$$\operatorname{Value}^\sigma_{w+c}(v) - \operatorname{Value}^\sigma_w(v) = \partial_w \operatorname{Value}^\sigma_w(v) \cdot c.$$

The following proposition is a trivial consequence of the characterisation of $\operatorname{Value}^\sigma$ given by (4.2).

Proposition 4.9. *Consider a vertex u and a joint strategy σ. Suppose that v is the distinguished vertex. The rate of change $\partial_w \operatorname{Value}^\sigma_w(u)$ is $\mathrm{D}^u_\sigma(v)$.*

Once again, we define the rate of change of the balance of a vertex v in a joint strategy σ to be $\partial_w \operatorname{Bal}^\sigma_w(v)$, with the understanding that:

$$\operatorname{Bal}^\sigma_{w+c}(v) - \operatorname{Bal}^\sigma_w(v) = \partial_w \operatorname{Bal}^\sigma_w(v) \cdot c.$$

We can obtain an expression for $\partial_w \operatorname{Bal}^\sigma_w(v)$ by substituting the result of Proposition 4.9 into the definition of balance given in (2.12).

Proposition 4.10. *Consider a vertex u and a joint strategy σ in the game G_w such*

that σ chooses the edge with the bonus at d. The rate of change $\partial_w \operatorname{Bal}_w^\sigma(u)$ is:

$$\partial_w \operatorname{Bal}_w^\sigma(u) = \begin{cases} \partial_w \operatorname{Value}_w^\sigma(u) - \beta \cdot \partial_w \operatorname{Value}_w^\sigma(\overline{\sigma}(u)) & \text{if } u \in V_{Max}, \\ \beta \cdot \partial_w \operatorname{Value}_w^\sigma(\overline{\sigma}(u)) - \partial_w \operatorname{Value}_w^\sigma(u) & \text{if } u \in V_{Min}. \end{cases}$$

Proof. The proof is very similar to the proof of Proposition 4.5. The proof of this proposition is simpler, because the bonus w is guaranteed to be on an edge chosen by σ. Therefore, we have $r_w^{\overline{\sigma}}(v) = r^{\overline{\sigma}}(v)$ for every vertex v, and so the careful consideration of $r_w^{\overline{\sigma}}(v)$ in Proposition 4.5 does not need to be repeated. □

With Propositions 4.9 and 4.10 in hand, the minimum ratio test from Lemke's algorithm can be reused with very little modification. The proof of the following proposition is identical to the proof given for Proposition 4.6.

Proposition 4.11. *Let $\sigma_{i,j}$ be a joint strategy, and let w_{j-1} be a rational value. Suppose that for every vertex $v \in P_i$ we have $\operatorname{Bal}_{w_{j-1}}^{\sigma_{i,j}}(v) \geq 0$. If we set:*

$$w_i = w_{i-1} + \min\{\frac{\operatorname{Bal}_w^\sigma(v)}{-\partial_w \operatorname{Bal}_w^\sigma(v)} \ : \ v \in P_i \text{ and } \partial_w \operatorname{Bal}_w^\sigma(v) < 0\},$$

then all of the following properties hold.

1. *For every vertex $v \in P_i$ we have $\operatorname{Bal}_{w_j}^{\sigma_{i,j}}(v) \geq 0$.*
2. *For every value $y > w_j$ there is some vertex $v \in P_i$ with $\operatorname{Bal}_{w_j}^{\sigma_{i,j}}(v) < 0$.*
3. *There is some vertex $v \in P_i$ with $\operatorname{Bal}_{w_j}^{\sigma_{i,j}}(v) = 0$.*

As with Lemke's algorithm, it is possible that the algorithm could make a degenerate step, where $w_{i+1} = w_i$. These cases will be discussed in Section 4.1.3, and for now we will assume that $w_{i+1} > w_i$.

Once w_i has been computed, the algorithm then switches a vertex that is indifferent when $\sigma_{i,j}$ is played on G_{w_j}. This produces the joint strategy $\sigma_{i,j+1}$ that will be considered in the next step of the major iteration. The algorithm stops when

it finds a pair $(\sigma_{i,k}, w_k)$ for which $\mathrm{Bal}_{w_k}^{\sigma_{i,k}}(d) \geq 0$. We define σ_{i+1} to be $\sigma_{i,k}$ with the vertex d switched to the edge $\overline{\sigma_i}(d)$. We now argue that this correctly implements a major iteration.

Proposition 4.12. *For every vertex $v \in P_i \cup \{d\}$ we have $\mathrm{Bal}^{\sigma_{i+1}}(v) \geq 0$.*

Proof. Since $\mathrm{Bal}_{w_{k-1}}^{\sigma_{i,k}}(d) < 0$ and $\mathrm{Bal}_{w_k}^{\sigma_{i,k}}(d) \geq 0$, we know that there is some value $w_{k-1} < y < w_k$ such that $\mathrm{Bal}_y^{\sigma_{i,k}}(d) = 0$. Consider the joint strategy $\sigma_{i,k}$ played in the game G_y. Since d is indifferent, switching it does not change the value of any vertex, and therefore $\mathrm{Bal}_y^{\sigma_{i+1}}(v) \geq 0$ for every vertex $v \in P_i \cup \{d\}$.

We must now argue that $\mathrm{Bal}^{\sigma_{i+1}}(v) \geq 0$ for every vertex $v \in P_i \cup \{d\}$. For every vertex $v \in P_i$, this holds because σ_{i+1} does not use the edge to which the bonus y has been attached. Therefore, the characterisation given by (4.2) implies that $\mathrm{Value}_y^{\sigma_{i+1}}(u) = \mathrm{Value}^{\sigma_{i+1}}(u)$ for every vertex u. This implies that $\mathrm{Bal}_y^{\sigma_{i+1}}(u) = \mathrm{Bal}^{\sigma_{i+1}}(u)$ for every vertex $u \neq d$.

The above reasoning does not hold for the vertex d, because d is the only vertex at which $r^{\overline{\sigma_i}}(d) \neq r_y^{\overline{\sigma_i}}(d)$. However, if $d \in V_{\mathrm{Max}}$ then:

$$\begin{aligned}
\mathrm{Bal}_y^{\sigma_i}(d) &= \mathrm{Value}_y^{\sigma_i}(v) - (r_y^{\overline{\sigma_i}}(v) + \beta \cdot \mathrm{Value}_y^{\sigma_i}(\overline{\sigma_i}(v))) \\
&= \mathrm{Value}^{\sigma_i}(v) - (r^{\overline{\sigma_i}}(v) + y + \beta \cdot \mathrm{Value}^{\sigma_i}(\overline{\sigma_i}(v))) \\
&\leq \mathrm{Value}^{\sigma_i}(v) - (r^{\overline{\sigma_i}}(v) + \beta \cdot \mathrm{Value}^{\sigma_i}(\overline{\sigma_i}(v))) \\
&= \mathrm{Bal}^{\sigma_i}(d).
\end{aligned}$$

Therefore, $\mathrm{Bal}_y^{\sigma_i}(d)$ must be positive. The proof for the case where $d \in V_{\mathrm{Min}}$ is symmetric. □

The Cottle-Dantzig algorithm for discounted games is shown as Algorithm 3. Note that in major iteration i, the algorithm only ever switches vertices in P_i. Therefore, the algorithm can consider at most $2^{|P_i|}$ joint strategies in major iteration i. Therefore, the largest number of joint strategies that the algorithm can consider

Algorithm 3 Cottle-Dantzig(G, σ)

$i := 0$; $\sigma_0 = \sigma$;
$P_0 := \{v \in V \ : \ \mathrm{Bal}^{\sigma_0}(v) \geq 0\}$
while $P \neq V$ **do**
  $j := 0$; $\sigma_{i,0} = \sigma_i$; $w_{-1} := 0$;
  $P_i := \{v \in V \ : \ \mathrm{Bal}^{\sigma_i}(v) \geq 0\}$
  $d :=$ Some vertex in $V \setminus P_i$
  while $\mathrm{Bal}^{\sigma}_{w_i}(d) < 0$ **do**
    $w_{i+1} := w_i + \min\{-\frac{\mathrm{Bal}^{\sigma}_{w_i}(u)}{\partial_w \mathrm{Bal}^{\sigma}_{w}(u)} \ : \ u \in P_i\}$ and $\partial_w \mathrm{Bal}^{\sigma}_{w}(u) < 0\}$
    $\sigma_{i,j} := \sigma[\overline{\sigma}(v)/v]$ for some vertex v with $\mathrm{Bal}^{\sigma}_{w_i}(v) = 0$
    $j := j + 1$
  end while
  $\sigma := \sigma[\overline{\sigma}(d)/d]$; $i := i + 1$
end while

over all major iterations is $\sum_{i=0}^{|V|-1} 2^i = 2^{|V|} - 1$. Therefore, we have shown the following theorem.

Theorem 4.13. *Algorithm 3 terminates, with the optimal joint strategy, after at most* $2^{|V|} - 1$ *iterations.*

4.1.3 Degeneracy

Until now, we have ignored the possibility of reaching a joint strategy σ in which there is more than one indifferent vertex. In LCP algorithms this is known as a degenerate step. There are several methods in the LCP literature that can be used to resolve degeneracy. One method for is Bland's rule, which uses a least-index method to break ties, and another is to use lexicographic perturbations. Both of these methods are well-known, and are also used with the simplex method for linear programming [Chv83]. We will describe how a degenerate step in Lemke's algorithm can be resolved by Bland's rule. Our description can easily be adapted to resolve degenerate steps for the Cottle-Dantzig algorithm. Bland's rule has been chosen because it has a particularly simple interpretation in terms of discounted games.

In each degenerate step, we have a joint strategy σ and a parameter z such that σ is an optimal strategy in G_z, and there is more than one vertex v such that

$\mathrm{Bal}_z^\sigma(v) = 0$. Furthermore, we know that z cannot be decreased any further, because there is at least one vertex v with $\mathrm{Bal}_z^\sigma(v) = 0$ and $\partial_{-z} \mathrm{Bal}_z^\sigma(v) > 0$. This means that the balance of v would fall below 0 if z were decreased.

The task of the degeneracy resolution algorithm is to find a joint strategy σ' that is optimal for the game G_z such that every vertex v with $\mathrm{Bal}_z^\sigma(v) = 0$ has $\partial_{-z} \mathrm{Bal}_z^\sigma(v) < 0$. Moreover, only indifferent vertices may be switched during the procedure. Bland's rule assigns each vertex a unique index in the range 1 to $|V|$. Then, in each iteration, the algorithm switches the smallest vertex v such that $\partial_{-z} \mathrm{Bal}_z^\sigma(v) > 0$.

Bland's rule works because it can never cycle, and because there must be at least one joint strategy in which every vertex v with $\mathrm{Bal}_z^\sigma(v) = 0$ has $\partial_{-z} \mathrm{Bal}_z^\sigma(v) < 0$. If there are k indifferent vertices, then Bland's rule must terminate after at most 2^k steps. This exponential upper bound shows that the choice of degeneracy resolution method can have a severe effect on the running time of the algorithm. However, we are not aware of any discounted games upon which Bland's rule achieves this upper bound.

4.2 The Link With LCPs

We have given two algorithm that solve discounted game, and we have proved their correctness. These algorithms are clearly inspired by their LCP counterparts, however in this section we will show a stronger property: each algorithm for discounted games behaves identically to the corresponding algorithm for LCPs.

This section is divided into two parts. In the first part we justify the definition of the modified games used in the discounted game algorithms by showing how they correspond to the almost complementary solutions that are considered by the LCP versions of these algorithms. In the second part we argue that Lemke's algorithm for discounted games will perform the same sequence of steps as Lemke's algorithm for

LCPs. Since the Cottle-Dantzig algorithm uses the same machinery to perform each major iteration, this reasoning can easily be adapted to argue that the discounted game version of the Cottle-Dantzig algorithm will behave in the same way as the LCP version.

4.2.1 Correctness Of The Modified Games

We begin by considering Lemke's algorithm. The LCP version of Lemke's algorithm considers almost complementary solutions to the modified game, and the discounted game version of this algorithm considers positional joint strategies that are optimal for a modified game. We will show that there is a link between these two concepts: for each almost complementary solution found by the LCP version of the algorithm, there is a positional joint strategy and a modified game that correspond to this solution.

Recall that Lemke's algorithm modifies the LCP with a positive covering vector d, and a scalar z_0. The following Proposition shows how these modifications affect the optimality equations for the discounted game.

Proposition 4.14. *Let M and q be the result of reducing a discounted game to an LCP. Every solution of:*

$$\begin{aligned} w &= Mz + q + d \cdot z_0, \\ w, z &\geq 0, \\ z_i \cdot w_i &= 0 \qquad\qquad \textit{for } i = 1, 2 \ldots, n, \end{aligned}$$

corresponds to a solution of:

$$
\begin{aligned}
V(v) - w(v) &= (r_\lambda(v) - d_v \cdot z_0) + \beta \cdot V(\lambda(s)) && \text{if } v \in V_{Max},\\
V(v) - z(v) &= r_\rho(v) + \beta \cdot V(\rho(s)) && \text{if } v \in V_{Max},\\
V(v) + w(v) &= (r_\lambda(v) + d_v \cdot z_0) + \beta \cdot V(\lambda(s)) && \text{if } v \in V_{Min},\\
V(v) + z(v) &= r_\rho(v) + \beta \cdot V(\rho(s)) && \text{if } v \in V_{Min},\\
w(v),\ z(v) &\geq 0,\\
w(v) \cdot z(v) &= 0.
\end{aligned}
$$

Proof. We will reverse the manipulations that were used to find M and q in Section 2.5.2. By substituting the definitions of M and q into $w = Mz + q + d \cdot z_0$, we obtain:

$$w = (\widehat{I} - \beta\widehat{T_\lambda})(\widehat{I} - \beta\widehat{T_\rho})^{-1}(z + \widehat{I}r_\rho) - \widehat{I}r_\lambda + d \cdot z_0.$$

If we define $V = (\widehat{I} - \beta\widehat{T_\rho})^{-1}(z + \widehat{I}r_\rho)$, then we can rewrite the above equation, and then rearrange it as follows:

$$
\begin{aligned}
w &= (\widehat{I} - \beta\widehat{T_\lambda})V - \widehat{I}r_\lambda + d \cdot z_0,\\
\widehat{I} \cdot V - w &= \widehat{I}r_\lambda - d \cdot z_0 + \beta\widehat{T_\lambda} \cdot V. \qquad (4.6)
\end{aligned}
$$

We can also rearrange $V = (\widehat{I} - \beta\widehat{T_\rho})^{-1}(z + \widehat{I}r_\rho)$ to obtain:

$$\widehat{I} \cdot V - z = \widehat{I}r_\rho + \beta\widehat{T_\rho} \cdot V. \qquad (4.7)$$

Rewriting Equations 4.6 and 4.7 in component form gives the desired system of equations. □

Proposition 4.14 shows how the modifications made by Lemke's algorithm can be seen as modifying the rewards on the left hand edges at each vertex. Note that the

modifications made in Definition 4.1 precisely capture this change. Therefore, the system of equations that we found in Proposition 4.14 correspond to the optimality equations for G_{z_0}.

We now argue that each almost complementary solution that is considered by the LCP version of the algorithm corresponds to a pair (σ, z) such that σ is an optimal strategy for G_z. Let α be an almost complementary solution to the modified system of equations. We will define a joint strategy σ_α that uses the edges for which the corresponding slack variables are non-basic in α (recall that a variable is non-basic if it is chosen to be 0). By definition, we have that there is exactly one index v such that both $w_v = 0$ and $z_v = 0$. The LCP version of Lemke's algorithm will select one of these two variables to be the next driving variable. At this vertex, the joint strategy σ_α will choose the edge whose corresponding slack variable is the next driving variable. For every $v \in V$ we define:

$$\sigma_\alpha(v) = \begin{cases} \lambda(v) & \text{if } w_i = 0 \text{ and either } z_i > 0 \text{ or } w_i \text{ is the next driving variable,} \\ \rho(v) & \text{otherwise.} \end{cases}$$

Since α is a solution to $w = Mz + q + d \cdot z_0$, where both w and z satisfy both non-negativity constraint and the complementarity condition, we have that $\text{Value}^{\sigma_\alpha}$ must be a solution to the optimality equations for G_{z_0}. This implies that σ_α is an optimal strategy for the game G_{z_0}.

In summary, we have shown that each almost complementary solution α that is considered by the LCP version of the algorithm can be represented as a pair (σ_α, z_0) such that σ_α is an optimal strategy for G_{z_0}. Therefore, it is possible for the discounted game version of the algorithm to behave in the same way as the LCP version. In the next section, we will argue that this is indeed the case.

We now turn our attention to the Cottle-Dantzig algorithm. Recall that this algorithm considers almost complementary partial solutions, which allow both the

distinguished variable and its complement to be basic variables. This is problematic for the discounted game algorithm, because positional strategies can only capture solutions in which at least one of the two slack variables associated with each vertex is 0. The modified games considered by the discounted game version of the Cottle-Dantzig algorithm overcome this problem by representing the second non-zero slack variable at the distinguished vertex as a modification to the game.

Suppose that z_v is the distinguished variable in some major iteration. We can rewrite Equations (2.13) and (2.14) as:

$$V(v) = (r_\lambda(v) + w(v)) + \beta \cdot V(\lambda(s)) \quad \text{if } v \in V_{\text{Max}},$$
$$V(v) = (r_\lambda(v) - w(v)) + \beta \cdot V(\lambda(s)) \quad \text{if } v \in V_{\text{Min}}.$$

This implies that we can view the additional non-zero slack variable w_v as modifying the reward on left hand edge at v. Since there is no longer a slack variable on the left hand side of the equation, this characterisation only holds for positional joint strategies σ such that $\sigma(v) = \lambda(v)$. If w_v was chosen to be the driving variable, then the additional slack variable z_v would modify the rewards on the right hand edges in a similar fashion. These observations are implemented by the modified games given in Definition 4.8.

Once again, for each almost complementary partial solution α, we define a joint strategy σ_α. This strategy is similar to the strategy used for Lemke's algorithm, with the condition that, if v is the index of the distinguished variable, then $\sigma_\alpha(v)$ should always choose the edge whose slack variable is the complement of the distinguished variable.

$$\sigma_\alpha(v) = \begin{cases} \lambda(v) & \text{if } w_i = 0 \text{ and either } z_i > 0 \text{ or } w_i \text{ is the next driving variable,} \\ \lambda(v) & \text{if } z_i \text{ is the distinguished variable} \\ \rho(v) & \text{otherwise.} \end{cases}$$

Therefore, for each almost complementary partial solution α, there is a pair (σ_α, x), where x is the complement of the distinguished variable, that represents this solution. Therefore, it is possible for the discounted game version of the algorithm to behave in the same way as LCP version.

4.2.2 Lemke's Algorithm

In this section we argue that Lemke's algorithm for discounted games behaves in the same way as Lemke's algorithm for LCPs. Since the Cottle-Dantzig algorithm uses the same machinery, similar arguments can be used to show that the Cottle-Dantzig algorithm for discounted games behaves in the same way as the Cottle-Dantzig algorithm for LCPs.

We argue that the two algorithms initialise themselves in the same way. The algorithm for LCPs starts by selecting every variable w_i to be basic, and every variable z_i to be non-basic. It then finds an initial almost complementary solution α_0, by finding the index i that minimizes $\min(q_i/d_i \ : \ 1 \leq i \leq n)$, and swaps z_0 with w_i in a pivot operation. Therefore z_i will be the driving variable in the next iteration. The algorithm for discounted games starts with the joint strategy ρ, which always picks the right-hand successor of each vertex, and it is easy to see that this is the strategy σ_{α_0}.

We now argue that the first parameter computed by the discounted game version of the algorithm is the same as the value of z_0 in α_0. To see why this holds, we will state a result that was proved by Jurdziński and Savani.

Proposition 4.15 ([JS08])**.** *If the vector q arises from the reduction of a discounted game, then for each $v \in V$ we have $q_v = \mathrm{Bal}^\rho(v)$.*

This implies that Proposition 4.2 also uses $\min(q_i/d_i \ : \ 1 \leq i \leq n)$ to find the initial parameter. Therefore, the discounted game version of the algorithm will choose the pair (σ_{α_0}, x), where x is the value of z_0 in α.

We now argue that if the LCP version of the algorithm moves from α_i to α_{i+1}, then the discounted game version of the algorithm will move from (σ_{α_i}, x) to $(\sigma_{\alpha_{i+1}}, y)$, where x is the value of z_0 in α_i, and y is the value of z_0 in α_{i+1}. The most obvious difference between the two algorithms is that the discounted game algorithm always tries to decrease the parameter z, whereas the LCP version of the algorithm has a driving variable that it tries to increase. We argue that these two operations are equivalent.

Recall that in each iteration the LCP algorithm has a system of the following form:

$$x_B = M' x_N + q'.$$

The vector x_B is the n-dimensional set of basic variables, the vector x_N is the $(n+1)$-dimensional vector of non-basic variables, the n by $n+1$ matrix M' has been obtained through a sequence of pivot operations applied to the matrix M and the covering vector d, and the vector q' has been obtained by the same sequence of pivot operations applied to q. If the driving variable is the i-th component of x_N, z_0 is the j-th component of x_B, and e is the i-th column in M, then we have have:

$$z_0 = e_j \cdot (x_N)_i + q'_j.$$

This implies that there is a linear dependence between z_0 and the driving variable $(x_N)_i$. It turns out that if M is a P-matrix, then e_j is always negative. This means that increasing the driving variable causes z_0 to decrease, and decreasing z_0 causes the driving variable to increase.

This relationship means that both algorithms will compute the same blocking variable. If the LCP algorithm finds that it can increase $(x_N)_i$ to some value y without a basic variable becoming negative, then the discounted game algorithm will find that it can decrease z by $-e_j \cdot (x_N)_i$ before the balance of some vertex becomes negative.

When the LCP algorithm has computed a blocking variable, it performs a pivot to swap the driving variable with the blocking variable. This means that the driving variable becomes basic, and the blocking variable becomes non-basic. If v is the vertex that corresponds to the driving variable, then we know that both w_v and z_v were non-basic in α_i. If we suppose that w_v was the driving variable, then we know that the joint strategy σ_{α_i} chose $\sigma(v) = \lambda(v)$. In α_{i+1}, we know that w_v will be basic, and z_v will be non-basic. This means that the joint strategy $\sigma_{\alpha_{i+1}}$ must choose $\sigma_{\alpha_{i+1}}(v) = \rho(v)$. This corresponds exactly to the behaviour of the discounted game algorithm.

4.3 Exponential Lower Bounds

In this section we will present exponential lower bounds for the algorithms that we have described. Figure 4.5 shows the family of examples that will be used to prove these bounds. These examples have been considered before: Björklund and Vorobyov showed that the single switch policy could take an exponential number of iterations when it is used with their strategy improvement algorithm [BV07], and a similar example was used by Melekopoglou and Condon to show that single switch strategy improvement for Markov decision processes could take an exponential number of steps [MC94]. It turns out that both Lemke's algorithm and the Cottle-Dantzig algorithm can be made to take exponential time on these examples.

Figure 4.5 presents a family of mean-payoff games, and it is easy to see that these examples could be expressed as parity games. If each reward $\pm|V|^c$ is replaced with a priority c, then applying the reduction of Puri shown in Section 2.5.1 will produce the exactly the same game. To obtain a discounted game, we would then apply the reduction of Zwick and Paterson shown in the same section. Therefore, we will establish exponential lower bounds even in the case where the input is a parity or mean-payoff game.

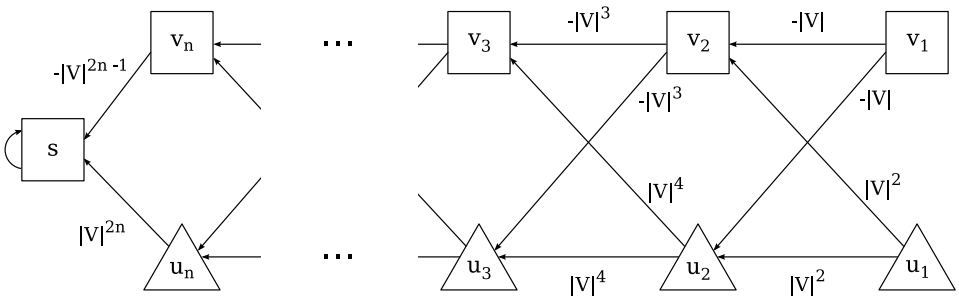

Figure 4.5: The game $\mathcal{G}_n$.

The reduction of Zwick and Paterson will set the discount factor to be very close to 1. To simplify the exposition in this section, we will assume that the discount factor β is actually equal to 1. This assumption is forbidden by the definition of a discounted game, however since the game contains one cycle, and since the sum of the rewards on this cycle is 0, the value of every vertex under every joint strategy will still be finite. The discount factor chosen by the reduction from mean-payoff games is close enough to 1 to ensure that the algorithms will behave as we describe.

Every play in the game $\mathcal{G}_n$ must eventually arrive at the sink vertex s, after which no further reward can be earned. We will therefore be interested in the finite prefix of each play that occurs before the sink is reached. If $\text{Play}(v_0, \sigma) = \langle v_0, v_1, \dots v_n, s, s, \dots \rangle$ for some joint strategy σ, then we define $\text{Prefix}(v_0, \sigma) = \langle v_0, v_1, \dots v_n \rangle$. In our setting, where the discount factor is 1, we have $\text{Value}^\sigma(v) = \sum_{u \in \text{Prefix}(v,\sigma)} r^\sigma(u)$.

The game $\mathcal{G}_n$ consists of $2n$ states $\{v_1, v_2, \dots, v_n\} \cup \{u_1, u_2, \dots u_n\}$ and a sink vertex s. Note that although the vertices v_n, u_n, and s only have one outgoing edge, we can express this game in our model by duplicating the outgoing edges from these vertices. For example, we have $\lambda(u_n) = \rho(u_n) = s$, and we have $r^\lambda(u_n) = r^\rho(u_n) = |V|^{2n}$. For every other vertex, we define the right-hand edge to be the edge that leads to the vertex owned by the same player. This means that $\rho(v_i) = v_{i+1}$ and $\rho(u_i) = u_{i+1}$, for all $i < n$.

4.3.1 Lemke's Algorithm

In this section, we describe how Lemke's algorithm behaves when it is applied to the example. We will show that the behaviour of Lemke's algorithm depends on which vertices are switchable in the original game. In fact, we will show that Lemke's algorithm always switches the switchable vertex with the smallest index. We go on to show that this property causes Lemke's algorithm to take an exponential number of steps to find an optimal strategy for the original game.

Lemke's algorithm allows the user to pick the covering vector. Our exponential lower bound will be for the case where this is a unit vector. In other words, we set $d_v = 1$, for every vertex v.

We say that a joint strategy σ is *symmetric* if for all i in the range $1 \leq i < n$ we have that $\sigma(v_i) = v_{i+1}$ if and only if $\sigma(u_i) = u_{i+1}$. In other words, a symmetric joint strategy is symmetric in the horizontal line separating the vertices v_i from the vertices u_i in Figure 4.5. We have introduced this concept because Lemke's algorithm will always switch the vertex v_i directly before or after it switches the vertex u_i. Therefore, we can restrict ourselves to considering only symmetric joint strategies in this section.

We begin by analysing the balance of each vertex in the original game when a symmetric joint strategy is played. The next proposition shows that for every index i we have $\mathrm{Bal}^\sigma(v_i) = \mathrm{Bal}^\sigma(u_i)$, and gives a formula that can be used to derive the balance of these vertices. In this section, we will prove properties about $\mathrm{Bal}^\sigma(v_i)$. The equality that we are about to prove implies that these properties also hold for the vertices u_i.

Proposition 4.16. *For every symmetric joint strategy σ, and every index $i < n$,*

we have:

$$\mathrm{Bal}^{\sigma}(v_i) = \mathrm{Bal}^{\sigma}(u_i) = \begin{cases} \mathrm{Bal}^{\sigma}(v_{i+1}) - |V|^{2(i+1)} - |V|^{2i+1} & \text{if } \sigma(v_i) = \rho(v_i), \\ -\mathrm{Bal}^{\sigma}(v_{i+1}) + |V|^{2(i+1)} + |V|^{2i+1} & \text{if } \sigma(v_i) = \lambda(v_i). \end{cases}$$

Proof. We begin by proving that $\mathrm{Bal}^{\sigma}(v_i) = \mathrm{Bal}^{\sigma}(u_i)$ for every index $i < n$. For each vertex v_i we can use the fact that $r^{\lambda}(v_i) = r^{\rho}(v_i)$ to obtain:

$$\begin{aligned} \mathrm{Bal}^{\sigma}(v_i) &= \mathrm{Value}^{\sigma}(v_i) - (r^{\sigma}(v_i) + \mathrm{Value}^{\sigma}(\overline{\sigma}(v_i)) \\ &= \mathrm{Value}^{\sigma}(\sigma(v_i)) - \mathrm{Value}^{\sigma}(\overline{\sigma}(v_i)). \end{aligned} \tag{4.8}$$

Applying the same technique to the vertex u_i gives:

$$\mathrm{Bal}^{\sigma}(u_i) = \mathrm{Value}^{\sigma}(\overline{\sigma}(u_i)) - \mathrm{Value}^{\sigma}(\sigma(u_i)).$$

The fact that $\mathrm{Bal}^{\sigma}(v_i) = \mathrm{Bal}^{\sigma}(u_i)$ then follows because, by symmetry of σ, we have that $\sigma(u_i) = \overline{\sigma}(v_i)$ and $\overline{\sigma}(u_i) = \sigma(v_i)$.

We now prove the characterisation for $\mathrm{Bal}^{\sigma}(v_i)$. We first consider the case of $\sigma(v_i) = \rho(v_i)$. In this case, we can use the formula given by (4.8) as follows:

$$\begin{aligned} \mathrm{Bal}^{\sigma}(v_i) &= \mathrm{Value}^{\sigma}(\sigma(v_i)) - \mathrm{Value}^{\sigma}(\overline{\sigma}(v_i)) \\ &= \mathrm{Value}^{\sigma}(v_{i+1}) - \mathrm{Value}^{\sigma}(u_i) \\ &= \mathrm{Value}^{\sigma}(\sigma(v_{i+1})) - \mathrm{Value}^{\sigma}(\sigma(u_i)) - |V|^{2(i+1)} - |V|^{2i+1} \\ &= \mathrm{Value}^{\sigma}(\sigma(v_{i+1})) - \mathrm{Value}^{\sigma}(\overline{\sigma}(v_i)) - |V|^{2(i+1)} - |V|^{2i+1} \\ &= \mathrm{Bal}^{\sigma}(v_{i+1}) - |V|^{2(i+1)} - |V|^{2i+1}. \end{aligned}$$

The case where $\sigma(v_i) = \lambda(v_i)$ can be proved using identical methods. □

We now give a simple characterisation for when a vertex is switchable in the original game when a symmetric joint strategy is played. This characterisation is

derived from the rewards on the outgoing edges of v_n and u_n. Max wants to avoid reaching the vertex v_n because it has a very large negative reward, and Min wants to avoid reaching the vertex u_n because it has a very large positive reward. In a symmetric joint strategy σ, we have that v_n is reached from some vertex v_i if and only if $\text{Prefix}(v_i, \sigma)$ uses an even number of left edges. This is the characterisation that is used in the following proposition.

