Logical Methods in Computer Science
Volume 22, Issue 1, 2026, pp. 17:1–17:28
https://lmcs.episciences.org/

Submitted Apr. 08, 2024
Published Mar. 05, 2026

AN OBJECTIVE IMPROVEMENT APPROACH TO SOLVING DISCOUNTED PAYOFF GAMES [*]

DANIELE DELL'ERBA [a], ARTHUR DUMAS [b], AND SVEN SCHEWE [c]

[a] Middlesex University, London, United Kingdom
e-mail address: d.dellerba@mdx.ac.uk

[b] ENS Rennes, France
e-mail address: arthur.dumas@ens.rennes.fr

[c] University of Liverpool, United Kingdom
e-mail address: sven.schewe@liverpool.ac.uk

Abstract. While discounted payoff games and classic games that reduce to them, like parity and mean-payoff games, are symmetric, their solutions are not. We have taken a fresh view on the properties that optimal solutions need to have, and devised a novel way to converge to them, which is entirely symmetric. We achieve this by building a constraint system that uses every edge to define an inequation, and update the objective function by taking a single outgoing edge for each vertex into account. These edges loosely represent strategies of both players, where the objective function intuitively asks to make the inequation to these edges sharp. In fact, where they are not sharp, there is an 'error' represented by the difference between the two sides of the inequation, which is 0 where the inequation is sharp. Hence, the objective is to minimise the sum of these errors. For co-optimal strategies, and only for them, it can be achieved that all selected inequations are sharp or, equivalently, that the sum of these errors is zero. While no co-optimal strategies have been found, we step-wise improve the error by improving the solution for a given objective function or by improving the objective function for a given solution. This also challenges the gospel that methods for solving payoff games are either based on strategy improvement or on value iteration.

1. Introduction

We study turn-based zero sum games played between two players on directed graphs. The two players take turns to move a token along the vertices of a finite labelled graph with the goal to optimise their adversarial objectives.

Various classes of graph games are characterised by the objective of the players, for instance, in *parity games* the objective is to optimise the parity of the dominating colour occurring infinitely often, while in *discounted and mean-payoff games* the objective of the players is to minimise resp. maximise the discounted and limit-average sum of the colours.

Solving graph games is the central and most expensive step in many model checking [Koz83, EJS93, Wil01, dAHM01, AHK02, SF06a], satisfiability checking [Koz83, Var98,

Key words and phrases: Discounted Payoff Games, Payoff Games, Objective Improvement, Strategy Improvement.

[*] This work is based on [DDS23], which appeared in GandALF'23.

Wil01], and synthesis [Pit06, SF06b] algorithms. Progress in algorithms for solving graph games will, therefore, allow for the development of more efficient model checkers and contribute to bringing synthesis techniques to practice.

There is a hierarchy among the graph games mentioned earlier, with simple and well-known reductions from parity games to mean payoff games, from mean-payoff games to discounted payoff games, and from discounted payoff games to simple stochastic games such as the ones from [ZP96], while no reductions are known in the other direction. Therefore, one can solve instances of all these games by using an algorithm for stochastic games. All of these games are in UP and co-UP [Jur98], while no tractable algorithm is known.

Most research has focused on parity games: as the most special class of games, algorithms have the option to use the special structure of their problems, and they are most directly linked to the synthesis and verification problems mentioned earlier. Parity games have thus enjoyed a special status among graph games and the quest for efficient algorithms [EL86, EJ91, McN93, ZP96, BCJ$^+$97, Zie98, Obd03, BDM16, BDM18b, BDM18a] to solve them has been an active field of research during the last decades, which has received further boost with the arrival of quasi-polynomial techniques [JL17, FJdK$^+$19, LB20, LPSW20, DS22, CJK$^+$22, BDM$^+$24].

Interestingly, the only class of efficient techniques for solving parity games that does not (yet) have a quasi-polynomial approach is strategy improvement algorithms [Lud95, Pur95, VJ00, BV07, Sch08, Fea10, STV15], a class of algorithms closely related to the Simplex for linear programming, known to perform well in practice. Most of these algorithms reduce to mean [BV07, Sch08, STV15, BDM20] or discounted [Lud95, Pur95, FGO20, Koz21] payoff games.

With the exception of the case in which the fixed-point of discounted payoff games is explicitly computed [ZP96], all these algorithms share a disappointing feature: they are inherently non-symmetric approaches for solving an inherently symmetric problem. However, some of these approaches have a degree of symmetry. Recursive approaches treat even and odd colours symmetrically, one at a time, but they treat the two players very differently for a given colour. Symmetric strategy improvement [STV15] runs a strategy improvement algorithms for both players in parallel, using the intermediate results of each of them to inform the updates of the other, but at heart, these are still two intertwined strategy improvement algorithms that, individually, are not symmetric. This is due to the fact that applying strategy improvement itself symmetrically can lead to cycles [Con93].

The key contribution of this paper is to devise a new class of algorithms to solve discounted payoff games, which is entirely symmetric. Like strategy improvement algorithms, it seeks to find co-optimal strategies and improves strategies while they are not optimal. However, in order to do so, it does not distinguish between the strategies of the two players. This seems surprising, as maximising and minimising appear to pull in opposing directions.

Similarly to strategy improvement approaches, the new objective improvement approach turns the edges of a game into constraints (here called inequations) and minimises an objective function. However, while strategy improvement algorithms take only the edges in the strategy of one player (and all edges of the other player) into account and then finds the optimal response by solving the resulting one-player game, objective improvement always takes all edges into account. The strategies under consideration then form a subset of the inequations, and the goal would be to make them *sharp* (i.e., strict, tight, satisfied as equations), which only works when both strategies are optimal. When they are not, then there is some *offset* for each of the inequations, and the objective is to reduce this offset in every improvement step. This treats the strategies of both players completely symmetrically.

Organisation of the Paper. The paper is organised as follows. After the preliminaries in Section 2, we start by outlining our method and using a simple game to explain it in Section 3. We then formally introduce our objective improvement algorithm in Section 4, keeping the question of how to choose better strategies abstract. Section 5 discusses how to find better strategies and Section 6 provides an experimental evaluation of the algorithm. We finally wrap up with a discussion of our results in Section 7.

2. Preliminaries

A *discounted payoff game* (DPG) is a tuple $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, where $V = V_{\min} \cup V_{\max}$ are the vertices of the game, partitioned into two disjoint sets $V_{\min}$ and $V_{\max}$, such that the pair (V, E) is a finite directed graph without sinks. The vertices in $V_{\max}$ (*resp*, $V_{\min}$) are controlled by Player Max or maximiser (*resp*, Player Min or minimiser) and $E \subseteq V \times V$ is the edge relation. Every edge has a weight represented by the function $w : E \to \mathbb{R}$, and a *discount factor* represented by the function $\lambda : E \to [0, 1)$. When the discount factor is uniform, i.e., the same for every edge, it is represented by a constant value $\lambda \in [0, 1)$. For ease of notation, we write w_e and λ_e instead of $w(e)$ and $\lambda(e)$. A *play* on $\mathcal{G}$ from a vertex v is an infinite path, which can be represented as a sequence of edges $\rho = e_0e_1e_2\ldots$ such that, for every $i \in \mathbb{N}$, $e_i = (v_i, v_{i+1}) \in E$ and $v_0 = v$. By ρ_i we refer to the i-th edge of the play. The *outcome* of a discounted game $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ for a play ρ is $\mathsf{out}(\rho) = \sum_{i=0}^{\infty} w_{e_i} \prod_{j=0}^{i-1} \lambda_{e_j}$. For games with a constant discount factor, this simplifies into $\mathsf{out}(\rho) = \sum_{i=0}^{\infty} w_{e_i} \lambda^i$.

A *positional strategy* for Max is a function $\sigma_{\max} : V_{\max} \to V$ that maps each Max vertex to a vertex according to the set of edges, i.e., $(v, \sigma_{\max}(v)) \in E$. Positional Min strategies are defined accordingly, and we call the set of positional Min and Max strategies $\Sigma_{\min}$ and $\Sigma_{\max}$, respectively.

A pair of strategies $\sigma_{\min}$ and $\sigma_{\max}$, one for each player Min and Max, defines a unique play $\rho(v, \sigma_{\min}, \sigma_{\max})$ from each vertex $v \in V$. Discounted payoff games are positionally determined [ZP96]:

$$\max_{\sigma_{\max}\in\Sigma_{\max}} \min_{\sigma_{\min}\in\Sigma_{\min}} \mathsf{out}(\rho(v, \sigma_{\min}, \sigma_{\max})) = \min_{\sigma_{\min}\in\Sigma_{\min}} \max_{\sigma_{\max}\in\Sigma_{\max}} \mathsf{out}(\rho(v, \sigma_{\min}, \sigma_{\max}))$$

holds for all $v \in V$, and neither the strategy, nor the value computed from the strategy, changes when we allow more powerful classes of strategies that allow for using memory and/or randomisation for one or both players.

The *value* of a minimiser strategy σ, denoted by $\mathsf{minival}_\sigma : V \to \mathbb{R}$, is defined as

$$\mathsf{minival}_\sigma : v \mapsto \max_{\sigma_{\max}\in\Sigma_{\max}} \mathsf{out}(\rho(v, \sigma, \sigma_{\max})) \ ,$$

The *value* of a maximiser strategy σ, denoted by $\mathsf{maxival}_\sigma : V \to \mathbb{R}$, is defined as

$$\mathsf{maxival}_\sigma : v \mapsto \min_{\sigma_{\min}\in\Sigma_{\min}} \mathsf{out}(\rho(v, \sigma_{\min}, \sigma)) \ .$$

The resulting *value of* $\mathcal{G}$, denoted by $\mathsf{val}_\mathcal{G} : V \to \mathbb{R}$, is defined as

$$\mathsf{val}_\mathcal{G} : v \mapsto \max_{\sigma_{\max}\in\Sigma_{\max}} \min_{\sigma_{\min}\in\Sigma_{\min}} \mathsf{out}(\rho(v, \sigma_{\min}, \sigma_{\max})) \ .$$

We have

$$\mathsf{val}_G(v) = \max_{\sigma_{\max}\in\Sigma_{\max}} \mathsf{maxival}_{\sigma_{\max}}(v) = \min_{\sigma_{\min}\in\Sigma_{\min}} \mathsf{minival}_{\sigma_{\min}}(v)$$

for all $v \in V$. A positional maximiser (resp. minimiser) strategy σ is said to be *optimal* if, and only if, $\mathsf{val}_{\mathcal{G}}(v) = w_{(v,\sigma(v))} + \lambda_{(v,\sigma(v))}\mathsf{val}_{\mathcal{G}}(\sigma(v))$ holds for all maximiser (resp. minimiser) vertices. Such optimal strategies always exist.

The valuation $\mathsf{val}_{\mathcal{G}}$ is the sole valuation that satisfies the fixed point equations

$$\mathsf{val}_{\mathcal{G}}(v) = \max_{(v,v')\in E} w_{(v,v')} + \lambda_{(v,v')}\mathsf{val}_{\mathcal{G}}(v')$$

for all maximiser vertices $v \in V_{\max}$ and

$$\mathsf{val}_{\mathcal{G}}(v) = \min_{(v,v')\in E} w_{(v,v')} + \lambda_{(v,v')}\mathsf{val}_{\mathcal{G}}(v')$$

for all minimiser vertices $v \in V_{\min}$.

Likewise, we define the value of a pair of strategies $\sigma_{\min}$ and $\sigma_{\max}$, denoted $\mathsf{val}_{\sigma_{\min},\sigma_{\max}} : V \to \mathbb{R}$, as

$$\mathsf{val}_{\sigma_{\min},\sigma_{\max}} : v \mapsto \mathsf{out}(\rho(v, \sigma_{\min}, \sigma_{\max})) \ .$$

As we treat both players symmetrically in this paper, we define a *pair of strategies* $\sigma : V \mapsto V$ whose restriction to $V_{\min}$ and $V_{\max}$ are a minimiser strategy $\sigma_{\min}$ and a maximiser strategy $\sigma_{\max}$, respectively. We then write $\rho(v, \sigma)$ instead of $\rho(v, \sigma_{\min}, \sigma_{\max})$ and val_σ instead of $\mathsf{val}_{\sigma_{\min},\sigma_{\max}}$. To ease the reading, when the strategy is clear from the context, we shall drop the subscript.

