From Parity and Payoff Games to Linear Programming

Sven Schewe

University of Liverpool
sven.schewe@liverpool.ac.uk

Abstract. This paper establishes a surprising reduction from parity and mean payoff games to linear programming problems. While such a connection is trivial for solitary games, it is surprising for two player games, because the players have opposing objectives, whose natural translations into an optimisation problem are minimisation and maximisation, respectively. Our reduction to linear programming circumvents the need for concurrent minimisation and maximisation by replacing one of them, the maximisation, by approximation. The resulting optimisation problem can be translated to a linear programme by a simple space transformation, which is inexpensive in the unit cost model, but results in an exponential growth of the coefficients. The discovered connection opens up unexpected applications – like μ-calculus model checking – of linear programming in the unit cost model, and thus turns the intriguing academic problem of finding a polynomial time algorithm for linear programming in this model of computation (and subsequently a strongly polynomial algorithm) into a problem of paramount practical importance: All advancements in this area can immediately be applied to accelerate solving parity and payoff games, or to improve their complexity analysis.

1 Introduction

This paper links two intriguing open complexity problems, solving parity and payoff games and solving linear programming problems in the unit cost model.

Linear programming [1–7], the problem of maximising $\mathbf{c}^T\mathbf{x}$ under the side conditions $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$, is one of the most researched problems in computer science and discrete mathematics. The interest in linear programming has two sources: It has a significant practical impact, because a wide range of optimisation problems in economy and operations research can be approached with linear programming, and it is the source of a range of challenges that remained unresolved for years.

The most prominent open challenge is Smale's 9^{th} problem [8], which asks if linear programming has a polynomial time solution in the unit cost model. In the unit cost model, we assume that all arithmetic operations have an identical unit cost. This model is inspired by the desire of mathematicians to use real numbers instead of rationals, but it can also be applied to handle large numbers,

whose representation in binary led to an exponential (or higher) blow-up in the size of the problem description.

Dantzig's simplex algorithm addresses both complexity models alike. While highly efficient in practice [1], Klee and Minty [2] showed that the worst case running time of the simplex algorithm is exponential in the size of the linear programme. Their original proof referred to a particular Pivot rule, but it has proven to be very flexible with respect to the chosen Pivot rule, and could be extended to every suggested deterministic Pivot rule that does not depend on the history of previous updates. The proof is also independent of the chosen cost model, because the path constructed by the Pivot rule has exponential length.

While the complexity for the unit cost model is still open, polynomial time algorithms for the Turing model like Kachian's ellipsoid method [3] and interior point methods such as Karmarkar's algorithm [4] are known for decades. Unfortunately, these algorithms depend on the size of the binary representation of the numbers, and do not provide any insight for the unit cost model. Apart from the mathematical interest in determining the unit cost complexity of solving linear programmes, such an algorithm is a prerequisite of finding a *strongly polynomial* time algorithm. The requirement for an algorithm to be strongly polynomial is slightly higher: It also requires that the intermediate arithmetic operations can be performed in polynomial time, which usually depends on the representation of the numbers. Thus, a polynomial time algorithm in the unit cost model will usually provide a polynomial time algorithm for some representation of the numbers different from the usual binary representation.

Current attempts to finding a strongly polynomial time algorithm focus on the Simplex algorithms. Randomised update strategies, for example, have been suggested early on for the simplex technique, but their complexity has not yet been analysed successfully. For the simplest of these techniques, which uses a random edge Pivot rule that chooses a profitable base change uniformly at random, the number of arithmetic steps needed on non-degenerated simplices is merely known to be at least quadratic in the size of the constraint system [5]. Shadow vertex techniques [6, 7] for exploring the simplex allowed for randomised polynomial time procedures for the approximation [6] and computation [7] of the solution to linear programmes.