Proposition 4.17. *If σ is a symmetric joint strategy, then a vertex v is switchable if and only if* $\text{Prefix}(v, \sigma)$ *uses an even number of left edges*

Proof. We will prove this claim by induction over the indices i. The inductive hypothesis is as follows. If $\text{Prefix}(v_i, \sigma)$ uses an even number of edges then:

$$\text{Bal}^\sigma(v_i) \leq -|V|^{2n} - |V|^{2n-1} + \sum_{j=2i+1}^{2n-2} |V|^j.$$

On the other hand, if $\text{Prefix}(v_i, \sigma)$ uses an odd number of left edges then:

$$\text{Bal}^\sigma(v_i) \geq |V|^{2n} + |V|^{2n-1} - \sum_{j=2i+1}^{2n-2} |V|^j.$$

This inductive hypothesis is sufficient to prove the claim, because $|V|^{2n} + |V|^{2n-1} - \sum_{j=i+2}^{2n-2} |V|^j$ is positive for all $i \geq 0$, and $-|V|^{2n} - |V|^{2n-1} + \sum_{j=i+2}^{2n-2} |V|^j$ is negative for all $i \geq 0$. In this proof we will only consider the vertices v_i, this is because Proposition 4.16 has shown that $\text{Bal}^\sigma(v_i) = \text{Bal}^\sigma(u_i)$ for all i.

In the base case, we consider the vertex v_{n-1}. It can easily be seen that:

$$\text{Bal}^\sigma(v_{n-1}) = \begin{cases} -|V|^{2n} - |V|^{2n-1} & \text{if } \sigma(v_{n-1}) = \rho(v_{n-1}), \\ |V|^{2n} + |V|^{2n-1} & \text{if } \sigma(v_{n-1}) = \lambda(v_{n-1}). \end{cases}$$

Therefore, the induction hypothesis is satisfied at the state v_i.

For the inductive step, we consider a vertex v_i. We will prove the claim for the

case where $\sigma(v_i) = \lambda(v_i)$ and where $\text{Prefix}(v_i, \sigma)$ uses an even number of left edges. The other cases can be proved using identical methods. Applying Proposition 4.16 and the inductive hypothesis gives:

$$\begin{aligned}
\text{Bal}^\sigma(v_i) &= \text{Bal}^\sigma(v_{i+1}) + |V|^{2(i+1)} + |V|^{2i+1} \\
&\leq (-|V|^{2n} - |V|^{2n-1} + \sum_{j=2(i+1)+1}^{2n-2} |V|^j) + |V|^{2(i+1)} + |V|^{2i+1} \\
&\leq -|V|^{2n} - |V|^{2n-1} + \sum_{j=2i+1}^{2n-2} |V|^j.
\end{aligned}$$

□

We now begin to analyse the behaviour of Lemke's algorithm on these examples. In order to achieve this, we must know the value of $\partial_{-z} \text{Bal}_z^\sigma(v)$ for every vertex v and every joint strategy σ. It turns out that our simplifying assumption of $\beta = 1$ gives us a simple formula for this quantity. This formulation also depends on how many left edges are used by $\text{Prefix}(v, \sigma)$.

Proposition 4.18. *If σ is a symmetric joint strategy, then $\partial_{-z} \text{Bal}_z^\sigma(v)$ is -1 if v is switchable, and 1 otherwise.*

Proof. We will give a proof for the case where $v_i \in V_{\text{Max}}$, and v_i is switchable. The other three cases can be proved with similar methods. Since v_i is switchable in σ, Proposition 4.17 implies that the path $\text{Prefix}(\sigma, v_i) = \langle v_i = x_i, x_{i+1}, \ldots, x_n \rangle$ must use an even number of left edges. We define $C_{\text{Max}} = \{x_j \in \text{Prefix}(\sigma, v_i) \cap V_{\text{Max}} : \sigma(x_i) = \lambda(x_i)\}$ to be the set of vertices in $\text{Prefix}(v_i, \sigma)$ at which σ chooses a left edge, and we define a similar set $C_{\text{Min}} = \{x_j \in \text{Prefix}(\sigma, v_i) \cap V_{\text{Min}} : \sigma(x_i) = \lambda(x_i)\}$ for Min. We argue that $|C_{\text{Max}}| = |C_{\text{Min}}|$. This holds because whenever a player chooses a left edge, they do not regain control until their opponent chooses a left edge. We use Proposition 4.4, and the fact that $\text{D}^{v_i}(u) = 1$ for every vertex u to

obtain:

$$\begin{aligned}\partial_{-z}\operatorname{Value}_z^\sigma(v_i) &= \sum_{u\in C_{\text{Max}}\cup C_{\text{Min}}} ([u\in V_{\text{Max}}]\cdot \mathrm{D}^v(u) - [u\in V_{\text{Min}}]\cdot \mathrm{D}^v(u)) \\ &= \sum_{u\in C_{\text{Max}}\cup C_{\text{Min}}} ([x\in V_{\text{Max}}] - [x\in V_{\text{Min}}]) \\ &= \sum_{u\in C_{\text{Max}}} 1 - \sum_{u\in C_{\text{Min}}} 1 = 0.\end{aligned}$$

If $\operatorname{Prefix}(\overline{\sigma}(v_i),\sigma) = \langle x_{i+1},\ldots,x_n\rangle$ then $\pi = \langle v_i, x_{i+1},\ldots,x_n\rangle$ is the path that starts at v_i, takes the edge chosen by $\overline{\sigma}(v)$, and then follows σ. We begin by proving that this path uses an odd number of left edges. If $\sigma(v_i) = \rho(v_i) = v_{i+1}$, then the fact that $\operatorname{Prefix}(v_i,\sigma)$ uses an even number of left edges implies that $\operatorname{Prefix}(v_{i+1},\sigma)$ uses an even number of left edges, and by symmetry we have that $\operatorname{Prefix}(\overline{\sigma}(v_i),\sigma)$ uses an even number of left edges. Since $\overline{\sigma}(v_i)$ chooses a left edge at v_i, we have that the path $\langle v_i, x_{i+1},\ldots,x_n\rangle$ must use an odd number of left edges. The case where $\sigma(v_i) = \lambda(v_i) = v_{i+1}$ is similar.

Since the path π starts at $v_i \in V_{\text{Max}}$ and it uses an odd number of left edges, we can use a similar argument to the first paragraph to conclude that $[\overline{\sigma}(v_i) = \lambda(v_i)] + \beta\cdot\partial_{-z}\operatorname{Value}_z^\sigma(\overline{\sigma}(v_i)) = 1$. Substituting this into the formula given by Proposition 4.5 gives:

$$\partial_{-z}\operatorname{Bal}_z^\sigma(v_i) = \partial_{-z}\operatorname{Value}_z^\sigma(v_i) - ([\overline{\sigma}(v_i) = \lambda(v_i)] + \beta\cdot\partial_{-z}\operatorname{Value}_z^\sigma(\overline{\sigma}(v_i))) = 0-1 = -1.$$

□

We claim that for every symmetric joint strategy σ, Lemke's algorithm will choose $z = |\operatorname{Bal}^\sigma(v_i)|$, where i is the smallest index for which v_i is switchable when σ is played in the original game. In order to prove this claim, we must argue that no vertex has a negative balance in the game G_z. We begin by considering the vertices that are switchable in the original game. Proposition 4.18 implies that these vertices have $\partial_{-z}\operatorname{Bal}_z^\sigma(v) = -1$. Therefore, in order for these vertices to not be switchable

in the game G_z, we must have that $\text{Bal}^\sigma(v_i) > \text{Bal}^\sigma(v)$ for each vertex v that is switchable when σ is played in the original game. The next Proposition confirms that this is the case.

Proposition 4.19. *If v_i and v_j are both switchable in a symmetric joint strategy σ, and $i < j$, then* $\text{Bal}^\sigma(v_i) < \text{Bal}^\sigma(v_j)$.

Proof. Let $S = \{i \ : \ v_i \text{ is switchable in } \sigma\}$ be the set of indices of the switchable vertices in σ. We begin by arguing that for each vertex v_i such that $i \in S$, we have $v_j \in \text{Prefix}(v_i, \sigma)$, where $j = \min(S \cap \{i+1, i+2, \ldots, n\})$. Proposition 4.17 implies that for every $i \in S$ we have that $\text{Prefix}(v_i, \sigma)$ uses an even number of left edges, and for every $i \notin S$ we have that $\text{Prefix}(v_i, \sigma)$ uses an odd number of left edges. We have two cases to consider. If $\sigma(v_i) = \rho(v_i) = v_{i+1}$, then $\text{Prefix}(v_{i+1}, \sigma)$ must use an even number of left edges, which implies that $i + 1 \in S$.

The other case is when $\sigma(v_i) = \lambda(v_i) = u_{i+1}$. In this case we have that $\langle v_i, u_{i+1}, u_{i+2}, \ldots, u_{i+k}, v_{i+k+1} \rangle$ is a prefix of $\text{Prefix}(v_i, \sigma)$, and we argue that $i+k+1 \in S$ and that there is no element j in the range $i < j < i+k+1$ such that $j \in S$. Since $\text{Prefix}(v_i, \sigma)$ uses an even number of left edges, and σ chooses a left edge at v_i, we have that $\text{Prefix}(u_{i+j}, \sigma)$ uses an odd number of left edges for every j in the range $1 \leq j \leq k$. By symmetry of σ, we also have that for every j in the range $1 \leq j \leq k$, the path $\text{Prefix}(v_{i+j}, \sigma)$ uses an odd number of left edges, and therefore $j \notin S$. Since σ must use a left edge when moving from u_{i+k} to v_{i+k+1}, we have that $\text{Prefix}(v_{i+k+1}, \sigma)$ must use an even number of left edges, and therefore $j \in S$.

We now prove that for each $i \in S$ we have $\text{Bal}^\sigma(v_i) < \text{Bal}^\sigma(v_j)$, where $j = \min(S \cap \{i+1, i+2, \ldots, n\})$. From our arguments so far, we know that $\text{Prefix}(v_i, \sigma)$ starts at v_i, and then passes through a (potentially empty) set of Min vertices $\{u_{i+1}, u_{i+2}, \ldots, u_{i+k}\}$ before arriving at v_j. Repeated application of Proposition 4.16

gives:

$$\mathrm{Bal}^{\sigma}(v_i) = \mathrm{Bal}^{\sigma}(v_j) - |V|^{2j} - |V|^{2j-1} + \sum_{k=2(i+1)+1}^{2j-2} |V|^k \leq \mathrm{Bal}^{\sigma}(v_j). \qquad \square$$

The next Proposition proves a similar property for a vertex v that is not switchable when σ is played in the original game. In this case, Proposition 4.18 implies that $\partial_{-z} \mathrm{Bal}^{\sigma}_{z}(v) = 1$. Therefore, in order for the balance of v to remain positive when σ is played in G_z, we must have that $|\mathrm{Bal}^{\sigma}(v_i)| \leq \mathrm{Bal}^{\sigma}(v)$. The next proposition confirms that this property holds.

Proposition 4.20. *Let i be the smallest index such that v_i is switchable in a symmetric joint strategy σ. For every vertex v that is not switchable in σ we have $|\mathrm{Bal}^{\sigma}(v_i)| \leq \mathrm{Bal}^{\sigma}(v)$.*

Proof. We begin by arguing that this property holds for every index $j < i$. Proposition 4.17 implies that $\mathrm{Prefix}(v_i, \sigma)$ uses an even number of left edges, and that $\mathrm{Prefix}(v_{i-1}, \sigma)$ uses an odd number of left edges. Therefore, we have $\sigma(v_{i-1}) = \lambda(v_{i-1})$. Proposition 4.16 gives:

$$\mathrm{Bal}^{\sigma}(v_{i-1}) = -\mathrm{Bal}^{\sigma}(v_i) + |V|^{2i} + |V|^{2i-1}.$$

This implies that the proposition holds for the vertex v_{i-1}. Since i is the smallest index such that v_i is switchable, we have that $\sigma(v_j) = \rho(v_j)$ for every index $j < i+1$. Repeated application of Proposition 4.16 gives:

$$\mathrm{Bal}^{\sigma}(v_j) = -\mathrm{Bal}^{\sigma}(v_i) + |V|^{2i} + |V|^{2i-1} - \sum_{k=j+1}^{i-2} (|V|^{2(k+1)} + |V|^{2k+1}) > -\mathrm{Bal}^{\sigma}(v_i).$$

Since $\mathrm{Bal}^{\sigma}(v_i)$ is negative, we have that $|\mathrm{Bal}^{\sigma}(v_i)| \leq \mathrm{Bal}^{\sigma}(v)$.

We now turn our attention to the vertices v_j with $j > i$. We know that $\mathrm{Prefix}(v_i)$ must pass through either v_j or u_j, and we know that $\mathrm{Bal}^{\sigma}(v_j) = \mathrm{Bal}^{\sigma}(u_j)$.

Let k be the largest index such that $k < j$, and $\sigma(v_k) = \lambda(v_k)$. Proposition 4.16 implies that:

$$\operatorname{Bal}^\sigma(v_k) = -\operatorname{Bal}^\sigma(v_j) + \sum_{l=k}^{j-1}(|V|^{2(l+1)} + |V|^{2l+1}).$$

Moreover, no matter what moves σ makes on the indices between i and k, we have the following inequality:

$$\begin{aligned}\operatorname{Bal}^\sigma(v_i) &\geq -\operatorname{Bal}^\sigma(v_j) + \sum_{l=k}^{j-1}(|V|^{2(l+1)} + |V|^{2l+1}) - \sum_{l=i}^{k-1}(|V|^{2(l+1)} + |V|^{2l+1}) \\ &\geq -\operatorname{Bal}^\sigma(v_j).\end{aligned}$$

Since $\operatorname{Bal}^\sigma(v_j)$ is positive, we have that $|\operatorname{Bal}^\sigma(v_i)| \leq \operatorname{Bal}^\sigma(v)$. □

We can now prove the most important claim, which is that Lemke's algorithm will always choose $z = |\operatorname{Bal}^\sigma(v_i)|$. The next proposition shows that this value of z satisfies all three invariants that Lemke's algorithm requires, and it is not difficult to see that no other value of z can satisfy all three of these requirements. Therefore, Lemke's algorithm must choose this value for z when it considers the joint strategy σ.

Proposition 4.21. *Let σ be a symmetric joint strategy, let i be the smallest index for which v_i is switchable, and let $z = -\operatorname{Bal}^\sigma(v_i)$. The following statements are true.*

1. *The strategy σ is optimal for the game G_z.*
2. *For every parameter $y < z$ the strategy σ is not optimal for the game G_y.*
3. *We have $\operatorname{Bal}^\sigma_z(v_i) = \operatorname{Bal}^\sigma_z(u_i) = 0$.*

Proof. If v is a vertex that is switchable when σ is played in G_0, then Proposition 4.19 combined with Proposition 4.18 implies that $\operatorname{Bal}^\sigma_z(v) \geq 0$. On the other hand, if v is a vertex that is not switchable when σ is played in G_0, then Proposition 4.20 combined with Proposition 4.18 implies that $\operatorname{Bal}^\sigma_z(v) \geq 0$. Therefore, we have that

$\mathrm{Bal}_z^\sigma(v) \geq 0$ for every vertex v, which implies that σ is an optimal strategy for the game G_z.

Proposition 4.18 implies that $\partial_{-z} \mathrm{Bal}_z^\sigma(v_i) = -1$. Therefore, we have that both $\mathrm{Bal}_z^\sigma(v_i) = 0$, and $\mathrm{Bal}_z^\sigma(u_i) = 0$. We also have that both $\mathrm{Bal}_y^\sigma(v_i) < 0$, and $\mathrm{Bal}_y^\sigma(u_i) < 0$, for every parameter $y < z$. □

Finally, we must argue that this behaviour forces Lemke's algorithm to take an exponential number of steps. The property given by Proposition 4.21 implies that Lemke's algorithm works from the right: it must find an optimal strategy for the vertices v_1 through v_{i-1} and u_1 through u_{i-1}, before it can switch the vertices v_i or u_i. After these vertices are switched, every prefix that used an even number of left edges now uses an odd number of left edges, and every prefix that used an odd number of left edges now uses an even number of left edges. In other words, a vertex v_j with $j < i$ is switchable after this operation, if and only if it was not switchable before the operation. We can use this property to prove that Lemke's algorithm takes exponential time.

Theorem 4.22. *Lemke's algorithm performs $2^{n+1} - 2$ iterations on the game $\mathcal{G}_n$.*

Proof. Suppose that Lemke's algorithm passes through the sequence of strategies $\langle \sigma_0, \sigma_1, \ldots, \sigma_k \rangle$ before it switches one of the vertices v_i or u_i for the first time. Lemke's algorithm will produce two strategies σ_{k+1} and σ_{k+2} in which is switches the vertices v_i and u_i in an order that is determined by the degeneracy resolution rule. Since Lemke's algorithm works from the right, we know that Lemke's algorithm cannot switch a vertex v_j with $j > i$ before it finds an optimal strategy for the vertices v_1 through v_i, and u_1 through u_i. Our principal claim is that Lemke's algorithm will take k more iterations before it achieves this.

For each joint strategy σ_j, let σ'_j be that strategy with the vertices v_i and u_i switched, and note that $\sigma_{k+2} = \sigma'_k$. We argue that Lemke's algorithm will move from the strategy σ'_j to the strategy σ'_{j-1}. To see why, suppose that k is the smallest

index such that v_k is switchable when σ'_j is played on G_0. Lemke's algorithm will switch both v_k and u_k to produce the strategy τ. Since every vertex with index smaller than k was not switchable in σ'_j, we have that every vertex with index smaller than k is switchable in τ. We also have that v_k and u_k are not switchable in τ. Furthermore, if τ' is the strategy τ with the vertices v_i and u_i switched, then we have that the vertices v_k and u_k are switchable in τ', and that no vertex with index smaller than k is switchable in τ'. It follows that $\tau' = \sigma_{j-1}$, and that $\tau = \sigma'_{j-1}$. Therefore, after arriving at the strategy $\sigma_{k+2} = \sigma'_k$, Lemke's algorithm will move through the sequence of strategies $\langle \sigma'_k, \sigma'_{k-1}, \ldots, \sigma'_0 \rangle$.

Therefore, we have the following recursion for the first time that Lemke's algorithm arrives at a joint strategy that is optimal for the first i indices. The expression $T(1)$ is derived from the fact that the initial strategy selects $\rho(v_1)$ and $\rho(u_1)$, and that Lemke's algorithm will spend two iterations switching these vertices to $\lambda(v_1)$ and $\lambda(u_1)$.

$$T(1) = 2,$$
$$T(n) = T(n-1) + 2 + T(n-1).$$

It can easily be verified that $T(n) = 2^{n+1} - 2$. □

4.3.2 The Cottle-Dantzig Algorithm

In this section, we show that the Cottle-Dantzig algorithm takes an exponential number of steps when it is applied to the family of examples $\mathcal{G}_n$. There are two parameters that the user is allowed to choose for this algorithm: the initial joint strategy and the order in which the vertices are chosen as the distinguished vertex. We define the initial joint strategy σ_0 to be $\sigma_0(v_i) = \rho(v_i)$ and $\sigma_0(u_i) = \lambda(u_i)$, for all i.

It is not difficult to verify that each vertex u_i has a positive balance in σ_0,

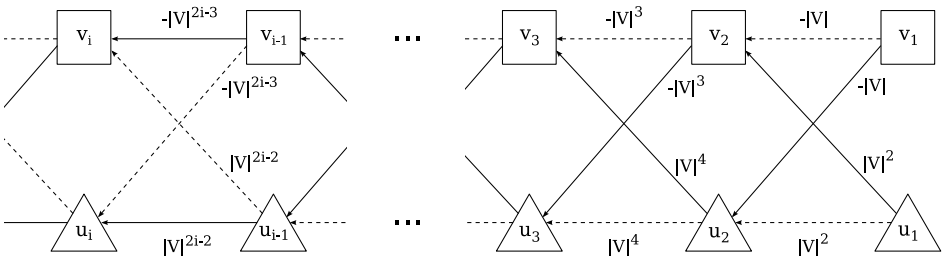

Figure 4.6: The joint strategy σ_0 that is considered at the start of major iteration i for the first i vertices.

and each vertex v_i has a negative balance. Therefore, the set P_0 will contain every vertex u_i, and these vertices cannot be chosen as distinguished variables. For major iteration i, we will choose the vertex v_i to be the distinguished variable. In other words, the vertices v_i are chosen in order from 1 to n. To establish our exponential lower bound, we will prove that major iteration i takes $2^i - 1$ steps to be completed.

The joint strategy σ_0 that the algorithm considers at the start of major iteration i is represented by the dashed lines in Figure 4.6. It is not difficult to see that in major iteration i the algorithm will only modify the strategy at the vertices v_j and u_j with $j < i$. This follows from the fact that there is no path from any vertex u_j with $j > i$ to the distinguished vertex v_i, and therefore the balance of these vertices cannot be affected by the bonus that is added to the outgoing edge of v_i.

Unlike Lemke's algorithm, the overall strategy will not be symmetric until the optimal strategy is found, but during major iteration i the strategy will be symmetric on the vertices v_j and u_j with $j < i$. This is shown as the symmetric portion in Figure 4.6. In this section, we will call a strategy symmetric if it is symmetric for every vertex index that is smaller than i. The Cottle-Dantzig algorithm will always switch the vertex v_i immediately before or after it switched the vertex u_i, and therefore it can be seen as a process that moves between symmetric joint strategies.

Our first proposition gives a simple characterisation of $\partial_z \operatorname{Bal}_z^\sigma(v)$ for each

vertex in the symmetric portion, and for every joint strategy σ. Once again, this relies on whether an odd or even number of left edges are used by $\text{Prefix}(v, \sigma)$. However, for the Cottle-Dantzig algorithm, we are only interested in the portion of $\text{Prefix}(v, \sigma)$ that lies in the symmetric portion. We are not interested in the number of left edges used by $\text{Prefix}(v, \sigma)$ after it has passed through either v_i or u_i.

Proposition 4.23. *Suppose that the algorithm is in the major iteration i and is considering a symmetric joint strategy σ, and let v be some vertex v_j or u_j with $j < i$. If $\pi = \langle v = x_0, x_1, \dots x_k \rangle$ is the prefix of $\text{Prefix}(v, \sigma)$ such that x_k is either v_i or u_i, then we have:*

$$\partial_z \operatorname{Bal}_z^\sigma(v) = \begin{cases} -1 & \text{If } \pi \text{ uses an odd number of left edges,} \\ 1 & \text{otherwise.} \end{cases}$$

Proof. We will prove this proposition for a vertex v_j with $j < i$. The proof for the vertices u_j will be entirely symmetrical. If π uses an odd number of left edges, then we must have that $x_k = u_i$. This implies that $\partial_z \operatorname{Value}_z^\sigma(v_j) = 0$. The symmetry of σ implies that $\text{Prefix}(\overline{\sigma}(v_j), \sigma)$ must pass through the vertex v_i. Therefore, we have that $\partial_z \operatorname{Value}_z^\sigma(\overline{\sigma}(v_j)) = 1$. Substituting these into the definition of $\partial_z \operatorname{Bal}_z^\sigma(v)$ gives:

$$\partial_z \operatorname{Bal}_z^\sigma(v_j) = \partial_z \operatorname{Value}_z^\sigma(v_j) - \beta \cdot \partial_z \operatorname{Value}_z^\sigma(\overline{\sigma}(v_j)) = 0 - 1 = -1.$$

On the other hand, if π uses an even number of left edges, then we must have that $x_k = v_i$. This implies that $\partial_z \operatorname{Value}_z^\sigma(v_j) = 1$ and $\partial_z \operatorname{Value}_z^\sigma(\overline{\sigma}(v_j)) = 0$. By performing the same substitution, we can conclude that $\partial_z \operatorname{Bal}_z^\sigma(v_j) = 1$. □

Proposition 4.23 implies that for the initial joint strategy σ_0 in major iteration i, every vertex v_j or u_j with $j < i$ has $\partial_z \operatorname{Bal}_z^{\sigma_0}(v) = -1$. This means that as w is increased, the balance of every vertex in the symmetric portion will decrease.

Therefore, to find the first parameter w that is chosen in each major iteration, we must find the vertex in the symmetric portion that has the smallest balance. The next proposition gives a characterisation for the balance of each vertex in the symmetric portion for the initial strategy, which can easily be verified by tracing the paths used by the strategy shown in Figure 4.6.

Proposition 4.24. *For every index j where $j < i$, we have:*

$$\mathrm{Bal}^{\sigma_0}(v_i) = \mathrm{Bal}^{\sigma_0}(u_i) = |V|^{2i} + |V|^{2i-1} - \sum_{k=j+1}^{i-1} (|V|^{2k} + |V|^{2k-1}).$$

This proposition indicates that in the initial strategy, the vertices v_1 and u_1 have the smallest balance among the vertices in the symmetric portion. Therefore, the Cottle-Dantzig algorithm will choose a parameter w that makes these vertices indifferent.

Recall that our goal is to show that major iteration i will take $2^i - 2$ steps. Our proof of this will be by induction over the first time that a vertex v_j has balance 0. We define, for $1 \leq j < i$, the quantity k_j to be the number of strategies that the Cottle-Dantzig algorithm passes through before the vertex v_j has balance 0 for the first time. Furthermore, we define w_j to be the value of the parameter when this occurs. The inductive hypothesis is as follows.

- For each j we have that $k_j = 2^{j+1} - 2$.
- For each j we have that $w_j = |V|^{2i} + |V|^{2i-1} - \sum_{k=j+1}^{i-1}(|V|^{2k} + |V|^{2k-1})$.
- In the first k_j iterations, no vertex with index larger than j will be switched.

From our arguments so far, we know that $k_1 = 0$ and $w_1 = |V|^{2i} + |V|^{2i-1} - \sum_{k=2}^{i-1}(|V|^{2k} + |V|^{2k-1})$, and we know that no vertex with index larger than 1 has been switched. This proves the base case of the induction.

We will now prove the inductive step. Suppose that the algorithm has passed through the sequence of strategies $\langle \sigma_0, \sigma_1, \ldots \sigma_{k_j} \rangle$, to arrive at the joint strategy σ_{k_j}

in which the vertex v_j has balance 0 for the first time. We know that the parameter w has been set to w_j in this iteration.

By the inductive hypothesis, we know that no vertex v_l with l in the range $j < l < i$ has been switched since the start of this major iteration. This implies that $\sigma_{k_j} = \sigma_0$ on these vertices. It follows that the balance of v_l has been continuously decreasing since the start of the major iteration, and therefore we have $\mathrm{Bal}_{w_j}^{\sigma_{k_j}}(v_l) = \mathrm{Bal}_0^{\sigma_0}(v_l) - w_j$. The first thing that we will prove is that the balance of v_l will remain positive even if w is raised to $2w_j - w_1$.

Proposition 4.25. *If $w = 2w_j - w_1$ then $\mathrm{Bal}_w^{\sigma_0}(v_l) - w \geq 0$ and $\mathrm{Bal}_w^{\sigma_0}(u_l) - w \geq 0$ for each l in the range $j < l < i$.*

Proof. We prove the proposition for the vertices v_l with l in the range $j < l < i$. The proof for the vertices u_l is entirely symmetric. Proposition 4.24 implies that the balance of v_l in G_0 was:

$$\mathrm{Bal}^{\sigma_0}(v_l) = |V|^{2i} + |V|^{2i-1} - \sum_{k=l+1}^{i-1} (|V|^{2k} + |V|^{2k-1}).$$

The inductive hypothesis gives the following two equalities.

$$w_j = |V|^{2i} + |V|^{2i-1} - \sum_{k=j+1}^{i-1} (|V|^{2k} + |V|^{2k-1})$$
$$w_1 = |V|^{2i} + |V|^{2i-1} - \sum_{k=2}^{i-1} (|V|^{2k} + |V|^{2k-1})$$

Simple arithmetic gives the following expression for $2w_j - w_1$:

$$|V|^{2i} + |V|^{2i-1} - \sum_{k=j+1}^{i-1} (|V|^{2k} + |V|^{2k-1}) + \sum_{k=2}^{j} (|V|^{2k} + |V|^{2k-1}).$$

Finally, we obtain:

$$\mathrm{Bal}^{\sigma_0}(v_l) - (2 \cdot w_j - w_1) = \sum_{k=j+1}^{l} (|V|^{2k} + |V|^{2k-1}) - \sum_{k=2}^{j} (|V|^{2k} + |V|^{2k-1}).$$

Since $l \geq j + 1$, this expression must be positive. □

Our next task is to show that the Cottle-Dantzig algorithm will pass through $k_j + 2$ further strategies after it has reached the joint strategy σ_{k_j}. For each joint strategy σ_i we define:

$$\sigma'_l(v) = \begin{cases} \overline{\sigma}_l(v) & \text{if } v = v_j \text{ or } v = u_j, \\ \sigma_l(v) & \text{otherwise.} \end{cases}$$

In other words, σ'_l is σ_l in which the vertices with index j have been switched. To prove this claim, we will show that the Cottle-Dantzig algorithm passes through each of the strategies σ_l with $1 \leq l \leq k_j$.

Proposition 4.26. *After arriving at the joint strategy σ_{k_j}, the Cottle-Dantzig algorithm will pass through $k_j + 2$ further strategies while raising the parameter w to at most $2w_j - w_1$. No vertex v_l with l in the range $j < l < i$ is switched during this sequence.*

Proof. Since both vertices with index j are indifferent in σ_{k_j} the algorithm will produce two strategies σ_{k_j+1} and σ_{k_j+2}, in order to switch these two vertices. Note that $\sigma_{k_j+2} = \sigma'_{k_j}$. Proposition 4.23 implies that for every l in the range $1 \leq l \leq k_j$ and every vertex v_m with $m < j$ we have:

$$\partial_z \mathrm{Bal}_z^{\sigma_l}(v_m) = -\partial_z \mathrm{Bal}_z^{\sigma'_l}(v_m).$$

This is because after switching the vertex pair with index j the path from every vertex with index less than j sees an extra left edge. So, as w is increased the

balance of every vertex σ'_l will move in a direction that is opposite to the way that it moved in σ_l. Therefore, if w is raised to $2w_j - w_1$ then Cottle-Dantzig algorithm will pass through the sequence of strategies $\langle \sigma'_{k_j}, \sigma'_{k_j-1}, \ldots \sigma'_0 \rangle$. This is because w was increased by $w_j - w_1$ as the algorithm moved from σ_0 to σ_{k_j}. Proposition 4.25 confirms that no vertex with index greater than j can be switched during this process. □

We have provided a proof for the first and third conditions of the inductive hypothesis. To complete the proof, the next proposition shows that the vertices v_{j+1} and u_{j+1} have balance 0 for the first time in the iteration where the Cottle-Dantzig algorithm considers the joint strategy σ'_0.

Proposition 4.27. *We have* $k_{j+1} = 2^{j+2} - 2$ *and* $w_{j+1} = |V|^{2i} + |V|^{2i-1} - \sum_{k=j+2}^{i-1}(|V|^{2k} + |V|^{2k-1})$.

Proof. We must show that after arriving at the joint strategy σ'_0, The Cottle-Dantzig algorithm sets the parameter w to w_{j+1}, and that the balance of v_{j+1} is 0 when σ'_0 is played in $G_{w_{j+1}}$. Note that in σ'_0 the path from every vertex v_l with $l < j$ uses precisely two left edges: one which belongs to either v_j or u_j and one that belongs to v_{i-1} or u_{i-1}. From this we can conclude, by Proposition 4.23, that $\partial_z \operatorname{Bal}_z^{\sigma'_0}(v) = 1$ for every vertex v with index smaller than or equal to j. This implies that once the algorithm has reached σ'_0 it can continue to raise w until the balance of some vertex with index greater than j becomes 0.