If both of these strategies are optimal, we call σ a joint *co-optimal* strategy. This is the case if, and only if, $\mathsf{val}_{\mathcal{G}} = \mathsf{val}_\sigma$ holds.

We define a *solution* to a discounted payoff game $\mathcal{G}$ to be a valuation val for the vertices such that, for every edge $e = (v, v')$, it holds that[1]

- $\mathsf{val}(v) \leq w_e + \lambda_e\mathsf{val}(v')$ if v is a minimiser vertex and
- $\mathsf{val}(v) \geq w_e + \lambda_e\mathsf{val}(v')$ if v is a maximiser vertex.

For co-optimal strategies σ, val_σ that is equal to $\mathsf{val}_{\mathcal{G}}$ is clearly a solution, as all of these inequations are satisfied, while for a pair of strategies σ that is not co-optimal, val_σ is not a solution[2].

Note that we are interested in the *value* of each vertex, not merely if the value is greater or equal than a given threshold value.

2.1. **Simplex method.**

A common way to compute $\mathsf{val}_{\mathcal{G}}$ is to create a sequence of linear programming instances from $\mathcal{G}$. In this paragraph, we provide the background for solving a linear programming instance and outline the Simplex method that is the standard algorithm used to find a solution.

A linear programming problem requires to find a (solution) vector $x \in \mathbb{R}^n$ that minimises (or maximises) $c \cdot x$, where $c \in \mathbb{R}^n$ is a vector of constants, such that $A \cdot x \leq b$ (or $\geq b$), where $A \in \mathbb{R}^{m\times n}$ is a coefficient matrix and $b \in \mathbb{R}^m$ is a vector of constants. Given a game $\mathcal{G}$, we can iteratively define an objective function represented by the vector c, the input matrix A,

[1]These are the constraints represented in H in Section 4.

[2]For every joint strategy σ of both players and every vertex $v \in V$, we have $\mathsf{val}_\sigma(v) = w_{(v,\sigma(v))} + \lambda_{(v,\sigma(v))}\mathsf{val}_\sigma(\sigma(v))$ by definition. Thus, if val_σ is a solution, then $\mathsf{val}_{\mathcal{G}}$ also satisfies the fixed point equations that define $\mathsf{val}_{\mathcal{G}}$, and we have that $\mathsf{val}_\sigma = \mathsf{val}_{\mathcal{G}}$ holds and that σ is co-optimal.

and the vector b, such that, in the sequence of linear programming instances, the solution x converges to $\mathsf{val}_{\mathcal{G}}$. In this translation from $\mathcal{G}$ to the linear programming instance, the matrix A is composed of m rows and n columns where m is the number of edges and n the number of vertices in $\mathcal{G}$. Since the constrains in A represent the edges, that are binary relations, only two columns in each row contain a nonzero value (with the exception of self-loops that have only one nonzero coefficient). The vector b contains the weight of the edges. Hence, every row i can be expressed as $a_{ij} \cdot x_j + a_{ik} \cdot x_k \geq b_i$, if $e = (j, k)$ is an edge of $\mathcal{G}$ and j is a maximiser vertex (otherwise the constraint is $\leq b_i$), with $a_{ij} = 1$, $a_{ik} = -\lambda_e$, and $b_i = w_e$. Finally, c is the unit vector with a positive sign for the maximiser vertices and negative sign for the minimiser, and the objective function maximises the sum of the values of x.

As we use no details of any implementation of a simplex algorithm, we only rehash the principle way it operates here. From a graphical point of view, the set of constraints, i.e., the rows of A and b, define a region called *polytope* (or: simplex; hence the name of the approach) in the space $\mathbb{R}^n$. Any point that lies within the polytope (including its surface) is a solution.

Broadly speaking, the simplex algorithm traverses the corner of the polytope so that the value the linear objective function takes for the corner is improving – in our case decreasing as we minimise. More precisely, it picks subsets of n inequations that are made sharp: when read as equations, they define a valuation in $\mathbb{R}^n$ that satisfies all m inequations; they are bases and define the corners of the simplex.

For this, the simplex algorithm operates in two phases. In a first phase it finds *any* basis that defines such a corner. We will not discuss this phase as it is not relevant for this article[3].

In the second phase, the simplex algorithm essentially updates the basis to a *neighbouring* one, meaning that only one inequation is removed from and one inequation is entered into the set of sharp inequations. When making this pick, the simplex algorithm only allows for such updates that are (1) still bases (i.e. their rows in A must be linearly independent to that they define a point in $\mathbb{R}^n$); (2) the point in $\mathbb{R}^n$ they define satisfies all m inequations, and (3) the value of the objective function improves.

For non-optimal solutions (w.r.t. the objective function), this is always possible, though in some cases the improvement in (3) is not strict. Even in those cases, there is still a series of changes that all satisfy (1-3) to an optimal solution. Those cases are called degenerate, that is, when the current solution satisfies more inequations than necessary and, although there exists a neighbouring solution leading to a strict improvement, picking it requires a basis change. While we do not discuss how they are treated in simplex algorithms, we point out that the problems and their solutions are similar to what we describe in Sections 5.2 and 5.3, respectively.

3. Outline and Motivation Example

We start by considering the simple discounted payoff game of Figure 1, assuming that it has some uniform discount factor $\lambda \in [0, 1)$. In this game, the minimiser (who owns the right vertex, depicted as a square) has only one option: she always has to use the self-loop, which earns her an immediate reward of 1. The overall reward the minimiser reaps for a play that starts in her vertex is therefore $1 + \lambda + \lambda^2 + \ldots = \frac{1}{1-\lambda}$.

[3]It is also the case that, in all but the first call to a linear program, we already have a basis to start from.

The maximiser (who owns the left vertex, b, marked by a circle) can choose to either use the self-loop, or to move to the minimiser vertex (marked by a square), both yielding no immediate reward.

If the maximiser decides to stay forever in his vertex (using the self-loop), his overall reward in the play that starts at (and, due to his choice, always stays in) his vertex, is 0. If he decides to move on to the minimiser vertex the n^{th}-time, then the reward is $\frac{\lambda^n}{1-\lambda}$.

The optimal decision of the maximiser is, therefore, to move on the first time, which yields the maximal reward of $\frac{\lambda}{1-\lambda}$. Every vertex v has some outgoing edge(s) $e = (v, v')$ where $\mathsf{val}_{\mathcal{G}}(v) = w_e + \lambda_e \mathsf{val}_{\mathcal{G}}(v')$ holds [ZP96]; these edges correspond to the optimal decisions for the respective player.

For our running example game of Figure 1 with a fixed discount factor $\lambda \in [0, 1)$, the constraints that a solution must satisfy are:

(1)	$\mathsf{val}(a) \leq 1 + \lambda\mathsf{val}(a)$	for the self-loop of the minimiser vertex;
(2)	$\mathsf{val}(b) \geq \lambda\mathsf{val}(b)$	for the self-loop of the maximiser vertex; and
(3)	$\mathsf{val}(b) \geq \lambda\mathsf{val}(a)$	for the transition from the maximiser to the minimiser vertex.

The unique solution that satisfies these inequations and produces a sharp inequation (i.e. satisfied as equation) for some outgoing edge of each vertex assigns $\mathsf{val}(a) = \frac{1}{1-\lambda}$ and $\mathsf{val}(b) = \frac{\lambda}{1-\lambda}$. This solution also defines the optimal strategies of the players (to stay for the minimiser, and to move on for the maximiser).

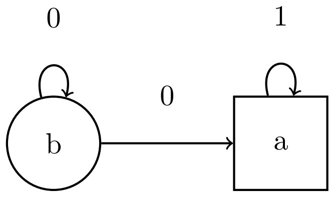

Figure 1: A discounted Payoff Game. Maximiser vertices are depicted as a circle, minimizer ones as a square.

Solving a discounted payoff game means finding this solution and/or these strategies.

We discuss a symmetric approach to find this unique valuation. Our approach adjusts linear programming in a natural way that treats both players symmetrically: we maintain the set of inequations for the complete time, while approximating the goal of "one equality per vertex" by the objective function. To do that, we initially fix an *arbitrary* outgoing edge for every vertex (a strategy), and minimise the sum of the distances between the left and right sides of the inequations defined by these edges, which we call the *offset* of this edge. This means, for an edge $e = (v, v')$, to minimise the difference between $\mathsf{val}(v)$ (left side of the inequation) and $w_e + \lambda_e\mathsf{val}(v')$ (right side).

To make this clear, we consider again the example of Figure 1 and use both self-loops as the strategies for the players fixed at the beginning in our running example. The offset for the selected outgoing edge of the minimiser vertex a is equal to $1 - (1-\lambda)\mathsf{val}(a)$, while the offset for the selected outgoing edge of the maximiser vertex b is equal to $(1-\lambda)\mathsf{val}(b)$. The resulting overall objective consists, therefore, in minimising the value $1 - (1-\lambda)\mathsf{val}(a) + (1-\lambda)\mathsf{val}(b)$.

This term is always non-negative, since it corresponds to the sum of the edges' contributions that are all non-negative. Moreover, when only optimal strategies are selected to form this objective function, the value 0 can be taken, and where it is taken, it defines the correct valuation of the game.

As the maximiser's choice to take the self-loop is not optimal, the resulting objective function the strategies define, that is $1 - (1-\lambda)\mathsf{val}(a) + (1-\lambda)\mathsf{val}(b)$, cannot reach 0. But let us take a look at what an optimal solution w.r.t. this objective function looks like.

Optimal solutions can be taken from the corners of the polytope defined by the inequations, also known as its simplex. In this case, the optimal solution (w.r.t. this initial objective function) is defined by making inequations (1) and (3) sharp: this provides the values $\mathsf{val}(a) = \frac{1}{1-\lambda}$ and $\mathsf{val}(b) = \frac{\lambda}{1-\lambda}$; the objective function takes the value λ at this point.

For comparison, in the other corner of the simplex, defined by making inequations (2) and (3) sharp, we obtain the values $\mathsf{val}(a) = \mathsf{val}(b) = 0$; the objective function takes the value 1 at this point. Finally, if we consider the last combination, making (1) and (2) sharp, this provides the values $\mathsf{val}(a) = \frac{1}{1-\lambda}$ and $\mathsf{val}(b) = 0$, so that inequation (3) is not satisfied; this is therefore not a corner of the simplex.

Thus, in this toy example, while selecting the wrong edge cannot result in the objective function taking the value 0, we still found the optimal solution. In general, we might need to update the objective function. To update the objective function, we change the outgoing edges of some (or all) vertices so that the overall value of the objective function goes down. Note that this can be done not only when the linear program returns an optimal solution but also during its computation. For example, when using a simplex method, updating the objective function can be used as an alternative pivoting rule at any point during the traversal of the simplex.

In general, the valuation returned as solution is computed by using objective functions based on strategies that are not necessarily optimal. We now adjust the example so that choosing both self-loops is not only non-optimal, but also does not define the correct valuation of the DPG. To this end, we use different discount factors[4] for the game of Figure 1: we choose $\frac{1}{3}$ for the self-loop of the maximiser vertex and $\frac{2}{3}$ for the other two transitions, so that the three resulting inequations are (1) $\mathsf{val}(a) \leq 1 + \frac{2}{3}\mathsf{val}(a)$; (2) $\mathsf{val}(b) \geq \frac{1}{3}\mathsf{val}(b)$; and (3) $\mathsf{val}(b) \geq \frac{2}{3}\mathsf{val}(a)$. Choosing both self loops results in the objective function to minimise $1 - \frac{1}{3}\mathsf{val}(a) + \frac{2}{3}\mathsf{val}(b)$.