Parity and Payoff Games are finite two player zero sum games of infinite duration. They are played on labelled directed graphs, whose vertices are partitioned into two sets of vertices owned by two players with opposing objectives. Intuitively, they are played by placing a pebble on the game graph. In each step, the owner of a vertex chooses a successor vertex of the digraph. This way, an infinite run of the game is constructed, where the objectives of the players is to minimise and maximise the average payoff of the moves in mean payoff games, or to enforce parity or imparity of the highest colour occurring infinitely many times in parity games, respectively. These games play a central role in model checking [9–13], satisfiability checking [11, 9, 14, 15], and synthesis [16, 17], and numerous algorithms for solving them have been studied [9, 18–34]. Mean payoff games [34–36] have further applications in economic game theory.

The complexity of solving parity games is equivalent to the complexity of μ-calculus model checking [9]. The complexity of solving parity and mean payoff games is known to be in the intersection of UP and coUP [22], but its membership in P is still open. Up to a recent complexity analysis by Friedman [37], strategy improvement algorithms [25–29] have been considered candidates for a polynomial running time. To the best of my knowledge, all algorithms proposed so far for solving parity or mean payoff games are now known not to be in P.

The Reduction proposed in this paper reduces testing if there are states in a mean payoff game with value ≥ 0 to solving a linear programming problem. It is simple to extend this non-emptiness test to finding the 0 mean partition of a mean payoff game and hence to solving parity and mean payoff games.

Our reduction goes through an intermediate representation of the non-emptiness problem to a non-standard optimisation problem that treats minimisation and maximisation quite differently: While minimisation is represented in a standard way — replacing minimisation $v = \min\{t_1, t_2, \ldots, t_n\}$ over n terms by n inequalities $v \leq t_i$ and maximising over the possible outcome — maximisation is replaced by an approximation within small margins. We use $v = \log_b(b^{t_1} + b^{t_2} + b^{t_n})$ instead of $v = \max\{t_1, t_2, \ldots, t_n\}$, which keeps the error of the operation sufficiently small if the basis b is big enough:

$$\max\{t_1, t_2, \ldots, t_n\} \leq \log_b \left(b^{t_1} + b^{t_2} + b^{t_n}\right) \leq \log_b n + \max\{t_1, t_2, \ldots, t_n\}.$$

From there it is a small step to building a linear programme that computes *the exponent* of the solution to this non-standard optimisation problem.

Starting from parity games or from mean payoff games with a binary representation of the edge weights, this linear programme is cheap to compute in the unit cost model (bi-linear in the number of vertices and edges and linear in the size of the representation), but the constants are exponential in the edge weights of the mean payoff game, and doubly exponential in the number of colours for parity games.

The benefit of the proposed reduction for game solving is therefore not immediate, but depends on future results in linear programming. There has, however, been recent progress in the quest for polynomial time algorithms in the unit cost model, or even strongly polynomial algorithms. Shadow vertex techniques [6, 7], for example, seem to be promising candidates; Kelner and Spielman [7] use this quest as the main motivation for their randomised polynomial time algorithm.

But the benefit of the reduction is bi-directional. While solving finite games of infinite duration automatically profits from future progress in the theory of linear programming, linear programming profits from the problem: The proposed reduction contributes an important natural problem class that can be reduced to linear programming, but requires a polynomial time algorithm in the unit cost model, because the constants are too large for efficient binary representation. Opening up a range of model checking problems to linear programming, the proposed reduction lifts the problem of finding such algorithms from a problem of mere academic interest to one of practical importance. Furthermore, it is

unintuitive that the complexity of inherently discrete combinatorial problems like μ-calculus model checking or solving parity games should depend on the cost model of arithmetic operations. Provided a polynomial time algorithm for solving linear programming problems in the unit cost model is found, it thus seems likely that a polynomial time algorithm for solving parity games can be inferred.

2 Finite Games of Infinite Duration

Finite games of infinite duration (ω-games) are played by two players, a Maximiser and a Minimiser, with opposing objectives. ω-games are composed of a finite arena and an evaluation function.

Arenas. A finite *arena* is a triple $\mathcal{A} = (V_{\max}, V_{\min}, E)$ consisting of

- a set $V = V_{\max} \cup V_{\min}$ of vertices that is partitioned into two disjoint sets $V_{\max}$ and $V_{\min}$, called the vertices owned by the *Maximiser* and *Minimiser*, respectively, and
- a set $E \subseteq V \times V$ of edges, such that (V, E) is a directed graph without sinks.