We know that the balance of every vertex with index higher than j has decreased in every iteration. Therefore, to find the first vertex whose balance becomes 0 as w is increased we need only to find the vertex whose balance was the smallest, among those vertices with indices higher than j, at the start of major iteration i. From Proposition 4.24 this is clearly the vertex v_{j+1}. Moreover, we know that w must be set to $|V|^{2i} + |V|^{2i-1} - \sum_{k=j+2}^{i-1}(|V|^{2k} + |V|^{2k-1})$ in order to achieve this. □

We now know that each major iteration must take at least $2^i - 2$ steps. We have therefore shown the following theorem, which is an exponential lower bound for the Cottle-Dantzig algorithm.

Theorem 4.28. *Consider an order in which all Min vertices precede Max vertices, and Max vertices are ordered from right to left. The Cottle-Dantzig algorithm performs* $2^{n+1} - 2n - 1$ *iterations.*

4.4 Concluding Remarks

In this chapter, we have studied two classical pivoting algorithms from the LCP literature. We have shown how these algorithms can be interpreted as strategy iteration algorithms when they are applied to discounted games. We have also constructed a family of examples upon which both of these algorithms take exponential time.

It should be stressed that the lower bounds that have been shown in this chapter depend on specific choices for user-supplied parameters. The exponential lower bound for Lemke's algorithm requires that the covering vector is chosen to be a unit vector, and the lower bound for the Cottle-Dantzig algorithm requires a specific ordering over the choice of the distinguished vertex. There are other choices of these parameters for which the algorithms behave well: there is a choice of covering vector that makes Lemke's algorithm terminate in a linear number of steps on our examples, and there is an ordering for distinguished vertices that makes the Cottle-Dantzig algorithm terminate in a linear number of steps.

This situation can be compared with the choice of switching policy in strategy improvement. The fact that some switching policies take exponential time does not rule out the possibility of a switching policy that always terminates in polynomial time. Therefore, the effect that the choice of covering vector and ordering over distinguished vertices has upon the running time of the respective algorithms should be studied. For example, in strategy improvement it is known that for each input

instance there is a switching policy that causes strategy improvement to terminate after a linear number of iterations. It would be interesting to see if this result can be replicated for our algorithms. This would involve proving that for each example there is a choice of covering vector that makes Lemke's algorithm terminate in polynomial time, or that there is a choice of ordering over distinguished vertices that makes the Cottle-Dantzig algorithm terminate in polynomial time. On the other hand, if this property does not hold, then it would also be interesting to extend our lower bound for arbitrarily chosen covering vectors and arbitrarily chosen orderings over distinguished vertices.

A more challenging problem is to show that these parameters can be chosen in a way that guarantees that the algorithms will terminate quickly. There are previous results that indicate that this may be fruitful. Adler and Megiddo studied the performance of Lemke's algorithm for the LCPs arising from randomly chosen linear programs [AM85]. They showed that if a unit covering vector is used, then the expected running time of Lemke's algorithm is exponential. On the other hand, they carefully construct a covering vector that causes Lemke's algorithm to terminate after an expected quadratic number of steps.

Chapter 5

Greedy Strategy Improvement For Markov Decision Processes

The greedy switching policy for strategy improvement is probably the most natural switching policy. It was long thought that strategy improvement algorithms equipped with this switching policy could be proved to terminate in polynomial time. In the game setting, these hopes were dashed by the result of Friedmann [Fri09], which gives a family of parity games upon which the strategy improvement algorithm of Vöge and Jurdziński [VJ00] took exponential time. It was later shown that these examples could be generalised to provide exponential lower bounds for strategy improvement on mean-payoff, discounted, and simple stochastic games [And09].

However, the lower bounds that have been discovered so far apply only to games. The running time of greedy strategy improvement for Markov decision processes has been left open. It is possible to imagine that greedy strategy improvement could be exponential for games and polynomial for MDPs. This possibility is highlighted by the fact that critical structures in Friedmann's examples rely on the behaviour of the second player.

In this chapter we show how Friedmann's examples can be adapted to provide exponential lower bounds in the Markov decision process setting. We show how the

second player in Friedmann's examples can be replaced with a random action in the Markov decision process setting. We produce a family of Markov decision processes upon which greedy strategy improvement for the average-reward criteria take an exponential number of steps.

5.1 The Strategy Improvement Algorithm

The family of examples that will be used to show the exponential lower bound have a special form. Each example contains a sink, which is a vertex s with a single outgoing action (s, s) such that $r(s, s) = 0$. Moreover, every run under every strategy that is considered by the strategy improvement algorithm will arrive at this sink. This means that we will produce an exponential sequence of strategies, such that $G^\sigma(v) = 0$ for each strategy σ in this sequence, and for every vertex v in the example.

This property allows us to simplify our notation. Recall that the strategy improvement algorithm for the average-reward criterion has two different types of iteration: if there is a gain-switchable action, then the algorithm will only switch gain-switchable actions, and if there are no gain-switchable actions, then the algorithm will switch bias-switchable actions. However, since the gain of each strategy is always 0, we know that the algorithm algorithm can never encounter a gain-switchable action. From this, it follows that only bias switchable actions are switched during our exponential sequence of strategies. Therefore, we can completely ignore the gain component of each strategy during our proofs.

For the sake of notational simplicity, we will use this knowledge to simplify the optimality equations given in (2.6)-(2.7). Since we know that $G^\sigma(v) = 0$ for every strategy σ that we will consider, we can simplify Equation 2.7 to obtain the

following optimality equation:

$$B(v) = \max_{a \in A_v} \left(\mathrm{r}(v, a) + \sum_{v' \in V} p(v'|v, a) \cdot B(v') \right). \tag{5.1}$$

Note that this equation bears some resemblance to the optimality equation that is used in the total-reward setting [Put05, Chapter 7]. However, our proof will not imply an exponential lower bound for the total-reward criterion, because our examples do not fall into a class of models upon which strategy improvement has been shown to be correct.

We now define some simplified notation that will be used with the optimality equation (5.1). Since the gain component is being ignored, we define $\mathrm{Val}^{\sigma}(v) = B^{\sigma}(v)$ for every vertex v. We define the appeal of an action a under a strategy σ to be:

$$\mathrm{Appeal}^{\sigma}(a) = \mathrm{r}(v, a) + \sum_{v' \in S} p(v'|v, a) \cdot \mathrm{Value}^{\sigma}(v').$$

The action a is switchable at a vertex v in a strategy σ if $\mathrm{Appeal}^{\sigma}(a) > \mathrm{Val}^{\sigma}(v)$. Theorem 3.2 implies that if σ' is obtained from σ by switching a subset of the switchable actions, then we have $\mathrm{Val}^{\sigma}(v) \leq \mathrm{Val}^{\sigma'}(v)$ for every vertex v, and there is some vertex at which this inequality is strict. In this context, the greedy switching policy chooses, at every vertex v, the switchable action a such that $\mathrm{Appeal}^{\sigma}(a) \geq \mathrm{Appeal}^{\sigma}(b)$, for every switchable action b. Ties are broken using the same methods that were shown in Section 3.2.4.

5.2 The Family of Examples

In this section we describe the family of examples that will be used to show the exponential lower bound for greedy strategy improvement on Markov decision processes. Our goal is to force greedy strategy improvement to mimic a binary counter. We will define the family of examples $\mathcal{G}_n$, where n is the number of bits in the binary

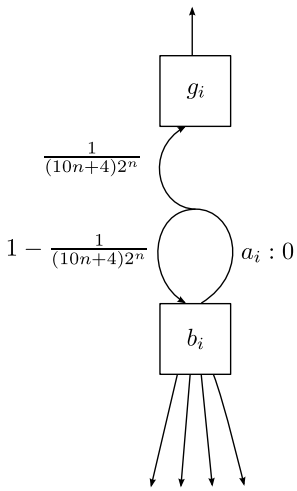

Figure 5.1: The structure for the bit with index i.

counter. In this section, we will describe how to construct the game $\mathcal{G}_n$.

5.2.1 The Bit Gadget

Figure 5.1 shows the gadget that will be used to represent a bit of the binary counter. It consists of a vertex b_i, which has a special action a_i. The action a_i is a probabilistic action that has reward 0. When the action a_i is chosen at b_i there is a very large probability of returning to the vertex b_i, and a very small probability of moving to the vertex g_i. There will be n copies of this structure in the example $\mathcal{G}_n$, which will be indexed with the integers 1 through n. The bit with index 1 will be the least significant bit, and the bit with index n will be the most significant bit.

For each strategy σ, the state of the bit represented by b_i will be determined by the choice that σ makes at b_i. The bit is a 1 in the strategy if $\sigma(b_i) = a_i$, and it is a 0 if $\sigma(b_i) \neq a_i$. We will represent the configuration of a binary counter as a non-empty set $B \subseteq \{1, 2, \ldots, n\}$ that contains the indices of the bits that are 1. If a configuration B is a strict subset of $\{1, 2, \ldots n\}$, then we say that B is *improvable*.

Definition 5.1 (Configuration)**.** *A configuration is a set $B \subseteq \{1, 2, \ldots, n\}$ such that $B \neq \emptyset$. An improvable configuration is a set $B \subset \{1, 2, \ldots, n\}$ such that $B \neq \emptyset$.*

A strategy σ represents a configuration B if $\sigma(b_i) = a_i$ for every index $i \in B$, and $\sigma(b_i) \neq a_i$ for every every index $i \notin B$. For a set of natural numbers B we define $B^{>i}$ to be the set $B \setminus \{k \in \mathbb{N} \ : \ k \leq i\}$. We define analogous notations for $<$, $\geq$, and $\leq$.

Our objective is to force greedy strategy improvement to pass through at least one strategy for each configuration: we will begin at a strategy that represents the configuration $\{1\}$, and after visiting at least one strategy that represents each configuration, we will finally arrive at a strategy that represents the configuration $\{1, 2, \ldots, n\}$. Since the MDPs that we will construct will be polynomially sized in n, if this property can be achieved, then an exponential lower bound will obviously follow. As we have mentioned, the algorithm will will traverse configurations in the same order as a binary counter. Suppose that greedy strategy improvement is currently considering a strategy that represents the improvable configuration B. If $i = \min(\{1, 2, \ldots, n\} \setminus B)$, then we want to force greedy strategy improvement to move to a strategy that represents the configuration $(B \setminus \{1, 2, \ldots, i-1\}) \cup \{i\}$.

This operation occurs in two phases. In the *flip* phase, greedy strategy improvement moves from a strategy that represents the configuration B to a strategy that represents the configuration $B \cup \{i\}$, where i is the smallest index such that $i \notin B$. In other words, in the first phase, the vertex b_i must be switched to the action a_i, and the strategy at each other vertex b_j must not be changed. Once this has occurred, greedy strategy improvement will enter the *reset* phase where it moves to a strategy that represents the configuration $(B \setminus \{1, 2, \ldots, i-1\}) \cup \{i\}$. This means that every vertex b_j with $j < i$ will be switched away from the action a_j. Once this strategy has been reached, greedy strategy improvement will return to the flip phase for the new configuration.

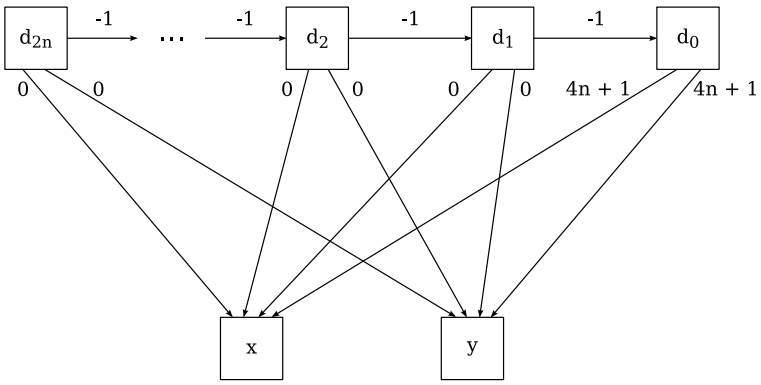

Figure 5.2: The deceleration lane.

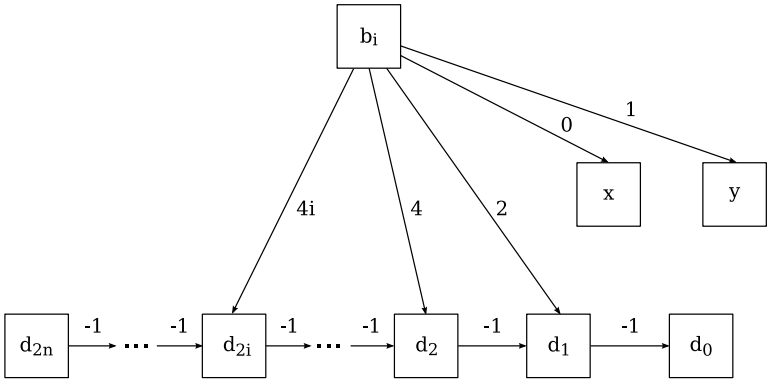

Figure 5.3: The outgoing actions from the vertex b_i.

5.2.2 The Deceleration Lane

Figure 5.2 shows a gadget called the deceleration lane. The example $\mathcal{G}_n$ will contain one instance of this gadget. It consists of two vertices x and y, which have outgoing actions to other parts of the example, and a sequence of vertices d_i. Each vertex d_i with $i > 0$ has an action to the vertex d_{i-1} with reward -1, an action to the vertex x with reward 0, and an action to the vertex y with reward 0. The vertex d_0 is different: it has an action to the vertex y with reward $4n+1$ and an action to the vertex x with reward $4n+1$.

The deceleration lane plays a key role in implementing the flip phase. In this phase, we must ensure that the action a_i is switched at the vertex b_i, where i

is the smallest index such that $i \notin B$, and that no other bit changes state. This is achieved by connecting each bit vertex to the deceleration lane, and Figure 5.3 shows how each vertex b_i is connected to the deceleration lane. The vertex b_i has exactly $2i$ outgoing actions to the deceleration lane, and these actions lead to the vertex d_1 through d_{2i}.

The principal idea is that the deceleration lane can prevent the action a_i from being switched at the vertex b_i for exactly $2i$ iterations. It does this by going through a sequence of $2n$ strategies. Consider an index i such that $i \notin B$. In the first strategy, the action (b_i, d_1) will be the most appealing action at the vertex b_i, and in the j-th strategy, the action (b_i, d_j) will be the most appealing action at the vertex b_i. Since the most appealing action at this vertex is not a_i, greedy strategy improvement cannot switch this action. Therefore, the bit with index i cannot be set to 1 during this sequence.

We can now see why the smallest 0 bit is set to 1. If i is the smallest index such that $i \notin B$, then every vertex b_j with $j > i$ has at least two more outgoing actions to the deceleration lane. Therefore, the action a_i can be switched at the vertex b_i at least two iterations before the action a_j can be switched at the vertex b_j. This allows greedy strategy improvement to move to a strategy that represents the configuration $B \cup \{i\}$, which correctly implements the flip phase.

The deceleration lane also has an important role to play in the reset phase. During this phase, the deceleration lane must be switched back to its initial strategy, which was the strategy where the action (b_i, d_1) is the most appealing action at every vertex b_j with $j \notin (B \cup \{i\}) \setminus \{1, 2, \ldots, i-1\}$. This must occur in order for greedy strategy improvement to continue with the flip phase for the next configuration immediately after the reset phase has been completed. We will later show how the choice of outgoing actions from the vertices x and y achieves this.

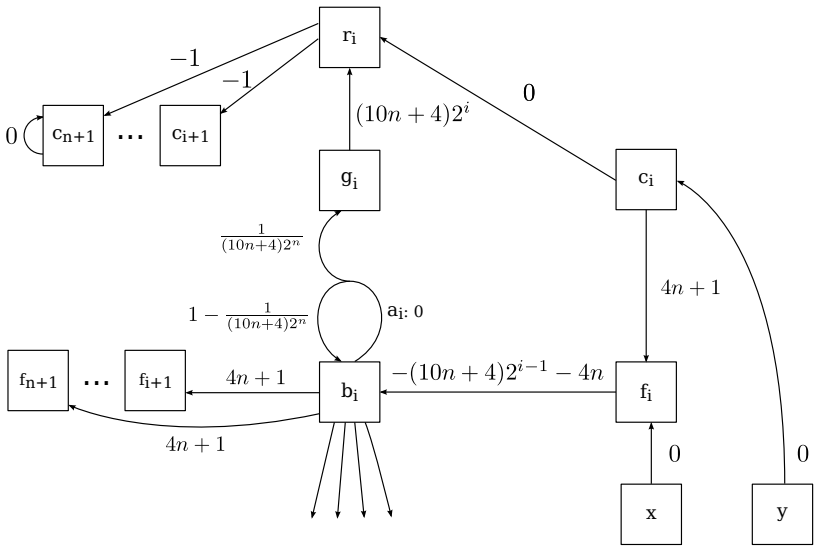

Figure 5.4: The structure associated with the vertex b_i.

5.2.3 Reset Structure

Each vertex b_i has an associated structure that is called the reset structure, which is shown in Figure 5.4. The vertex c_i is called the *choice* vertex. The idea is that the action (c_i, f_i) should be chosen at the vertex c_i if and only if the index $i \in B$. It is not difficult to see why this should be the case: if $i \in B$ then choosing the action (c_i, f_i) results in a penalty from the edge (f_i, b_i), but this is offset by the larger reward on the edge (g_i, r_i). On the other hand, if $i \notin B$ then only the penalty on the edge (f_i, b_i) is seen.

During the flip phase, greedy strategy improvement does not switch any vertex in the reset structure. However, during the reset phase, it is the reset structure that causes each vertex b_j with $j < i$ to be switched away from the action a_j. This occurs because each vertex b_j has an action (b_j, f_k) for every index $k > j$. When the vertex b_i is switched to the action a_i, the valuation of the vertex f_i rises, because of the reward on the action (g_i, r_i). This rise is enough to ensure that each action (b_j, f_i) is the most appealing action at the vertex b_j, for each $j < i$. Therefore, greedy strategy improvement will move to a strategy that represents the

Vertex	Target	Range	Reward
d_0	y		$4n+1$
d_0	x		$4n+1$
d_i	x	$1 \le i \le 2n$	0
d_i	y	$1 \le i \le 2n$	0
d_i	d_{i-1}	$1 \le i \le 2n$	-1
b_i	x	$1 \le i \le n$	0
b_i	y	$1 \le i \le n$	1
b_i	d_j	$1 \le i \le n$, $1 \le j \le 2i$	$2j$
b_i	f_j	$1 \le i \le n$, $i < j \le n$	$4n+1$
c_i	f_i	$1 \le i \le n$	$4n+1$
c_i	r_i	$1 \le i \le n$	0
c_{n+1}	c_{n+1}		0
f_i	b_i	$1 \le i \le n$	$-(10n+4)2^{i-1} - 4n$
g_i	r_i	$1 \le i \le n$	$(10n+4)2^i$
r_i	c_j	$1 \le i \le n$, $i < j \le n+1$	-1
y	c_j	$1 \le j \le n$	0
x	f_j	$1 \le j \le n$	0

Table 5.1: The deterministic actions in the game $\mathcal{G}_n$.

configuration $(B \cup \{i\}) \setminus \{1, 2, \ldots, i-1\}$.

Figure 5.4 also specifies the outgoing actions from the vertices x and y. There is an action (y, c_i) with reward 0 for every i in the range $1 \le i \le n$, and there is an action (x, f_i) with reward 0 for every i in the range $1 \le i \le n$.

We have now completed the description of our examples. We will now formally specify $\mathcal{G}_n$. The MDP contains the vertices x and y and the vertices c_i, f_i, b_i, g_i, and r_i for i in the range $1 \le i \le n$. It also contains the vertex c_{n+1}, and the vertices d_i for every i in the range $0 \le i \le 2n$. The deterministic actions in the MDP are given by Table 5.2.3. Each vertex b_i also has a probabilistic action a_i with reward 0, where $\mathrm{p}(g_i|b_i, a_i) = 1/((10n+4)2^n)$ and $\mathrm{p}(b_i|b_i, a_i) = 1 - 1/((10n+4)2^n)$. Figure 5.5 shows the example $\mathcal{G}_2$.

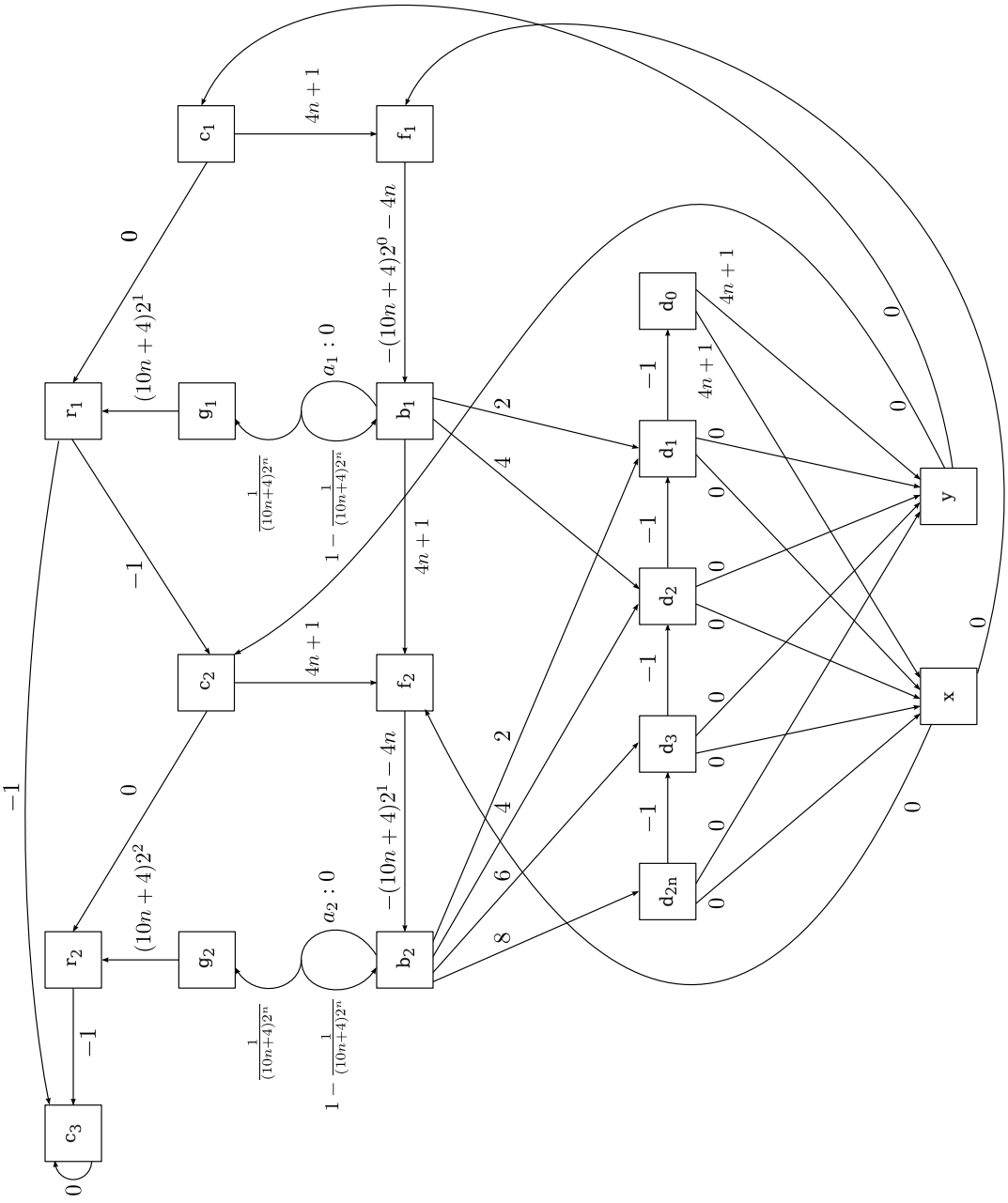

Figure 5.5: The MDP $\mathcal{G}_2$.

5.3 The Flip Phase

We begin by describing the sequence of strategies that greedy strategy improvement will pass through during the flip phase. We will specify a sequence of partial strategies for each gadget individually, and then combine these into a sequence of full strategies for the entire example.

For the deceleration lane, we begin with a strategy that picks the action (d_i, y) for every vertex d_i. In the first iteration, the action (d_1, d_0) will be switched. The action (d_i, d_{i-1}) can only be switched after every action (d_j, d_{j-1}) with $j < i$ has been switched. Therefore, we have the following sequence of partial strategies for the deceleration lane. For every j such that $j \geq 0$ we define a partial strategy:

$$\sigma_j(s) = \begin{cases} d_{k-1} & \text{if } s = d_k \text{ and } 1 \leq k \leq j, \\ y & \text{otherwise.} \end{cases} \tag{5.2}$$

We will give two sequences of partial strategies for the vertices b_i, one strategy for the vertices where $i \in B$, and one strategy for the vertices where $i \notin B$. As we have described, these vertices should follow the updates made by the deceleration lane. By this we mean that greedy strategy improvement will switch the action (b_i, d_j) in the iteration immediately after the action (d_j, d_{j-1}) is switched in the deceleration lane. The initial strategy for b_i will select the action y, and the behaviour that we have described will hold from the second iteration onwards. In the first iteration, greedy strategy improvement will switch the edge (b_i, d_{2i}). Formally, for every j in the range $0 \leq j \leq 2i + 1$ we define a partial strategy for the vertices

in the deceleration lane and each vertex b_i:

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

We also give a sequence of strategies for the vertices b_i, where $i \in B$. In order for the configuration to remain constant, these vertices should not switch away from the action a_i as the deceleration lane is being updated. Formally, for every j such that $j \geq 0$, we define a partial strategy:

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

Finally, we give a strategy for the vertices in the reset structure. Let B be a configuration. As we know, the vertices in the reset structure will not be switched away from this strategy while the deceleration lane is being updated. We have already described how the choice at the vertex c_i depends on whether $i \in B$. Each vertex r_i selects the action (r_i, c_j), where j is the smallest index such that $j > i$ and $j \in B$. If i is the largest index in B then the vertex r_i moves directly to the sink vertex c_{n+1}. The vertices x and y both move to the reset structure associated with

the smallest index i such that $i \in B$.

$$\sigma^B(c_i) = \begin{cases} f_i & \text{if } i \in B, \\ r_i & \text{if } i \notin B, \end{cases}$$

$$\sigma^B(r_i) = c_{\min(B^{>i} \cup \{n+1\})},$$

$$\sigma^B(y) = c_{\min(B)},$$

$$\sigma^B(x) = f_{\min(B)}.$$

We can now define the sequence of strategies that greedy strategy improvement will pass through during the flip phase, which combines the partial strategies that we have defined. For each improvable configuration B, we define Sequence$(B) = \langle \sigma_0^B, \sigma_1^B, \ldots \sigma_{2j+1}^B \rangle$, where $j = \min(\{1, 2, \ldots, n\} \setminus B)$ and:

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

We also define the strategy $\sigma_0^{\{1,2,\ldots,n\}}$ analogously. The rest of this section is dedicated to showing that if greedy strategy improvement is applied to the strategy σ_0^B, then it will pass through the sequence of strategies in Sequence(B).

5.3.1 Breaking The Example Down

In order to simplify our exposition, we will provide proofs for each gadget separately. To do this, we first need to provide some preliminary proofs that will be used to break the example down into pieces. These proofs concern the valuation of the vertices c_i. These vertices are important, because they are on the border between the gadgets

in the example. For example, we know that the vertex y in the deceleration lane always chooses an action of the form (y, c_i), and by knowing bounds of the valuation of c_i we will be able to prove properties of the deceleration lane irrespective of the strategy that is being played on the vertices in the reset structure.

Recall that for each improvable configuration B, the strategy σ_j^B chooses the action (c_i, f_i) if and only if $i \in B$. This implies that if we follow the strategy σ_j^B from some vertex c_i where $i \in B$, then we will pass through every vertex c_k where $k \in B^{>i}$. On the other hand, if we follow the strategy σ_j^B from some vertex c_i where $i \notin B$, then we will move to the vertex r_i, and then to the vertex c_k where $k = \min(B^{>i} \cup \{n+1\})$. The next proposition uses these facts to give a formula for the valuation of each vertex c_i in terms of the configuration B.

Proposition 5.2. *Let B be an improvable configuration and σ be a member of* Sequence(B). *For every i we have:*

$$\mathrm{Val}^\sigma(c_i) = \begin{cases} \sum_{j \in B^{\geq i}} (10n+4)(2^j - 2^{j-1}) & \text{if } i \in B, \\ \sum_{j \in B^{\geq i}} (10n+4)(2^j - 2^{j-1}) - 1 & \text{otherwise.} \end{cases}$$

Proof. We first consider the case where $i \in B$. If $k = \min(B^{>i} \cup \{n+1\})$ then the definition of σ gives:

$$\begin{aligned} \mathrm{Val}^\sigma(c_i) &= \mathrm{r}(c_i, f_i) + \mathrm{r}(f_i, b_i) + \mathrm{Val}^\sigma(b_i) \\ &= \mathrm{r}(c_i, f_i) + \mathrm{r}(f_i, b_i) + \mathrm{r}(g_i, r_i) + \mathrm{r}(r_i, c_k) + \mathrm{Val}^\sigma(c_k) \\ &= (4n+1) - ((10n+4)2^{i-1} - 4n) + (10n+4)2^i - 1 + \mathrm{Val}^\sigma(c_k) \\ &= (10n+4)(2^i - 2^{i-1}) + \mathrm{Val}^\sigma(c_k). \end{aligned}$$

If $k = n+1$ then we are done because $\mathrm{Val}^\sigma(c_{n+1}) = 0$. Otherwise, repeated substi-

tution of the above expression for $\mathrm{Val}^\sigma(c_k)$ gives:

$$\begin{aligned}\mathrm{Val}^\sigma(c_i) &= \sum_{j \in B^{\geq i}} (10n+4)(2^j - 2^{j-1}) + \mathrm{Val}^\sigma(c_{j+1}) \\ &= \sum_{j \in B^{\geq i}} (10n+4)(2^j - 2^{j-1}).\end{aligned}$$

We now consider the case where $i \notin B$. If $k = \min(B^{>i} \cup \{n+1\})$, then the definition of σ gives:

$$\begin{aligned}\mathrm{Val}^\sigma(c_i) &= \mathrm{r}(c_i, r_i) + \mathrm{r}(r_i, c_k) + \mathrm{Val}^\sigma(c_k) \\ &= \mathrm{Val}^\sigma(c_k) - 1 \\ &= \sum_{j \in B^{\geq i}} (10n+4)(2^j - 2^{j-1}) - 1.\end{aligned}$$

□

The characterisation given by Proposition 5.2 gives some important properties about the valuation of the vertex c_i. One obvious property is that passing through a vertex b_i, where $i \in B$, provides a positive reward. Therefore, if i and j are both indices in B and $i < j$, then the valuation of c_i will be larger than the valuation of c_j.

Proposition 5.3. *Let B be an improvable configuration and σ be a member of* Sequence(B). *For every $i \in B$ and $j \in B$ such that $i < j$, we have* $\mathrm{Val}^\sigma(c_i) > \mathrm{Val}^\sigma(c_j)$.