Making the adjusted inequations (2) and (3) sharp still results in the values $\mathsf{val}(a) = \mathsf{val}(b) = 0$, and the objective function still takes the value of 1. While making inequations (1) and (3) sharp provides the values $\mathsf{val}(a) = 3$ and $\mathsf{val}(b) = 2$; the objective function takes the value $\frac{4}{3}$ at this point. Finally, if we consider the last combination, making (1) and (2) sharp still conflicts with inequation (3).

Thus, $\mathsf{val}(a) = \mathsf{val}(b) = 0$ would be the optimal solution for the given objective function, which is not the valuation of the game. We will then update the candidate strategies so that the sum of the offsets goes down.

3.1. **Comparison with strategy improvement.**

The closest relatives to our new approach are strategy improvement algorithms. Classic strategy improvement approaches solve the problem of finding the valuation of a game (and usually also co-optimal strategies) by (1) fixing a strategy for one of the players (we assume w.l.o.g. that this is the maximiser), (2) finding a valuation function for the one player game that results from fixing this strategy (often together with an optimal counter strategy for their opponent), and (3) updating the strategy of the maximiser by applying local improvements. This is repeated until no local improvements are available, which entails that the constraint system is satisfied.

[4]Note that we can also replace the transitions with a smaller discount factor by multiple transitions with a larger discount factor. This would allow for keeping the discount factor uniform, but needlessly complicate the discussion and inflate the size of the example.

Algorithm 1: Strategy Improvement

```
   input  : A discounted payoff game
            $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$
   output : The valuation val of $\mathcal{G}$
 1 $f \leftarrow$ ObjectiveFunction($\mathcal{G}$)
 2 $\sigma \leftarrow$ ChooseInitialStrategy($\mathcal{G}$)
 3 while true do
 4     $H \leftarrow$ Inequations($\mathcal{G}, \sigma$)
 5     val $\leftarrow$ LinearProgramming($H, f$)
 6     if $\sigma$ is optimal then
           return val
       end
 7     $\sigma \leftarrow$ ChooseBetterStrategy($\mathcal{G}$, val, $\sigma$)
   end
```

These steps can be identified in Algorithm 1. The objective function is fixed at the beginning (Line 1) and corresponds to the sum of the values of all the vertices. The algorithm simply maximises it[5] by searching for a solution that maximises the value of all vertices. (1) The strategy of one of the two players (for convention the maximiser – if we want to instead fix the strategy of the minimiser, we would have to minimise the sum of the values for all vertices instead) is initialised at Line 2. (2) Before the computation of the valuation at Line 5, the set of constraints is updated accordingly to the selected strategy of the maximiser (Line 4). At this point, if the strategy is optimal w.r.t. the current valuation, then the algorithm ends. Otherwise, (3) there exists an improvement, which is applied at Line 7.

For Step (2) of this approach, we can use linear programming, which does invite a comparison with our technique. The linear program for solving Step (2) would not use all inequations: it would, instead, replace the inequations defined by the currently selected edges of the maximiser by equations, while dropping the inequations defined by the other maximiser transitions. The objective function would then be to maximise the values of all vertices while still complying with the remaining (in)equations.

Thus, in our novel symmetric approach, the constraints remain while the objective is updated; in strategy improvement, instead, the objective remains while the constraints are updated. Moreover, the players and their strategies are treated quite differently in strategy improvement algorithms: while the candidate strategy of the maximiser results in making the inequations of the selected edges sharp (and dropping all other inequations of maximiser edges), the optimal counter strategy is found by maximising the objective. This is again in contrast to our novel symmetric approach, which treats both players equally.

A small further difference is in the valuations that can be taken: the valuations that strategy improvement algorithms can take are the valuations of strategies, while the valuations our objective improvement algorithm can take on the way are the corners of the simplex defined by the inequations. Except for the only intersection point between the two (the

[5]The optimal minimiser strategy chooses edge that minimises the value for every minimiser vertex. In the linear program, this is revlected by an inequeation for every outgoing edge, like the inequation $\mathsf{val}(a) \leq 1 + \lambda\mathsf{val}(a)$ from the running example, and then maximises the value, which forces the selected value to be the minimal value such that one such inequation is sharp.

	Objective Improvement	**Strategy Improvement**
players	symmetric treatment	asymmetric treatment
constraints	remain the same: one inequation per edge	change: one inequation for each edge defined by the current strategy for the strategy player, one inequation for every edge of their opponent
objective	minimise errors for selected edges	maximise values
update	objective: one edge for each vertex	strategy: one edge for each vertex of the strategy player
valuations	corners of simplex	defined by strategies

Table 1: A comparison of the novel objective improvement with classic strategy improvement.

valuation of the game), these corners of the simplex do not relate to the value of strategies. Table 1 summarises these observations.

4. General Objective Improvement

In this section, we present the approach outlined in the previous section more formally, while keeping the most complex step – updating the candidate strategy to one which is *better* in that it defines an optimisation function that can take a smaller value – abstract. (We turn back to the question of how to find better strategies in Section 5.) This allows for discussing the principal properties more clearly.

```
Algorithm 2: Objective Improvement
   input  : A discounted payoff game
            $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$
   output : The valuation val of $\mathcal{G}$
1  $H \leftarrow$ Inequations($\mathcal{G}$)
2  $\sigma \leftarrow$ ChooseInitialStrategies($\mathcal{G}$)
3  while true do
4      $f_\sigma \leftarrow$ ObjectiveFunction($\mathcal{G}, \sigma$)
5      val $\leftarrow$ LinearProgramming($H, f_\sigma$)
6      if $f_\sigma$(val) $= 0$ then
           return val
       end
7      $\sigma \leftarrow$ ChooseBetterStrategies($\mathcal{G}$, val, $\sigma$)
   end
```

A general outline of our *objective improvement* approach is reported in Algorithm 2. Before describing the procedures called by the algorithm, we first outline the principle.

When running on a discounted payoff game $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, the algorithm uses a set of inequations defined by the edges of the game and the owner of the source of each edge. This set of inequations denoted by H contains one inequation for each edge and (different to strategy improvement approaches whose set of inequations is a subset of H) H never changes.

The inequations from H are computed by a function called Inequations (Line 1) that, given the discounted game $\mathcal{G}$, returns the set made up of one inequation per edge $e = (v, v') \in E$, defined as follows:

$$I_e = \begin{cases} \mathsf{val}(v) \geq w_e + \lambda_e \mathsf{val}(v') & \text{if } v \in V_{\max}, \\ \mathsf{val}(v) \leq w_e + \lambda_e \mathsf{val}(v') & \text{otherwise.} \end{cases}$$

The set $H = \{I_e \mid e \in E\}$ is defined as the set of all inequations for the edges of the game.

The algorithm also handles strategies for both players, treated as a single strategy σ. They are initialised (for example randomly) by the function ChooseInitialStrategies at Line 2.

This joint strategy is used to define an objective function f_σ by calling function ObjectiveFunction at Line 4, whose value on an evaluation val is: $f_\sigma(\mathsf{val}) = \sum_{v \in V} f_\sigma(\mathsf{val}, v)$ with the following objective function components:

$$f_\sigma(\mathsf{val}, v) = \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))$$

where the offset of an edge (v, v') for a valuation is defined as follows:

$$\mathsf{offset}(\mathsf{val}, (v, v')) = \begin{cases} \mathsf{val}(v) - (w_{(v,v')} + \lambda_{(v,v')} \mathsf{val}(v')) & \text{if } v \in V_{\max}, \\ (w_{(v,v')} + \lambda_{(v,v')} \mathsf{val}(v')) - \mathsf{val}(v) & \text{otherwise.} \end{cases}$$

This objective function f_σ is given to a linear programming algorithm, alongside with the inequations set H. We underline that, due to the inequation $I_{(v,v')}$, the value of $\mathsf{offset}(\mathsf{val}, (v, v'))$ is non-negative for all $(v, v') \in E$ in any solution val (optimal or not) that satisfies the system of inequations H.

Observation 4.1. At Line 6 of Algorithm 2, the value of $f_\sigma(\mathsf{val})$ is non-negative.

We put a further restriction on val, returned by the call LinearProgramming, in that we require it to be the solution to a *basis* $\mathbf{b}$ in H. Such a basis consists of $|V|$ inequations that are satisfied sharply (again, as equations), such that, according to the definition of basis, these $|V|$ equations uniquely define the values of all vertices. We refer to this unique set of values as *the evaluation of* $\mathbf{b}$, denote by $\mathsf{val}_\mathbf{b}$. Note that $\mathsf{val}_\mathbf{b}$ is in particular a solution of H, so that all $|E|$ inequations are satisfied, condition that in general does not holds.

The call LinearProgramming(H, f_σ) to some linear programming algorithm returns a solution val of the vertices that minimises f_σ while satisfying H (Line 5); for convenience, we require this solution to also be the evaluation $\mathsf{val}_\mathbf{b}$ for some basis $\mathbf{b}$ of H. (Note that the simplex algorithm, for example, only uses solutions of this form in every step.) We call this solution a *solution associated to* σ.

We say that a solution val *defines* strategies of both players if, for every vertex $v \in V$, the inequation of (at least) one of the outgoing edges of v is sharp. These are the strategies defined by using, for every vertex $v \in V$, an outgoing edge for which the inequation is sharp. Note that there can be more than one of these inequations for some of the vertices.

Observation 4.2. If, for a solution val of H, $f_\sigma(\mathsf{val}) = 0$ holds, then, for every vertex $v \in V$, the inequation $I_{(v,\sigma(v))}$ for the edge $(v, \sigma(v))$ is sharp and val therefore defines strategies for both players – those defined by σ, for example.

Instead of checking whether val defines the strategies of both players, we can use, alternatively, $f_\sigma(\mathsf{val}) = 0$ as a termination condition, as shown at Line 6 in Algorithm 2, since in this case σ must define co-optimal strategies.

Theorem 4.3. *If σ describes co-optimal strategies, then $f_\sigma(\mathsf{val}) = 0$ holds at Line 6 of Algorithm 2. If val (from Line 5 of Algorithm 2) defines joint strategies σ' for both players, then σ' is co-optimal and val is the valuation of $\mathcal{G}$.*

Proof. The valuation $\mathsf{val} = \mathsf{val}_\mathcal{G}$ of the game is the unique solution of H for which, for all vertices v, the inequation to (at least) one of the outgoing edges of v is sharp. This case implies that the edges for which they are sharp describe co-optimal strategies. This solution of H is thus the only one that *defines* strategies for both players, which shows the second claim. The uniqueness of the solution is a consequence of the sharpness: while inequalities define the solution region, equations select exactly one.

Moreover, if σ describes co-optimal strategies, then $f_\sigma(\mathsf{val}) = 0$ holds for $\mathsf{val} = \mathsf{val}_\mathcal{G}$ (and for this valuation only), which establishes the first claim. □

The theorem above ensures that, in case the condition at Line 6 holds, the algorithm terminates and provides the value of the game that then allows us to infer optimal strategies of both players. Otherwise, we have to improve the objective function and make another iteration of the while loop. At Line 7, $\mathsf{ChooseBetterStrategies}$ can be any procedure that, for $f_\sigma(\mathsf{val}) \neq 0$, provides a joint strategy σ' *better* than σ as defined below.

Definition 4.4 (Better Strategy)**.** A joint strategy σ' for both players is *better* than a joint strategy σ if, and only if,

$$\min_{\mathsf{val}'} f_{\sigma'}(\mathsf{val}') < \min_{\mathsf{val}} f_\sigma(\mathsf{val})$$

Given a game $\mathcal{G}$, a strategy σ' and the corresponding objective function $f_{\sigma'}$ computed by $\mathsf{ObjectiveFunction}(\mathcal{G}, \sigma')$, the call $\mathsf{LinearProgramming}(H, f_{\sigma'})$ computes the minimal value of $f_{\sigma'}$. If this value is strictly lower than the minimal value for f_σ, computed by $\mathsf{LinearProgramming}(H, f_\sigma)$, then σ' is better than σ.