Plays. Intuitively, a game is played by placing a token on a vertex of the arena. If the token is on a vertex $v \in V_{\max}$, the Maximiser chooses an edge $e = (v, v') \in E$ originating in v to a vertex $v' \in V$ and moves the token to v'. Symmetrically, the Minimiser chooses a successor in the same manner if the token is on one of her vertices $v \in V_{\min}$. In this way, they successively construct an infinite *play* $\vartheta = v_0 v_1 v_2 v_3 \ldots \in V^\omega$.

Strategies. For a finite arena $\mathcal{A} = (V_{\max}, V_{\min}, E)$, a (memoryless) *strategy* for the Maximiser is a function $f : V_{\max} \to V$ that maps every vertex $v \in V_{\max}$ of the Maximiser to a vertex $v' \in V$ such that there is an edge $(v, v') \in E$ from v to v'. A play is called *f-conform* if every decision of the Maximiser in the play is in accordance with f. For a strategy f of the Maximiser, we denote with $\mathcal{A}_f = (V_{\max}, V_{\min}, E_f)$ the arena obtained from $\mathcal{A}$ by deleting the transitions from vertices of the Maximiser that are not in accordance with f. The analogous definitions are made for the Minimiser.

Mean Payoff Games. A *mean payoff game* is a game $\mathcal{M} = (V_{\max}, V_{\min}, E, w)$ with arena $\mathcal{A} = (V_{\max}, V_{\min}, E)$ and a weight function $w : E \to \mathbb{Z}$ from the edges of the mean payoff game to the integers. Each play $\vartheta = v_0 v_1 v_2 \ldots$ of a mean payoff game is evaluated to $\mathit{value}(\vartheta) = \liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} w\big((v_{i-1}, v_i)\big)$.

As a variant, we can allow for real valued weight functions $w : E \to \mathbb{R}$. We then refer explicitly to a *real valued mean payoff game*.

The objective of the Maximiser and Minimiser are to maximise and minimise this value, respectively. For single player games where all vertices are owned by one player (or, likewise, all vertices owned by the other player have exactly one

successor), the optimal strategy for this player from some vertex v is to proceed to a cycle with maximal or minimal average weight a and henceforth follow it. The outcome $value(v) = a$ of this game when started in v is called the value of v. Mean payoff games are memoryless determined:

Proposition 1. *[34–36] For every real valued mean payoff game $\mathcal{M}$, there are Minimiser and Maximiser strategies f and g, respectively, such that the value of every vertex in $\mathcal{M}_f$ equals the value of every vertex in $\mathcal{M}_g$.* □

The 0 *mean partition* $M_{\geq 0} = \{v \in V \mid value(v) \geq 0\}$ of a mean payoff game is thus well-defined. We say that a vertex v is *winning* for the Maximiser if it is in $M_{\geq 0}$, and winning for the Minimiser otherwise.

Corollary 1. *[34–36] The 0 mean partition is well-defined, and both players have winning strategies for their respective winning region $M_{\geq 0}$ and $V \setminus M_{\geq 0}$.* □

Solving a mean payoff game can be reduced to finding the 0 mean partition of a number of games played on sub-arenas with a slightly adjusted weight function, because the average weight of a cycle is a multiple of $\frac{1}{n}$ for some $n \leq |V|$, and it hence suffices to know that the value is within an interval $[\frac{i}{n^2}, \frac{i+1}{n^2}[$ for some integer $i \in \mathbb{Z}$.

Corollary 2. *A mean payoff game with n vertices and maximal absolute edge weight a can be solved in time $O\big(n \log(a+n)\big)$ when using an oracle for the construction of 0 mean partitions.* □

Parity Games. A *parity game* is a game $\mathcal{P} = (V_{\max}, V_{\min}, E, \alpha)$ with arena $\mathcal{A} = (V_{\max}, V_{\min}, E)$ and a colouring function $\alpha : V \to \mathcal{C} \subset \mathbb{N}$ that maps each vertex of $\mathcal{P}$ to a natural number. $\mathcal{C}$ denotes the finite set of colours.