Proof. Let $C = B^{\geq i} \cap B^{<j}$ be the members of B that lie between indices i and $j-1$. Proposition 5.2, the fact that $i \in C$, and the fact that $2^j - 2^{j-1}$ is positive for every j imply that:

$$\mathrm{Val}^\sigma(c_i) = \sum_{j \in C} (10n+4)(2^j - 2^{j-1}) + \mathrm{Val}^\sigma(c_j) > \mathrm{Val}^\sigma(c_j).$$

□

Proposition 5.3 provides a lower bound for the valuation of a vertex c_i in

terms of a vertex c_j with $j > i$. Some of our proofs will also require a corresponding upper bound. The next proposition provides such a bound.

Proposition 5.4. *Let B be an improvable configuration and σ be a member of* Sequence(B). *For every $i \in B$ and $j \in B$ such that $j > i$, we have:*

$$\mathrm{Val}^\sigma(c_i) \leq \mathrm{Val}^\sigma(c_j) + (10n+4)(2^{j-1} - 2^{i-1}).$$

Proof. Let $C = B^{\geq i} \cap B^{<j}$ be the members of B that lie between indices i and $j-1$. Using Proposition 5.2 gives:

$$\begin{aligned}\mathrm{Val}^\sigma(c_i) &= \sum_{k \in B^{\geq i}} (10n+4)(2^k - 2^{k-1}) \\ &= \sum_{k \in C} (10n+4)(2^k - 2^{k-1}) + \mathrm{Val}^\sigma(c_j).\end{aligned}$$

We use the fact that $(10n+4)(2^k - 2^{k-1}) > 0$ for all k and the fact that $i > 0$ to obtain:

$$\begin{aligned}\sum_{k \in C} (10n+4)(2^k - 2^{k-1}) &\leq \sum_{k=i}^{j-1} (10n+4)(2^k - 2^{k-1}) \\ &= (10n+4)(2^{j-1} - 2^{i-1}).\end{aligned}$$

Therefore, we have:

$$\mathrm{Val}^\sigma(c_i) \leq \mathrm{Val}^\sigma(c_j) + (10n+4)(2^{j-1} - 2^{i-1}). \qquad \square$$

5.3.2 The Deceleration Lane

In this section we will prove that the deceleration lane behaves as we have described. In particular, we will provide a proof of the following proposition.

Proposition 5.5. *If* $\mathrm{Val}^{\sigma_i}(y) > \mathrm{Val}^{\sigma_i}(x)$ *for every i in the range $0 \leq i \leq 2n$,*

then applying greedy strategy improvement to σ_0 produces the sequence of strategies $\langle \sigma_0, \sigma_1, \ldots, \sigma_{2n} \rangle$.

This Proposition may seem strange at first sight, because each strategy σ_i is a partial strategy that is defined only for the vertices of the deceleration lane, and strategy improvement works only with full strategies. However, since the vertices x and y are the only vertices at which it is possible to leave the deceleration lane, placing an assumption on the valuations of these vertices will allow us to prove how greedy strategy improvement behaves on the deceleration lane. These proofs will hold irrespective of the decisions that are made outside the deceleration lane. In other words, if greedy strategy improvement arrives at a strategy σ that is consistent with σ_0 on the vertices in the deceleration lane, and if $\mathrm{Val}^\sigma(y) > \mathrm{Val}^\sigma(x)$, then Proposition 5.5 implies that greedy strategy improvement will move to a strategy σ' that is consistent with σ_1 on the vertices in the deceleration lane. This approach allows us to prove properties of the deceleration lane without having to worry about the behaviour of greedy strategy improvement elsewhere in the example.

Of course, this approach will only work if we can prove that $\mathrm{Val}^\sigma(y) > \mathrm{Val}^\sigma(x)$. The next proposition confirms that this property holds for every strategy in Sequence(B).

Proposition 5.6. *For every improvable configuration B and every strategy σ in* Sequence(B) *we have* $\mathrm{Val}^\sigma(y) > \mathrm{Val}^\sigma(x)$.

Proof. By the definition of σ, we have that there is some index $i \in B$ such that $\sigma(y) = c_i$ and $\sigma(x) = f_i$. Moreover, since $i \in B$ we have that $\sigma(c_i) = f_i$. We therefore have the following two equalities:

$$\mathrm{Val}^\sigma(y) = \mathrm{r}(y, c_i) + \mathrm{r}(c_i, f_i) + \mathrm{Val}^\sigma(f_i) = \mathrm{Val}^\sigma(f_i) + 4n + 1,$$
$$\mathrm{Val}^\sigma(x) = \mathrm{r}(x, f_i) + \mathrm{Val}^\sigma(f_i) = \mathrm{Val}^\sigma(f_i).$$

Clearly, since $4n + 1 > 0$ we have $\mathrm{Val}^\sigma(y) > \mathrm{Val}^\sigma(x)$. □

We now turn our attention to proving that greedy strategy improvement moves from the strategy σ_i to the strategy σ_{i+1}. To prove properties of greedy strategy improvement, we must know the appeal of every action leaving the vertices d_j. The next proposition gives a characterisation for the appeal of the actions (d_j, d_{j-1}) for each strategy σ_i.

Proposition 5.7. *For each strategy σ_i we have:*

$$\text{Appeal}^{\sigma_i}(d_j, d_{j-1}) = \begin{cases} \text{Val}^{\sigma_i}(y) + 4n - j + 1 & \text{if } 1 \leq j \leq i+1, \\ \text{Val}^{\sigma_i}(y) - 1 & \text{if } i+1 < j \leq 2n. \end{cases}$$

Proof. We begin by considering the case where $1 \leq j \leq i+1$. By the definition of σ_i we have that $\sigma_i(d_j) = d_{j-1}$ for all vertices d_j with $1 \leq j \leq i$, and we have that $\sigma_i(d_0) = y$. Using the definition of appeal, and applying the optimality equation (5.1) repeatedly gives, for every action (d_j, d_{j-1}) with $0 \leq j \leq i+1$:

$$\begin{aligned} \text{Appeal}^{\sigma_i}(d_j, d_{j-1}) &= \text{r}(d_j, d_{j-1}) + \text{Val}^{\sigma_i}(d_j) \\ &= \sum_{k=1}^{j} \text{r}(d_k, d_{k-1}) + \text{r}(d_0, y) + \text{Val}^{\sigma_i}(y) \\ &= -j + (4n+1) + \text{Val}^{\sigma_i}(y). \end{aligned}$$

We now consider the case where $i+1 < j \leq 2n$. By definition we have that $\sigma_i(d_{j-1}) = y$. Using the definition of appeal and the optimality equation gives:

$$\begin{aligned} \text{Appeal}^{\sigma_i}(d_j, d_{j-1}) &= \text{r}(d_j, d_{j-1}) + \text{Val}^{\sigma_i}(d_{j-1}) \\ &= \text{r}(d_j, d_{j-1}) + \text{r}(d_{j-1}, d_y) + \text{Val}^{\sigma_i}(y) \\ &= \text{Val}^{\sigma_i}(y) - 1. \end{aligned}$$

□

Proposition 5.7 confirms that an action (d_j, d_{j-1}) is switchable only after every action (d_k, d_{k-1}) with $k < j$ has been switched by greedy strategy improve-

ment. It can clearly be seen that the only action of this form that is switchable in the strategy σ_i is the action (d_{i+1}, d_i). Every other action of this form has either already been switched, or is obviously not switchable.

We must also consider the actions (d_i, x). The next proposition confirms that these actions will not be switchable in the strategy σ_i.

Proposition 5.8. *If* $\mathrm{Val}^{\sigma_i}(x) < \mathrm{Val}^{\sigma_i}(y)$ *then* $\mathrm{Appeal}^{\sigma_i}(d_j, x) < \mathrm{Appeal}^{\sigma_i}(d_j, y)$ *for all* j.

Proof. Using the definition of appeal for the vertex d_j gives two equalities:

$$\mathrm{Appeal}^{\sigma_i}(d_j, y) = \mathrm{r}(d_j, y) + \mathrm{Val}^{\sigma_i}(y),$$
$$\mathrm{Appeal}^{\sigma_i}(d_j, x) = \mathrm{r}(d_j, x) + \mathrm{Val}^{\sigma_i}(x).$$

Observe that for all j we have $\mathrm{r}(d_j, y) = \mathrm{r}(d_j, x)$. Therefore we can conclude that when $\mathrm{Val}^{\sigma_i}(x) < \mathrm{Val}^{\sigma_i}(y)$ we have $\mathrm{Appeal}^{\sigma_i}(d_j, x) < \mathrm{Appeal}^{\sigma_i}(d_j, y)$. □

In summary, Proposition 5.7 has shown that there is exactly one action of the form (d_j, d_{j-1}) that is switchable in σ_i, and that this action is (d_{i+1}, d_i). Proposition 5.8 has shown that no action of the form (d_j, x) is switchable in σ_i. It is obvious that no action of the form (d_j, y) is switchable in σ_i. Therefore, greedy strategy improvement will switch the action (d_{i+1}, d_i) in the strategy σ_i, which creates the strategy σ_{i+1}. Therefore, we have shown Proposition 5.5.

5.3.3 Zero Bits

In this section we will prove that greedy strategy improvement behaves as we describe for the vertices that represent the 0 bits of a configuration. As with our proof of the deceleration lane behaviour, we will provide a proof that deals with partial strategies under the assumption that $\mathrm{Val}^{\sigma}(y) > \mathrm{Val}^{\sigma}(x)$ always holds. Proposition 5.6 implies that this assumption is valid. Therefore, this section is concerned

with proving the following proposition.

Proposition 5.9. *If* $\mathrm{Val}^{\sigma_j^o}(y) > \mathrm{Val}^{\sigma_j^o}(x)$ *for every* j *in the range* $0 \leq i \leq 2i+1$*, then applying greedy strategy improvement to* σ_0^o *produces the sequence of strategies* $\langle \sigma_0^o, \sigma_1^o, \ldots, \sigma_{2i+1}^o \rangle$.

The key property that will be used to prove this Proposition is that the appeal of the action a_i at each vertex b_i, where $i \notin B$, is bounded. Since greedy strategy improvement always switches the action with maximal appeal, we will prove this proposition by showing that there is always some action at b_i that has a larger appeal than the action a_i. The next proposition gives the bound for the action a_i.

Proposition 5.10. *For every improvable configuration* B*, and every strategy* σ *in* $\mathrm{Sequence}(B)$*, we have that if* $\sigma(b_i) \neq a_i$ *then* $\mathrm{Appeal}^{\sigma}(b_i, a_i) < \mathrm{Val}^{\sigma}(b_i) + 1$.

Proof. To prove this, we will first show that $\mathrm{Val}^{\sigma}(b_i) > 0$, and that $\mathrm{Val}^{\sigma}(g_i) \leq (10n+4)2^n$, for each strategy σ. We begin by showing that $\mathrm{Val}^{\sigma}(b_i) > 0$. We know that $\sigma(b_i) = d_k$ for some k, and that $\sigma(d_l) = d_{l-1}$ for all l in the range $1 \leq l \leq k$. We can therefore apply Proposition 5.7 and Proposition 5.2 to get:

$$\begin{aligned}\mathrm{Val}^{\sigma}(b_i) &= 4n + k + 1 + \mathrm{Val}^{\sigma}(y) \\ &= 4n + k + 1 + \sum_{j \in B^{\geq i}} (10n+4)(2^j - 2^{j-1}).\end{aligned}$$

Since $k > 0$ we have that $(4n + k + 1) > 0$, and we have already argued that the summation will be non-negative. This implies that the entire expression will be positive. Therefore, we have shown that $\mathrm{Val}^{\sigma}(b_i) > 0$ for all i.

Secondly, we argue that $\mathrm{Val}^{\sigma}(g_i) \leq (10n+4)2^n$. If $k = \min(B \cup \{n+1\})$ then we have:

$$\mathrm{Val}^{\sigma}(g_i) = (10n+4)2^i + \mathrm{Val}^{\sigma}(r_i) = (10n+4)2^i - 1 + \mathrm{Val}^{\sigma}(c_k).$$

If $k = n+1$ then we are done because $\mathrm{Val}^{\sigma}(c_{n+1}) = 0$ and $(10n+4)2^i - 1 < (10n+4)2^n$

for all $i \leq n$. Otherwise, we can apply Proposition 5.2 and the fact that $k-1 \geq i$ to obtain:

$$\begin{aligned}
\mathrm{Val}^{\sigma}(g_i) &\leq (10n+4)2^i - 1 + \mathrm{Val}^{\sigma}(c_{n+1}) + (10n+4)(2^n - 2^{k-1}) \\
&\leq \mathrm{Val}^{\sigma}(c_{n+1}) + (10n+4)2^n - 1 \\
&\leq (10n+4)2^n.
\end{aligned}$$

Finally, we can use these two inequalities to prove the proposition:

$$\begin{aligned}
\mathrm{Appeal}(b_i, a_i) &= \mathrm{r}(b_i, a_i) + \sum_{s \in S} p(s|b_i, a_i)\,\mathrm{Val}^{\sigma}(s) \\
&= 0 + (1 - \frac{2^{-(n)}}{10n+4})\,\mathrm{Val}^{\sigma}(b_i) + \frac{2^{-(n)}}{10n+4}\,\mathrm{Val}^{\sigma}(g_i) \\
&< \mathrm{Val}^{\sigma}(b_i) + \frac{2^{-n}}{10n+4}\,\mathrm{Val}^{\sigma}(g_i) \\
&\leq \mathrm{Val}^{\sigma}(b_i) + \frac{2^{-n}}{10n+4} \cdot (10n+4)2^n \\
&= \mathrm{Val}^{\sigma}(b_i) + 1.
\end{aligned}$$

□

Recall that Proposition 5.7 gave a characterisation for the appeal of the action (d_k, d_{k-1}) in terms of the current strategy for the deceleration lane. The next proposition provides an analogous characterisation for the actions (b_i, d_k) in terms of the strategies σ_j^o and σ_j^c.

Proposition 5.11. *If σ is either σ_j^o or σ_j^c then we have:*

$$\mathrm{Appeal}^{\sigma}(b_i, d_k) = \begin{cases} \mathrm{Val}^{\sigma}(y) + 4n + k + 1 & \text{if } 1 \leq k \leq j, \\ \mathrm{Val}^{\sigma}(y) + 2k & \text{if } j < k \leq 2i. \end{cases}$$

Proof. We first consider the case where $1 \leq k \leq j$. Since $\sigma(d_k) = \sigma_j(d_k)$ for every

vertex d_k, we have:

$$\begin{aligned}\text{Appeal}^{\sigma}(b_i, d_k) &= \text{r}(b_i, d_k) + \text{Val}^{\sigma}(d_k) \\ &= 2k + (\text{Val}^{\sigma}(y) + 4n - k + 1) \\ &= \text{Val}^{\sigma}(y) + 4n + k + 1.\end{aligned}$$

In the case where $j < k \leq 2i$, the fact that $\sigma(d_k) = y$ when $1 < k \leq 2n$ gives:

$$\begin{aligned}\text{Appeal}^{\sigma}(b_i, d_k) &= \text{r}(b_i, d_k) + \text{Val}^{\sigma}(d_k) \\ &= \text{r}(b_i, d_k) + \text{r}(d_k, y) + \text{Val}^{\sigma}(y) \\ &= 2k + \text{Val}^{\sigma}(y).\end{aligned}$$

□

The characterisation given by Proposition 5.11 explains the behaviour of greedy strategy improvement for bit vertices that represent a 0. In the strategy σ_0^o, no vertex d_k can satisfy $1 \leq k \leq j$, because $j = 0$. This implies that every action (b_i, d_k) will have appeal $2k$. Therefore, the most appealing action of this form will be (b_i, d_{2i}), and switching this action creates σ_1^o. In each subsequent iteration, it is obvious that the most appealing action of this form in σ_j^o will be (b_i, d_j), and switching this action creates σ_{j+1}^o.

The key property to note here is that the appeal of the action (b_i, d_j) in the strategy σ_j^o is $\text{Val}^{\sigma_j^o}(b_i) + 1$. Proposition 5.10 implies that the appeal of the action a_i must be smaller than $\text{Val}^{\sigma_j^o}(b_i) + 1$, which means that the action a_i will not be switched by greedy strategy improvement at the vertex b_i in this strategy.

So far, we have only considered actions of the form (b_i, d_k). To complete the proof of Proposition 5.9 we must also consider the other actions that leave the vertex b_i. It is obvious that the actions (b_i, x) and (b_i, y) will not be switchable in any strategy σ_j^o. However, this still leaves the actions (b_i, f_j) for each $j > i$. The next proposition confirms that these actions are not switchable in any strategy in

Sequence(B), which completes the proof of Proposition 5.9.

Proposition 5.12. *Let B be an improvable configuration and let σ be a member of* Sequence(B). *For each action (b_i, f_j) where $i \notin B$, we have* $\text{Appeal}^\sigma(b_i, f_j) < \text{Val}^\sigma(b_i)$.

Proof. To prove this proposition we must consider two cases. Firstly, when $j \in B$ we can apply Proposition 5.11, the fact that $k \geq 0$, the fact that $\min(B) \leq j$, and Proposition 5.3 to give:

$$\begin{aligned}
\text{Val}^\sigma(b_i) &= \text{Val}^\sigma(y) + 4n + k + 1 \geq \text{Val}^\sigma(c_{\min(B)}) + 4n + 1 \\
&\geq \text{Val}^\sigma(c_j) + 4n + 1 \\
&= \text{Val}^\sigma(f_j) + 8n + 2 \\
&> \text{Val}^\sigma(f_j) + 4n + 1 = \text{Appeal}^\sigma(b_i, f_j).
\end{aligned}$$

Secondly, we consider the case where $j \notin B$. In this case, the fact that $\sigma(b_i) = \sigma(b_j)$ gives:

$$\begin{aligned}
\text{Appeal}^\sigma(b_i, f_j) &= 4n + 1 + \text{Val}^\sigma(f_j) \\
&= -(10n + 4)2^{j-1} + 1 + \text{Val}^\sigma(b_j) \\
&= -(10n + 4)2^{j-1} + 1 + \text{Val}^\sigma(\sigma(b_j)) \\
&= -(10n + 4)2^{j-1} + 1 + \text{Val}^\sigma(\sigma(b_i)) < \text{Val}^\sigma(\sigma(b_i)).
\end{aligned}$$

□

5.3.4 One Bits

In this section, we consider the vertices b_i that represent 1 bits in a configuration. We must prove that greedy strategy improvement never switches away from the action a_i at these vertices. As usual, we will consider the partial sequence of strategies defined by σ_j^c. The purpose of this section is to provide a proof for the following proposition.

Proposition 5.13. *If* $\mathrm{Val}^{\sigma_i^c}(y) > \mathrm{Val}^{\sigma_i^c}(x)$ *for every* i *in the range* $0 \leq i \leq 2i+1$*, then applying greedy strategy improvement to* σ_0^c *produces the sequence of strategies* $\langle \sigma_0^c, \sigma_1^c, \ldots, \sigma_{2i+1}^c \rangle$.

Proposition 5.11 gives an upper bound for the appeal of an action (b_i, d_k) in terms of the valuation of the vertex y. It implies that no action of this form can have an appeal that is larger than $\mathrm{Val}^\sigma(y) + 6n + 1$. In order to show that greedy strategy improvement never switches away from the action a_i, we must show that the valuation of the vertex b_i is always larger than this amount. The next proposition provides a proof of this fact, which implies that no action of the form (b_i, d_k) where $i \in B$ is switchable.

Proposition 5.14. *For every improvable configuration* B *and every strategy* σ *in* $\mathrm{Sequence}(B)$ *we have* $\mathrm{Val}^\sigma(y) + 6n + 1 < \mathrm{Val}^\sigma(b_i)$*, for every index* $i \in B$.

Proof. The definition of σ implies that $\sigma(y) = c_{\min(B)}$. Applying Proposition 5.2 gives:

$$\begin{aligned}\mathrm{Val}^\sigma(y) &= \mathrm{Val}^\sigma(c_{\min(B)}) \\ &\leq \mathrm{Val}^\sigma(c_i) + (10n+4)(2^{i-1} - 2^{\min(B)-1}).\end{aligned}$$

Since $i \in B$ we have $\sigma(c_i) = f_i$. Therefore we can apply the optimality equation, Proposition 5.4, and the fact that $\min(B) - 1 \geq 0$ to obtain:

$$\begin{aligned}\mathrm{Val}^\sigma(y) &\leq \mathrm{Val}^\sigma(c_i) + (10n+4)(2^{i-1} - 2^{\min(B)-1}) \\ &\leq \mathrm{Val}^\sigma(c_i) + (10n+4)(2^{i-1} - 2^0) \\ &= \mathrm{Val}^\sigma(f_i) + (4n+1) + (10n+4)(2^{i-1} - 1) \\ &= \mathrm{Val}^\sigma(b_i) - (10n+4)2^{i-1} - 4n + (4n+1) + (10n+4)(2^{i-1} - 1) \\ &= \mathrm{Val}^\sigma(b_i) - (10n+3).\end{aligned}$$

It is now clear that $\mathrm{Val}^\sigma(y) + 6n + 1 < \mathrm{Val}^\sigma(b_i)$. □

To complete the proof of Proposition 5.13 we must consider the other actions that leave the vertex b_i. The actions (b_i, x) and (b_i, y) are obviously not switchable, which leaves only the actions of the form (b_i, f_j) with $j > i$. The next proposition is an analog of Proposition 5.12, which confirms that no action of this form can be switchable. This implies that there are no switchable actions at the vertex b_i, and so greedy strategy improvement will never switch away from the action a_i. Therefore, after proving this proposition, we will have completed the proof of Proposition 5.13.

Proposition 5.15. *Let B be an improvable configuration and let σ be a member of* Sequence(B). *For each action (b_i, f_j) where $i \in B$, we have* $\mathrm{Appeal}^\sigma(b_i, f_j) < \mathrm{Val}^\sigma(b_i)$.

Proof. We begin by considering the case where $j \in B$. In this case we can use the fact that $\min(B^{>i}) \leq j$, Proposition 5.3, and the fact that $i > 0$ to obtain:

$$\begin{aligned}
\mathrm{Val}^\sigma(b_i) &= (10n+4)2^i + \mathrm{Val}^\sigma(r_i) \\
&= (10n+4)2^i - 1 + \mathrm{Val}^\sigma(c_{\min(B^{>i})}) \\
&\geq (10n+4)2^i - 1 + \mathrm{Val}^\sigma(c_j) \\
&= (10n+4)2^i + 4n + \mathrm{Val}^\sigma(f_j) \\
&> \mathrm{Val}^\sigma(f_j) + 4n + 1 = \mathrm{Appeal}^\sigma(b_i, f_j).
\end{aligned}$$

Secondly, we consider the case where $j \notin B$. The fact that $k \leq 2n$, implies:

$$\begin{aligned}
\mathrm{Appeal}^\sigma(b_i, f_j) &= -(10n+4)2^j + 1 + \mathrm{Val}^\sigma(b_j) \\
&= -(10n+4)2^j + 4n + k + 2 + \mathrm{Val}^\sigma(y) \\
&\leq -(10n+4)2^j + 6n + 2 + \mathrm{Val}^\sigma(c_{\min(B)}).
\end{aligned}$$

Let $l = \min(B^{>j} \cup \{n+1\})$ be the smallest bit in the configuration that is larger

than j. By Proposition 5.2, and the fact that there is no bit in the configuration with an index m in the range $j \leq m < l$, we have:

$$\begin{aligned}\mathrm{Val}^{\sigma}(c_{\min(B)}) &= \sum_{j \in B^{<j}} (10n+4)(2^{j} - 2^{j-1}) + \mathrm{Val}^{\sigma}(c_l) \\ &\leq \mathrm{Val}^{\sigma}(c_l) + (10n+4)(2^{j-1} - 2^{0}).\end{aligned}$$

Therefore, we have:

$$\begin{aligned}\mathrm{Appeal}^{\sigma}(b_i, f_j) &\leq \mathrm{Val}^{\sigma}(c_l) + (10n+4)(2^{j-1} - 1 - 2^{j}) + 6n + 2 \\ &\leq \mathrm{Val}^{\sigma}(c_l) + (10n+4)(2^{j-1} - 2^{j}) \\ &< \mathrm{Val}^{\sigma}(c_l).\end{aligned}$$

However, Proposition 5.3, and the fact that $i \geq 0$ imply:

$$\begin{aligned}\mathrm{Val}^{\sigma}(b_i) &= (10n+4)2^{i} + \mathrm{Val}^{\sigma}(r_i) = (10n+4)2^{i} - 1 + \mathrm{Val}^{\sigma}(c_{\min(B^{>i})}) \\ &\geq (10n+4)2^{i} - 1 + \mathrm{Val}^{\sigma}(c_l) \\ &\geq \mathrm{Val}^{\sigma}(c_l).\end{aligned}$$

□

5.3.5 The Reset Structure

In this section, we will prove that greedy strategy improvement passes through the sequence of strategies given by Sequence(B). Since each strategy in this sequence is a complete strategy, we are now proving the behaviour of greedy strategy improvement for the full example. The purpose of this section is give a proof of the following proposition.

Proposition 5.16. *For every improvable configuration B, if greedy strategy improvement is applied to σ_0^B, then it will pass through the sequence of strategies given by* Sequence(B).

Propositions 5.5, 5.9, and 5.13 provide a proof of this proposition for the vertices d_k for all k, and for the vertices b_i for all i. Therefore, in this section we will concern ourselves with the vertices x and y, and the vertices c_i and r_i for all i. Note that none of the vertices are switched during Sequence(B), so we must therefore show that there is never a switchable action at these vertices.

We begin with the vertex y. This vertex has an outgoing action (y, c_j) for each j in the range $1 \leq j \leq n+1$. We have already described the properties of the vertex c_i in Section 5.3.1. The next proposition uses these properties to show that the vertex c_i with $i = \min(B)$ has the largest valuation amongst the vertices c_j. Since each action (y, c_j) has the same reward, this implies that no action can be switchable at the vertex y. Therefore, greedy strategy improvement will not switch away from this action as it passes through Sequence(B).

We will prove a version of the proposition that is more general than is needed for the vertex y. This is because, we also want to apply the proposition to prove that each vertex r_i is not switched by greedy strategy improvement. Whereas the vertex y always chooses the action (y, c_j) such that $j = \min(B)$, the vertex r_i must choose the action (r_i, c_k) such that $k = \min(B^{>i})$. The next proposition will also show that the vertex c_k has a largest valuation amongst the vertices c_l with $l \in B^{>i}$. As usual, since every action leaving the vertex r_i has the same reward, this implies that no action at r_i will become switchable.

Proposition 5.17. *Let B be an improvable configuration and σ be a member of* Sequence(B). *For each k in the range $1 \leq k \leq n$, if $i = \min(B^{>k})$ then we have $\mathrm{Val}^\sigma(c_i) > \mathrm{Val}^\sigma(c_j)$ for every j such that $j > k$ and $j \neq i$.*

Proof. Proposition 5.3 implies that the vertex c_i has a higher valuation than every other vertex c_j with $j \in B^{>k}$. To complete the proof we must eliminate the vertices c_j with $j \in \{k+1, k+2, \ldots n\} \setminus B^{>k}$. We will accomplish this by arguing that for every such vertex c_j there is some index $l \in B^{>j} \cup \{n+1\}$ such that $\mathrm{Val}^\sigma(c_l) > \mathrm{Val}^\sigma(c_j)$. We choose $l = \min(B^{>j} \cup \{n+1\})$ to be the smallest index

in $B^{>j}$ that is larger than j, or the index of the sink if j is the largest index in $B^{>k}$. Since $j \notin B^{>k}$ we have:

$$\begin{aligned}\mathrm{Val}^{\sigma}(c_j) &= \mathrm{r}(c_j, r_j) + \mathrm{r}(r_j, c_k) + \mathrm{Val}^{\sigma}(c_k) \\ &= \mathrm{Val}^{\sigma}(c_k) - 1 < \mathrm{Val}^{\sigma}(c_k).\end{aligned}$$ □

We now turn our attention to the vertex x. Much like the vertex y, this vertex has an outgoing action (x, f_j) for each j in the range $1 \leq j \leq n$. Our proof for this vertex will split into two cases. We begin by considering the actions (x, f_j) for which $j \notin B$. The next proposition shows that these actions are not switchable for any strategy in Sequence(B).

Proposition 5.18. *Let B be an improvable configuration and σ be a member of* Sequence(B). *If $i \notin B$ we have $\mathrm{Val}^{\sigma}(f_i) + 4n + 1 < \mathrm{Val}^{\sigma}(c_k) - 1$, where $k = \min(B^{>i} \cup \{n+1\})$.*

Proof. Using Proposition 5.11, and the fact that $i \leq n$, gives the following expression for the value of f_i.

$$\begin{aligned}\mathrm{Val}^{\sigma}(f_i) &\leq -(10n+4)2^{i-1} - 4n + 6n + 1 + \mathrm{Val}^{\sigma}(y) \\ &= -(10n+4)2^{i-1} + 2n + 1 + \mathrm{Val}^{\sigma}(y).\end{aligned}$$

Using Proposition 5.2 to obtain the valuation of y in terms of the vertex c_k gives:

$$\begin{aligned}
\mathrm{Val}^\sigma(f_i) &= -(10n+4)2^{i-1} + 2n + 1 + \sum_{j \in B^{<i}} (10n+4)(2^j - 2^{j-1}) + \mathrm{Val}^\sigma(c_k) \\
&\leq -(10n+4)2^{i-1} + 2n + 1 + \sum_{j=0}^{i-1} (10n+4)(2^j - 2^{j-1}) + \mathrm{Val}^\sigma(c_k) \\
&= -(10n+4)2^{i-1} + 2n + 1 + (10n+4)(2^{i-1} - 2^0) + \mathrm{Val}^\sigma(c_k) \\
&= 2n + 1 + -(10n+4) + \mathrm{Val}^\sigma(c_k) \\
&= -8n - 3 + \mathrm{Val}^\sigma(c_k).
\end{aligned}$$

Therefore $\mathrm{Val}^\sigma(f_i) + 4n + 1 < \mathrm{Val}^\sigma(c_k) - 1$. □

Now that we know that each action (x, f_j) where $j \notin B$ cannot be switchable, we can consider the actions (x, f_j) where $j \in B$. We know that $\sigma(c_j) = f_j$ for each $j \in B$. Therefore, the valuation of the vertex f_j is strongly related to the valuation of c_j, when $j \in B$. Since we know that the vertex c_i where $i = \min(B)$ has a larger valuation than any other vertex c_j, we can use this connection to argue that the vertex f_i has a larger valuation than any other vertex f_j. Since each action (x, f_j) has the same reward, this implies that no action will become switchable at x as greedy strategy improvement moves through the strategies in Sequence(B).