While we discuss how to implement this key function in the next section, we observe here that the algorithm terminates with a correct result with any implementation that chooses a better objective function in each round: correctness is due to it only terminating when val *defines* strategies for both players, which implies (cf. Theorem 4.3) that val is the valuation of $\mathcal{G}$ ($\mathsf{val} = \mathsf{val}_\mathcal{G}$) and all strategies defined by val are co-optimal. Termination is obtained by a finite number of positional strategies: by Observation 4.1, the value of the objective function of all of them is non-negative, while the objective function of an optimal solution to co-optimal strategies is 0 (cf. Theorem 4.3), which meets the termination condition of Line 6 (cf. Observation 4.2).

Corollary 4.5. *Algorithm 2 always terminates with the correct value.*

5. Choosing Better Strategies

In this section, we discuss sufficient criteria for finding a procedure that efficiently implements $\mathsf{ChooseBetterStrategies}$. For this, we make four observations described in the next subsections:

(1) All local improvements can be applied. A strategy σ' is a local improvement to a strategy σ if $f_{\sigma'}(\mathsf{val}) < f_\sigma(\mathsf{val})$ holds for the current solution val (Section 5.1).

(2) If the current solution val does not *define* a pair of strategies σ for both players and has no local improvements, then a better strategy σ' can be found applying only switches from and to edges that already have offset 0 (Section 5.2).
(3) The improvement mentioned in the previous point can be found for special games – the sharp and improving games defined in Section 5.3 – by trying a single-edge switch.
(4) Games can almost surely be made sharp and improving by adding random noise that retains optimal strategies (Section 5.4).

Together, these four points provide efficient means for finding increasingly better strategies, and thus to find the co-optimal strategies and the valuation of the discounted payoff game.

As a small side observation, when using a simplex-based technique to implement $\mathsf{LinearProgramming}$ at Line 5 of Algorithm 2, then the pivoting of the objective function from point (1) and the pivoting of the basis can be mixed (this will be discussed in Section 5.5).

5.1. **Local Improvements.**

The simplest and most common case of creating better strategies σ' from a solution for the objective f_σ for a strategy σ is to consider *local improvements*. Much like local improvements in strategy iteration approaches, local improvements consider, for each vertex v, a successor $v' \neq \sigma(v)$, such that $\mathsf{offset}(\mathsf{val}, (v, v')) < \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))$ for the current solution val, which is optimal for the objective function f_σ.

To be more general, our approach does not necessarily require one to select only local improvements, but it can work with global improvements, though we cannot see any practical use of choosing differently. For example, if we treat the function as a global improvement approach, we can update the value of a vertex v so that it increases by 1 and update the value of another vertex v' so that it decreases by 2. The overall value of the function will decrease even if locally some components increased their value. Interestingly, this cannot be done with a strategy improvement approach, as it requires one to always locally improve the value of each vertex when updating.

Lemma 5.1. *If* val *is an optimal solution for the linear programming problem at Line 5 of Algorithm 2 and* $f_{\sigma'}(\mathsf{val}) < f_\sigma(\mathsf{val})$*, then* σ' *is better than* σ*.*

Proof. The solution val is, being an optimal solution for the objective f_σ, a solution to the system of inequations H. For a solution val' that is optimal for $f_{\sigma'}$, we thus have $f_{\sigma'}(\mathsf{val}') \leq f_{\sigma'}(\mathsf{val}) < f_\sigma(\mathsf{val})$, which implies that σ' is better than σ according to Definition 4.4 of a better strategy. □

This allows us to identify local improvements cheaply.

Corollary 5.2. *If* val *is an optimal solution for the linear programming problem at Line 5 of Algorithm 2, there is an edge* (v, v') *such that* $\mathsf{offset}(\mathsf{val}, (v, v')) < \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))$*, and* σ' *is a strategy for both players s.t.* $\mathsf{offset}(\mathsf{val}, (v, v')) \geq \mathsf{offset}(\mathsf{val}, (v, \sigma'(v)))$ *holds for all edges* (v, v')*, then* σ' *is better than* σ*.*

We say that val *identifies* these strategies.

5.2. **No Local Improvements.** The absence of local improvements means that, for all vertices $v \in V$ and all outgoing edges $(v, v') \in E$, $\mathsf{offset}(\mathsf{val}, (v, v')) \geq \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))$.

We define for a solution val optimal for f_σ (like the val produced in Line 5 of Algorithm 2):

- $S^\sigma_{\mathsf{val}} = \{(v, v') \in E \mid \mathsf{offset}(\mathsf{val}, (v, v')) \leq \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))\}$ as the set of *at least stale* edges; naturally, every vertex has at least one outgoing stale edge: the one defined by σ;
- $E_{\mathsf{val}} = \{(v, v') \in E \mid \mathsf{offset}(\mathsf{val}, (v, v')) = 0\}$ as the set of edges, for which the inequation for val is sharp; in particular, all edges in the basis of H that defines val are sharp (and at least stale); and
- E^σ_{val} as any set of edges between E_{val} and S^σ_{val} (i.e. $E_{\mathsf{val}} \subseteq E^\sigma_{\mathsf{val}} \subseteq S^\sigma_{\mathsf{val}}$) such that E^σ_{val} contains an outgoing edge for every vertex $v \in V$; we are interested to deal with sets that retain the game property that every vertex has a successor, we can do that by adding (non-sharp) at least stale edges to E_{val}.

Note that S^σ_{val} is such a set, and, therefore, an adequate set is easy to identify. However, we might be interested in keeping the set small and choosing the edges defined by E_{val} plus one outgoing edge for every vertex v that does not have an outgoing edge in E_{val}. Where there is no local improvement, the most natural solution is to choose the edge $(v, \sigma(v)) \in E^\sigma_{\mathsf{val}}$ defined by σ for each such vertex v.

Observation 5.3. If $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ is a DPG and σ a strategy for both players such that val is an optimal solution for the objective f_σ to the system of inequations H, then $\mathcal{G}' = (V_{\min}, V_{\max}, E^\sigma_{\mathsf{val}}, w, \lambda)$ is also a DPG.

This simply holds because every vertex $v \in V$ retains at least one outgoing transition.

Lemma 5.4. *Let $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ be a DPG, σ a strategy for both players,* val *an optimal solution returned at Line 5 of Algorithm 2 for f_σ. If* val *does not define strategies of both players, then there is a better strategy σ' such that, for all $v \in V$, $(v, \sigma'(v)) \in E^\sigma_{\mathsf{val}}$.*

Proof. By Observation 5.3, $\mathcal{G}' = (V_{\min}, V_{\max}, E^\sigma_{\mathsf{val}}, w, \lambda)$ is a DPG. Let val' be the value of $\mathcal{G}'$, and σ' be the strategies for the two players defined by it.

If val' is also a solution of $\mathcal{G}$, then we are done. However, this need not be the case, as the set of inequations H' for $\mathcal{G}'$ is smaller than the set of inequations H for $\mathcal{G}$, so val' might violate some of the inequations that are in H, but not in H'. Given that val' is a solution for $\mathcal{G}'$, it satisfies all inequations in H'. Moreover, since val also satisfies all inequations of H', it follows that the same inequations hold for every convex combination of val and val'.

We now note that the inequations of H that are not in H' are not sharp for val. Thus, there is an $\varepsilon \in (0, 1]$ such that the convex combination $\mathsf{val}_\varepsilon = \varepsilon \cdot \mathsf{val}' + (1 - \varepsilon) \cdot \mathsf{val}$ is a solution to those inequations.

We have $\mathsf{offset}(\mathsf{val}, (v, \sigma'(v))) \leq \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))$ for every vertex v, as σ' only uses the at least stale edges from E^σ_{val}, so that in particular $f_{\sigma'}(\mathsf{val}) \leq f_\sigma(\mathsf{val})$ holds. We also have $f_\sigma(\mathsf{val}) > 0$ by assumption and $f_{\sigma'}(\mathsf{val}') = 0$ by the optimality of val' for H'.

For an optimal solution val'' of H for the objective $f_{\sigma'}$, this provides $f_{\sigma'}(\mathsf{val}'') \leq f_{\sigma'}(\mathsf{val}_\varepsilon) < f_\sigma(\mathsf{val})$.

Therefore, σ' is better than σ. □

Where val does not define strategies of both players, there are always at least $|V|$ inequations sharp, and thus there are vertices with multiple outgoing edges in E^σ_{val} that all have 0 offset.

While Lemma 5.4 also holds when there is a local improvement, we would not use it then, as finding local improvements is much easier.

When there is no local improvement, the most natural choice is $E^{\sigma}_{\mathsf{val}} = E_{\mathsf{val}} \cup \{(v, \sigma(v)) \mid v \in V\}$: this translates into keeping all transitions, for which the offset is *not* 0. As a result, σ' would change some of those for which the offset already is 0, but agree with σ for all vertices where this is currently not the case. This is a slightly surprising choice, since to progress one intuitively has to improve on the transitions whose offset is positive—which are then the ones one keeps.

5.3. **Games with Efficient Objective Improvement.**

In this subsection, we consider sufficient conditions for finding better strategies efficiently. Note that we only have to consider cases where the termination condition (Line 6 of Algorithm 2) is not met.

The simplest condition for efficiently finding better strategies is the existence of local improvements. In particular, it is easy to find, for a given solution val, strategies σ' for both players such that $f_{\sigma'}(\mathsf{val}) \leq f_{\sigma''}(\mathsf{val})$ holds for all strategies σ''. When there are local improvements, we can obtain a better strategy simply by applying them. This leaves the case in which there are no local improvements, but where val also does not *define* strategies for the two players. We have seen that we can obtain a better strategy by only swapping edges, for which the inequations are sharp (Lemma 5.4).

We now describe two conditions that, when simultaneously met, allow us to efficiently find better strategies: that the games are *sharp* and *improving*.

Sharp games. To do this efficiently, it helps if there are always $|V|$ inequations that are sharp for a solution val that minimises $f_\sigma(\mathsf{val})$ for some σ. Of course, if val is a solution returned by the simplex method, then there must be at least $|V|$ sharp inequations, as val is the solution defined by making $|V|$ inequations sharp (namely those that form the basis). By requiring that there are exactly $|V|$ sharp inequations we require that an optimal solution defines a basis. We call such a set of inequations H and games that define them *sharp* games.

Definition 5.5 (Sharp Game)**.** A game $\mathcal{G}$ is called *sharp* for a solution val if, and only if, val satisfies exactly $|V|$ inequations H computed by $\mathsf{Inequations}(\mathcal{G})$ sharply. It is called *sharp* if no solution satisfies strictly more than $|V|$ inequations sharply.

Improving games. The second condition, which allows us to identify better strategies efficiently, is to assume that, for every strategy σ for both players, if a solution val defined by a basis is not optimal for f_σ under the constraints H, then there is a single basis change that improves it. We call such games *improving* and show that all sharp games are improving.

DPGs are not always improving. Being improving is a very useful property as it removes the problem that the simplex method can stall, as a single basis change is enough for improving games to improve the value of the objective function (hence the name). For non-improving games, adjacent solutions might not improve although the current basis does not define a solution that is optimal for the current objective function.

Observation 5.6. Bases define solutions, and solutions identify (sets of) strategies: as we have used in Corollary 5.2, it is easy to identify for a solution val the pair of strategies σ (or the pairs of strategies if there are many) for whom $f_\sigma(\mathsf{val})$ is the smallest among all pairs of strategies; they are simply those pairs that minimise the offset for every vertex.

We therefore say that a solution identifies these pairs of strategies, and that a basis defines the pairs of strategies defined by the solution it defines.

Definition 5.7 (Improving Game)**.** A game $\mathcal{G}$ is called *σ-improving* for an objective function f_σ defined by a joined strategy σ and a non-optimal solution val defined by a basis for f_σ if, and only if, there is a single basis change that leads to a better solution val' (having a smaller value of $f_\sigma(\mathsf{val}') < f_\sigma(\mathsf{val})$) defined by the new basis.

A game $\mathcal{G}$ is called *improving* if it is improving for all objective functions f_σ defined by a joined strategy σ and all solutions val defined by bases that are not optimal w.r.t. f_σ.