Each play is evaluated by the highest colour that occurs infinitely often. The Maximiser wins a play $\vartheta = v_0 v_1 v_2 v_3 \ldots$ if the highest colour occurring infinitely often in the sequence $\alpha(\vartheta) = \alpha(v_0)\alpha(v_1)\alpha(v_2)\alpha(v_3)\ldots$ is even, while the Minimiser wins if the highest colour occurring infinitely often in $\alpha(\vartheta)$ is odd. Without loss of generality, we assume that the highest occurring colour is bounded by the number of vertices in the arena. It is simple to reduce solving parity games to finding the 0 mean partition of a mean payoff game: One can simply translate a colour c to the weight $|V|^c$ [34].

Corollary 3. *[34] Parity games are memoryless determined, and solving them can be reduced in time $O(m\,n)$ to solving the 0 mean partition problem of a mean payoff game with the same arena, such that the Minimiser and Maximiser have the same winning regions and winning strategies.* □

3 Reduction

In this section, we describe a reduction from finding the 0 mean partition of a mean payoff game, to which solving parity and payoff games can be reduced in

polynomial time by Corollaries 2 and 3, to solving a linear programming problem. We first focus on the slightly simpler problem of testing $M_{\geq 0}$ for emptiness, and reduce this question to a linear programming problem.

The first important observation for our reduction is that membership in $M_{\geq 0}$ is invariant under increasing the weight function slightly: If every edge weight in a mean payoff game with n vertices is increased by some value in $[0, \frac{1}{n}[$, then the weight of every cycle is increased by a value in $[0, 1[$, and hence non-negative if, and only if, the original integer valued weight of the cycle is non-negative.

Lemma 1. *If we increase the weight function of an (integer valued) mean payoff game $\mathcal{M} = (V_{\max}, V_{\min}, E, w)$ with $n = |V|$ vertices for every edge by some non-negative value $< \frac{1}{n}$, then the same cycles as before have non-negative weight in the resulting real valued mean payoff game, and the 0 mean partition does not change.* □

This observation is used to replace maximisation in a natural representation of the objectives of both players in a mean payoff game (Subsection 3.1) by a logarithmic expression in Subsection 3.2, which is subsequently translated into a boundedness test for a linear programming problem in Subsection 3.3. In Subsection 3.4, we show how this boundedness test, which refers to a non-emptiness test of $M_{\geq 0}$, can be adjusted to a bounded optimisation problem that provides $M_{\geq 0}$, and discuss the complexity of the transformations in Subsection 3.5. An example that illustrates these transformations is provided in Section 4.

3.1 Basic Inequations

We now devise a set of inequalities that have a non-trivial solution if, and only if, $M_{\geq 0}$ is non-empty. For our reduction, we extend addition from $\mathbb{R}$ to $\mathbb{R} \cup \{-\infty\}$ in the usual way by choosing $a + (-\infty) = -\infty = -\infty + a$ for all $a \in \mathbb{R} \cup \{-\infty\}$. This motivates the definition of a family of *basic inequalities* for a mean payoff game $\mathcal{M} = (V_{\max}, V_{\min}, E, w)$ that contains one inequality

$$v \leq w\big((v, v')\big) + v'$$

for every edge (v, v') originating from a Minimiser vertex $v \in V_{\min}$, and one inequality

$$v \leq \max\big\{w\big((v, v')\big) \mid v' \in suc(v)\big\} + c_v$$

for every Maximiser vertex $v \in V_{\max}$, where $suc(v)$ denotes the set of successor vertices of v, and each $c_v \in [0, \frac{1}{|V|}[$ can be any sufficiently small slip value (cf. Lemma 1).

Every such system of inequalities has a trivial solution that assigns $-\infty$ to every vertex; but it also has a real valued solution for all vertices in $M_{\geq 0}$.