Proposition 5.19. *Let B be an improvable configuration and σ be a member of* Sequence(B). *If $i = \min(B)$ then we have $\mathrm{Val}^\sigma(f_i) > \mathrm{Val}^\sigma(f_j)$ for every $j \neq i$.*

Proof. We begin by arguing that $\mathrm{Val}^\sigma(f_i) > \mathrm{Val}^\sigma(f_j)$ for the case where $j \in B$. Since $\mathrm{Val}^\sigma(c_k) = \mathrm{Val}^\sigma(f_k) + 4n + 1$ for every $k \in B$, we can apply Proposition 5.3 to obtain:

$$\mathrm{Val}^\sigma(f_i) = \mathrm{Val}^\sigma(c_i) - 4n - 1 > \mathrm{Val}^\sigma(c_j) - 4n - 1 = \mathrm{Val}^\sigma(f_j).$$

We now complete the proof by arguing that for every vertex f_j with $j \notin B$, there is

some vertex f_k with $k \in B$ such that $\mathrm{Val}^\sigma(f_k) > \mathrm{Val}^\sigma(f_j)$. Proposition 5.18 implies that $\mathrm{Val}^\sigma(f_i) + 4n + 1 < \mathrm{Val}^\sigma(c_k)$ where $k = \min(B \cup \{n+1\})$. Therefore we have:

$$\mathrm{Val}^\sigma(f_j) < \mathrm{Val}^\sigma(c_k) - (4n+1) = \mathrm{Val}^\sigma(f_k). \qquad \square$$

Now we consider the vertex c_i. The action chosen at this vertex depends on whether the index i is contained in B. The next proposition shows that in either case, the vertex c_i will not be switched away from the action that it chose in the first strategy in Sequence(B). After proving this proposition we will have shown that greedy strategy improvement will not switch the vertices x and y, or the vertices c_i and r_i for every i. Therefore, we will have completed the proof of Proposition 5.16.

Proposition 5.20. *Let B be an improvable configuration and σ be a member of* Sequence(B). *The vertex c_i will not be switched away from $\sigma(c_i)$.*

Proof. First we will consider the case where $i \in B$, where we must show that the vertex c_i does not switch away from the action (c_i, f_i). In this case, the fact $2^i - 2^{i-1} > 0$ implies:

$$\begin{aligned}\mathrm{Appeal}^\sigma(c_i, f_i) &= (10n+4)(2^i - 2^{i-1}) + 1 + \mathrm{Val}^\sigma(r_i) \\ &> \mathrm{Val}^\sigma(r_i) = \mathrm{Appeal}^\sigma(c_i, r_i).\end{aligned}$$

Therefore, greedy strategy improvement will not switch away from the action (c_i, f_i).

Now we consider the case where $i \notin B$. In this case Proposition 5.18 implies that if $k = \min(B^{>i} \cup \{n+1\})$ then:

$$\mathrm{Appeal}^\sigma(c_i, f_i) = 4n + 1 + \mathrm{Val}^\sigma(f_i) < \mathrm{Val}^\sigma(c_k) - 1.$$

We also have that $\mathrm{Val}^\sigma(c_i) = \mathrm{Val}^\sigma(c_k) - 1$. Therefore, greedy strategy improvement will not switch away from the current action at c_k. $\square$

5.4 The Reset Phase

In this section, we finish our description of the behaviour of greedy strategy improvement on the examples $\mathcal{G}_n$ by describing the reset phase. Let B be an improvable configuration, and let $i = \min(\{1, 2, \ldots, n\} \setminus B)$ be the smallest index that is not in B. We define the configuration $B' = (B \cup \{i\}) \setminus \{1, 2, \ldots, i-1\}$, which is the configuration that a binary counter would move to from B. We will show that greedy strategy improvement moves from the final strategy in Sequence(B) to the strategy $\sigma_0^{B'}$.

We begin by describing the sequence of strategies that greedy strategy improvement passes through. The sequence begins at the final strategy in Sequence(B), which is σ_{2i+1}^B. At every vertex other than b_i, greedy strategy improvement moves to the strategy σ_{2i+2}^B. However, since this strategy cannot be switched at the vertex b_i, the action a_i is switched instead. Note that σ_{2i+2} is well defined even if $i = n$. This is because, although σ_{2i+2}^o is not well defined in this case, every vertex b_j with $j \neq i$ must play σ_{2i+2}^c, and this strategy is well defined. This comment also applies to the strategy σ_{2i+3}^B, which is used in subsequent reset strategies. Therefore, greedy strategy improvement will move to the strategy σ_{R1}^B, which is defined as, for every vertex v:

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

Switching the action a_i to create σ_{R1}^B will cause the valuation of the vertex f_i to dramatically increase. This causes greedy strategy improvement to switch the action (x, f_i), the action (c_i, f_i), and the actions (b_j, f_i) for $j < i$. The vertex b_i is not switched away from the action a_i, and every other vertex is switched to the strategy σ_{2i+3}^B. Therefore, greedy strategy improvement moves to the strategy σ_{R2}^B, which is defined as, for every vertex v:

$$\sigma^B_{\mathrm{R2}}(v) = \begin{cases} a_i & \text{if } v = b_i \\ f_i & \text{if } v = x \text{ or } v = c_i \text{ or } v = b_j \text{ with } j < i, \\ \sigma^B_{2i+3}(v) & \text{otherwise.} \end{cases}$$

When greedy strategy improvement moves to the strategy σ^B_{R2} it switches the vertex x to the action (x, f_i), which causes the valuation of x to increase. This causes greedy strategy improvement to switch the action (d_k, x) at every state d_k, and the action (b_j, x) at every state b_j with $j \notin B'$. The increase in valuation that occurred when the vertex c_i was switched to (c_i, f_i) causes greedy strategy improvement to switch the action (y, c_i), and the actions (r_j, c_i), with $j < i$. It is worth noting that at this point that the vertices in the set $\{c_j, r_j \ : \ j \geq i\} \cup \{b_j \ : \ j \in B'\} \cup \{x\}$ are now playing $\sigma^{B'}_0$. For every vertex v, we define the strategy σ^B_{R3} as:

$$\sigma^B_{\mathrm{R3}}(v) = \begin{cases} \sigma^{B'}_0(v) & \text{if } v \in \{c_j, r_j \ : \ j \geq i\} \cup \{b_j \ : \ j \in B'\} \cup \{x\}, \\ c_i & v = y \text{ or } v = r_j \text{ with } j < i, \\ x & v = d_k \text{ for some } k \text{ or } v = b_j \text{ with } j \notin B', \\ \sigma^B_{2i+3}(s) & \text{if } v = c_j \text{ with } j < i. \end{cases}$$

We can now describe how greedy strategy improvement behaves once it has arrived at the strategy σ^B_{R3}. The increase in valuation that was obtained when the vertex y was switched to the action (y, c_i) causes greedy strategy improvement to switch the actions (d_k, y) for every vertex d_k, and the actions (b_j, y) for every vertex b_j with $j \notin B'$. Also, for every j in the range $1 < j < i$ the increase in valuation that was obtained when the vertex r_j was switched to the action (r_j, c_i) causes greedy strategy improvement to switch the action (c_j, r_j). After making these

switches, greedy strategy improvement arrives at the strategy $\sigma_0^{B'}$, which completes the reset phase for the configuration B. In the remainder of this section, we will prove that greedy strategy improvement behaves as we have described.

5.4.1 The First Reset Strategy

The purpose of this section is to provide a proof for the following proposition.

Proposition 5.21. *Greedy strategy improvement moves from the strategy σ_{2i+1}^B to the strategy σ_{R1}^B.*

We will begin by considering the vertex b_i. At this vertex we must show that greedy strategy improvement switches the action a_i. The next proposition shows that a_i is indeed the most appealing action at b_i in σ_{2i+1}^B. This proposition also proves the claim for the strategy σ_{R1}^B, as this fact will be needed in the subsequent section.

Proposition 5.22. *If σ is σ_{2i+1}^B or σ_{R1}^B, then the action a_i is the most appealing action at b_i.*

Proof. We will begin by showing that every action other than a_i is not switchable in σ. Note that since $\mathrm{Val}^{\sigma_{\mathrm{R1}}^B}(b_i) > \mathrm{Val}^{\sigma_{2i+1}^B}(b_i)$, it is sufficient to prove that the appeal of every action is strictly less than $\mathrm{Val}^{\sigma_{2i+1}^B}(b_i)$.

We first consider the actions of the form (b_i, d_k). It is not difficult to verify that $\mathrm{Val}^{\sigma_{2i+1}^B}(d_k) = \mathrm{Val}^{\sigma_{\mathrm{R1}}^B}(d_k)$ for every $k \leq 2i$. Applying Proposition 5.11 to the strategy σ_{2i+1}^B, using the fact that $k \leq 2i$ gives:

$$\begin{aligned}
\mathrm{Appeal}^{\sigma}(b_i, d_k) &= \mathrm{Val}^{\sigma_{2i+1}^B}(y) + 4n + k + 1 \\
&\leq \mathrm{Val}^{\sigma_{2i+1}^B}(y) + 4n + 2i + 1 \\
&= \mathrm{Appeal}^{\sigma_{2i+1}^B}(b_i, d_{2i}) = \mathrm{Val}^{\sigma_{2i+1}^B}(b_i).
\end{aligned}$$

We now consider the actions (b_i, y) and (b_i, x). Again, it can easily be verified

that $\mathrm{Val}^{\sigma^B_{2i+1}}(v) = \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(v)$ when $v = y$ and when $v = x$. Therefore, for the action (b_i, y) we have:

$$\mathrm{Appeal}^{\sigma}(b_i, y) = \mathrm{Val}^{\sigma^B_{2i+1}}(y) + 1 < \mathrm{Val}^{\sigma^B_{2i+1}}(y) + 4n + 2i + 1 = \mathrm{Val}^{\sigma^B_{2i+1}}(b_i).$$

For the action (b_i, x), Proposition 5.6 gives:

$$\mathrm{Appeal}^{\sigma}(b_i, x) = \mathrm{Val}^{\sigma^B_{2i+1}}(x) < \mathrm{Val}^{\sigma^B_{2i+1}}(y) < \mathrm{Val}^{\sigma^B_{2i+1}}(b_i).$$

Finally, we consider the actions (b_i, f_j) with $j > i$. Again, it can easily be verified that $\mathrm{Val}^{\sigma^B_{2i+1}}(f_j) = \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_j)$ for the vertices f_j with $j > i$. Proposition 5.12 implies that $\mathrm{Appeal}^{\sigma^B_{2i+1}}(b_i, f_j) < \mathrm{Val}^{\sigma^B_{2i+1}}(b_i)$.

We now complete the proof by considering the action a_i. In the strategy σ^B_{R1}, we have $\sigma^B_{\mathrm{R1}}(b_i) = a_i$. Therefore, the fact that no action is switchable at b_i in σ^B_{R1} implies that a_i is the most appealing action at b_i. In the strategy σ^B_{2i+1}, we will show that the action a_i is switchable. We begin by showing that $\mathrm{Val}^{\sigma^B_{2i+1}}(g_i) > \mathrm{Val}^{\sigma^B_{2i+1}}(b_i)$. Let $k = \min(B^{>i} \cup \{n+1\})$ be the smallest index in B that is bigger than i, or the index of the sink if i is the highest bit. Using Proposition 5.11, the fact that $i \leq n$, and Proposition 5.2 gives:

$$\begin{aligned}
\mathrm{Val}^{\sigma^B_{2i+1}}(b_i) &\leq 6n + 1 + \mathrm{Val}^{\sigma^B_{2i+1}}(y) \\
&= 6n + 1 + \sum_{j \in B^{\leq i}} (10n+4)(2^j - 2^{j-1}) + \mathrm{Val}^{\sigma^B_{2i+1}}(c_k) \\
&\leq 6n + 1 + \sum_{j=1}^{i-1} (10n+4)(2^j - 2^{j-1}) + \mathrm{Val}^{\sigma^B_{2i+1}}(c_k) \\
&= 6n + 1 + (10n+4)(2^{i-1} - 2^0) + \mathrm{Val}^{\sigma^B_{2i+1}}(c_k) \\
&= (10n+4)2^{i-1} - 4n - 3 + \mathrm{Val}^{\sigma^B_{2i+1}}(c_k).
\end{aligned}$$

The valuation of the vertex g_i is:

$$\mathrm{Val}^{\sigma^B_{2i+1}}(g_i) = (10n+4)2^i + \mathrm{Val}^{\sigma^B_{2i+1}}(r_i) = (10n+4)2^i - 1 + \mathrm{Val}^{\sigma^B_{2i+1}}(c_k).$$

Since $(10n+4)2^i - 1 > (10n+4)2^{i-1} - 4n - 3$ for every i, we have that $\mathrm{Val}^{\sigma^B_{2i+1}}(g_i) > \mathrm{Val}^{\sigma^B_{2i+1}}(b_i)$. Now we can conclude:

$$\begin{aligned}
\mathrm{Appeal}^{\sigma^B_{2i+1}}(b_i, a_i) &= (1 - \frac{2^{-n}}{10n+4})\,\mathrm{Val}^{\sigma^B_{2i+1}}(b_i) + \frac{2^{-n}}{10n+4}\,\mathrm{Val}^{\sigma^B_{2i+1}}(g_i) \\
&> (1 - \frac{2^{-n}}{10n+4})\,\mathrm{Val}^{\sigma^B_{2i+1}}(b_i) + \frac{2^{-n}}{10n+4}\,\mathrm{Val}^{\sigma^B_{2i+1}}(b_i) \\
&= \mathrm{Val}^{\sigma^B_{2i+1}}(b_i).
\end{aligned}$$

□

Now that we know that the vertex b_i is switched to the action a_i, we must consider every other vertex in the MDP. We must argue that these vertices are switched from the strategy σ^B_{2i+1} to the strategy σ^B_{2i+2}. Proposition 5.16 provides a proof that greedy strategy improvement moves from the strategy σ^B_j to the strategy σ^B_{j+1} when $j \leq 2i+1$. No modifications are needed in order to reuse this proof to show that greedy strategy improvement moves from the strategy σ^B_{2i+1} to the strategy σ^B_{2i+2} at every vertex other than b_i. This is because the vertex b_i is the only strategy at which σ^B_{2i+2} is not well defined. Therefore, the proof of Proposition 5.21 has been completed.

5.4.2 The Second Reset Strategy

The purpose of this section is to provide a proof for the following proposition.

Proposition 5.23. *Greedy strategy improvement moves from the strategy σ^B_{R1} to the strategy σ^B_{R2}.*

For the vertex b_i, the fact that greedy strategy improvement does not switch away from the action a_i is implied by Proposition 5.22. The following proposition will prove useful throughout this proof.

Proposition 5.24. *For every vertex v such that $v \neq b_i$ and $v \neq f_i$, we have* $\mathrm{Val}^{\sigma^B_{R1}}(v) = \mathrm{Val}^{\sigma^B_{2i+2}}(v)$.

Proof. The only difference between σ^B_{R1} and σ^B_{2i+2} is that the vertex b_i selects the action a_i in the strategy σ^B_{R1}. Note that if we select a vertex v such that $v \neq b_i$ and $v \neq f_i$, and apply the strategy σ^B_{R1}, then the vertex b_i will never be reached from v. Therefore, we must have that $\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(v) = \mathrm{Val}^{\sigma^B_{2i+2}}(v)$. □

Consider a vertex v in the set $V \setminus (\{b_i, f_i, x, c_i\} \cup \{b_j \ : \ j < i\})$. Proposition 5.24 implies that we can apply Proposition 5.16 to argue that greedy strategy improvement moves to σ^B_{2i+3} at this vertex. This is because there is no outgoing edge from v to b_i or f_i, and therefore greedy strategy improvement will switch the same actions as it would in σ^B_{2i+2} at the vertex v.

All that remains is to prove that every vertex that has an outgoing action to the vertex f_i will switch to that action. The following proposition shows that the vertex f_i has a larger valuation than every other vertex f_j.

Proposition 5.25. *We have* $\mathrm{Val}^{\sigma^B_{R1}}(f_i) > \mathrm{Val}^{\sigma^B_{R1}}(f_j)$ *for every* $j \neq i$.

Proof. For every vertex f_j where $j \neq i$, Proposition 5.24 implies that $\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_j) = \mathrm{Val}^{\sigma^B_{2i+2}}(f_j)$. Therefore, we can apply Proposition 5.19 to argue that, if $k = \min(B)$, then $\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_k) > \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_j)$, for every j such that $j \neq i$ and $j \neq k$. Therefore, to prove this claim it is sufficient to show that $\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_i) > \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_k)$.

If $l = \min(B^{>i} \cup \{n+1\})$, then we have:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_i) = (10n+4)(2^i - 2^{i-1}) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l).$$

Moreover, we can express the valuation of f_k as:

$$\begin{aligned}\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_k) &= \sum_{j \in B^{<i}} (10n+4)(2^j - 2^{j-1}) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l) \\ &\leq \sum_{j=1}^{i-1} (10n+4)(2^j - 2^{j-1}) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l) \\ &= (10n+4)(2^{i-1} - 2^0) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l).\end{aligned}$$

Since $(10n+4)(2^i - 2^{i-1}) > (10n+4)(2^{i-1} - 2^0)$ for every $i > 0$ we can conclude that $\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_i) > \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_k)$. □

Firstly, we will consider the vertex x. Proposition 5.25 implies that x will be switched to the action (x, f_i). This is because every outgoing action from x has the form (x, f_j), and each of these actions has the same reward. Therefore, the fact that $\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_i) > \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(f_j)$ for all $j \neq i$ implies that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(x, f_i) > \mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(x, f_j)$ for all $j \neq i$.

Now we will prove the claim for the vertex c_i. Using the fact that $2^i - 2^{i-1} > 0$ for every i gives:

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(c_i, f_i) &= (10n+4)(2^i - 2^{i-1}) + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(r_i) \\ &> \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(r_i) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(c_i, r_i).\end{aligned}$$

Therefore (c_i, f_i) is the most appealing action at the vertex c_i.

Finally, we will consider the vertices b_j with $j < i$. We will begin by proving that every action at b_j other than (b_j, f_i) is not switchable. Proposition 5.24 implies that, for every action a at the state b_j other than (b_j, f_i), we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(a) = \mathrm{Appeal}^{\sigma^B_{2i+2}}(a)$. Since $j \in B$, Proposition 5.16 implies that greedy strategy improvement would not switch away from the action a_j at the vertex b_j in the strategy σ^B_{2i+2}. This implies that $\mathrm{Appeal}^{\sigma^B_{2i+2}}(a) \leq \mathrm{Val}^{\sigma^B_{2i+2}}(b_j)$, and therefore we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(a) \leq \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(b_j)$. To complete the proof, we must argue

that the action (b_j, f_i) is switchable at the vertex b_j.

In the proof of Proposition 5.25 we derived an expression for the valuation of f_i in terms of the vertex c_l, where $l = \min(B^{>i} \cup \{n+1\})$. We can use this to obtain:

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(b_j, f_i) &= (10n+4)(2^i - 2^{i-1}) + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l) \\ &= (10n+4)2^{i-1} + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l).\end{aligned}$$

We can also express the valuation of the vertex b_j as:

$$\begin{aligned}\mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(b_j) &= (10n+4)2^j - 1 + \sum_{k \in B^{>j} \cap B^{<j}} (10n+4)(2^k - 2^{k-1}) + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l) \\ &\leq (10n+4)2^j - 1 + \sum_{k=j+1}^{i-1} (10n+4)(2^k - 2^{k-1}) + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l) \\ &= (10n+4)2^j - 1 + (10n+4)(2^{i-1} - 2^j) + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l) \\ &= (10n+4)2^{i-1} - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(c_l).\end{aligned}$$

Since $(10n+4)2^{i-1} > (10n+4)2^{i-1} - 1$ we have that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R1}}}(b_j, f_i) > \mathrm{Val}^{\sigma^B_{\mathrm{R1}}}(b_j)$. Since (b_j, f_i) is the only switchable action at the vertex b_j, we have that this action must be switched by greedy strategy improvement at every vertex b_j with $j < i$.

5.4.3 The Third Reset Strategy

The purpose of this section is to provide a proof for the following proposition.

Proposition 5.26. *Greedy strategy improvement moves from the strategy σ^B_{R2} to the strategy σ^B_{R3}.*

We will begin by considering the vertices in the set $\{r_j \,:\, j \geq i\} \cup \{c_j \,:\, j \in B'\}$. For each vertex v in this set, we have that $\sigma^B_{\mathrm{R2}}(v) = \sigma^{B'}_0(v)$. We must show that greedy strategy improvement does not switch away from the strategy $\sigma^{B'}_0$. The

following proposition shows that the value of each of these vertices under σ^B_{R2} is the same as it is under $\sigma_0^{B'}(v)$.

Proposition 5.27. *We have* $\mathrm{Val}^{\sigma^B_{R2}}(v) = \mathrm{Val}^{\sigma_0^{B'}}(v)$ *for every* $v \in \{c_j, r_j \ : \ j \geq i\} \cup \{b_j, f_j \ : \ j \in B'\}$.

Proof. This follows from the fact that $\sigma^B_{\mathrm{R2}}(v) = \sigma_0^{B'}(v)$ on every vertex in $v \in S = \{c_j, r_j : j \geq i\} \cup \{b_j : j \in B'\}$, and we also have that $\sigma^B_{\mathrm{R2}}(v) \in S$ for every vertex $v \in S$. Therefore, we must have that $\Omega_v^{\sigma^B_{\mathrm{R2}}} = \Omega_v^{\sigma_0^{B'}}$ for every vertex $v \in S$, which implies the claimed result. □

We can now argue that a vertex r_j with $j \geq i$ will not be switched away from the strategy $\sigma_0^{B'}(r_j)$. Since every outgoing action from the vertex r_j is of the form (r_j, v), where $v \in \{c_j \ : \ j > i\}$, Proposition 5.27 implies that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(r_j, v) = \mathrm{Appeal}^{\sigma_0^{B'}}(r_j, v)$ for every outgoing action from r_j. The claim then follows from the fact that Proposition 5.16 implies that if greedy strategy improvement is applied to $\sigma_0^{B'}$, then it will not switch away from $\sigma_0^{B'}$.

The same argument can also be used to prove that a vertex c_j with $j \in B'$ will not be switched away from the strategy $\sigma_0^{B'}(c_j)$. Once again, this is because we have that every outgoing action from the vertex c_j is of the form (c_j, v) where $v \in \{c_j, r_j \ : \ j \geq i\} \cup \{b_j, f_j \ : \ j \in B'\}$. Therefore, we can apply Propositions 5.27 and 5.16 in the same way in order to prove the claim.

Next, we will move on to consider the vertex x, and the vertices c_j with $j \notin B'$ and $j \geq i$. In both cases, the proofs for these vertices depend on the valuation of the vertices f_j with $j \notin B'$. The following proposition proves a useful inequality for these vertices.

Proposition 5.28. *For every* $j \notin B'$ *we have* $\mathrm{Val}^{\sigma^B_{R2}}(f_j) \leq \mathrm{Val}^{\sigma^B_{R2}}(f_i)$. *Moreover, if* $j > i$ *then we have* $\mathrm{Val}^{\sigma^B_{R2}}(f_j) \leq \mathrm{Val}^{\sigma^B_{R2}}(f_i) - (10n + 4)2^{j-1} - 1$.

Proof. We first consider the vertices f_j where $j < i$. At these vertices we have:

$$\begin{aligned}\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j) &= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) - (10n+4)2^{j-1} - 1 \\ &= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) + 4n + 1 - (10n+4)2^{j-1}\end{aligned}$$

Therefore, the claim follows from the fact that $4n + 1 - (10n + 4)2^{j-1} > 0$ when $j > 0$.

We now consider the vertices f_j where $j > i$ and $j \notin B'$. Since $j \notin B'$ we must have $\sigma^B_{\mathrm{R2}}(b_j) = d_k$ for some k. Therefore, we have:

$$\begin{aligned}\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j) &= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) - (10n+4)2^{j-1} - 4n \\ &\leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) - (10n+4)2^{j-1} + 2n + 1\end{aligned}$$

In the case where $i > 1$, we have $\sigma^B_{\mathrm{R2}}(y) = c_1$, and we have $\sigma^B_{\mathrm{R2}}(c_1) = f_1$. We can use the formula that we obtained for the vertices f_j with $j < i$ to obtain:

$$\begin{aligned}\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j) &\leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) - (10n+4)2^{j-1} + 2n + 1 \\ &= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_1) - (10n+4)2^{j-1} + 6n + 2 \\ &= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) - (10n+4)2^{0} - (10n+4)2^{j-1} + 10n + 3 \\ &= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) - (10n+4)2^{j-1} - 1\end{aligned}$$

In the case where $i = 1$, we have that $\sigma^B_{\mathrm{R2}}(y) = c_l$, where $l > 1$. Moreover, we have that $\sigma^B_{\mathrm{R2}}(b_1) = a_1$ and $\sigma^B_{\mathrm{R2}}(r_1) = c_l$. Therefore, we have the following two facts:

$$\begin{aligned}\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) &= (10n+4)(2^1 - 2^0) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l) \\ \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j) &\leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l) - (10n+4)2^{j-1} + 2n + 1\end{aligned}$$

From this it is easy to see that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j) \leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) - (10n+4)2^{j-1} - 1$. □

We can now prove that greedy strategy improvement does not switch the vertex x away from the action (x, f_i) in the strategy σ^B_{R2}. Since Proposition 5.27 implies that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j) = \mathrm{Val}^{\sigma_0^{B'}}(f_j)$ for every $j \in B'$, we can use Proposition 5.19 to conclude that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) > \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j)$ with $j \in B'$ and $j \neq i$. Moreover, Proposition 5.28 implies that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) > \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j)$ for every $j \notin B'$. Therefore, the action (x, f_i) must be the most appealing action at the vertex x in the strategy σ^B_{R2}.

We can also prove that greedy strategy improvement does not switch the vertices c_j with $j \notin B'$ and $j > i$ away from the action (c_j, r_j). To do this, we will prove that the action (c_j, f_j) is not switchable in σ^B_{R2}. Note that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_j, f_j) = 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_j)$, and therefore Proposition 5.28 implies that:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_j, f_j) \leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) - (10n + 4)2^{j-1} + 4n.$$

Following the strategy σ^B_{R2} through the vertices f_i, b_i, g_i, and r_i gives:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_j, f_j) \leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_{\min(B^{>i} \cup \{n+1\})}) + (10n + 4)(2^i - 2^{j-1} - 2^{i-1}) - 1$$

Now we can apply Proposition 5.27 and Proposition 5.2 to conclude the following two facts:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_j, f_j) \leq \sum_{m \in B'^{>i}} (10n + 4)(2^m - 2^{m-1}) + (10n + 4)(2^i - 2^{j-1} - 2^{i-1}) - 1$$

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) = \sum_{m \in B'^{\geq j}} (10n + 4)(2^m - 2^{m-1})$$

Therefore, we have:

$$\begin{aligned}
\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_j, f_j) &\leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) + (10n+4)\Big(\sum_{m \in B' < j \cap B' > i} (2^m - 2^{m-1}) \\
&\quad + 2^i - 2^{j-1} - 2^{i-1}\Big) - 1 \\
&\leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) + (10n+4)\Big(\sum_{m=i+1}^{j-1} (2^m - 2^{m-1}) \\
&\quad + 2^i - 2^{j-1} - 2^{i-1}\Big) - 1 \\
&= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) + (10n+4)(2^{j-1} - 2^i + 2^i - 2^{j-1} - 2^{i-1}) - 1 \\
&= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) - (10n+4)2^{i-1} - 1
\end{aligned}$$

Therefore, the action (c_j, f_j) is not switchable in the strategy σ^B_{R2}, which implies that greedy strategy improvement cannot switch away from the action (c_j, r_j).

We now turn our attention to the vertices d_k. We must prove that greedy strategy improvement switches the action (d_k, x) at these vertices. The next proposition proves that the valuation of x must be significantly larger than the valuation of y in the strategy σ^B_{R2}, which is a fact that will be used to prove the behaviour of greedy strategy improvement at each vertex d_k.

Proposition 5.29. *We have* $\mathrm{Val}^{\sigma^B_{R2}}(y) + 6n + 1 < \mathrm{Val}^{\sigma^B_{R2}}(x)$.

Proof. Let $l = \min(B \cup \{n+1\})$. We first consider the case where $l < i$. It is not difficult to see that if $l < i$ then $l = 1$, since i is the smallest index that is not contained in B. In this case we can express the valuation of y in terms of the valuation of the vertex f_i as:

$$\begin{aligned}
\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) &= -(10n+4)2^{l-1} + 4n + 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) \\
&= -(10n+4) + 4n + 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i) \\
&= -6n - 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i).
\end{aligned}$$

Moreover, we can express the valuation of x as:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) = \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i).$$

Therefore, we have $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) + 6n + 1 < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x)$.

The second case that we must consider is when $l > i$, which occurs only when $i = 1$. In this case we can express the valuation of y in terms of the valuation of c_l as:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) = \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l).$$

Similarly, we can express the valuation of x in terms of the valuation of c_l. Our derivation uses the fact that $i = 1$.

$$\begin{aligned}
\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) &= -(10n+4)2^{i-1} - 4n + (10n+4)2^i - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l) \\
&= (10n+4) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l) \\
&= 6n + 3 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l).
\end{aligned}$$

Once again it is clear that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) + 6n + 1 < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x)$. □

We can now prove that the action (d_k, x) is switched at the vertex d_k. Proposition 5.29, and the fact that $\mathrm{r}(d_k, x) = \mathrm{r}(d_k, y)$ for every k, imply:

$$\begin{aligned}
\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(d_k, y) &= \mathrm{r}(d_k, y) + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(d_k, y) \\
&< \mathrm{r}(d_k, y) + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(d_k, x) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(d_k, x).
\end{aligned}$$

Every vertex d_k with $k \geq 1$ has an additional action (d_k, d_{k-1}), for which we consider two cases. Since $\sigma^B_{\mathrm{R2}}(d_k) = \sigma^B_{2i+3}(d_k)$ for every k, we can use Proposition 5.7. When $1 \leq k \leq 2i + 4$ this proposition, in combintation with Proposition 5.29 gives:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(d_k, d_{k-1}) = 4n - k + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(d_k, x).$$

In the case where $k > 2i + 4$, Proposition 5.7 and Proposition 5.29 imply:

$$\text{Appeal}^{\sigma^B_{\text{R2}}}(d_k, d_{k-1}) = \text{Val}^{\sigma^B_{\text{R2}}}(y) - 1 < \text{Val}^{\sigma^B_{\text{R2}}}(x) = \text{Appeal}^{\sigma^B_{\text{R2}}}(d_k, d_{k-1}).$$

We have therefore shown that the action (d_k, x) is the most appealing action at the vertex d_k.

We now turn our attention to the vertex y. For this vertex, we must prove that greedy strategy improvement switches the action (y, c_i), and to do this it is sufficient to prove that $\text{Val}^{\sigma^B_{\text{R2}}}(c_i) > \text{Val}^{\sigma^B_{\text{R2}}}(c_j)$ for every $j \neq i$. For the vertices c_j with $j \geq i$, we can use Proposition 5.27 and Proposition 5.2 to prove that $\text{Val}^{\sigma^B_{\text{R2}}}(c_i) > \text{Val}^{\sigma^B_{\text{R2}}}(c_j)$ where $j > i$.

We will now deal with the vertices c_j with $j < i$. At these vertices we have $\sigma^B_{\text{R2}}(c_j) = f_j$, $\sigma^B_{\text{R2}}(f_j) = b_j$, and $\sigma^B_{\text{R2}}(b_j) = f_i$. Therefore, we have:

$$\text{Val}^{\sigma^B_{\text{R2}}}(c_j) = -(10n + 4)2^{j-1} + 4n + 2 + \text{Val}^{\sigma^B_{\text{R2}}}(f_i)$$

On the other hand, since $\sigma^B_{\text{R2}}(c_i) = f_i$, we have:

$$\text{Val}^{\sigma^B_{\text{R2}}}(c_i) = 4n + 1 + \text{Val}^{\sigma^B_{\text{R2}}}(f_i)$$

Therefore, it is obvious that $\text{Val}^{\sigma^B_{\text{R2}}}(c_i) > \text{Val}^{\sigma^B_{\text{R2}}}(c_j)$ when $j < i$, and we have completed the proof that greedy strategy improvement switches the action (y, c_i) at the vertex y.