Theorem 5.8. *Every sharp game $\mathcal{G}$ is improving.*

Proof. As our definition of sharp games simply says that every basic solution is nondegenerate, every step of a simplex algorithm (outside of optimal solutions) will provide a strict improvement [BT97, Theorem 3.3], so that $\mathcal{G}$ is σ-improving for every joined strategy σ and all solutions val defined by bases that are not optimal w.r.t. f_σ. □

We call a solution val' whose basis can be obtained from that of val by a single change to the basis of val a neighbouring solution to val. We show that, for improving games, we can refine the result of Lemma 5.4 so that the better strategy σ' also guarantees $f_{\sigma'}(\mathsf{val}') < f_\sigma(\mathsf{val})$ for some neighbouring solution val' to val.

This allows us to consider $O(|E|)$ basis changes and, where they define a solution, to seek optimal strategies for a given solution. Finding an optimal strategy for a given solution is straightforward.

Theorem 5.9. *Let $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ be an improving DPG, σ a strategy for both players that is not co-optimal,* val *an optimal solution returned at Line 5 of Algorithm 2 for f_σ. Then there is (at least) one neighbouring solution* val'' *to* val *such that there is a better strategy σ' that satisfies $f_{\sigma'}(\mathsf{val}'') < f_\sigma(\mathsf{val})$.*

This strategy σ' can be selected in such a way that $(v, \sigma'(v)) \in E^\sigma_{\mathsf{val}}$ holds for all $v \in V$ for the given set E^σ_{val}.

Proof. By Lemma 5.4, we can obtain a better strategy σ' from transitions in E^σ_{val}; for σ' therefore $f_{\sigma'}(\mathsf{val}) = f_\sigma(\mathsf{val})$ holds, and val is not optimal for $f_{\sigma'}$.

Let $\mathbf{b}$ be a basis that defines val. As $\mathcal{G}$ is improving, it is σ'-improving, so that there is a basis $\mathbf{b}'$ that neighbours $\mathbf{b}$ and defines a solution val' with $f_{\sigma'}(\mathsf{val}') < f_{\sigma'}(\mathsf{val}) = f_\sigma(\mathsf{val})$. □

We observe that any set E^σ_{val} can be selected, including the set $E_{\mathsf{val}} \cup \{(v, \sigma(v) \mid v \in V\}$, to select a better strategy σ' from.

While we can restrict the selection of σ' to the strategies that comply with the restriction $(v, \sigma'(v)) \in E^\sigma_{\mathsf{val}}$, there is no particular reason for doing so; as soon as we have a neighbouring solution val', we can identify a pair of strategies σ' for which $f_{\sigma'}(\mathsf{val}')$ is minimal and select σ' if $f_{\sigma'}(\mathsf{val}') < f_\sigma(\mathsf{val})$ holds.

5.4. Making Games Sharp (and Thus Improving).

Naturally, not every game is sharp. In this subsection, we discuss how to almost surely make games sharp by adding sufficiently small random noise to the edge weights. Note that these are 'global' adjustments of the game that only need to be applied once, as it is the game that becomes sharp.

We first create notation for expressing how much we can change edge weights by adding small noise, such that joint co-optimal strategies of the resulting game are joint co-optimal strategies in the original game. To this end, we define the *gap* of a game.

Gap of a game. Before defining the gap of the game, we recall that val_σ is the value of the joint strategy σ (whereas val denotes the outcome of the optimisation problem). We use this solution to argue that a small distortion of the edge weights cannot turn non-co-optimal strategies into co-optimal ones, and we define the gap of the game to reason about how small is small enough to retain non-co-optimality. The value of a joint strategy itself is useful for this because it is much easier to access than the result of optimisation. As a valuation of a non-co-optimal joint strategy is not a solution of a game, we will encounter negative offsets: they occur for exactly those edges, for which the associated inequation is not satisfied by val_σ.

Definition 5.10 (Contraction and Gap of a Game)**.** Given a game $\mathcal{G}$, we call its *contraction* $\lambda^* = \max\{\lambda_e \mid e \in E\}$. For a non-co-optimal joint strategy σ, we call the *gap* of σ for $\mathcal{G}$ the value $\gamma_\sigma = \min\{-\mathsf{offset}(\mathsf{val}_\sigma, e) \mid e \in E, \mathsf{offset}(\mathsf{val}_\sigma, e) < 0\}$. We call the *gap* of $\mathcal{G}$ the value $\gamma = \min\{\gamma_\sigma \mid \sigma \text{ is a non-co-optimal joint strategy}\}$, i.e. the minimal such value.

Note that $\gamma_\sigma > 0$ always holds for non-co-optimal strategies; for the minimal[6] value γ, $\gamma > 0$ thus always holds.

In a strategy improvement approach, where there is an edge e with $-\mathsf{offset}(\mathsf{val}_\sigma, e) > 0$, this would constitute a profitable switch, allowing the player to improve their return. While we do not use it this way, we argue that the gap γ allows us to infer that, when we change all edge weights by a sufficiently small factor (which we can derive from γ_σ, or γ, and the contraction of the game), a non-co-optimal strategy σ is still non-co-optimal after adjusting the weights this way.

We now use the gap of a game γ to define the magnitude of a change to all weights, such that all strategies that used to have a gap still have one, and thus that all co-optimal joint strategies from the distorted game are also co-optimal joint strategies in the undistorted game.

Lemma 5.11. *Let $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ be a DPG with contraction λ^* and gap γ, and let $\mathcal{G}' = (V_{\min}, V_{\max}, E, w', \lambda)$ differ from $\mathcal{G}$ only in the edge weights such that, for all $e \in E$, $|w_e - w'_e| < \frac{1-\lambda^*}{2}\gamma$ holds. Then any joint co-optimal strategy from $\mathcal{G}'$ is also co-optimal for $\mathcal{G}$.*

Proof. We argue that the small weight disturbance, $|w_e - w'_e| < \frac{1-\lambda^*}{2}\gamma$ for all $e \in E$, provides a small difference in the value that does not affect co-optimality. By using the definition of val_σ and the triangle inequality we can estimate the difference between the values. Precisely, for all joint strategies σ, we have for val_σ on $\mathcal{G}$, and val'_σ on $\mathcal{G}'$, that $|\mathsf{val}_\sigma(v) - \mathsf{val}'_\sigma(v)| < \frac{1}{1-\lambda^*}\frac{1-\lambda^*}{2}\gamma = \frac{\gamma}{2}$. Indeed, σ defines a play $\rho = e_0e_1e_2\ldots$, and we have $\mathsf{val}_\sigma(v) = \mathsf{out}(\rho) = \sum_{i=0}^{\infty} w_{e_i}\prod_{j=0}^{i-1}\lambda_{e_j}$ and $\mathsf{val}'_\sigma(v) = \sum_{i=0}^{\infty} w'_{e_i}\prod_{j=0}^{i-1}\lambda_{e_j}$. This provides

$$|\mathsf{val}_\sigma(v) - \mathsf{val}'_\sigma(v)| \leq \sum_{i=0}^{\infty}|w_{e_i} - w'_{e_i}|\prod_{j=0}^{i-1}\lambda_{e_j} < \frac{1-\lambda^*}{2}\gamma\sum_{i=0}^{\infty}\prod_{j=0}^{i-1}\lambda_{e_j} \leq \frac{1-\lambda^*}{2}\gamma\sum_{i=0}^{\infty}(\lambda^*)^i = \frac{\gamma}{2}.$$

If we assume that σ is not co-optimal for $\mathcal{G}$, then we have an edge $e = (v, v')$ with $-\mathsf{offset}(\mathsf{val}_\sigma, e) = \gamma_\sigma \geq \gamma$. Using offset' to indicate the use of w'_e for $\mathcal{G}'$, by applying the triangle inequality we get

[6]This skips over the case where all strategies are co-optimal, but that case is trivial to check and such games are trivial to solve, so that we ignore this case in this subsection.

$$|\mathsf{offset}'(\mathsf{val}'_\sigma, e) - \mathsf{offset}(\mathsf{val}_\sigma, e)| \leq |\mathsf{val}'_\sigma(v) - \mathsf{val}_\sigma(v)| + |w_e - w'_e| + \lambda_e|\mathsf{val}_\sigma(v') - \mathsf{val}'_\sigma(v')|$$
$$< \tfrac{\gamma}{2} + \tfrac{1-\lambda^*}{2}\gamma + \lambda_e\tfrac{\gamma}{2} \leq \gamma.$$

Together with the fact that $-\mathsf{offset}(\mathsf{val}_\sigma, e) \geq \gamma$, this provides $\mathsf{offset}'(\mathsf{val}'_\sigma, e) < 0$. Thus, σ is not co-optimal for $\mathcal{G}'$ either. □

Lemma 5.12. *Given a DPGs $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, the DPG $\mathcal{G}' = (V_{\min}, V_{\max}, E, w', \lambda)$ resulting from $\mathcal{G}$ by adding independently uniformly at random drawn values from an interval $(-\varepsilon, \varepsilon)$ to every edge weight, will almost surely result in a sharp game.*

Note that these distributions are continuous. They could be replaced by any other distribution over these intervals that has weight 0 for all individual points.

Proof. In this proof, we show a stronger property: every two different bases almost surely define different valuations (regardless of whether or not these valuations are solutions).

To this end, we can select two arbitrary different sets of $|V|$ edges that define potential bases $\mathbf{b}_1$ and $\mathbf{b}_2$; they can be selected before drawing $\mathcal{G}'$.

What we show is that it happens with probability 0 that $\mathbf{b}_1$ and $\mathbf{b}_2$ are bases *and* define the same valuations (regardless of whether or not these valuations are solutions) in $\mathcal{G}$. If $\mathbf{b}_1$ or $\mathbf{b}_2$ is not a basis (which would happen if the inequations are not linearly independent, e.g. if a vertex v does not appear at all in these inequations) we are done, too.

As $\mathbf{b}_1$ and $\mathbf{b}_2$ are different, there will be one inequation that corresponds to an edge $e = (v, v')$ that occurs in $\mathbf{b}_2$, but not in $\mathbf{b}_1$. As all weight disturbances are drawn independently, we assume without loss of generality that the weight disturbance to this edge is drawn last.

Now, the valuation val_1 defined by $\mathbf{b}_1$ does not depend on this final draw. For val_1, there is a value $w'_e = \mathsf{val}_1(v) - \lambda_e\mathsf{val}_1(v')$ that defines the weight w'_e that e would need to have, in order to make the inequation sharp for e.

For the valuation val_1 defined by $\mathbf{b}_1$ to be equal to the valuation val_2 defined by $\mathbf{b}_2$, the weight for the edge e (after adding the drawn distortion) needs to be exactly w'_e. (This is a necessary condition, but not necessarily a sufficient condition.) There is at most one value for the disturbance that would provide for this, and this disturbance for the weight of e is sampled with a likelihood of 0. □

Putting these two results together, we get the following:

Corollary 5.13. *Given a pair of DPGs $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ with contraction λ^* and gap γ, and $\mathcal{G}' = (V_{\min}, V_{\max}, E, w', \lambda)$ obtained from $\mathcal{G}$ by adding independently uniformly at random drawn values from an interval $(-\varepsilon, \varepsilon)$ to every edge weight, for some $\varepsilon < \frac{1-\lambda^*}{2}\gamma$, it holds that any joint co-optimal strategy from $\mathcal{G}'$ is also co-optimal for $\mathcal{G}$, and $\mathcal{G}'$ is almost surely sharp.*

Note that we can estimate the gap cheaply when all coefficients in $\mathcal{G}$ are rational. To estimate the gap

$$\gamma = \min\{-\mathsf{offset}(\mathsf{val}_\sigma, e) \mid \sigma \text{ is a non-co-optimal joint strategy}, e \in E, \mathsf{offset}(\mathsf{val}_\sigma, e) < 0\}\ ,$$

we first fix a joint non-co-optimal strategy σ and an edge $e = (v, w)$ with $\mathsf{offset}(\mathsf{val}_\sigma, e) < 0$.