Lemma 2. *For every such system of inequalities for a mean payoff game $\mathcal{M} = (V_{\max}, V_{\min}, E, w)$, a vertex $v \in V$ has a real valued solution if, and only if, v is in $M_{\geq 0}$.*

Proof. '$\Leftarrow$:' Let us fix an optimal strategy for the Maximiser in the mean payoff game and consider the system of inequalities that contain one equation $v \leq w(e) + v'$ for every edge of the resulting singleton game. (A solution to this set of inequalities is obviously a solution to the original set of inequalities.) This set of inequalities has obviously a solution that is real valued for every vertex $v \in M_{\geq 0}$ (and sets $v = -\infty$ for every vertex v not in $M_{\geq 0}$)[1].

'$\Rightarrow$' A real valued solution for a vertex v defines a strategy for the Maximiser that witnesses $\mathit{value}(v) \geq 0$: If $v \leq \max\{w\big((v,v')\big) \mid v' \in \mathit{suc}(v)\} + c_v$ holds true, then $v \leq w\big((v,v')\big) + v' + c_v$ holds for some $v' \in \mathit{suc}(v)$ in particular, and we choose a Maximiser strategy that fixes such a successor for every Maximiser vertex. By a simple inductive argument, every vertex u reachable from v in the resulting singleton game is real valued, and, for every cycle reachable from v, the sum of the edge weights and vertex slips is non-negative. As the sum of the vertex slips is strictly smaller than 1 in every cycle, the sum of the edge weights is strictly greater than -1, and hence non-negative. □

Naturally, having one real valued solution implies having unbounded solutions, because adding the same value $r \in \mathbb{R}$ to every value of a solution provides a new solution.

3.2 Logarithmic Inequations

These observations set the ground for a reduction to linear programming: For a sufficiently large basis $b > 1$, $\log_b \sum_{v' \in \mathit{suc}(v)} b^{w((v,v'))}\, b^{v'}$ equals $\max\{w\big((v,v')\big) + v' \mid v' \in \mathit{suc}(v)\} + c_v$ for some slip value $c_v \in [0, \frac{1}{n}[$, because

$$\max_{v' \in \mathit{suc}(v)} \{w\big((v,v')\big) + v'\} \leq \log_b \sum_{v' \in \mathit{suc}(v)} b^{w((v,v'))}\, b^{v'} \leq \log_b |\mathit{suc}(v)| + \max_{v' \in \mathit{suc}(v)} \{w\big((v,v')\big) + v'\}$$

holds true. (For the extension to $\mathbb{R} \cup \{-\infty\}$ we use the usual convention $b^{-\infty} = 0$ and $\log_b 0 = -\infty$.) Choosing a basis $b > {n_{out}}^{|V|}$ that is greater than the $|V|$-th power of the maximal out-degree n_{out} of Maximiser vertices guarantees $\log_b i < \frac{1}{n}$, and hence that the small error caused by moving from minimisation to the logarithm of the sum of the exponents is within the margins allowed for by Lemma 1.

Corollary 4. *The system of inequalities consisting of the Minimiser inequalities and the adjusted Maximiser inequalities have a real valued solution for a vertex $v \in V$ if, and only if, $v \in M_{\geq 0}$.* □

[1] Starting with the digraph with states $M_{\geq 0}$ and the respective edges defined by the fixed Maximiser strategy, we can apply the following algorithm until values are assigned to all vertices in $M_{\geq 0}$: (1) pick a vertex v in a leaf component of the digraph that is the minimum of the weighted distance to plus the value assigned to any vertex v' that is already removed from the graph (or an arbitrary value if no such vertex exists), and then (2) remove v from the graph.

Note that the reduction uses estimation from below (through the inequality) as well as estimations from above (through the slip) at the same time, which is sound only because the slip values are within the small margins allowed for by Lemma 1.

3.3 Linear Inequations

The resulting optimisation problem can be translated into a standard linear programming problem by a simple space transformation: As the exponential function $v \mapsto b^v$ is a strictly monotone ascending mapping from $\mathbb{R} \cup \{-\infty\}$ onto $\mathbb{R}^{\geq 0}$, we can simply replace the Minimiser inequalities by

$$b^v \leq b^{w((v,v'))} \cdot b^{v'},$$

for every edge (v, v') originating from a Minimiser vertex $v \in V_{\min}$, and the adjusted Maximiser inequalities by

$$b^v \leq \sum_{v' \in suc(v)} b^{w((v,v'))}\, b^{v'}$$

for every vertex $v \in V_{\max}$ owned by the Maximiser, and require $b^v \geq 0$ for all vertices $v \in V$ of the game.