The arguments that we have just made can also be used for the vertices r_j with $j < i$. At these vertices, we must show that greedy strategy improvement switches the action (r_j, c_i). Every outgoing action from the vertex r_j has the same reward, and leads to a vertex c_k. We have already shown that $\text{Val}^{\sigma^B_{\text{R2}}}(c_i) > \text{Val}^{\sigma^B_{\text{R2}}}(c_j)$ for every $j \neq i$, which implies that the action (r_j, c_i) must be the most appealing action at r_j. Therefore, greedy strategy improvement will switch the action (r_j, c_i)

at the vertex r_j in the strategy σ^B_{R2}

We will now consider the vertices c_j with $j < i-1$. At these vertices we must show that greedy strategy improvement does not switch away from the action (c_j, f_j). In order to do this, we will show that the action (c_j, r_j) is not switchable at the vertex c_j. Note that we have $\sigma^B_{\mathrm{R2}}(r_j) = c_{j+1}$, $\sigma^B_{\mathrm{R2}}(c_{j+1}) = f_{j+1}$, $\sigma^B_{\mathrm{R2}}(f_{j+1}) = b_{j+1}$ and $\sigma^B_{\mathrm{R2}}(b_{j+1}) = f_i$. Therefore, we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_j, r_j) = -(10n+4)2^j + 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i).$$

Whereas, the fact that $\sigma^B_{\mathrm{R2}}(c_j) = f_j$, that $\sigma^B_{\mathrm{R2}}(f_j) = b_j$, and that $\sigma^B_{\mathrm{R2}}(b_j) = f_i$ imply:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) = -(10n+4)2^{j-1} + 4n + 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i).$$

Since $2^j > 2^{j-1}$, we have that the action (c_j, r_j) is not switchable at the vertices c_j with $j < i-1$ in the strategy σ^B_{R2}. This implies that greedy strategy improvement cannot switch away from the action (c_j, f_j) at these vertices.

We now consider the vertex c_{i-1}, for which we must prove that greedy strategy improvement does not switch away from the action (c_{i-1}, f_{i-1}). The arguments used in the previous paragraph imply that:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_{i-1}) = -(10n+4)2^{i-2} + 4n + 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i).$$

Moreover, we have that $\sigma^B_{\mathrm{R2}}(f_i) = b_i$, $\sigma^B_{\mathrm{R2}}(b_i) = a_i$, and $\sigma^B_{\mathrm{R2}}(r_i) = c_l$, where $l = \min(B^{>i} \cup \{n+1\})$. This implies that:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_{i-1}) = (10n+4)(2^i - 2^{i-1} - 2^{i-2}) + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l) > \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l).$$

On the other hand, we have that $\sigma^B_{\mathrm{R2}}(r_{i-1}) = c_l$, which implies that:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(c_{i-1}, r_{i-1}) = \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l) - 1 < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l).$$

Therefore, the action (c_{i-1}, r_{i-1}) is not switchable in the strategy σ^B_{R2}.

Finally, to complete the proof of Proposition 5.26, we must consider the vertices b_j. We start by considering the vertices b_j with $j \notin B'$. For these vertices, we must show that the action (b_j, x) is the most appealing action. We will begin by showing that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, x) > \mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, d_k)$ for every k. Since $\sigma^B_{\mathrm{R2}}(d_k) = \sigma^B_{2i+3}(d_k)$ for every k, we can apply Proposition 5.11 to argue that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, d_k) \leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) + 6n + 1$. Then, we can apply Proposition 5.29 to conclude that:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, d_k) \leq \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) + 6n + 1 < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, x).$$

We can also apply Proposition 5.29 to argue that the action (b_j, x) is more appealing than the action (b_j, y):

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, y) = 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, x).$$

We now consider the actions (b_j, f_k) where $k \notin B'$ and $k < i$. Since $k > j$, we must also have that $j < i$. Therefore, it must be the case that $\sigma^B_{\mathrm{R2}}(b_j) = f_i$ and that $\sigma^B_{\mathrm{R2}}(b_k) = f_i$. Therefore, we have the following two facts:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, f_k) = 4n + 2 - (10n + 4)2^{k-1} + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i)$$
$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) = 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i)$$

Therefore, the action (b_j, f_k) is not switchable in the strategy σ^B_{R2}.

Now we condiser the actions (b_j, f_k) where $k \notin B'$ and $k > i$. In this case we have $\sigma^B_{\mathrm{R2}}(b_j) = d_l$ for some l. The following derrivation uses this fact, along with

Proposition 5.11 and Proposition 5.29, to obtain:

$$\begin{aligned}\text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, f_k) &= 1 - (10n+4)2^{k-1} + \text{Val}^{\sigma^B_{\text{R2}}}(b_j) \\ &\leq 1 - (10n+4)2^{k-1} + 6n + 1 + \text{Val}^{\sigma^B_{\text{R2}}}(y) \\ &< 1 + \text{Val}^{\sigma^B_{\text{R2}}}(y) = \text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, y) < \text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, x)\end{aligned}$$

Therefore, the action (b_j, x) is more appealing than the action (b_j, f_k).

We move on to consider the actions (b_j, f_k) where $k \in B'$. Since Proposition 5.27 implies that $\text{Val}^{\sigma^B_{\text{R2}}}(f_k) = \text{Val}^{\sigma_0^{B'}}(f_k)$ for all $k \in B'$, we can apply Proposition 5.19 to argue that $\text{Val}^{\sigma^B_{\text{R2}}}(f_i) > \text{Val}^{\sigma^B_{\text{R2}}}(f_k)$ with $k \neq i$. If $j < i$, then we have that $\sigma^B_{\text{R2}}(b_j) = f_i$, which immidiately implies that none of the actions (b_j, f_k) where $k \in B'$ can be switchable at b_j in σ^B_{R2}. For the vertices b_j with $j > i$, we will argue that $\text{Val}^{\sigma^B_{\text{R2}}}(x) > \text{Val}^{\sigma^B_{\text{R2}}}(f_l)$ where $l = \min(B'^{>i})$. We can assume that l is well defined since otherwise there would be no action (b_j, f_k) with $k \in B'$. By following the strategy σ^B_{R2} from x we obtain:

$$\text{Val}^{\sigma^B_{\text{R2}}}(x) = -(10n+4)(2^i - 2^{i-1}) - \text{Val}^{\sigma^B_{\text{R2}}}(f_l) > \text{Val}^{\sigma^B_{\text{R2}}}(f_l) + 4n + 1$$

Since Proposition 5.19 implies that $\text{Val}^{\sigma^B_{\text{R2}}}(f_l) > \text{Val}^{\sigma^B_{\text{R2}}}(f_m)$ with $m > l$ and $m \in B'$, we must have that $\text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, f_k) < \text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, x)$.

Finally, we condiser the action a_j. Proposition 5.27 and Proposition 5.2 imply that $\text{Val}(g_j) < (10n+4)2^n$. From this we can derrive, in a similar manner to Proposition 5.10, that $\text{Appeal}^{\sigma^B_{\text{R2}}}(a_j) < \text{Val}^{\sigma^B_{\text{R2}}}(b_j) + 1$. In the case where $j > i$, we have that $\sigma^B_{\text{R2}}(b_j) = d_k$ for some k. Therefore, we can apply Proposition 5.11 to conclude:

$$\text{Val}^{\sigma^B_{\text{R2}}}(b_j) \leq 6n + 1 + \text{Val}^{\sigma^B_{\text{R2}}}(y)$$

Proposition 5.29 implies that $\text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, x) > 6n + 1 + \text{Val}^{\sigma^B_{\text{R2}}}(y)$, and since every valuation in σ^B_{R2} is an integer, we have that $\text{Appeal}^{\sigma^B_{\text{R2}}}(b_j, x) \geq \text{Val}^{\sigma^B_{\text{R2}}}(b_j) + 1$. This

implies that the action (b_j, x) is more appealing than the action a_j at the state b_j in the strategy σ^B_{R2}.

We now consider the case where $j < i-1$. In this case we have the following two facts:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) = 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i)$$
$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(g_j) = (10n+4)(2^j - 2^j) - 1 + 4n + 1 - 4n + 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_i)$$

This implies that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(g_j) = \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j)$, and therefore we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(a_j) = \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j)$. This implies that the action a_i is not switchable in the strategy σ^B_{R2} at the vertex b_j.

Finally, we consider the case where $j = i-1$. If $l = \min(B^{>i} \cup \{n+1\})$ then we have the following two facts:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(g_{i-1}) = (10n+4)2^{i-1} - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l)$$
$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1}) = 4n + 1 + (10n+4)(2^i - 2^{i-1}) - 4n - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_l)$$

This implies that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(g_{i-1}) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1})$, which gives:

$$\begin{aligned}
\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_{i-1}, a_{i-1}) &= \mathrm{r}(b_{i-1}, a_{i-1}) + \sum_{s \in S} p(s|b_{i-1}, a_{i-1}) \, \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1}(s) \\
&= 0 + (1 - \frac{2^{-n}}{10n+4}) \, \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1}) + \frac{2^{-(n)}}{10n+4} \, \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(g_{i-1}) \\
&< 0 + (1 - \frac{2^{-n}}{10n+4}) \, \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1}) + \frac{2^{-(n)}}{10n+4} \, \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1}) \\
&= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_{i-1})
\end{aligned}$$

Therefore, the action a_{i-1} cannot be switchable at the vertex b_{i-1} in the strategy σ^B_{R2}.

To complete the proof of Proposition 5.26, we consider the vertices b_j where $j \in B'$. We will begin by showing that the actions (b_j, x), (b_j, y), (b_j, f_k) with $k \notin B'$,

and (b_j, d_k) for all k, are not switchable in the strategy σ^B_{R2}. For the action (b_j, x) we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, x) = \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x)$, and for the action (b_j, y) Proposition 5.29 gives:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, y) = 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x)$$

For the actions (b_j, d_k), Proposition 5.11 and Proposition 5.29 imply:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, d_k) \leq 6n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(y) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x)$$

For the actions (b_j, f_k) with $k \notin B'$, we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, f_k) = 4n + 1 - (10n + 4)2^{k-1} - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j)$$

Since $k \notin B'$ we must have $\sigma^B_{\mathrm{R2}}(b_k) = d_l$ for some l, therefore we can use the same arguments as we did for the action (b_j, d_k) to conclude:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R2}}}(b_j, f_k) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x)$$

Therefore, to show that the actions (b_j, x), (b_j, y), (b_j, f_k) with $k \notin B'$, and (b_j, d_k) for all k are not switchable, we must show that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j)$. Since $\sigma^B_{\mathrm{R2}}(x) = f_i$ we have:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(x) = -(10n + 4)2^{i-1} - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_i) < \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_i)$$

If $j = i$ then the proof is complete. On the other hand, if $j \neq i$ then the fact that $j \in B'$ implies that $j > i$. If we follow the strategy σ^B_{R2} from the vertex b_i then

Proposition 5.27 and Proposition 5.4 give:

$$\begin{aligned}
\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_i) &= (10n+4)2^i - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_{\min(B'>i)}) \\
&\leq (10n+4)2^i - 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(c_j) + (10n+4)(2^{j-1} - 2^i) \\
&= (10n+4)2^i + \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) + (10n+4)(2^{j-1} - 2^i) - (10n+4)2^{j-1} \\
&= \mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j)
\end{aligned}$$

Therefore, none of the actions that we are considering are switchable at the vertex b_j in the strategy σ^B_{R2}.

We now consider the actions (b_j, f_k) with $k \in B'$. Proposition 5.27 implies that $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(b_j) = \mathrm{Val}^{\sigma^{B'}_0}(b_j)$ and $\mathrm{Val}^{\sigma^B_{\mathrm{R2}}}(f_k) = \mathrm{Val}^{\sigma^{B'}_0}(f_k)$. This implies that the action (b_j, f_k) is switchable in the strategy $\sigma^{B'}_0$ if and only if it is switchable in the strategy σ^B_{R2}. Therefore, Proposition 5.16 implies that the action (b_j, f_k) is not switchable at the vertex b_j in the strategy σ^B_{R2}. Moreover, since we have now shown that there are no switchable actions at the vertex b_j, we have shown that greedy strategy improvement will not switch away from the action a_j at this vertex.

5.4.4 The Final Reset Strategy

The purpose of this section is to provide a proof for the following proposition.

Proposition 5.30. *Greedy strategy improvement moves from the strategy σ^B_{R3} to the strategy $\sigma^{B'}_0$.*

We begin by stating an analogue of Proposition 5.27. However, since the strategy σ^B_{R3} is closer to the strategy $\sigma^{B'}_0$, more vertices can be included in this proposition.

Proposition 5.31. *We have $\mathrm{Val}^{\sigma^B_{R3}}(v) = \mathrm{Val}^{\sigma^{B'}_0}(v)$ for every $v \in \{c_j, \ : \ j \geq i\} \cup \{r_j \ : \ 1 \leq j \leq n\} \cup \{b_j, f_j \ : \ j \in B'\} \cup \{x, y\}$.*

Proof. The proof of this proposition uses exactly the same reasoning as the proof of Proposition 5.27. This is because σ^B_{R3} is closed on the set $\{c_j, \,:\, j \geq i\} \cup \{r_j \,:\, 1 \leq j \leq n\} \cup \{b_j, f_j \,:\, j \in B'\} \cup \{x, y\}$. □

The arguments that we used to show that a vertex r_j with $j \geq i$ will not be switched away from the strategy σ^B_{R2} can also be applied to the strategy σ^B_{R3}. Since every outgoing action from the vertex r_j is of the form (r_j, v), where $v \in \{c_j \,:\, j > i\}$, Proposition 5.31 implies that $\mathrm{Appeal}^{\sigma_0^{B'}}(r_j, v) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(r_j, v)$ for every outgoing action from r_j. The claim then follows from the fact that Proposition 5.16 implies that if greedy strategy improvement is applied to $\sigma_0^{B'}$, then it will not switch away from $\sigma_0^{B'}$.

The same argument can also be used to prove that a vertex c_j with $j \in B'$ will not be switched away from the strategy $\sigma_0^{B'}(c_j)$. Once again, this is because we have that every outgoing action from the vertex c_j is of the form (c_j, v) where $v \in \{c_j, r_j \,:\, j \geq i\} \cup \{b_j, f_j \,:\, j \in B'\}$. Therefore, we can apply Propositions 5.31 and 5.16 in the same way in order to prove the claim.

We will now consider the vertices r_j with $j < i$, where we must show that (r_j, c_i) is the most appealing action. We will begin by showing that the actions (r_j, c_k) with $k < i$ are not switchable. Since we have $\sigma^B_{\mathrm{R3}}(c_k) = f_k$, $\sigma^B_{\mathrm{R3}}(b_k) = x$, and $\sigma^B_{\mathrm{R3}}(x) = f_i$ we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, c_k) = -(10n + 4)2^{k-1} + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i).$$

On the other hand, since $\sigma^B_{\mathrm{R3}}(r_j) = c_i$ and $\sigma^B_{\mathrm{R3}}(c_i) = f_i$ we have:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(r_j) = 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i).$$

Therefore, the actions (r_j, c_k) with $k < i$ are not switchable in σ^B_{R3}. Proposition 5.31 and Proposition 5.2 then imply that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(r_j, c_i) > \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(r_j, c_k)$ with for

every $k > i$. Therefore, greedy strategy improvement will not switch away from the action (r_j, c_i) in the strategy σ^B_{R3}.

We will now consider the vertices c_j with $j \notin B'$ and $j < i$. At these vertices we must argue that greedy strategy improvement switches the action (c_j, r_i). Since $\sigma^B_{\mathrm{R3}}(c_j) = f_j$, $\sigma^B_{\mathrm{R3}}(b_j) = x$, and $\sigma^B_{\mathrm{R3}}(x) = f_i$ we have:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_j) = 1 - (10n+4)2^{j-1} + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i)$$

On the other hand, since $\sigma^B_{\mathrm{R3}}(r_j) = c_i$ and $\sigma^B_{\mathrm{R3}}(c_i) = f_i$ we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(c_j, r_j) = -1 + 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i)$$

Therefore, we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(c_j, r_j) > \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_j)$, which implies that greedy strategy improvement will switch the action (c_j, r_j) at the vertex c_j in the strategy σ^B_{R3}.

Now we will consider the vertices c_j with $j \notin B'$ and $j > i$. At these vertices we must argue that greedy strategy improvement does not switch away from the action (c_j, r_j). We will do this by arguing that the action (c_j, f_j) is not switchable. As we have argued previously, we have:

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(c_j, f_j) &= 1 - (10n+4)2^{j-1} + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i) \\ &= (10n+4)(2^i - 2^{i-1} - 2^{j-1}) + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_{\min(B'_{>i}) \cup \{n+1\}})\end{aligned}$$

We can then apply Proposition 5.31 and Proposition 5.4 to show:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(c_j, f_j) \leq (10n+4)(2^i - 2^{i-1} - 2^{j-1} - 2^i + 2^{j-1}) + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_j) < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_j)$$

Therefore, the action (c_j, r_j) is not switchable in the strategy σ^B_{R3}.

We will now consider the vertex x, where we must show that the most appealing action is (x, f_i). We will begin by showing that the actions (x, f_j) where

$j \notin B'$ are not switchable in σ^B_{R3}. Since $\sigma^B_{\mathrm{R3}}(b_j) = x$, we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(x, f_j) = -(10n+4)2^{j-1} - 4n + \mathrm{Val}^{\sigma^b_{\mathrm{R3}}}(x) < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x)$$

Therefore, the actions (x, f_j) where $j \notin B'$ are not switchable in σ^B_{R3}. For the actions (x, f_j) where $j \in B'$ and $j \neq i$, Proposition 5.31 and Proposition 5.19 imply that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(x, f_i) > \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(x, f_j)$, which implies that greedy strategy improvement will not switch away from the action (x, f_i).

We can apply the same reasoning for the vertex y, where we must show that (y, c_i) is the most appealing action. For the actions (y, c_j) with $j \notin B'$ we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(y, c_j) = -(10n+4)2^{j-1} + 1 + \mathrm{Val}^{\sigma^b_{\mathrm{R3}}}(x) < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x) + 4n + 1.$$

Since $\sigma^b_{\mathrm{R3}}(y) = c_i$ and $\sigma^b_{\mathrm{R3}}(x) = f_i$ we must have $\mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(y) = \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x) + 4n + 1$. Therefore we have shown that the actions (y, c_j) with $j \notin B'$ are not switchable in the strategy σ^B_{R3}. Proposition 5.31 and Proposition 5.2 imply that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(c_i) > \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(c_j)$ for every $j \in B$ such that $j \neq i$. Therefore greedy strategy improvement will not switch away from the action (y, c_i) in the strategy σ^B_{R3}.

We now consider the vertices d_k for all k. For every vertex d_k, we must show that (d_k, y) is the most appealing action. For the action (d_k, d_{k-1}) we have:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(d_k, d_{k-1}) = \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x) - 1 < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(d_k, x).$$

For the action (d_k, x), the fact that $\mathrm{r}(d_k, y) = \mathrm{r}(d_k, x)$ implies:

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(d_k, x) &= \mathrm{r}(d_k, x) + \mathrm{Val}(f_i) \\ &< \mathrm{r}(d_k, y) + 4n + 1 + \mathrm{Val}(f_i) = \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(d_k, y).\end{aligned}$$

Therefore, the action (d_k, y) is the most appealing action at the vertex d_k.

Finally, we consider the vertices b_j. We begin with the case when $j \notin B'$. In this case we must show that (b_j, y) is the most appealing action at the vertex b_j. We will first argue that the action (b_j, x) is not switchable. Since $\sigma^B_{\mathrm{R3}}(x) = f_i$, $\sigma^B_{\mathrm{R3}}(y) = c_i$ and $\sigma^B_{\mathrm{R3}}(c_i) = f_i$, we have the following two equalities:

$$\mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x) = \mathrm{Val}(f_i)$$
$$\mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(y) = \mathrm{Val}(f_i) + 4n + 1$$

Therefore, we must have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, x) < \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y)$. These two equalities can also be used to prove that the actions of the form (b_j, d_k) are not switchable. This is because we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, d_k) \leq 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x)$ and we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) = 4n + 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x)$.

We now consider the actions of the form (b_j, f_k). We will first prove that the actions (b_j, f_k) where $k \notin B'$ are not switched by greedy strategy improvement. Since $k \notin B'$, we have:

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, f_k) &= -(10n+4)2^{k-1} + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(x) \\ &< \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(y) + 4n + 2 = \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y)\end{aligned}$$

Therefore, these actions will not be switched by greedy strategy improvement in the strategy σ^B_{R3}. We now consider the actions (b_j, f_k) where $k \in B'$. We will prove that these actions are not switchable in σ^B_{R3}. The appeal of the action (b_j, f_k) is:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, f_k) = 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_k).$$

On the other hand, the appeal of the action (b_j, y) can be expressed as:

$$\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) = (10n+4)(2^i - 2^{i-1}) - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_{\min(B' > i)})$$

We can then use Proposition 5.31 and Proposition 5.4 to conclude:

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) &= (10n+4)(2^i - 2^{i-1} - 2^i + 2^{k-1}) - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_k) \\ &= (10n+4)(2^{k-1} - 2^{i-1}) - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_k) \\ &= (10n+4)(2^{k-1} - 2^{i-1}) + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_k) > 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_k)\end{aligned}$$

Therefore, the action (b_j, f_k) will not be switched by greedy strategy improvement.

Finally, we consider the action a_j. We will first argue that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) > \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(b_j) + 1$. This holds because we have $\mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(b_j) = \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i)$, and we have $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) = 4n + 2 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(f_i)$. On the other hand Proposition 5.31 and Proposition 5.10 imply that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, a_j) < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(b_j) + 1$. Therefore, greedy strategy improvement will not switch the action a_j.

To complete the proof of Proposition 5.30, we will show that greedy strategy improvement does not switch away from the action a_j at every vertex b_j with $j \in B'$. We can use exactly the same arguments as we used for the vertices b_j with $j \notin B'$ to argue that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) > \mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, a)$ for every action $a \in \{(b_j, d_k) \ : \ 1 \leq k \leq n\} \cup \{(b_j, f_k \ : \ j < k \leq n\} \cup \{(b_j, x)\}$. Therefore, we can prove the claim by showing that $\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(b_j)$. As we have done previously, we can use Proposition 5.31 and Proposition 5.4 to obtain the following characterisation for the appeal of the action (b_j, y):

$$\begin{aligned}\mathrm{Appeal}^{\sigma^B_{\mathrm{R3}}}(b_j, y) &= (10n+4)(2^i - 2^{i-1} - 2^i + 2^{j-1}) - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_j) \\ &= (10n+4)(2^{j-1} - 2^{i-1}) - 4n + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(c_k) \\ &= -(10n+4)(2^{i-1}) - 4n + 1 + \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(b_j) < \mathrm{Val}^{\sigma^B_{\mathrm{R3}}}(b_j)\end{aligned}$$

Therefore, greedy strategy improvement will not switch away from the action a_j at the vertex b_j in the strategy σ^B_{R3}.

5.5 Exponential Lower Bounds For The Average Reward Criterion

We can now state the main theorem of this Chapter. The proof of this theorem follows from Proposition 5.16 in the flip phase, and Propositions 5.21, 5.23, 5.26, and 5.30 in the reset phase.

Theorem 5.32. *When greedy strategy improvement for the average-reward criterion is applied to the strategy $\sigma_0^{\{1\}}$ it will take at least $2^n - 1$ iterations to find the optimal strategy.*

5.6 Concluding Remarks

In this chapter we have shown how Friedmann's lower bounds for the greedy switching policy in the strategy improvement setting can be extended to apply to the Markov decision process setting. We have shown lower bounds for the the average-reward criterion, but lower bounds for the discounted-reward criterion have been left open. We suspect that a smart choice of discount factor will allow the same construction to be used to prove an exponential lower bound in this setting. This is because the valuation of a vertex under the discounted-reward criterion approaches the valuation of that vertex under the total-reward criterion as the discount factor approaches 1. Therefore, if the discount factor is chosen to be sufficiently close to 1, then the appeal of each action under the discounted-reward criterion will be close to appeal of that action under the total-reward criterion. If it can be shown that the ordering of successor actions at each vertex does not change, then greedy strategy improvement will make the same decisions at those vertices, and an exponential lower bound would follow.

Another open question is whether the sub-exponential lower bounds of Friedmann, Hansen, and Zwick for the random facet switching policy [FHZ10] can be

generalised to the Markov decision process setting. Their construction also uses the machinery of a binary counter, but they have a more complex gadget to represent each bit. If this lower bound can be extended to the Markov decision process setting, then it would seem likely that the lower bound could also be extended to the random facet pivot rule for linear programming. This is because strategy improvement algorithms that only switch one action in each iteration are strongly related to the behaviour of the simplex method as it is applied to the dual of the reduction from Markov decision processes to linear programming.

Chapter 6

Non-oblivious Strategy Improvement

The switching policies for strategy improvement that have been considered so far have been general switching policies, which are not concerned with the type of game or MDP that they are solving. Either these switching policies follow a simple rule, such as "switch some switchable edge", or they use a rule that depends on the appeal of an edge, such as "switch the edge with maximal appeal". This generality allows the switching policies to be applied to a wide variety of models, such as both MDPs and games, but it prevents the switching policy from using more complex rules that depend on the model to which it is being applied. It is for this reason that we refer to these switching polices as *oblivious* switching policies. In this chapter, we develop *non-oblivious* switching policies for the Björklund-Vorobyov strategy improvement algorithm.

In the first part of this chapter we study switchable edges, and we show that these edges can be classified into two types: cross edges and back edges. We show that the effect of switching a switchable cross edge can be very different to the effect of switching a switchable back edge. The traditional switching policies that we have described do not consider this distinction, and they are therefore unable to exploit

it in their decision making.

This however, is only the first step. We go on to show how each switchable back edge corresponds to a structure in a mean-payoff game that we call a snare. We show that if a switching policy is aware of a snare in the game, then it can make better decisions. Therefore, we propose a family of switching policies that remember the snares that have been seen in the game, and then use this knowledge to make better decisions in future iterations. We also show how a traditional switching policy can be modified to make use of these techniques.

Finally, we examine the behaviour of our techniques on the families of examples that have been used to show super-polynomial lower bounds. In particular, we consider the examples of Friedmann, which show a lower bound for the greedy and optimal switching policies [Fri09], and we consider the examples of Friedmann, Hansen, and Zwick, which show a sub-exponential lower bound for the random facet switching policy [FHZ10]. We show that modifying each of these switching policies to make use of our techniques allows these switching policies to avoid super-polynomial behaviour on their respective examples.

6.1 Classifying Profitable Edges

In this section we study switchable edges. We are interested in how the choice of edge to switch affects the behaviour of a strategy improvement algorithm. We show that, in the Björklund-Vorobyov strategy improvement algorithm, there are two different types of switchable edge, and that these types of edge behave differently when strategy improvement switches them. In the first part of this section we define our classification of switchable edges, and in the second part we show the effect of switching both of these types of edges.

6.1.1 Strategy Trees

The purpose of this section is to show how a strategy and its best response can be viewed as a tree, and to classify switchable edges by their position in this tree. The strategy tree will only contain vertices with finite valuation. This means that for each Max strategy σ, if a vertex v has $\text{Val}^\sigma(v) = \infty$, then it will not be contained in the strategy tree for σ. To make this more natural, we will rephrase the problem slightly so that the set of vertices with infinite valuation can be ignored.

We define the *positive-cycle* problem to be the problem of finding a strategy σ with $\text{Val}^\sigma(v) = \infty$ for some vertex v, or to prove that there is no strategy with this property. This is clearly a weakening of the zero-mean partition problem, which required us to find every vertex v for which there is some strategy σ with $\text{Val}^\sigma(v) = \infty$. Strategy improvement can be applied to solve the positive-cycle problem simply by executing it as normal, but stopping in the first iteration in which some vertex has an infinite valuation.

In this section we will show how a switching policy for the positive-cycle problem can be adapted to find the optimal strategy. Let $\alpha(\sigma, G)$ be a switching policy for the positive cycle problem on the graph G. Algorithm 4 shows a switching policy that uses α to find the optimal strategy. The algorithm first finds the set U that contains every vertex with a finite valuation in the current strategy. For a set of vertices W we define $G \restriction W$ to be the sub-game induced by W, which is G with every vertex not in W removed. The algorithm runs α on $G \restriction U$ until α produces a strategy in which some vertex has an infinite valuation. When this occurs, the algorithm will continually switch switchable edges (v, u) where $\text{Val}^\sigma(u) = \infty$ until no such edges remain. It then recomputes the set U, by removing every vertex whose valuation has risen to ∞.

To prove the correctness of this approach, we will prove the following proposition, which shows that the valuation of some vertex rises to ∞ every time the second inner while loop in Algorithm 4 is executed.

Algorithm 4 ZeroMeanPartition(σ, α, G)

```
U := V
while σ is not optimal do
  while Val^σ(v) < ∞ for every v in U do
    σ := α(σ, G ↾ U)
  end while
  while There is a switchable edge (v, u) with Val^σ(u) = ∞ do
    σ := σ[v ↦ u]
  end while
  U := U \ {v ∈ V : Val^σ(v) = ∞}
end while
```

Proposition 6.1. *If (v, u) is a switchable edge in σ with $\mathrm{Val}^\sigma(u) = \infty$, then we have $\mathrm{Val}^{\sigma[v \mapsto u]}(v) = \infty$.*

Proof. Since the edge (v, u) is switchable in σ, we have by Theorem 3.6 that $\sigma[v \mapsto u]$ is admissible. Moreover, since $\mathrm{Val}^\sigma(u) = \infty$, we have have $\mathrm{Val}^{\sigma[v \mapsto u]}(u) = \infty$. Therefore, we have $\mathrm{Play}(u, \sigma[v \mapsto u], \tau) = \langle u, v_1, \ldots v_k, \langle c_0, c_1, \ldots c_l \rangle^\omega \rangle$ with $\sum_{i=0}^{l} \mathrm{r}(c_i) > 0$ for every Min strategy τ. In the strategy $\sigma[v \mapsto u]$, every play starting at v moves directly to u. Therefore, we have $\mathrm{Play}(v, \sigma[v \mapsto u], \tau) = \langle v, u, v_1, \ldots, v_k, \langle c_0, c_1, \ldots, c_l \rangle^\omega \rangle$ with $\sum_{i=0}^{l} \mathrm{r}(c_i) > 0$ for every Min strategy τ. This implies that $\mathrm{Val}^{\sigma[v \mapsto u]}(v) = \infty$. □

Suppose that the first inner while loop terminates with a strategy σ. We will have that either there is some vertex v contained in U with $\mathrm{Val}^\sigma(v) = \infty$, or we will have that σ is an optimal strategy for the sub-game induced by the set of vertices U. In the first case, we will have that $|U|$ will strictly decrease in the next iteration of the outer while loop. In the second case, we argue that σ is also optimal for the full game. Since σ is optimal for the sub-game induced by U, there cannot be a switchable edge (v, u) where both v and u are contained in U, and there obviously cannot be a switchable edge (v, u) where both v and u are both contained in $V \setminus U$. This leaves only the case of switchable edges (v, u) where $v \in U$ and $u \in V \setminus U$. This case is dealt with by the second inner while loop, which only terminates when no such edges exist in the current strategy. Moreover, Proposition 6.1 implies that

the second inner while loop can execute at most $|V|$ times during the entire run of the algorithm. Therefore, we have the following bound on the running time of Algorithm 4.