Starting in v, σ defines a run $\rho = e_1e_2e_3\dots$ in the form of a "lasso path", which consists of a (possibly empty) initial path $e_1, \dots, e_k$, followed by an infinitely often repeated cycle $e'_1, \dots, e'_\ell$, where all edges occur only once.

In this run, an edge e_i occurs only once and contributes $(\prod_{j=1}^{i-1}\lambda_{e_j})w_{e_i}$ to the value of this run, while an edge e'_i occurs infinitely often and contributes the value $\frac{(\prod_{j=1}^{k}\lambda_{e_j})\cdot(\prod_{j=1}^{i-1}\lambda_{e'_j})\cdot w_{e'_i}}{1-\prod_{j=1}^{\ell}\lambda_{e'_j}}$ to the value of the run. Now all we need to do is to estimate a common denominator.

To do this, let $\mathsf{nom}(r)$ and $\mathsf{denom}(r)$ be the nominator and denominator of a rational number r. It is easy to see that

$$\mathsf{common}' = \prod_{v\in V}\mathsf{denom}(\lambda_{(v,\sigma(v))})\cdot\mathsf{denom}(w_{(v,\sigma(v))})$$

is a common denominator of all expressions of the first type, $(\prod_{j=1}^{i-1}\lambda_{e_j})w_{e_i}$, and of the nominator from the second, $(\prod_{j=1}^{k}\lambda_{e_j})\cdot(\prod_{j=1}^{i-1}\lambda_{e'_j})\cdot w_{e'_i}$.

This leaves the contribution of the denominator of the fraction from the second form, which is the nominator of the term $1-\prod_{j=1}^{\ell}\lambda_{e'_j}$. But since the term is between 0 and 1, its value is strictly smaller than the value of its denominator; we thus have for

$$n_1 = \mathsf{nom}(1-\prod_{j=1}^{\ell}\lambda_{e'_j}) < \prod_{j=1}^{\ell}\mathsf{denom}(\lambda_{e'_j}) \le \prod_{v\in V}\mathsf{denom}(\lambda_{(v,\sigma(v))})$$

that

$$\mathsf{common}'\cdot n_1 \text{ is a denominator for } \mathsf{out}(\rho)\ .$$

To obtain $-\mathsf{offset}(\mathsf{val}_\sigma, e)$, we have to compare with a run starting in v, but taking the edge $e \neq (v,\sigma(v))$ first.

This defines a run $\rho' = d_1d_2d_3\dots$ (with $d_1 = e$), again in the form of a "lasso path" that consists of an initial path $d_1,\dots,d_{k'}$, followed by an infinitely often repeated cycle $d'_1,\dots,d'_{\ell'}$, where every edge on the path $d_1,\dots,d_{k'},d'_1,\dots,d'_{\ell'}$ occurs only once.

In this run, an edge d_i occurs only once and contributes $(\prod_{j=1}^{i-1}\lambda_{d_j})w_{d_i}$ to the value of this run, while an edge d'_i occurs infinitely often and contributes the value $\frac{(\prod_{j=1}^{k'}\lambda_{d_j})\cdot(\prod_{j=1}^{i-1}\lambda_{d'_j})\cdot w_{d'_i}}{1-\prod_{j=1}^{\ell'}\lambda_{d'_j}}$ to the value of the run.

It is easy to see that

$$\mathsf{common}'\cdot\mathsf{denom}(\lambda_e)\cdot\mathsf{denom}(w_e)$$

is a common denominator of all expressions of the first type, $(\prod_{j=1}^{i-1}\lambda_{d_j})w_{d_i}$, and of the nominator from the second, $(\prod_{j=1}^{k'}\lambda_{d_j})\cdot(\prod_{j=1}^{i-1}\lambda_{d'_j})\cdot w_{d'_i}$.

This again leaves the contribution of the denominator of the fraction from the second form, which is the nominator of the term $1-\prod_{j=1}^{\ell'}\lambda_{d'_j}$. Again a term between 0 and 1, its value is strictly smaller than the value of its denominator; we thus have for

$$n_2 = \mathsf{nom}(1-\prod_{j=1}^{\ell'}\lambda_{d'_j}) < \prod_{j=1}^{\ell'}\mathsf{denom}(\lambda_{d'_j}) \le \prod_{v\in V}\mathsf{denom}(\lambda_{(v,\sigma(v))})$$

that

$$\mathsf{common}'\cdot\mathsf{denom}(\lambda_e)\cdot\mathsf{denom}(w_e)\cdot n_2 \text{ is a denominator for } \mathsf{out}(\rho')\ .$$

Consequently,

$$\mathsf{common}'\cdot\mathsf{denom}(\lambda_e)\cdot\mathsf{denom}(w_e)\cdot n_1\cdot n_2 \text{ is a denominator for } \mathsf{offset}(\mathsf{val}_\sigma, e)\ .$$

Putting the estimates together, we get that

$$\mathsf{bound} = \mathsf{denom}(\lambda_e) \cdot \mathsf{denom}(w_e) \cdot \prod_{v \in V} \mathsf{denom}(\lambda_{(v,\sigma(v))})^3 \cdot \mathsf{denom}(w_{(v,\sigma(v))})$$

is smaller than the smallest denominator of $\mathsf{offset}(\mathsf{val}_\sigma, e)$. We then use this to estimate $\gamma \geq -\mathsf{offset}(\mathsf{val}_\sigma, e) > 1/\mathsf{bound}$.

This value is easily estimated by using the highest possible denominators available in $\mathcal{G}$—maximising $\mathsf{denom}(\lambda_e) \cdot \mathsf{denom}(w_e)$ over all edges $e \in E$ and maximising $\mathsf{denom}(\lambda_{(v,\sigma(v))})^3 \cdot \mathsf{denom}(w_{(v,\sigma(v))})$ individually for every vertex $v \in V$. Its representation is polynomial in the size of $\mathcal{G}$.

Thus, we can almost surely obtain sharpness by adding small noise to the weights, and the resulting sharp games are also improving by Theorem 5.8. This guarantees cheap progress for the case where there are no local improvements.

5.5. **Mixing Pivoting on the Simplex and of the Objective.** When using a simplex based technique to implement LinearProgramming (Line 5 of Algorithm 2), then the algorithm mixes three approaches that stepwise reduce the value of $f_\sigma(\mathsf{val})$:

(1) The simplex algorithm updates the basis, changing val (while retaining the objective function f_σ).
(2) Local updates, that change the objective function f_σ (through updating σ) and retain val.
(3) Non-local updates.

Non-local updates are more complex than the other two, and the correctness proofs make use of the optimality (w.r.t. f_σ) of the current solution. For both reasons, it seems natural to take non-local updates as a last resort.

The other two updates, however, can be freely mixed, as they both bring down the value of $f_\sigma(\mathsf{val})$ by applying local changes. That the improvements from (1) are given preference in the algorithm is a choice made to keep the implementation of the algorithm for using linear programs open, allowing, for example, to use ellipsoid methods [Kha79] or inner point methods [Kar84] to keep this step tractable.

6. Experimental Evaluation

To evaluate the performance of the novel approach, we have implemented it in an experimental framework in C++. Together with the objective improvement algorithm (OI, for short), we have also implemented the classic asymmetric variant of the strategy improvement algorithm (SI, for short) that has been described in Section 3. Note that our implementation uses floating-point numbers with double precision (type `double`), while the approach itself assumes infinite precision. As a consequence, match checks are not possible and the solutions can be affected by an ϵ error due to the precision of the library (that comes from the external linear programming solver too). For most parts, this could be rectified by using fractions. However, while fractions with infinite precision are available with the right libraries, their use would lead to an overhead and would not solve the precision problem of the external linear programming solver. We also note that the small noise added in Section 5.4 to almost surely make a game sharp and improving requires a continuous value space.

Evaluation comes with a number of difficulties. First, there are no actual concrete benchmarks for DPGs. We have therefore translated parity games that model synthesis problems into DPGs. The translation process is exponential in the number of priorities of the parity games. For this reason we could translate and test only two types of concrete games: the Elevator and Language Inclusion. Besides these concrete games, we have also generated random games of various sizes. We have considered games with rational weights in the interval $[-250, 250]$ on the edges, as games with integer weights represent a restricted class of games. The range of rational weights is not crucial as it suffice to multiply their values to scale the interval up or down. Moreover, we use a uniform discount factor of 0.9, since for games in which each vertex has different weights on the edges, non-uniform discount factors do not make games any easier or harder to solve. Using a low discount factor (*e.g.* 0.5 or below) make, instead, games easier to solve, because with small discount factors the solution (*i.e.* the valuation $\mathsf{val}_{\mathcal{G}}$) of the vertices converges after few steps, while with a discount factor close to 1 the valuation requires many steps to converge. Finally, it is worth noting that games with such random weights on the edges are also almost surely sharp, hence, it has not been necessarily to implement the *sharpening* technique described in Section 5.4.

The second obstacle is that, for an implementation, one has to choose *how* to update the current "state", both for SI and OI. For SI, we use a greedy all switch rule: we update all local decisions where there is a local improvement available, and where there are multiple, we choose one with the maximal local improvement. For OI, there are more choices to make. As mentioned in Section 5.5, in simplex-based approaches, we can freely mix (1) the standard updates of the basis for a fixed objective function, and (2) updates of the objective function itself, and in most steps, both options are available. For (1), there are a plethora of update rules for the simplex, and for (2), there is also a range of ways of how to select the update. We have dodged this question by not directly implementing a solver, but by, instead, solving the linear programs with an off-the-shelf solver. We have used the LP solver provided by the ALGLIB library[7] for both OI and SI. This does not restrict us to the use of simplex-based solvers, while it would restrict simplex-based solutions to always giving priority to (1) over (2). Where the solver does provide a solution for a given objective function, we use rule (2) with a greedy all switch rule.

One cost of doing this is that we do not have a similarly straightforward integration of (3) non-local updates, where we are in the optimal solution for an objective function that cannot be improved with rule (2). Instead of using the method from Theorem 5.9 to slightly adjust a simplex method to find an improvement for improving games (knowing that the games we create are almost surely improving), we use the general rule from Lemma 5.4, trying random stale strategies. In principle, this might lead to large plateaux that are hard to exit, but we considered this very unlikely, at least for the improving games we create, and our experiments have confirmed this.

This setup allows us to fairly compare OI with SI by counting the number of linear program instances that need to be solved on the way.

As our first benchmark set we selected games with two outgoing edges per vertex. This type of games have a relatively small strategy space, which should make them a relatively easy target for SI in that SI should require fewer steps to identify the optimal strategy than OI. Figures 2 and 3 show the number of iterations and strategy updates required to solve a game of size ranging from 100 to 1000 vertices. We selected 1000 games, each of the ten

[7]Library libalglib version 3.16.0

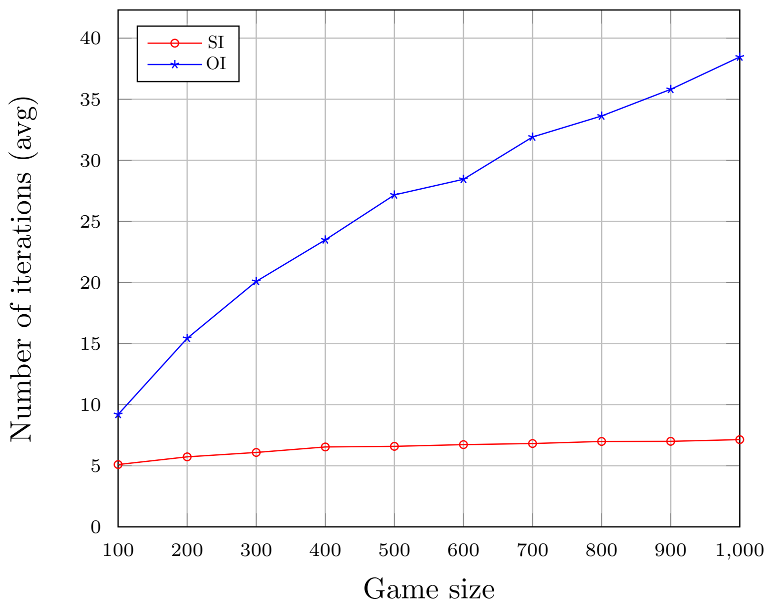

Figure 2: Comparisons of the average number of iterations (LP calls) on games with two successors for each vertex.

points in the graph shows the mean value for a cluster of 100 games of the corresponding size.