Reading the b^v as variables, this provides us with a linear constraint system

$$A\mathbf{x} \leq \mathbf{0}, \text{ subject to } \mathbf{x} \geq \mathbf{0},$$

and Corollary 4 implies that this constraint system has a solution different from $\mathbf{x} = \mathbf{0}$ if, and only if, $M_{\geq 0}$ is non-empty for the defining mean payoff game. As every positive multiple of a solution to $A\mathbf{x} \leq \mathbf{0}$ and $\mathbf{x} \geq \mathbf{0}$ is again a solution, this implies the following corollary:

Corollary 5. *The resulting linear programme* maximise $\mathbf{1}^T\mathbf{x}$ for $A\mathbf{x} \leq \mathbf{0}$ and $\mathbf{x} \geq \mathbf{0}$ *is unbounded if $M_{\geq 0}$ is non-empty, and the constraint system has $\mathbf{x} = \mathbf{0}$ as the only solution if $M_{\geq 0} = \emptyset$ is empty.* □

3.4 From Qualitative to Quantitative Solutions

While solving the linear programme introduced in the previous subsection answers only the qualitative question of whether the linear programming problem is bounded, and hence if $M_{\geq 0}$ is non-empty, it is simple to extend the approach to a qualitative solution that provides us with $M_{\geq 0}$ and a strategy for the Maximiser that witnesses this. To achieve this, it suffices to bound the value of every vertex from above, for example, by adding a constraint $\mathbf{x} \leq \mathbf{1}$, or any other constraint $\mathbf{x} \leq \mathbf{d}$ for some constant vector $\mathbf{d} > \mathbf{0}$. (Where $>$ for the vector requires $>$ for every row.)

Proposition 2. *For every constant vector* $\mathbf{d} > \mathbf{0}$*, the solution to the linear programming problem* maximise $\mathbf{c}^T\mathbf{x}$ for $A\mathbf{x} \leq \mathbf{0}$, $\mathbf{x} \leq \mathbf{d}$, and $\mathbf{x} \geq \mathbf{0}$ *assigns a value* $\neq 0$ *to a variable if, and only if, it is in* $M_{\geq 0}$*. A witnessing strategy for the Maximiser in the defining mean payoff game can be inferred from the solution.*

Proof. For the solution of the linear programming problem it holds that if a Maximiser vertex v has some successor with non-zero value, or if a Minimiser vertex v has only non-zero successors, than the value b^v assigned to v by the solution is also non-zero. (Otherwise we could increase it, and hence $\mathbf{1}^T\mathbf{x}$, without changing any other value.) Hence, the logarithms of the solution define a solution to the system of logarithmic inequalities from the previous subsection, and we can infer a witnessing strategy for the Maximiser as described in the proof of Lemma 2.

Now consider a solution to the new linear programming problem defined by the sub-game of the mean payoff game that contains only the vertices with 0 values. If it had a solution different to $\mathbf{0}$, we could increase the solution of the linear programming problem we started with by ε times the solution of the new liner programming problem for a sufficiently small $\varepsilon > 0$. Hence $\mathbf{0}$ is the only solution to the new problem, and therefore there is no real valued solution for the basic or logarithmic inequalities defined by this sub-game. By Corollary 1 the Minimiser has thus a witnessing strategy for the 0 mean partition in the sub-game, which is also a witnessing strategy in the full game. □

3.5 Translation Complexity

The proposed translation of a given mean payoff game to a linear programming problem is cheap in the unit cost model:

Proposition 3. *A mean payoff game* $\mathcal{M}$ *with* n *vertices and* m *edges, and edge weights represented in binary can be translated in time* $O(|\mathcal{M}| + n\,m)$ *in the unit cost model. (Where* $|\mathcal{M}|$ *denotes the length of the representation of* $\mathcal{M}$*.)*

Proof. We have to compute the linear constraint $A\mathbf{x} \leq \mathbf{0}$, which requires the computation of the non-zero constants of A, and filling up A with 0s. As the rows of A refer to Maximiser vertices or edges originating from Minimiser vertices, and the columns refer to vertices, the latter requires $O(n\,m)$ steps.