Proposition 6.2. *If α is a switching policy that solves the positive-cycle problem in $O(f(n))$ iterations then Algorithm 4 is a switching policy that finds an optimal strategy in $O(n \cdot f(n))$ iterations.*

The purpose of defining the positive-cycle problem is to allow us to ignore the set of vertices with infinite valuation. For the rest of this chapter we will focus on devising switching policies for the positive cycle problem. Therefore, we can assume that every strategy σ that is considered by strategy improvement will have the property $\mathrm{Val}^\sigma(v) < \infty$ for every vertex v. Proposition 6.2 indicates that only a linear factor must be paid in terms of complexity to generalise our switching policies to solve the zero-mean partition.

With the assumption that every vertex has a finite valuation in place, we are now ready to define the strategy tree.

Definition 6.3 (Strategy Tree)**.** *Given a Max strategy σ and a Min strategy τ we define the tree $T^{\sigma,\tau} = (V, E')$ where $E' = \{(v, u) \ : \ \sigma(v) = u \text{ or } \tau(v) = u\}$.*

In other words, $T^{\sigma,\tau}$ is a tree rooted at the sink whose edges are those chosen by σ and τ. Recall that, although there may be many best responses to a particular strategy, the function $\mathrm{br}(\sigma)$ selects one of these best responses, and we make no assumptions about which best response this function selects. Therefore, there is a unique tree $T^{\sigma,\mathrm{br}(\sigma)}$, and we define T^σ to be shorthand for this tree. The define $\mathrm{Subtree}^\sigma(v) : V \to 2^V$ to be the function that gives the vertices in the subtree rooted at the vertex v in T^σ.

We can now define our classification for switchable edges. Let (v, u) be a switchable edge in the strategy σ. We call this a switchable *back edge* if u is in $\mathrm{Subtree}^\sigma(v)$, otherwise we call it a switchable *cross edge*.

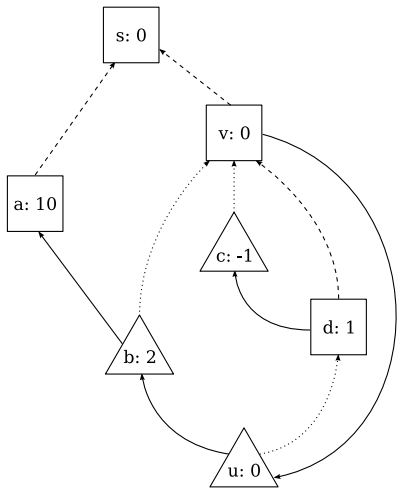

Figure 6.1: A strategy tree.

Figure 6.1 gives an example of a strategy tree. The dashed edges show a strategy for Max, and the dotted edges show the best response to this strategy. The strategy tree contains every vertex, and every edge that is either dashed or dotted. The subtree of v is the set $\{v, b, c, d, u\}$. The edge (v, u) is switchable because $\mathrm{Val}^\sigma(s) = 0$ and $\mathrm{Val}^\sigma(u) = 1$. Since u is contained in the subtree of v, the edge (v, u) is a switchable back edge.

6.1.2 The Effect Of Switching An Edge

In this section we will justify the usefulness of our classification of switchable edges by giving some basic results about how these edges affect the behaviour of strategy improvement. We will see that the result of switching a switchable cross edge is quite different to the result of switching a switchable back edge. The standard switching policies for strategy improvement are not aware of this distinction, and so they cannot exploit it in order to make better decisions.

We will measure the outcome of switching a switchable edge by the *increase* in valuation that occurs when the edge is switched. More specifically, if an edge (v, u) is switched in the strategy σ, we are interested in $\max_{w \in V}(\mathrm{Val}^{\sigma[v \mapsto u]}(w) - \mathrm{Val}^\sigma(w))$.

We begin by considering switchable cross edges. The first thing that we will show is a lower bound on how much valuations can increase when a switchable cross edge is switched. For every switchable cross edge (v, u) in the strategy σ we define the increase of that edge as:

$$\text{Increase}^{\sigma}(v, u) = \text{Val}^{\sigma}(u) - \text{Val}^{\sigma}(\sigma(v)).$$

The following lower bound has long been known for strategy improvement algorithms: when the switchable edge (v, u) is switched in a strategy σ, we will have that $\text{Val}^{\sigma[v \mapsto u]}(v) \geq \text{Val}^{\sigma}(u) + \text{r}(v)$. We will provide a short proof that this is correct for the Björklund-Vorobyov strategy improvement algorithm.

Proposition 6.4. *Let (v, u) be a switchable edge in a strategy σ. After switching the edge we have* $\text{Val}^{\sigma[v \mapsto u]}(v) - \text{Val}^{\sigma}(v) \geq \text{Increase}^{\sigma}(v, u)$.

Proof. It follows from the definition of a valuation that $\text{Val}^{\sigma}(v) = \text{Val}^{\sigma}(\sigma(v)) + \text{r}(v)$ and that $\text{Val}^{\sigma[v \mapsto u]}(v) = \text{Val}^{\sigma[v \mapsto u]}(u) + \text{r}(v)$. Since (v, u) is a switchable edge in σ, Theorem 3.6 implies that $\text{Val}^{\sigma[v \mapsto u]}(u) \geq \text{Val}^{\sigma}(u)$. Combining these gives:

$$\begin{aligned}
\text{Val}^{\sigma[v \mapsto u]}(v) - \text{Val}^{\sigma}(v) &= (\text{Val}^{\sigma[v \mapsto u]}(u) + \text{r}(v)) - (\text{Val}^{\sigma}(\sigma(v)) + \text{r}(v)) \\
&= \text{Val}^{\sigma[v \mapsto u]}(u) - \text{Val}^{\sigma}(\sigma(v)) \\
&\geq \text{Val}^{\sigma}(u) - \text{Val}^{\sigma}(\sigma(v)).
\end{aligned}$$

□

The lower bound given in Proposition 6.4 holds for both switchable cross edges and switchable back edges. However, for switchable cross edges we can show that this bound is tight.

Proposition 6.5. *Let (v, u) be a switchable cross edge in the strategy σ. For every vertex w we have* $\text{Val}^{\sigma[v \mapsto u]}(w) - \text{Val}^{\sigma}(w) \leq \text{Increase}^{\sigma}(v, u)$.

Proof. We assume that Min plays $\text{br}(\sigma)$ against $\sigma[v \to u]$. We will prove for every

vertex w that

$$\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w) - \mathrm{Val}^{\sigma}(w) \leq \mathrm{Increase}^{\sigma}(v,u).$$

This is sufficient to prove the proposition because, by properties of the best response, we have that $\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma[v\mapsto u])}(w) \leq \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w)$, and therefore

$$\mathrm{Val}^{\sigma[v\mapsto u]}(w) - \mathrm{Val}^{\sigma}(w) \leq \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w) - \mathrm{Val}^{\sigma}(w).$$

We shall first consider a vertex w that is not in $\mathrm{Subtree}^{\sigma}(v)$. Note that $\mathrm{Play}(w, \sigma[v \mapsto u], \mathrm{br}(\sigma))$ does not pass through through the vertex v, which implies that $\mathrm{Play}(w, \sigma[v \mapsto u], \mathrm{br}(\sigma)) = \mathrm{Play}(w, \sigma, \mathrm{br}(\sigma))$. Since the valuation of a vertex is determined entirely by the play, this implies that $\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w) = \mathrm{Val}^{\sigma}(w)$. Note that since (v,u) is a switchable edge it must be the case that $\mathrm{Increase}^{\sigma}(v,u) > 0$. We therefore have:

$$\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w) - \mathrm{Val}^{\sigma}(w) = 0 < \mathrm{Increase}^{\sigma}(v,u).$$

Now consider a vertex w that is in $\mathrm{Subtree}^{\sigma}(v)$. Let $\pi = \langle w = v_0, v_1, \ldots, v_k = v\rangle$ be the prefix of $\mathrm{Play}(w, \sigma, \mathrm{br}(\sigma))$ that ends at the vertex v. Since the only modification to σ was to change the successor of v, we also have that π is a prefix of $\mathrm{Play}(w, \sigma[v \mapsto u], \mathrm{br}(\sigma))$. If $c = \sum_{i=0}^{k} \mathrm{r}(v_i)$ then we the following two expressions for the valuation of w:

$$\mathrm{Val}^{\sigma}(w) = \mathrm{Val}^{\sigma}(v) + c,$$
$$\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w) = \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(v) + c.$$

Therefore, the increase of the valuation of w can be represented as:

$$\begin{aligned}\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(w) - \mathrm{Val}^{\sigma}(w) &= (\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(v) + c) - (\mathrm{Val}^{\sigma}(v) + c) \\ &= \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(v) - \mathrm{Val}^{\sigma}(v).\end{aligned}$$

To complete the proof we must show that $\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(v) - \mathrm{Val}^{\sigma}(v) \leq \mathrm{Increase}^{\sigma}(v,u)$. By definition we have that $\mathrm{Val}^{\sigma}(v) = \mathrm{Val}^{\sigma}(\sigma(v)) + \mathrm{r}(v)$ and we have that $\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(v) = \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(u) + \mathrm{r}(v)$. Since (v,u) is a switchable cross edge, we have that u is not in the subtree of v, and so we have already shown that $\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(u) = \mathrm{Val}^{\sigma}(u)$. Combining these facts gives:

$$\begin{aligned}\mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(v) - \mathrm{Val}^{\sigma}(v) &= \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(u) + \mathrm{r}(v) - (\mathrm{Val}^{\sigma}(\sigma(v)) + \mathrm{r}(v)) \\ &= \mathrm{Val}^{\sigma[v\mapsto u],\mathrm{br}(\sigma)}(u) - \mathrm{Val}^{\sigma}(\sigma(v)) \\ &= \mathrm{Val}^{\sigma}(u) - \mathrm{Val}^{\sigma}(\sigma(v)) \\ &= \mathrm{Increase}^{\sigma}(v,u).\end{aligned}$$

□

The lower bound given by Proposition 6.4 is the one that is generally used in the design of switching policies. This can be seen, for example, in the greedy switching policy that was defined in Section 3.2.4. When there is more than one switchable edge at a vertex v, the greedy policy chooses the switchable edge (v,u) which has the largest value of $\mathrm{Appeal}^{\sigma}(v,u)$. The reason for this choice is that Proposition 6.4 guarantees the largest increase in valuations for the edge (v,u). With Proposition 6.5 we have shown that this is a valid approach when the edge (v,u) is a switchable cross edge. In the remainder of this section, we will argue that relying on Proposition 6.4 is not a good idea when considering switchable back edges.

For a switchable back edge, the lower bound given by Proposition 6.4 can dramatically underestimate the increase in valuation that is caused by switching the

edge. The example shown in Figure 6.1 is one such case. For the strategy σ that is shown, we have $\text{Val}^{\sigma}(u) = 1$, and therefore $\text{Increase}^{\sigma}(v, u) = 1$. However, when the edges (v, u) is switched, Min will use the edges (b, a) and (u, b) in the best response to $\sigma[v \mapsto u]$. This causes the valuation of u to rise to 12, and we therefore have that $\text{Val}^{\sigma[v \mapsto u]}(u) - \text{Val}^{\sigma}(u) = 11$, which is clearly much larger than the increase that was implied by Proposition 6.4.

The problem with the lower bound given in Proposition 6.4 is that it does not consider the effect that switching an edge has on the behaviour of the opponent. We will now formalize what this effect is. For a switchable back edge (v, u) in a strategy σ we define the critical set, which is the vertices in $\text{Subtree}^{\sigma}(v)$ that Min can reach from u when Max plays σ.

Definition 6.6 (Critical Set)**.** *If (v, u) is a switchable back edge in the strategy σ, then we define the critical set as*

$$\begin{aligned}\text{Critical}^{\sigma}(v, u) = \{w \in \text{Subtree}^{\sigma}(v) \ : \ & \textit{there is a path } \langle u, u_1, \ldots, u_k = w\rangle, \\ & \textit{where for all } i \textit{ with } 1 \leq i \leq k \textit{ we have } u_i \in \text{Subtree}^{\sigma}(v), \\ & \textit{and if } u_i \in V_{Max} \textit{ then } u_{i+1} = \sigma(u_i)\}.\end{aligned}$$

In the example given by Figure 6.1, the critical set for the switchable back edge (v, u) is $\{v, b, d, u\}$. The vertex b is in the critical set because it is in the subtree of v, and Min can reach it from u when Max plays σ. On the other hand, the vertex c is not in the critical set because $\sigma(d) = v$, and therefore Min cannot reach c from u when Max plays σ. The vertex a is not in the critical set because it is not in the subtree of v.

We can now explain why Min must use the edge (b, a) in the best response to $\sigma[v \mapsto u]$ in this example. This happens because $\sigma[v \mapsto u]$ is a winning strategy for the vertices in the critical set of (v, u). It can be verified that if Min does not choose an edge that leaves this critical set, then a positive cycle must be formed. Therefore,

Min must use the edge (b, a) in the best response to secure a finite valuation. The fact that $\sigma[v \mapsto u]$ is a winning strategy for the critical set of (v, u) is a general property, which we will now prove. In this proof we will refer to valuations from multiple games. We use $\mathrm{Val}^{\sigma}_{G}(v)$ to give the valuation of the vertex v when σ is played against $\mathrm{br}(\sigma)$ in the game G. We extend all of our notations in a similar manner, by placing the game in the subscript.

Proposition 6.7. *Let (v, u) be a switchable back edge in the strategy σ and let $C = \mathrm{Critical}^{\sigma}(v, u)$. The strategy $\sigma[v \mapsto u]$ is winning for every vertex in $G \restriction C$.*

Proof. Since C is a critical set it must be the case that every vertex in C must be in the subtree of v according to σ, and this implies that $\sigma[v \mapsto u](w)$ is not the sink, for every vertex w in C. Note that only paths ending at the sink can have finite valuations, and that no such paths can exist when $\sigma[v \mapsto u]$ is played in $G \restriction C$. Therefore, we must argue that $\sigma[v \mapsto u]$ is an admissible strategy for $G \restriction C$.

Suppose for the sake of contradiction that $\sigma[v \mapsto u]$ is inadmissible. This implies that there is some vertex v and some Min strategy τ for which $\mathrm{Play}_{G \restriction C}(v, \sigma[v \mapsto u], \tau)$ ends in a negative cycle. We define τ' to be a strategy G that follows τ on the vertices in $G \restriction C$ and makes arbitrary decisions at the other vertices. For every vertex w in V_{Min} we choose some edge (w, x) and define:

$$\tau'(w) = \begin{cases} \tau(w) & \text{if } w \in C, \\ x & \text{otherwise.} \end{cases}$$

Now consider $\sigma[v \mapsto u]$ played against τ' on the game G. Note that neither of the two strategies choose an edge that leaves the set C and so $\mathrm{Play}_{G}(w, \sigma[v \mapsto u], \tau') = \mathrm{Play}_{G \restriction C}(w, \sigma[v \mapsto u], \tau)$ for every vertex w in C. Therefore, $\mathrm{Play}_{G}(w, \sigma[v \mapsto u], \tau')$ must end in a negative cycle in G. This implies that $\sigma[v \mapsto u]$ is inadmissible in G, which contradicts Theorem 3.6 because (v, u) was a switchable edge in σ. Therefore, $\sigma[v \mapsto u]$ is an admissible strategy in $G \restriction U$. □

In our running example, the edge (b, a) allowed Min to leave the critical set of the edge (v, u). We call these edges *escapes*. An escape edge from a given set of vertices is an edge that Min can use to leave that set of vertices.

Definition 6.8 (Escapes)**.** *Let W be a set of vertices. We define the escapes from W as* $\mathrm{Esc}(W) = \{(v, u) \in E \ : \ v \in W \cap V_{Min} \text{ and } u \notin W\}$.

The effect of switching a switchable back edge is to force Min to use an escape edge from the critical set of that back edge.

Proposition 6.9. *Let (v, u) be a switchable back edge in the strategy σ. There is some edge (x, y) in* $\mathrm{Esc}(\mathrm{Critical}^{\sigma}(v, u))$ *such that* $\mathrm{br}(\sigma[v \mapsto u])(x) = y$.

Proof. Consider a strategy τ for player Min for which there is no edge (x, y) in $\mathrm{Esc}(\mathrm{Critical}^{\sigma}(v, u))$ with $\tau(x) = y$. We argue that $\mathrm{Val}^{\sigma[v \mapsto u], \tau}(w) = \infty$ for every vertex w in $\mathrm{Critical}^{\sigma}(v, u)$. Note that neither $\sigma[v \mapsto u]$ or τ chooses an edge that leaves $\mathrm{Critical}^{\sigma}(v, u)$, which implies that $\mathrm{Play}(w, \sigma[v \mapsto u], \tau)$ does not leave $\mathrm{Critical}^{\sigma}(v, u)$, for every vertex w in $\mathrm{Critical}^{\sigma}(v, u)$. By Proposition 6.7 we have that $\sigma[v \mapsto u]$ is a winning strategy for $G \restriction \mathrm{Critical}^{\sigma}(v, u)$, and therefore $\mathrm{Val}^{\sigma[v \mapsto u], \tau}(w) = \infty$ for every vertex w in $\mathrm{Critical}^{\sigma}(v, u)$.

We will now construct a strategy for Min which, when played against $\sigma[v \mapsto u]$, guarantees a finite valuation for some vertex in $\mathrm{Critical}^{\sigma}(v, u)$. Let (x, y) be some edge in $\mathrm{Esc}(\mathrm{Critical}^{\sigma}(v, u))$. We define the Min strategy τ, for every vertex w in V_{Min} as:

$$\tau(w) = \begin{cases} y & \text{if } w = x, \\ \mathrm{br}(\sigma)(w) & \text{otherwise.} \end{cases}$$

By definition of the critical set we have that y cannot be in the subtree of v, since otherwise it would also be in $\mathrm{Critical}^{\sigma}(v, u)$. This implies that $\mathrm{Play}(y, \sigma, \mathrm{br}(\sigma)) = \mathrm{Play}(y, \sigma[v \mapsto u], \tau)$, since $\tau = \mathrm{br}(\sigma)$ on every vertex that is not in $\mathrm{Subtree}^{\sigma}(v)$, and $\sigma = \sigma[v \mapsto u]$ on every vertex that is not v. From this we can conclude that

$\mathrm{Val}^{\sigma[v\mapsto u],\tau}(y) = \mathrm{Val}^{\sigma}(y) < \infty$. By construction of τ we have that $\mathrm{Val}^{\sigma[v\mapsto u],\tau}(x) = \mathrm{Val}^{\sigma[v\mapsto u],\tau}(y) + \mathrm{r}(x)$, and so we also have $\mathrm{Val}^{\sigma[v\mapsto u],\tau}(x) < \infty$.

In summary, we have shown that every Min strategy τ that does not use an edge in $\mathrm{Esc}(\mathrm{Critical}^{\sigma}(v,u))$ has the property $\mathrm{Val}^{\sigma[v\mapsto u],\tau}(w) = \infty$ for every vertex v in $\mathrm{Critical}^{\sigma}(v,u)$. We have also shown that there is a Min strategy τ which guarantees $\mathrm{Val}^{\sigma[v\mapsto u],\tau}(w) < \infty$ for some vertex w in $\mathrm{Critical}^{\sigma}(v,u)$. From the properties of a best response we can conclude that Min must use some edge in $\mathrm{Esc}(\mathrm{Critical}^{\sigma}(v,u))$. □

We will use the property given by Proposition 6.9 to define a better lower bound for the increase in valuation caused by switching a switchable back edge. We define:

$$\mathrm{EscInc}^{\sigma}(v,u) = \min\{(\mathrm{Val}^{\sigma}(y) + \mathrm{r}(x)) - \mathrm{Val}^{\sigma}(x) \ : \ (x,y) \in \mathrm{Esc}(\mathrm{Critical}^{\sigma}(v,u))\}.$$

The function EscInc captures the smallest increase in valuation that can be caused when Min chooses an escape from the critical set. In the example given by Figure 6.1 we have $\mathrm{EscInc}^{\sigma}(v,u) = 12$, which accurately captures the increase of valuation at u when the edge (v,u) is switched. We can now extend the Increase function to give a lower bound for every switchable edge (v,u) in the strategy σ.

$$\mathrm{NewInc}^{\sigma}(v,u) = \begin{cases} \mathrm{Increase}^{\sigma}(v,u) & \text{if } (v,u) \text{ is a cross edge,} \\ \max(\mathrm{Increase}^{\sigma}(v,u), \mathrm{EscInc}(v,u)) & \text{otherwise.} \end{cases}$$

We now prove that this lower bound is correct.

Proposition 6.10. *Let (v,u) be a switchable edge in the strategy σ. There is a vertex w such that* $\mathrm{Val}^{\sigma[v\mapsto u]}(w) - \mathrm{Val}^{\sigma}(w) \geq \mathrm{NewInc}^{\sigma}(v,u)$.

Proof. For switchable cross edges this proposition is implied by Proposition 6.4. For switchable back edges, if $\mathrm{Increase}^{\sigma}(v,u) \geq \mathrm{EscInc}(v,u))$ then this proposition is

again implied by Proposition 6.4. Otherwise, Proposition 6.9 implies that there is some edge (x, y) in $\text{Esc}(\text{Critical}^\sigma(v, u))$ such that $\text{br}(\sigma[v \mapsto u])(x) = y$. We therefore have that $\text{Val}^{\sigma[v\mapsto u]}(x) = \text{Val}^{\sigma[v\mapsto u]}(y) + \text{r}(x)$, and by Theorem 3.6 combined with the fact that (v, u) is a switchable edge we have $\text{Val}^{\sigma[v\mapsto u]}(y) \geq \text{Val}^\sigma(y)$. The increase at x is therefore

$$\begin{aligned}\text{Val}^{\sigma[v\mapsto u]}(x) - \text{Val}^\sigma(x) &= \text{Val}^{\sigma[v\mapsto u]}(y) + \text{r}(x) - \text{Val}^\sigma(x) \\ &\geq \text{Val}^\sigma(y) + \text{r}(x) - \text{Val}^\sigma(x) \\ &\geq \text{EscInc}(v, u).\end{aligned}$$

□

We close this section by using our new bounds to define a more intelligent version of the greedy policy. While the original policy picks, for each vertex v in a strategy σ, the edge (v, u) that maximizes $\text{Appeal}^\sigma(v, u)$, our version of the greedy policy will pick the edge that maximizes $\text{NewInc}^\sigma(v, u)$. Once again, to define this formally we require that each vertex v is given a unique index in the range $\{1, 2, \ldots, |V|\}$, which will denote as $\text{Index}(v)$.

$$\begin{aligned}\text{New-Greedy}^\sigma(F) = \{(v, u) \ :\ &\text{there is no edge } (v, w) \in F \text{ with} \\ &\text{NewInc}^\sigma(v, u) < \text{NewInc}^\sigma(v, w) \text{ or with } (\text{NewInc}^\sigma(v, u) = \text{NewInc}^\sigma(v, w) \\ &\qquad\qquad \text{and } \text{Index}(u) < \text{Index}(w))\}.\end{aligned}$$

6.2 Remembering Previous Iterations

In the previous section we classified switchable edges into two types: switchable cross edges and switchable back edges. We also showed how switching policies can be made aware of these concepts. However, the New-Greedy switching policy still takes an exponential number of steps on the examples of Friedmann, which implies that simply being aware of these concepts is not enough.

With the New-Greedy switching policy, we showed how awareness of switchable back edges in a strategy could be exploited when strategy improvement considers that strategy. However, we claim that a switchable back edge can be used to make better decisions in other iterations. In this section, we introduce a structure that is called a snare. The dictionary definition[1] of the word snare is "something that serves to entangle the unwary". This is a particularly apt metaphor for these structures since, as we will show, a winning strategy for a player must be careful to avoid being trapped by the snares that are present in that player's winning set.

A snare is a structure that is contained in a parity or mean-payoff game, and we will show that every switchable back edge that is encountered by strategy improvement must correspond to some snare that is embedded in the game. We propose that strategy improvement algorithms should remember a snare for each switchable back edge that they encounter, and we show how a switching policy that has remembered a snare can exploit this knowledge to make better progress.

6.2.1 Snares

In this section we introduce a structure that we call a snare. These structures are contained in a mean-payoff game, and we claim that the behaviour of strategy improvement on a given mean-payoff game is strongly related to the snares that exist in that game. In this section we will introduce these structures, and derive some of their general properties.

As usual, the definitions that we give in this section could be formalized for either player. We choose to focus on player Max because we chose Max to be the strategy improver. A snare for player Max is defined to be a sub-game for which player Max can guarantee a win from every vertex.

Definition 6.11 (Max Snare)**.** *For a game G, a snare is defined to be a tuple (W, χ) where $W \subseteq V$ and $\chi : W \cap V_{Max} \to W$ is a partial strategy for player Max that is*

[1] American Heritage Dictionary of the English Language, Fourth Edition

winning for every vertex in the sub-game $G \restriction W$.

Clearly, this definition is strongly related to the properties of switchable back edges that we exposed in Section 6.1.2. If (v, u) is a switchable back edge in a strategy σ, then we define $\text{Snare}^\sigma(v, u)$ to be $(\text{Critical}^\sigma(v, u), \chi)$, where χ is the strategy $\sigma[v \mapsto u]$ defined only for the vertices contained in $\text{Critical}^\sigma(v, u)$. For for every vertex w we have:

$$\chi(w) = \begin{cases} \sigma[v \mapsto u](w) & \text{if } w \in \text{Critical}^\sigma(v, u), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since Proposition 6.7 implies that χ is a winning strategy for the set $\text{Critical}^\sigma(v, u)$, we have that $\text{Snare}^\sigma(v, u)$ meets the definition of a snare for every switchable back edge (v, u). For the example shown in Figure 6.1, we have that $\text{Snare}^\sigma(v, u) = (\{v, u, b, d\}, \{v \mapsto u, d \mapsto v\})$.

Although our definition of a snare is closely tied to the properties of switchable back edges, we argue that applications of snares are not restricted to strategy improvement. In the following proposition, we will show that the existence of a snare in a game places restrictions upon the strategies that are winning for that game. Recall that in the zero-mean partition problem, a winning strategy for player Min is one that ensures a non-positive payoff.

Proposition 6.12. *Suppose that* τ *is a winning strategy for Min on a set of vertices* S. *If* (W, χ) *is a Max snare where* $W \subset S$, *then there is some edge* (v, u) *in* $\text{Esc}(W)$ *such that* $\tau(v) = u$.

Proof. For the sake of contradiction, suppose that τ is a winning strategy for S that does not choose an edge in $\text{Esc}(W)$. Since χ also does not choose an edge that leaves W, we have that $\text{Play}(v, \chi, \tau)$ never leaves the set W, for every vertex v in W. Furthermore, since χ is a winning strategy for the sub-game induced by W we have

$\mathcal{M}(\text{Play}(v, \chi, \tau)) > 0$ for every vertex v in W, which contradicts the fact that τ is a winning strategy for S. □

We will now argue that the conditions given by Proposition 6.12 must be observed in order for strategy improvement to terminate. We begin by defining a concept that we call snare consistency. This concept captures the idea that every winning strategy for Min must choose an escape from every Max snare. We say that a Max strategy is consistent with a snare if Min's best response chooses an escape from that snare.

Definition 6.13 (Snare Consistency)**.** *A strategy σ is said to be consistent with the snare (W, χ) if $\text{br}(\sigma)$ uses some edge in $\text{Esc}(W)$.*

We can show that strategy improvement cannot terminate unless the current strategy is consistent with every snare that exists in the game. This is because every strategy that is not consistent with some snare must contain a switchable edge.

Proposition 6.14. *Let σ be a strategy that is not consistent with a snare (W, χ). There is a switchable edge (v, u) in σ such that $\chi(v) = u$.*

Proof. In order to prove the claim we will construct an alternate game. We define the game $G' = (V, V_{\text{Max}}, V_{\text{Min}}, E', \text{r})$ where:

$$E' = \{(v, u) \ : \ \sigma(v) = u \text{ or } \text{br}_G(\sigma)(v) = u \text{ or } \chi(v) = u\}.$$

In other words, we construct a game where Min is forced to play $\text{br}_G(\sigma)(v)$ and Max's strategy can be constructed using a combination of the edges used by σ and χ. Since Min is forced to play $\text{br}_G(\sigma)(v)$ we have that $\text{Val}^\sigma_G(v) = \text{Val}^\sigma_{G'}(v)$ for every vertex v. To decide if an edge is switchable we compare two valuations, and since the valuation of σ is the same in both G and G' we have that an edge is switchable for σ in G if and only if it is switchable for σ in G'. Note also that the only way σ can be modified in G' is to choose an edge that is chosen by χ but not

by σ. Therefore, to prove our claim it is sufficient to show that σ has a switchable edge in G'.

We define the strategy:

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

We will argue that χ' is a better strategy than σ in G'. The definition of a snare implies that χ is a winning strategy for the sub-game induced by W, and by assumption we have that $\mathrm{br}(\sigma)$ does not use an edge in $\mathrm{Esc}(W)$. We therefore have that $\mathrm{Val}^{\chi'}_{G'}(v) = \infty$ for every vertex v in W. On the other hand, since we are considering the positive cycle problem, we know that $\mathrm{Val}^{\sigma}_{G'}(v) < \infty$ for every vertex v in W. This implies that σ is not an optimal strategy in G'. Theorem 3.6 implies that all non-optimal strategies must have at least one switchable edge, and the only edges that can be switchable in G' are those chosen by χ. Therefore there is some edge chosen by χ that is switchable for σ in G' and as we have argued this also means that the edge is switchable for σ in G. □

6.2.2 Using Snares To Guide Strategy Improvement

In the previous sections, we have shown that the switchable back edges that strategy improvement encounters are related to the snares that exist in the game. In this section, we will show how snares can be used to guide strategy improvement. We then propose a new kind of strategy improvement algorithm that remembers the switchable back edges that it encounters in previous iterations, and then uses those snares to guide itself in future iterations.

Proposition 6.14 implies that strategy improvement cannot terminate until the current strategy is consistent with every snare in the game. It therefore seems natural that strategy improvement algorithms should try to maintain consistency

with the snares that are known to exist in the game. We will give a method by which strategy improvement algorithms can achieve this.

We will give an efficient procedure FixSnare that takes a snare, and a strategy σ that is inconsistent with that snare. It will return a strategy σ' that is consistent with the snare. Moreover, we will have $\sigma \prec \sigma'$. This means that applying FixSnare during strategy improvement does not break the property given in Theorem 3.6, and therefore strategy improvement algorithms that use FixSnare are still guaranteed to terminate.

To define FixSnare we will use Proposition 6.14. Recall that this proposition implies that if a strategy σ is inconsistent with a snare (W, χ), then there is some switchable edge (v, u) in σ such that $\chi(v) = u$. Our procedure will actually be a strategy improvement switching policy, which will choose to switch an edge that is chosen by χ but not by the current strategy. As long as the current strategy remains inconsistent with (W, χ) such an edge is guaranteed to exist, and the policy terminates once the current strategy is consistent with the snare. This procedure is shown as Algorithm 5.