The number of iterations (Figure 2) represents how many times the linear programming solver has been called. As expected, SI is more efficient than OI at solving these games in terms of LP calls. What was less expected is how the advantage of SI over OI grows with the size of the game. Solving the linear programs, instead, is simpler for OI, at least when using a simplex style algorithm[8]. We also note that only 55 (5.5%) of the 1000 games considered overall required a non-local improvement step. In these rare cases, this has been solved by choosing a stale switch (i.e. a switch of strategy that leads to the same solution).

The number of updates, instead, counts how many times each vertex changed its strategy before the next call to the LP solver, and then provides the sum of these switches. If the ownership of the vertices of a game is not balanced (*e.g.* one player owns 80% of the vertices), most of the solving effort is charged to the LP solver, while the SI algorithm itself needs to update very few strategies. On the contrary, OI updates the strategies for both the players. It would therefore stand to reason to expect that the number of local strategy updates would give an extra advantage to SI. This is, however, not the case: Figure 3 shows that, while OI needs to update the strategy for at least twice as many vertices, it needs only around a third more local updates for this.

In the second benchmark set we selected games with more moves per vertex, in the range [5, 10]. In this case, the space of strategies is much wider than one of the games of the

[8]Simplex style algorithms operate in two phases: in the first phase, they look for any basis that defines a solution to the constraint system (Line 1), and in the second, they optimise (while loop). For OI, the first phase can be skipped for all but the first call; this is because the constraint system does not change, so that the previous solution can be used as a starting point. For SI, the constraint system changes; here, the previous solution is never a solution to the new linear program.

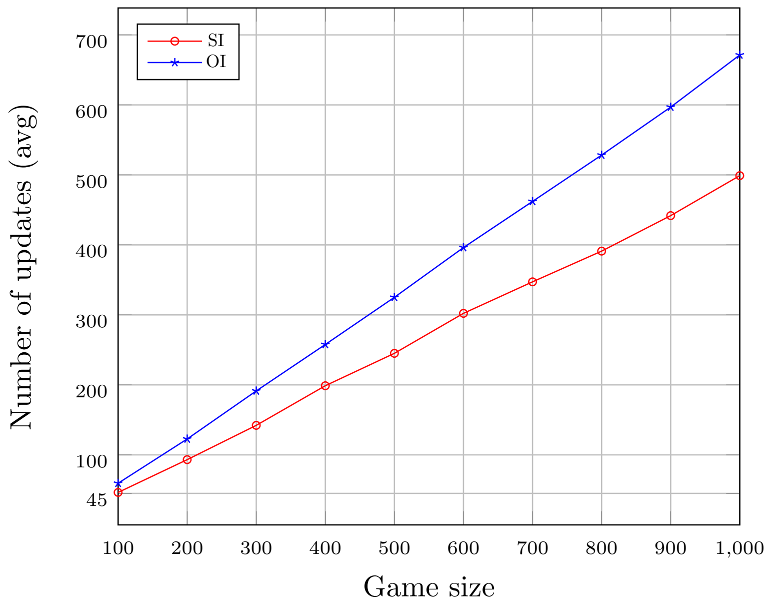

Figure 3: Comparisons of the average number of local strategy updates for games with two successors per vertex.

first benchmark set, and algorithms based on SI find these games harder to solve. Figures 4 and 5 show the comparison of OI and SI on the second benchmark set of 1000 games.

In Figure 4 we can observe a slowly but consistently growing gap in the number of LP calls (iterations). In addition to the advantage of simpler LP problems, we see that in these slightly more complex games OI also requires fewer calls, with SI needing some 2.5 to 3 times as many. For this benchmark set, only 7 (0.7%) of the games reached a non-improving state at any point.

Interestingly, Figure 5, instead, depicts a symmetric behaviour of the graph of Figure 3, but this time with the lanes of the two solvers swapped. Thus, although OI considers the strategy at all vertices (instead of at most half), the sum of local improvements is also slightly smaller.

When considering games with many transitions, namely games where the outdegree of every vertex is 10% of the number of vertices, the advantage of OI over SI grows further. Figure 6 shows a larger gap in the number of LP calls than Figure 4, and Figure 7 shows not only an advantage of OI over SI, but it also appears that the number of local strategy updates grows linearly for OI, but faster for SI. Only 3 (0.3%) of the games reached a non-improving state at some point of the solving process.

Finally, for the concrete problems translated from parity games, all these games turned out to be easy to solve so that a single LP call suffices to find the optimal solution. Therefore, in Table 2, instead of showing the number of iterations and updates, we show the solution time in seconds that provides a measure of the complexity of the game in terms of size of the linear system. Most of the games can be solved in less than one second. For the Elevator family of games, their size grows rapidly, so that we could not translate more than the first 5 instances. For the Language Inclusion games, instead, we report the games solved within 15 minutes.

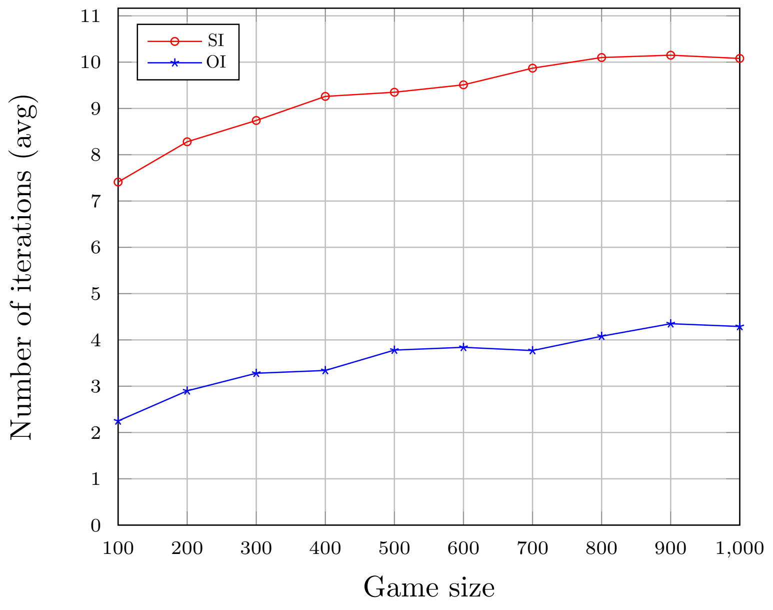

Figure 4: Comparisons of the average number of iterations (LP calls) for games with many (5 to 10) successors per vertex.

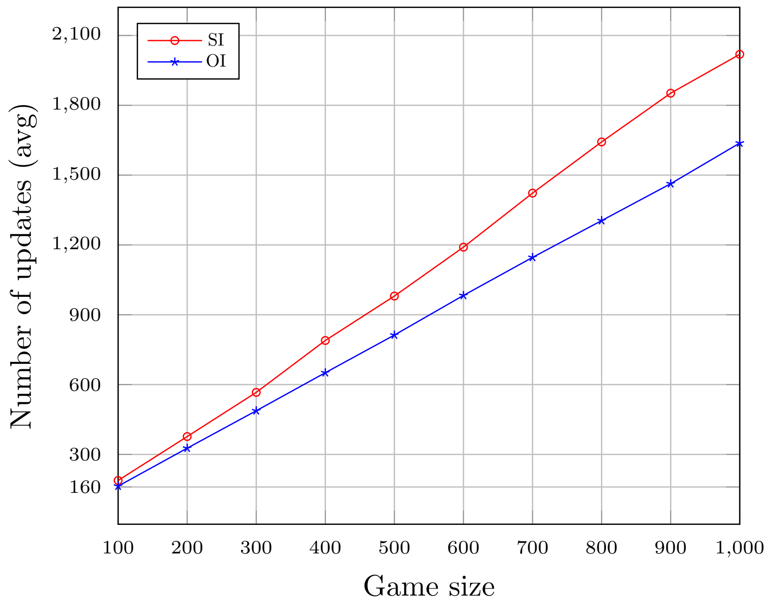

Figure 5: Comparisons of the average number of local strategy updates for games with many (5 to 10) successors per vertex.

7. Discussion

There is widespread belief that mean payoff and discounted payoff games have two types of algorithmic solutions: value iteration [FGO20, Koz21] and strategy improvement [Lud95,

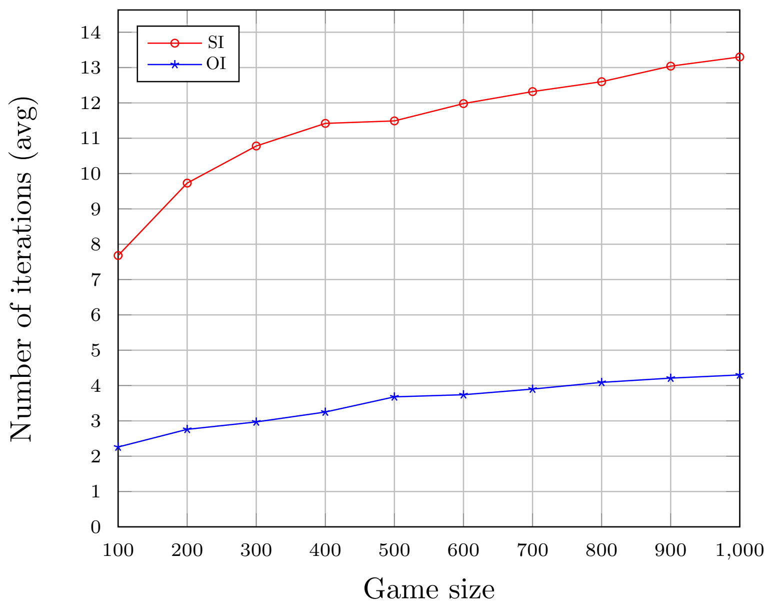

Figure 6: Comparisons of the average number of iterations (LP calls) for games with a linear number of successors per vertex (10%).

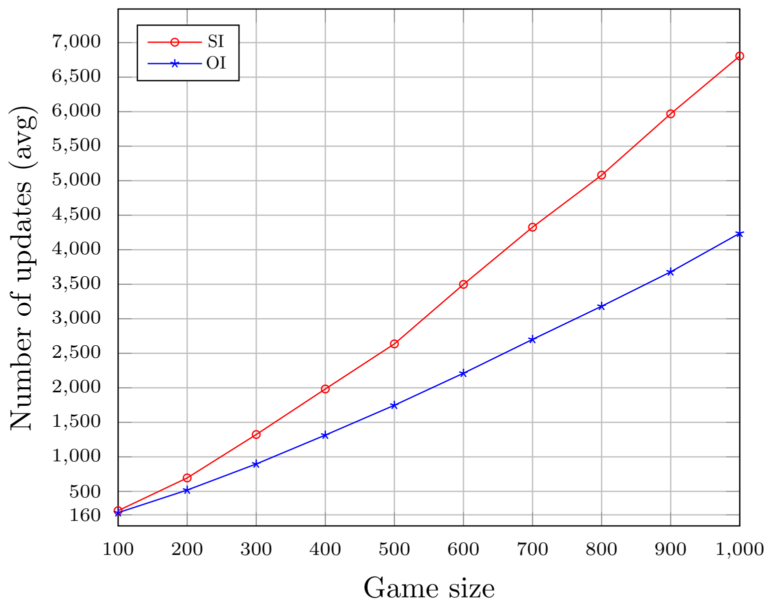

Figure 7: Comparisons of the average number of local strategy updates for games with a linear number of successors per vertex (10%).

Pur95, BV07, Sch08, STV15]. We have added a third method, which is structurally different and opens a new class of algorithms to attack these games. Moreover, our new symmetric approach has the same proximity to linear programming as strategy improvement algorithms, which is an indicator of efficiency.