Each edge refers to exactly one non-zero constant in A, and we need to translate the edge weight $w(e)$ to $b^{w(e)}$. We compute $b_0 = b$ (computing b is well within $O(n\,m)$), and then $b_i = b^{(2^i)} = b_{i-1}^2$ for all $i \leq \log_2 \max\{|w(e)| \mid e \in E\}$. $b^{w(e)}$ can then be expressed as a product of the respective b_i if $w(e) > 0$ is positive, as its reciprocal if $w(e) < 0$ is negative, and by 1 if $w(e) = 0$. The required time for the computation of $b^{w(e)}$ is therefore linear in the binary representation of $w(e)$, and computing all constants $b^{w(e)}$ requires $O(|\mathcal{M}|)$ operations. □

The translation of parity games to mean payoff games [34] discussed in Section 2 implies a likewise bound for parity games.

Fig. 1. Figure 1(a) shows a small example parity game. The vertices of the player with the objective to ensure parity are depicted as squares, while the positions of her opponent are depicted as circles. The vertices of the parity game are decorated with their respective colour. The parity game of Figure 1(a) is translated into the mean payoff game $\mathcal{M}$ of Figure 1(b). The edges of $\mathcal{M}$ are decorated with their weights, and the vertices with their name.

Corollary 6. *A parity game $\mathcal{P}$ with n vertices and m edges can be translated in time $O(nm)$ in the unit cost model.* □

Note that this also implies a polynomial bound on the cost of translating a mean payoff game whose edge weights are represented in unary — and hence of parity games with a bounded number of colours — in the Turing model of computation.

Corollary 7. *For parity games with a bounded number of colours and mean payoff games where the edge weights are represented in unary, the reduction results in a linear programme in binary representation that can be constructed in polynomial time.* □

As a result, the known polynomial bounds [3, 4] for solving linear programming problems in the Turing model of computation imply a polynomial bound for these sub-problems.

Corollary 8. *Parity games with a bounded number of colours and mean payoff games whose edge weights are represented unary can be solved in polynomial time.* □

Remark 1. If the algorithm requires non-degenerated linear programmes, then we can first apply the strongly polynomial standard ε-perturbation technique [38].

While the bounds provided by Corollary 8 are not new, they can be considered as a sanity check for new techniques: Besides its potential for parity and mean payoff games in general, the reduction is good enough to infer the relevant known polynomial bounds.

4 Example

This section contains an example reduction from solving the small parity game from Figure 1(a) to a linear programming problem.

Finding the winning region for the player that wins when the highest colour occurring infinitely many times is even can be reduced to finding the 0 mean partition of the mean payoff game from Figure 1(b). By Lemma 2, finding this 0 mean partition reduces to determining which variables can have a real value in a solution to any set

$$\begin{array}{lll} v_1 \leq \max\{v_2 - 1,\ v_4 - 1\} + c_{v_1} & v_3 \leq v_1 & v_4 \leq v_2 - 1 \\ v_2 \leq \max\{v_1,\ v_3 + 4,\ v_4 - 1\} + c_{v_2} & v_3 \leq v_4 - 1 & v_4 \leq v_3 + 4 \end{array}$$

of inequations, where $c_{v_1}, c_{v_2} < \frac{1}{4}$ can be any non-negative constant smaller than the reciprocal of the size of the game.

The maximal out-degree of a Maximiser vertex is 3, and choosing a basis b big enough to provide $\log_b 3 < \frac{1}{4}$, which holds for all $b > 3^4$, we can seek a solution to the inequations

$$\begin{array}{lll} v_1 \leq \log_b(b^{v_2 - 1} + b^{v_4 - 1}) & v_3 \leq v_1 & v_4 \leq v_2 - 1 \\ v_2 \leq \log_b(b^{v_1} + b^{v_3 + 4} + b^{v_4 - 1}) & v_3 \leq v_4 - 1 & v_4 \leq v_3 + 4 \end{array}$$

instead by Corollary 4, because $\log_b(b^{v_2} + b^{v_4 - 1}) = \max\{v_2,\ v_4 - 1\} + c_{v_1}$ and $\log_b(b^{v_1 - 1} + b^{v_3 + 4} + b^{v_4}) = \max\{v_1 - 1,\ v_3 + 4,\ v_4\} + c_{v_2}$ holds for some $c_{v_1} \leq \log_b 2 < \frac{1}{4}$ and $c_{v_2} \leq \log_b 3 < \frac{1}{4}$, respectively.