Algorithm 5 FixSnare$(\sigma, (W, \chi))$

```
while σ is inconsistent with (W, χ) do
  (v, w) := Some edge where χ(v) = w and (v, w) is switchable in σ.
  σ := σ[v ↦ u]
end while
return σ
```

Since FixSnare is implemented as a strategy improvement switching policy that switches only switchable edges, Theorem 3.6 implies that the strategy that is produced must be an improved strategy. The following proposition proves that the procedure FixSnare does indeed produce a strategy that is consistent with the snare that was given to it.

Proposition 6.15. *Let σ be a strategy that is not consistent with a snare (W, χ). Algorithm 5 will arrive at a strategy σ' which is consistent with (W, χ) after at*

most $|W|$ iterations.

Proof. By Proposition 6.14 we know that as long as the current strategy is not consistent with the snare (W, χ) there must be an edge (v, u) with $\chi(v) = u$ that is switchable in σ. The switching policy will always choose this edge, and will terminate once the current strategy is consistent with the snare. Therefore in each iteration the number of vertices upon which σ and χ differ decreases by 1. It follows that after at most $|W|$ iterations we will have $\sigma(v) = \chi(v)$ for every vertex v in W. Since χ is a winning strategy for the sub-game induced by W we have that player Min must choose some edge that leaves W to avoid losing once this strategy has been reached. □

We now define a strategy improvement algorithm that can exploit the properties of snares. This algorithm will record a snare for every switchable back edge that it encounters during its execution. In each iteration it can either switch a subset of switchable edges or run the procedure FixSnare on some recorded snare that the current strategy is inconsistent with. This algorithm is shown in Algorithm 6. We will describe the properties of the function Policy in the next section.

Algorithm 6 NonOblivious(σ)

```
S := ∅
while σ has a switchable edge do
    S := S ∪ {Snare^σ(v, u) : (v, u) is a switchable back edge in σ}
    σ := Policy(σ, S)
end while
return σ
```

6.2.3 Switching Policies For Snare Based Strategy Improvement

Traditional strategy improvement algorithms require a switching policy to decide which edges should be switched in each iteration, and our strategy improvement algorithm requires a similar function. However in our setting, the switching policy can decide to either switch a subset of switchable edges, or to run the procedure

FixSnare for some snare that has been recorded. There are clearly many possible switching policies for our algorithm, however in this section we will give one specific method of adapting a switching policy from traditional strategy improvement for use in our strategy improvement algorithm.

The switching policy for our algorithm will begin with a switching policy α from traditional strategy improvement, such as the greedy policy, the optimal policy, or the random-facet policy. Whenever our switching policy chooses not to use FixSnare, it will choose to switch the edges that would have been switched by α. Our goal is to only apply FixSnare when doing so would provide a better increase in valuation than the increase that would be obtained by applying α. The first part of this section is concerned with formalizing this idea.

The goal of our switching policy will be to obtain the largest possible increase in valuation in each step. This is a heuristic that has been used by traditional switching policies, such as the optimal switching policy given by Schewe [Sch08]. It is not difficult to see that strategy improvement must terminate after at most $|V| \cdot \sum_{v \in V} |w(v)|$ iterations. This is because the valuation of some vertex must increase by at least 1 in every iteration. From this we can see that strategy improvement to performs poorly in the situations where the valuations of the vertices increase slowly. In order for the algorithm to actually take $|V| \cdot \sum_{v \in V} |w(v)|$ iterations, each iteration of the algorithm must increase the valuation of exactly one vertex by 1. Therefore, by attempting to ensure a large increase in valuations, we are also attempting to minimize the number of iterations that the algorithm takes.

Another potential justification for this heuristic is that it attempts to maximize the number of strategies that strategy improvement eliminates in each iteration. Each run of strategy improvement produces a sequence of strategies that form a chain in the $\prec$ ordering. When strategy improvement chooses a successor to a strategy σ_i, it may only move to a strategy σ_{i+1} with the property $\sigma_i \prec \sigma_{i+1}$. If there is a strategy χ such that $\sigma_i \prec \chi$, but either $\chi \prec \sigma_{i+1}$ or χ is incomparable

with σ_{i+1}, then this strategy is eliminated. This is because strategy improvement can never visit the strategy χ after it has visited the strategy σ_{i+1}. It is not difficult to see that attempting to maximize the increase in valuation also maximizes the number of strategies eliminated by each step.

We now show how the increase in valuation of a traditional switching policy α can be determined. Since every iteration of strategy improvement takes polynomial time, we can simply switch the edges and measure the difference between the current strategy and the one that would be produced. Let σ be a strategy and let P be the set of edges that are switchable in σ. The increase of applying α is defined to be:

$$\text{PolicyIncrease}(\alpha, \sigma) = \sum_{v \in V} (\text{Val}^{\sigma[\alpha(P)]}(v) - \text{Val}^{\sigma}(v)).$$

We now give a lower bound on the increase in valuation that an application of FixSnare produces. Let (W, χ) be a snare and suppose that the current strategy σ is inconsistent with this snare. Our lower bound is based on the fact that FixSnare produces a strategy that is consistent with the snare. This means that Min's best response is not currently choosing an escape from the snare, but it will be forced to do so after FixSnare has been applied. It is easy to see that forcing the best response to use a different edge will cause an increase in valuation, since otherwise the best response would already be using that edge. Therefore, we can use the increase in valuation that will be obtained when Min is forced to use and escape. We define:

$$\text{SnareIncrease}^{\sigma}(W, \chi) = \min\{(\text{Val}^{\sigma}(y) + \text{r}(x)) - \text{Val}^{\sigma}(x) \ : \ (x, y) \in \text{Esc}(W)\}.$$

This expression gives the smallest possible increase in valuation that can happen when Min is forced to use an edge in $\text{Esc}(W)$. We can prove that applying FixSnare will cause an increase in valuation of at least this amount.

Proposition 6.16. *Let σ be a strategy that is not consistent with a snare (W, χ),*

and let σ' *be the result of* FixSnare$(\sigma, (W, \chi))$. *We have:*

$$\sum_{v \in V} (\mathrm{Val}^{\sigma'}(v) - \mathrm{Val}^{\sigma}(v)) \geq \mathrm{SnareIncrease}^{\sigma}(W, \chi).$$

Proof. We will prove this proposition by showing that there exists some vertex w with the property $\mathrm{Val}^{\sigma'}(w) - \mathrm{Val}^{\sigma}(w) \geq \mathrm{SnareIncrease}(W, \chi)$. Since the procedure FixSnare switches only switchable edges we have by Theorem 3.6 that $\mathrm{Val}^{\sigma'}(v) - \mathrm{Val}^{\sigma}(v) \geq 0$ for every vertex v. Therefore, this is sufficient to prove the proposition because $\sum_{v \in V} (\mathrm{Val}^{\sigma'}(v) - \mathrm{Val}^{\sigma}(v)) \geq \mathrm{Val}^{\sigma'}(w) - \mathrm{Val}^{\sigma}(w)$.

Proposition 6.15 implies that σ' is consistent with the snare (W, χ). By the definition of snare consistency, this implies that $\mathrm{br}(\sigma')$ must use some edge (w, x) in $\mathrm{Esc}(W)$. We therefore have that $\mathrm{Val}^{\sigma'}(w) = \mathrm{Val}^{\sigma'}(x) + \mathrm{r}(w)$. Since the FixSnare procedure switches only switchable edges, we have by Theorem 3.6 that $\mathrm{Val}^{\sigma'}(x) \geq \mathrm{Val}^{\sigma}(x)$. The increase at w is therefore:

$$\begin{aligned} \mathrm{Val}^{\sigma'}(w) - \mathrm{Val}^{\sigma}(w) &= \mathrm{Val}^{\sigma'}(x) + \mathrm{r}(w) - \mathrm{Val}^{\sigma}(w) \\ &\geq \mathrm{Val}^{\sigma}(x) + \mathrm{r}(w) - \mathrm{Val}^{\sigma}(w) \\ &\geq \mathrm{SnareIncrease}(W, \chi). \end{aligned}$$

□

We now have the tools necessary to construct our proposed augmentation scheme, which is shown as Algorithm 7. The idea is to compare the increase obtained by applying α and the increase obtained by applying FixSnare with the best snare that has been previously recorded, and then to only apply FixSnare when it is guaranteed to yield a larger increase in valuation.

6.3 Evaluation

Unfortunately, we have not been able to show good worst case upper bounds for the running time of our augmented switching policies. In this section, we justify their

Algorithm 7 Augment$(\alpha)(\sigma, S)$

$(W, \chi) := \text{argmax}_{(X,\mu)\in S}\, \text{SnareIncrease}^{\sigma}(X, \mu)$
if $\text{PolicyIncrease}(\alpha, \sigma) > \text{SnareIncrease}^{\sigma}(W, \chi)$ **then**
  $P := \{(v, u) \ : \ (v, u) \text{ is switchable in } \sigma\}$
  $\sigma := \sigma[\alpha(P)]$
else
  $\sigma := \text{FixSnare}(\sigma, (W, \chi))$
end if
return σ

usefulness by explaining how the augmentation helps traditional switching policies to avoid the pathological behaviour that occurs when they are run on the examples that have been used to show super-polynomial lower bounds.

6.3.1 Performance on Super Polynomial Time Examples

In this section, we analyse the examples that have been used to show super polynomial time lower bounds for the greedy switching policy, the optimal switching policy, and the random facet switching policy. The examples for the greedy and optimal switching policies were given by Friedmann [Fri09], and the examples for the random facet switching policy were given by Friedmann, Hansen, and Zwick [FHZ10]. We will explain why the lower bounds do not apply to the augmented versions of these switching policies.

We begin by describing the general themes that appear in both of the families of examples. These themes also appear in the family of MDPs that we constructed in Chapter 5. The examples use a structure to represent the state of a bit in a binary counter. Figure 6.2 shows these structures. Each of these gadgets contains exactly one state belonging to Min, and the state of the bit can be determined by the strategy that Min uses in the best response at this state. The bit represented by the gadget is a 1 for some strategy if and only if the best response to that strategy chooses the edge that leaves the gadget.

In the example of size n, there are n instances of the appropriate gadget.

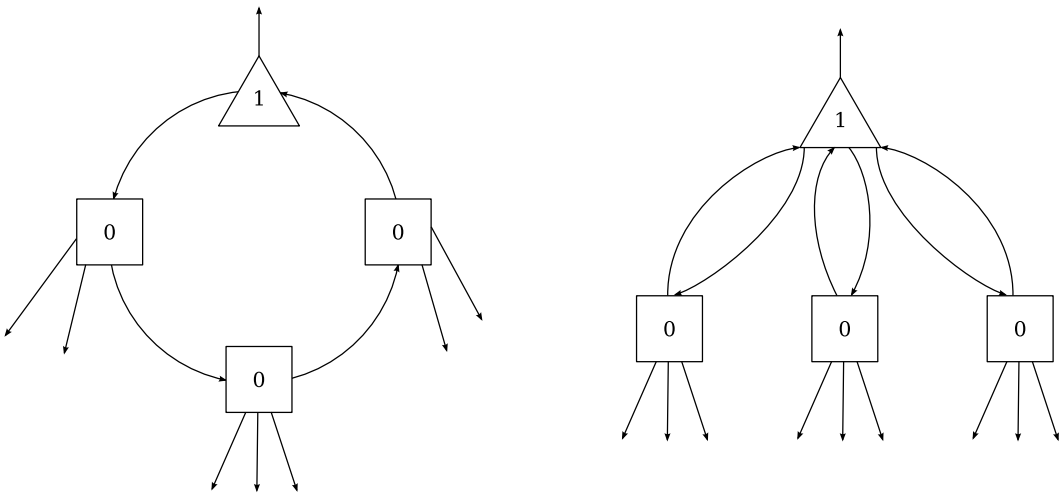

Figure 6.2: Left: the gadget that represents a bit in Friedmann's examples. Right: the gadget that represents a bit in the examples of Friedmann, Hansen, and Zwick.

The connections between the gadgets are designed to force a switching policy to take super-polynomial time. Friedmann's examples force the greedy and optimal switching policies to mimic a binary counter, which implies an exponential lower bound. The examples of Friedmann, Hansen, and Zwick force the random facet switching policy to mimic a *randomized counter*. This is a stochastic model that bears some similarity to a binary counter, and it is known that a process that mimics a randomized counter takes an expected sub-exponential number of steps to terminate.

To see why our augmentation procedure is helpful, it suffices to note that the bit gadgets are Max snares. It is not difficult to see that in both cases, if Max does not choose to leave the gadget, then Min is forced to use the edge that leaves the gadget. Furthermore, when a bit is set to 1, strategy improvement switches a switchable back edge that corresponds to this snare. Therefore, in the iteration immediately after a bit has been set to 1 for the first time, our augmentation procedure will have remembered the snare that corresponds to the gadget representing that bit.

We can now give a simple argument that shows that the augmented versions of the greedy and optimal policies must terminate in polynomial time on Friedmann's examples. Suppose that the switching policy sets the i-th bit to 1 for the first time, and that after doing so it sets every bit that is smaller than i to 0. When the bits are set to 0, Min's best response will no longer use the escape edge from the snare, and it turns out that fixing these snares gives a much larger increase in valuation than continuing to use the switching policy. Therefore, the augmentation procedure will spend $i - 1$ iterations to fix these snares, and it will arrive at a strategy in which the first i bits are 1. Clearly, this rules out the exponential behaviour, and the algorithm must terminate after a quadratic number of steps. Similar arguments can be made to show that the augmented version of random facet cannot take more than a quadratic number of steps when it is applied to the examples of Friedmann, Hansen, and Zwick.

6.3.2 Experimental Results

In order to test the effectiveness of the switching policies presented in this chapter experimentally, we have implemented them. This section gives a report of the results that we obtained using this implementation.

We began by testing our switching policies on the examples that have been used to prove super-polynomial lower bounds. Table 6.3.2 shows the number of iterations taken by the greedy switching policy and the augmented greedy switching policy on the family of examples given by Friedmann. It can clearly be seen that the greedy switching policy takes $9 \cdot 2^n - 9$ iterations to solve the example of size n, which matches the bound given by Friedmann in his paper. It can also be seen that the augmented greedy switching policy takes a linear number of steps to solve the examples. The number of iterations shown for the augmented version is the sum of the iterations spent by the greedy switching policy, and the iterations used by the procedure FixSnare.

n	$\lvert V\rvert$	Iterations	
		Original	Augmented
1	15	9	4
2	25	27	10
3	35	63	15
4	45	135	20
5	55	279	25
6	65	567	30
7	75	1143	35
8	85	2295	40
9	95	4599	45
10	105	9207	50

Table 6.1: The number of iterations taken by the greedy and augmented greedy switching policies on Friedmann's exponential time examples.

The reason why the augmented version takes a linear number of steps is that the initial strategy contains one switchable back edge for each of the bit gadgets in the game. Normally, greedy strategy improvement would not switch these edges because they do not have the largest appeal. However, these switchable back edges allow our augmented version to discover the snares that correspond to each of the bit gadgets. This causes the augmentation procedure to immediately run FixSnare for each of the bit gadgets in turn, until a strategy that represents a 1 for every bit is produced. After this, the greedy switching policy spends a few iterations to find a strategy that is optimal for every other vertex.

Table 6.2 shows the results that were obtained for the random facet switching policy on the examples of Friedmann, Hansen, and Zwick. The random facet switching policy uses a randomized rule to pick the next edge that should be switched, and therefore each run will take a different number of steps. We ran both the original and augmented switching policies nine times for each example size, and the table shows the smallest, largest, and average number of iterations that the switching policies took. The averages are rounded to the nearest integer.

It is obvious that the augmented version vastly outperforms the original

n	$\lvert V\rvert$	Original			Augmented		
		Min	Max	Average	Min	Max	Average
2	79	39	75	66	38	38	38
3	280	295	557	404	153	153	153
4	517	917	1383	1155	292	292	292
5	856	1729	3536	2644	495	495	495
6	1315	4170	7381	5205	774	774	774
7	1912	6374	21930	12299	1141	1141	1141
8	2185	10025	32867	16071	1304	1304	1304
9	2998	15265	85924	30070	1809	1809	1809
10	3331	30505	53560	42251	2010	2010	2010

Table 6.2: The number of iterations taken by random facet and augmented random facet on the examples of Friedmann, Hansen, and Zwick.

switching policy. It is interesting to note that the augmented version of the switching policy showed no variance in the number of iterations taken. This occurs due to an effect that is similar to the one we observed in Friedmann's examples. The augmentation procedure chooses to use FixSnare in the very first step, and control is not given to random facet until every bit gadget has been set to 1. There are still some edges left for random facet to switch at this point, however the order in which these edges are switched does not affect the running time of the switching policy. Therefore, the randomized choice made by random facet does not lead to a variance in the running time.

In addition to analysing the performance of our augmentation procedure on the super-polynomial time examples, we are also interested in the behaviour of this algorithm on more realistic inputs. Our implementation is capable of processing examples from the PGSolver collection [FL10], and the next examples are taken from that library. Parity games can be used as an algorithmic back end to μ-calculus model checking. PGSolver includes a family of examples that correspond to verification that an elevator system is fair. A fair elevator system is one that always eventually arrives at a floor from which a request has been made. PGSolver

	n	$\lvert V\rvert$	Iterations	
			Original	Augmented
FIFO	1	37	3	3
	2	145	5	5
	3	565	8	8
	4	2689	13	13
	5	15685	18	18
	6	108337	25	25
LIFO	1	37	3	3
	2	145	2	2
	3	589	2	2
	4	2833	2	2
	5	16357	2	2
	6	111457	2	2

Table 6.3: The number of iterations taken by the greedy and augmented greedy switching policies on the elevator verification examples.

provides examples that correspond to two possible elevator systems: one that uses a FIFO queue to store requests, and one that uses a LIFO queue. Clearly, the system that uses a FIFO queue will satisfy the property, and the system that uses a LIFO queue will not. The results of applying the greedy switching policy and the augmented greedy switching policy are shown in Table 6.3.

It can clearly be seen that the augmented version of the switching policy behaves in the same way as the original. The reason for this is that the augmented version never chooses to run FixSnare, because using the original switching policy gives a larger increase in valuation. One way of explaining this is that greedy strategy improvement is too fast on typical examples for the augmentation to make a difference. Since greedy strategy improvement takes only a handful of iterations to find an optimal strategy, the algorithm cannot find a significant number snares to run FixSnare on.

6.4 Concluding Remarks

In this chapter, we have studied switching policies for strategy improvement. For the strategy improvement algorithm that we have studied, we have shown that switchable edges can be classified into two distinct types. In doing so, we exposed a weakness of existing switching policies, because they do not take this classification into account. Moreover, we showed how switching policies can use this knowledge in future iterations by remembering snares, and we developed a snare based strategy improvement algorithm.

We developed an augmentation scheme, that allows oblivious switching policies to take advantage of snares, and we showed that the augmented versions of the greedy policy and the random facet policy terminate quickly on their super-polynomial time examples. In doing so, we demonstrated that snares are the key weakness that these examples exploit in order to force exponential time behaviour upon these switching policies. This is because, when the policies are modified so that they are aware of these structures, they no longer behave poorly.

There are a wide variety of questions that are raised by this work. Firstly, we have the structure of snares in parity and mean-payoff games. Proposition 6.12 implies that every optimal strategy for one of these games must use an escape from every snare that exists in the game. We therefore propose that a thorough and complete understanding of how snares arise in a game would be useful in order to devise a polynomial time algorithm that computes optimal strategies.

It is not currently clear how the snares in a game affect the difficulty of solving that game. It is not difficult, for example, to construct a game in which there an exponential number of Max snares: in a game in which every reward is positive there will be a snare for every connected subset of vertices. However, computing the zero-mean partition in a game where every reward is positive is obviously trivial. Clearly, the first challenge is to give a clear formulation of how the structure of the

snares in a given game affects the difficulty of solving it.

In our attempts to construct intelligent non-oblivious strategy improvement algorithms we have continually had problems with examples in which Max and Min snares overlap. By this we mean that the set of vertices that define the sub-games of the snares have a non empty intersection. We therefore think that studying how complex the overlapping of snares can be in a game may lead to further insight. There are reasons to believe that these overlappings cannot be totally arbitrary, since they arise from the structure of the game graph and the rewards assigned to the vertices.

We have presented a switching policy that passively records the snares that are discovered by a traditional switching policy, and then uses those snares when doing so is guaranteed to lead to a larger increase in valuation than using a given oblivious switching policy. While we have shown that this approach can clearly outperform traditional strategy improvement in some cases, it does not appear to immediately lead to a proof of polynomial time termination. It would be interesting to find an exponential time example for the augmented versions of the greedy policy or of the optimal policy. This may be significantly more difficult since it is no longer possible to trick strategy improvement into making slow progress by forcing it to repeatedly consider a small number of snares, which is the technique used by the examples of Friedman, and the examples of Friedmann, Hansen, and Zwick.

There is no inherent reason why strategy improvement algorithms should be obsessed with trying to increase valuations as much as possible in each iteration. Friedmann's exponential time example for the optimal switching policy demonstrates that doing so in no way guarantees that the algorithm will always make good progress. Our work uncovers an alternate objective that strategy improvement algorithms can use to measure their progress. Strategy improvement algorithms could actively try to discover the snares that exist in the game, or they could try and maintain consistency with as many snares as possible, for example. We believe that

there is much scope for intelligent snare based switching policies.

We have had some limited success in designing intelligent snare based strategy improvement algorithms for parity games. We have developed a switching policy which, when given a list of known snares in the game, either solves the game or finds a snare that is not in the list of known snares. This gives the rather weak result of a strategy improvement algorithm whose running time is polynomial in $|V|$ and k, where k is the number of Max snares that exist in the game. This is clearly unsatisfactory since we have already argued that k could be exponential in the number of vertices. However, this is one example of how snares can be applied to obtain new bounds for strategy improvement. As an aside, the techniques that we used to obtain this algorithm do not generalize to mean-payoff games. Finding a way to accomplish this task for mean-payoff games is an obvious starting point for designing intelligent snare based algorithms for this type of game.

Chapter 7

Conclusion

In this thesis, we have studied strategy iteration algorithms for infinite games and for Markov decision processes. In Chapter 4, we studied Lemke's algorithm and the Cottle-Dantzig algorithm, and showed how they behave when the input is an LCP that is derived from an infinite game. We also showed a family of examples upon which these algorithms take exponential time.

It is important to note that the lower bounds for Lemke's algorithm only hold for a specific choice of covering vector, and the lower bounds for the Cottle-Dantzig algorithm only hold for a specific ordering over distinguished vertices. We do not currently have a good understanding of how these parameters affect the running time of their respective algorithms. It is possible that the choice of these parameters could be as important as the choice of switching policy in strategy improvement. If this were the case, then perhaps our lower bounds could be seen in a similar light to the lower bounds given by Melekopoglou and Condon [MC94]: although they showed an exponential lower bound for a simple switching policy, we have seen that there is still potential for a switching policy that performs well.

In Chapter 5 we showed that Friedmann's exponential upper bounds can be extended to cover greedy strategy improvement for Markov decision processes. An important part of this result was showing how the snares used by Friedmann's

examples could be simulated using randomness. Currently, our concept of a snare only applies to mean-payoff games and parity games. It would be interesting to see if these concepts could be generalised to produce non-oblivious strategy improvement algorithms for Markov decision processes. Since Markov decision processes are already known to be solvable in polynomial time, perhaps this model would provide an easier starting point for proving good upper bounds on the running time of non-oblivious strategy improvement switching policies.

In Chapter 6 we laid the foundations of non-oblivious strategy improvement, and we provided a simple way of constructing a non-oblivious strategy improvement policy. Although we showed that our switching policies perform well in certain circumstances, we do not think that it will be easy to prove good upper bounds on the running time of these switching policies. Instead, we propose that research should be focused on the design of more advanced non-oblivious switching policies. In order to construct these, we will require a detailed understanding of snares, and how they arise in games. All of the concepts that we have introduced in this chapter can also be applied to the discrete strategy improvement algorithm for parity games given by Vöge and Jurdziński [VJ00], and we speculate that their algorithm may provide a better basis for the construction of non-oblivious switching policies. This is because their algorithm uses a discrete valuation, and it is likely to be easier to prove upper bounds using this valuation. Indeed, the non-oblivious switching policy for parity games that we mentioned in Section 6.4 makes use of a discrete valuation that was inspired by Vöge and Jurdziński's valuation.

Bibliography

[ADD00] R. B. Ash and C. A. Doléans-Dade. *Probability and Measure Theory*. Academic Press, 2000.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *The Annals of Mathematics*, 160(2):781–793, 2004.

[AM85] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the ACM*, 32(4):871–895, 1985.

[And09] D. Andersson. Extending Friedmann's lower bound to the Hoffman-Karp algorithm. Preprint, June 2009.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

[BV05] H. Björklund and S. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.

[BV07] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

[CDT09] X. Chen, X. Deng, and S. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56:14:1–14:57, May 2009.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[Chu89] S. J. Chung. NP-Completeness of the linear complementarity problem. *Journal of Optimization Theory and Applications*, 60(3):393–399, 1989.

[Chv83] V. Chvátal. *Linear Programming*. Freeman, 1983.

[Con92] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[Con93] A. Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[CPS92] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[Dan49] G. B. Dantzig. Programming of interdependent activities: II Mathematical model. *Econometrica*, 17(3/4):200–211, 1949.

[DC67] G. B. Dantzig and R. W. Cottle. Positive (semi-)definite programming. *Nonlinear Programming*, pages 55–73, 1967.

[d'E63] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[DGP06] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 71–78, New York, NY, USA, 2006. ACM.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, Washington, DC, USA, 1991. IEEE Computer Society Press.

[EJS93] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, 5th International Conference, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer-Verlag, 1993.

[Fat79] Y. Fathi. *Computational complexity of LCPs associated with positive definite symmetric matrices*, volume 17. Springer, Berlin / Heidelberg, 1979.

[Fea10a] J. Fearnley. Exponential lower bounds for policy iteration. In *37th International Colloquium on Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer Berlin / Heidelberg, 2010.

[Fea10b] J. Fearnley. Non-oblivious strategy improvement. In *16th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, 2010. To appear.

[FHZ10] O. Friedmann, T. Hansen, and U. Zwick. A subexponential lower bound for the random facet algorithm for parity games. Preprint, August 2010.

[FJS10] J. Fearnley, M. Jurdziński, and R. Savani. Linear complementarity algorithms for infinite games. In *SOFSEM '10: Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science*, pages 382–393, Berlin, Heidelberg, 2010. Springer-Verlag.

[FL10] O. Friedmann and M. Lange. Local strategy improvement for parity game solving. In *First International Symposium on Games, Automata, Logics and Formal Verification, GandALF*, 2010.

[Fri09] O. Friedmann. A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it. In *Logic in Computer Science*. IEEE, 2009.

[Fri10a] O. Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. Preprint, June 2010.

[Fri10b] O. Friedmann. Recursive solving of parity games requires exponential time. Preprint, July 2010.

[FV97] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

[GR05] B. Gärtner and L. Rüst. Simple stochastic games and P-matrix generalized linear complementarity problems. In *Fundamentals of Computation Theory*, volume 3623 of *LNCS*, pages 209–220. Springer, 2005.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

[HK66] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[How60] R. Howard. *Dynamic Programming and Markov Processes*. Technology Press and Wiley, 1960.

[JPZ06] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 117–123. ACM/SIAM, 2006.

[JS08] M. Jurdziński and R. Savani. A simple P-matrix linear complementarity problem for discounted games. In *Proceedings of Computability in Europe, Logic and Theory of Algorithms*, pages 283–293, 2008.

[Jur98] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[Jur00] M. Jurdziński. Small progress measures for solving parity games. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, 2000. Springer-Verlag.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm (Extended abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 475–482. ACM, 1992.

[Kar78] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, New York, NY, USA, 1984. ACM.

[Kha80] L. G. Khachiyan. Polynomial algorithms in linear programming. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 20:51–68, 1980.

[KL93] A. V. Karzanov and V. N. Lebedev. Cyclical games with prohibitions. *Mathematical Programming: Series A and B*, 60(3):277–293, 1993.

[KM70] V. Klee and G. J. Minty. How good is the simplex algorithm? *Inequalities III*, pages 159–175, 1970.

[Koz82] D. Kozen. Results on the propositional μ-calculus. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming*, pages 348–359, London, UK, 1982. Springer-Verlag.

[Lem65] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11(7):681–689, 1965.

[LH64] C. E. Lemke and J. T. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[LL69] T. M. Liggett and S. A. Lippman. Stochastic games with perfect information and time average payoff. *SIAM Review*, 11(4):604–607, 1969.

[Lud95] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117(1):151–155, 1995.

[Man60] A. S. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, 1960.

[MC94] M. Melekopoglou and A. Condon. On the complexity of the policy improvement algorithm for Markov decision processes. *ORSA Journal on Computing*, 6:188–192, 1994.

[McN93] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[Mos91] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.

[MS99] Y. Mansour and S. P. Singh. On the complexity of policy iteration. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 401–408. Morgan Kaufmann, 1999.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4–5):498–516, 1996.

[MTZ10] O. Madani, M. Thorup, and U. Zwick. Discounted deterministic Markov decision processes and discounted all-pairs shortest paths. *ACM Transactions on Algorithms*, 6(2):1–25, 2010.

[Mur74] K. G. Murty. Note on a Bard-type scheme for solving the complementarity problem. *Opsearch*, 11:123–130, 1974.

[Mur78] K. G. Murty. *Computational complexity of complementary pivot methods*, volume 7 of *Mathematical Programming Studies*. Springer Berlin Heidelberg, 1978.

[Mur88] K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann Verlag, 1988.

[Pap94] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[Pur95] A. Puri. *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, University of California, Berkeley, 1995.

[Put05] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. New York, NY, USA, 2005.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Sch07] S. Schewe. Solving parity games in big steps. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007.

[Sch08] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Computer Science Logic*, volume 5213 of *LNCS*, pages 369–384. Springer, 2008.

[Sha53] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095–1100, 1953.

[Sti95] C. Stirling. Local model checking games (Extended abstract). In *CONCUR'95: Concurrency Theory, 6th International Conference*, volume 962 of *LNCS*, pages 1–11. Springer-Verlag, 1995.

[SV07] O. Svensson and S. Vorobyov. Linear complementarity and P-matrices for stochastic games. In *Perspectives of Systems Informatics*, volume 4378 of *LNCS*, pages 409–423, Heidelberg, 2007. Springer.

[Tse90] P. Tseng. Solving H-horizon, stationary Markov decision problems in time proportional to log(H). *Operations Research Letters*, 9(5):287–297, 1990.

[VJ00] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In *Computer Aided Verification, 12th International Conference, CAV 2000, Proceedings*, volume 1855 of *LNCS*, pages 202–215, Chicago, IL, USA, 2000. Springer-Verlag.

[vM44] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[WD49] M. K. Wood and G. B. Dantzig. Programming of interdependent activities: I General discussion. *Econometrica*, 17(3/4):193–199, 1949.

[Wil88] K. Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20(1):69–81, 1988.

[Ye05] Y. Ye. A new complexity result on solving the Markov decision problem. *Mathematics of Operations Research*, 30:733–749, August 2005.

[Ye10] Y. Ye. The simplex method is strongly polynomial for the Markov decision problem with a fixed discount rate. Preprint, May 2010.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.

[ZP96] U. Zwick and M. S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.