Benchmark	Positions	Moves	Time
Elevator 1	36	54	0
Elevator 2	144	234	0
Elevator 3	564	950	0
Elevator 4	2688	4544	2
Elevator 5	15683	26354	16
Language Inclusion 1	170	1094	0
Language Inclusion 2	304	1222	0
Language Inclusion 3	428	878	0
Language Inclusion 4	628	1538	0
Language Inclusion 5	509	2126	0
Language Inclusion 6	835	2914	0
Language Inclusion 7	1658	4544	1
Language Inclusion 8	14578	17278	47
Language Inclusion 9	25838	29438	279

Table 2: Experiments on concrete verification problems.

The lack of a benchmarking framework for existing algorithms prevents us from testing and comparing an eventual implementation thoroughly, but we have compared OI against SI on random games and synthesis parity translated games in Section 6. To keep this comparison fair (and simple), we have put as much as possible of both problems into LP calls, using greedy all switch updates in both cases. We found that SI performs better on random games with very few (two) successors per vertex, while OI already comes out on top with few (five to ten), and shines with many (10% of the vertices) successors. This was a bit unexpected to us, as we thought that such a naive implementation of the new concept could not possibly compete.

Our results for random games with a low (five to ten) outdegree suggests that the number of local updates seen grows linearly with the size of the game for both SI and OI, and that the number of iterations initially grows and later plateaus. This may well be the same for games with a minimal outdegree of two, and the data clearly supports this for SI, but the number of LP calls growth almost linearly for OI. This might be an outlier in the behaviour and it is well possible that OI plateaus later, as it is unlikely that games with a low and very low outdegree behave fundamentally different. For SI, this was entirely expected, and it is perhaps not overly surprising for OI either. Generally speaking, OI appears to do better than SI measured in the number of LP calls on random games, except where the outdegree is tiny.

Naturally, a fresh approach opens the door to much follow-up research. A first target for such research is the questions on how to arrange the selection of better strategies to obtain fewer updates, either proven on benchmarks, or theoretically in worst, average, or smoothed analysis. Our implementation effectively updates the objective function only once an optimal solution for the current objective function is found. This is for two pragmatic reasons: it allows us to use a solver for linear programs as a black box, and we feel that it provides the best comparison with strategy improvement. In terms of the three update rules stated in Section 5.5, this approach used (1) basis change where available, then (2) update of the objective function through updating the joint strategy, before turning to (3) non-local updates as a last resort.

Seeing that (1) and (2) are equally cheap and simple, we could also favour (2) over (1): where we have a solution val defined by a basis, we can first find a joint strategy σ such that $f_\sigma(\mathsf{val})$ is minimal among all strategies, before turning to (1) and (3). This would also make sure that no corner of the polytope is ever visited twice. This could happen in principle when favouring (1) over (2), as a step that is good for one objective function might not be good for its successor, but as the value of the objective function goes down in every step, a return to a previously optimal solution is not possible.

From a theoretical perspective, it would in particular be interesting to establish non-trivial upper or lower bounds for various pivoting rules. Without such a study, a trivial bound for the proposed approach is provided by the number of strategies (exponential).

A second question is whether this method as whole can be turned into an inner point method [Kar84]. If so, this could be a first step towards showing tractability of discounted payoff games – which would immediately extend to mean-payoff and parity games.

Acknowledgements. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101032464. It was supported by the EPSRC through the projects EP/X017796/1 (Below the Branches of Universal Trees) and EP/X03688X/1 (TRUSTED: SecuriTy SummaRies for SecUre SofTwarE Development). We also thank Peter Austin from the University of Liverpool for providing an integration of the LP solver. We thank an anonymous reviewer for pointing out that all sharp games are improving, which simplifies our methods compared to the conference version and thus improves the results.

References

[AHK02] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

[BCJ+97] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. R. Marrero. An Improved Algorithm for the Evaluation of Fixpoint Expressions. *Theoretical Computer Science*, 178(1-2):237–255, 1997. doi:10.1016/S0304-3975(96)00228-9.

[BDM16] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Improving Priority Promotion for Parity Games. In *Haifa Verification Conference16*, LNCS 10028, pages 1–17. Springer, 2016. doi:10.1007/978-3-319-49052-6_8.

[BDM18a] M. Benerecetti, D. Dell'Erba, and F. Mogavero. A Delayed Promotion Policy for Parity Games. *Information and Computation*, 262(2):221–240, 2018. doi:10.1016/j.ic.2018.09.005.

[BDM18b] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving Parity Games via Priority Promotion. *Formal Methods in System Design*, 52(2):193–226, 2018.

[BDM20] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving Mean-Payoff Games via Quasi Dominions. In *Tools and Algorithms for the Construction and Analysis of Systems'20*, LNCS 12079, pages 289–306. Springer, 2020. doi:10.1007/978-3-030-45237-7_18.

[BDM+24] M. Benerecetti, D. Dell'Erba, F. Mogavero, S. Schewe, and D. Wojtczak. Priority Promotion with Parysian Flair. *Journal of Computer and System Sciences*, 2024. doi:10.1016/j.jcss.2024.103580.

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to linear optimization*, volume 6 of *Athena scientific optimization and computation series*. Athena Scientific, 1997.

[BV07] H. Björklund and S. G. Vorobyov. A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean-Payoff Games. *Discrete Applied Mathematics*, 155(2):210–229, 2007. doi:10.1016/j.dam.2006.04.029.

[CJK+22] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding Parity Games in Quasi-polynomial Time. *SIAM Journal on Computing*, 51(2):17–152, 2022. doi:10.1137/17M1145288.

[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. doi:10.1090/dimacs/013/04.

[dAHM01] L. de Alfaro, T. A. Henzinger, and R. Majumdar. From Verification to Control: Dynamic Programs for Omega-Regular Objectives. In *Logic in Computer Science'01*, pages 279–290. IEEE Computer Society, 2001. doi:10.1109/LICS.2001.932504.

[DDS23] D. Dell'Erba, A. Dumas, and S. Schewe. An Objective Improvement Approach to Solving Discounted Payoff Games. In *Games, Automata, Logics, and Formal Verification'23*, EPTCS 390, pages 203–219, 2023.

[DS22] D. Dell'Erba and S. Schewe. Smaller Progress Measures and Separating Automata for Parity Games. *Frontiers in Computer Science*, 4, 2022. doi:10.3389/fcomp.2022.936903.

[EJ91] E. A. Emerson and C. S. Jutla. Tree Automata, muCalculus, and Determinacy. In *Foundation of Computer Science'91*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.

[EJS93] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On Model-Checking for Fragments of muCalculus. In *Computer Aided Verification'93*, LNCS 697, pages 385–396. Springer, 1993. doi:10.1007/3-540-56922-7_32.

[EL86] E. A. Emerson and C. L. Lei. Efficient Model Checking in Fragments of the Propositional muCalculus. In *Logic in Computer Science'86*, pages 267–278. IEEE Computer Society, 1986.

[Fea10] J. Fearnley. Non-Oblivious Strategy Improvement. In *Logic for Programming Artificial Intelligence and Reasoning'10*, LNCS 6355, pages 212–230. Springer, 2010. doi:10.1007/978-3-642-17511-4_13.

[FGO20] N. Fijalkow, P. Gawrychowski, and P. Ohlmann. Value Iteration Using Universal Graphs and the Complexity of Mean Payoff Games. In *Mathematical Foundations of Computer Science'20*, LIPIcs 170, pages 1–15. Leibniz-Zentrum fuer Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.34.

[FJdK$^+$19] J. Fearnley, S. Jain, B. de Keijzer, S. Schewe, F. Stephan, and D. Wojtczak. An Ordered Approach to Solving Parity Games in Quasi Polynomial Time and Quasi Linear Space. *Software Tools for Technology Transfer*, 21(3):325–349, 2019. doi:10.1007/s10009-019-00509-3.

[JL17] M. Jurdziński and R. Lazić. Succinct Progress Measures for Solving Parity Games. In *Logic in Computer Science'17*, pages 1–9. Association for Computing Machinery, 2017. doi:10.1109/LICS.2017.8005092.

[Jur98] M. Jurdziński. Deciding the Winner in Parity Games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998. doi:10.1016/S0020-0190(98)00150-1.

[Kar84] N. Karmarkar. A New Polynomial-time Algorithm for Linear Programming. In *Symposium on Theory of Computing'84*, pages 302–311. Association for Computing Machinery, 1984.

[Kha79] L. G. Khachian. A Polynomial Algorithm in Linear Programming. *USSR Computational Mathematics and Mathematical Physics*, 244:1093–1096, 1979.

[Koz83] D. Kozen. Results on the Propositional muCalculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[Koz21] A. Kozachinskiy. Polyhedral Value Iteration for Discounted Games and Energy Games. In *Symposium on Discrete Algorithms'21*, page 600–616. SIAM, 2021. doi:10.1137/1.9781611976465.37.

[LB20] K. Lehtinen and U. Boker. Register Games. *Logical Methods in Computer Science*, 16(2), 2020. doi:10.23638/LMCS-16(2:6)2020.

[LPSW20] K. Lehtinen, P. Parys, S. Schewe, and D. Wojtczak. A Recursive Approach to Solving Parity Games in Quasipolynomial Time. *Logical Methods in Computer Science*, 18(1), 20220. doi:10.46298/lmcs-18(1:8)2022.

[Lud95] W. Ludwig. A Subexponential Randomized Algorithm for the Simple Stochastic Game Problem. *Information and Computation*, 117(1):151–155, 1995. doi:10.1006/inco.1995.1035.

[McN93] R. McNaughton. Infinite Games Played on Finite Graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993. doi:10.1016/0168-0072(93)90036-D.

[Obd03] J. Obdrzálek. Fast Mu-Calculus Model Checking when Tree-Width Is Bounded. In *Computer Aided Verification'03*, LNCS 2725, pages 80–92. Springer, 2003. doi:10.1007/978-3-540-45069-6_7.

[Pit06] N. Piterman. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. In *Logic in Computer Science'06*, pages 255–264. IEEE Computer Society, 2006. doi:10.2168/LMCS-3(3:5)2007.

[Pur95] A. Puri. *Theory of Hybrid Systems and Discrete Event Systems.* PhD thesis, University of California, Berkeley, USA, 1995.

[Sch08] S. Schewe. An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games. In *Computer Science Logic'08*, LNCS 5213, pages 369–384. Springer, 2008. `doi:10.1007/978-3-540-87531-4\_27`.

[SF06a] S. Schewe and B. Finkbeiner. Satisfiability and Finite Model Property for the Alternating-Time muCalculus. In *Computer Science Logic'06*, LNCS 6247, pages 591–605. Springer, 2006. `doi:10.1007/11874683\_39`.

[SF06b] S. Schewe and B. Finkbeiner. Synthesis of Asynchronous Systems. In *Symposium on Logic-based Program Synthesis and Transformation'06*, LNCS 4407, pages 127–142. Springer, 2006. `doi:10.1007/978-3-540-71410-1\_10`.

[STV15] S. Schewe, A. Trivedi, and T. Varghese. Symmetric Strategy Improvement. In *International Colloquium on Automata, Languages, and Programming'15*, LNCS 9135, pages 388–400. Springer, 2015. `doi:10.1007/978-3-662-47666-6\_31`.

[Var98] M. Y. Vardi. Reasoning about The Past with Two-Way Automata. In *International Colloquium on Automata, Languages, and Programming'98*, LNCS 1443, pages 628–641. Springer, 1998. `doi:10.1007/BFb0055090`.

[VJ00] J. Vöge and M. Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *Computer Aided Verification'00*, LNCS 1855, pages 202–215. Springer, 2000. `doi:10.1007/10722167\_18`.

[Wil01] T. Wilke. Alternating Tree Automata, Parity Games, and Modal muCalculus. *Bulletin of the Belgian Mathematical Society*, 8(2):359–391, 2001.

[Zie98] W. Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. `doi:10.1016/S0304-3975(98)00009-7`.

[ZP96] U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996. `doi:10.1016/0304-3975(95)00188-3`.