This system of inequations on the domain $[-\infty, \infty[$ can be rewritten as the system

$$\begin{array}{lll} b^{v_1} \leq b^{-1} \cdot b^{v_2} + b^{-1} \cdot b^{v_4} & b^{v_3} \leq b^{v_1} & b^{v_4} \leq b^{-1} \cdot b^{v_2} \\ b^{v_2} \leq b^{v_1} + b^4 \cdot b^{v_3} + b^{-1} \cdot b^{v_4} & b^{v_3} \leq b^{-1} \cdot b^{v_4} & b^{v_4} \leq b^4 \cdot b^{v_3} \end{array}$$

of inequations. Finally, the individual b^{v_i} can be treated as variables after adding the constraints $0 \leq b^{v_1},\ b^{v_2},\ b^{v_3},\ b^{v_4}$.

For finding a witnessing strategy for the Maximiser—and hence a winning strategy for the player that wants to ensure parity in the game from Figure 1(a)—it suffices to add the additional constraint $b^{v_1},\ b^{v_2},\ b^{v_3},\ b^{v_4} \leq 1$ and maximise $b^{v_1} + b^{v_2} + b^{v_3} + b^{v_4}$.

Note that the constraints in the linear programming problem reach $b^4 = 45212176$ for $b = 82$ even in this tiny example.

5 Discussion

The introduced reduction from solving parity and mean payoff games to linear programming opens up the well developed class of linear programming techniques to the analysis of these classes of ω-games. It also links their complexity to the complexity of linear programming in the unit cost model.

As the unit cost complexity of linear programming is not known, there is no immediate *practical* benefit attached to this reduction, but the drawn connections between linear programmes and finite games of infinite durations link two intriguing open problems. The potential benefit for the two areas are quite different in nature: The linear programming community gains a natural and important class of problems that would benefit from a polynomial time algorithm

for linear programming, while the game solving community will automatically profit from future developments of polynomial time algorithms.

References

1. Smale, S.: On the average number of steps of the simplex method of linear programming. Mathematical Programming **27** (1983) 241–262
2. Klee, F., Minty, G.J.: How good is the simplex algorithm? Inequalities III (1972) 159–175
3. Khachian, L.G.: A polynomial algorithm in linear programming. Doklady Akademii Nauk SSSR **244** (1979) 1093–1096
4. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of STOC '84. ACM Press (1984) 302–311
5. Gärtner, B., Henk, M., Ziegler, G.M.: Randomized simplex algorithms on Klee-Minty cubes. Combinatorica **18** (1994) 502–510
6. Spielman, D.A., Teng, S.H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. Journal of the ACM **51** (2004) 385–463
7. Kelner, J.A., Spielman, D.A.: A randomized polynomial-time simplex algorithm for linear programming. In: Proceedings of STOC '06. ACM Press (2006) 51–60
8. Smale, S.: Mathematical problems for the next century. The Mathematical Inteligencer **20** (1998) 7–15
9. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science **27** (1983) 333–354
10. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Proceedings of CAV '93. Volume 2725 of Lecture Notes in Computer Science, Springer-Verlag (1993) 385–396
11. Wilke, T.: Alternating tree automata, parity games, and modal μ-calculus. Bulletin of the Belgian Mathematical Society **8** (2001)
12. de Alfaro, L., Henzinger, T.A., Majumdar, R.: From verification to control: Dynamic programs for omega-regular objectives. In: Proceedings of LICS '01. IEEE Computer Society Press (2001) 279–290
13. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49** (2002) 672–713
14. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Proceedings of ICALP '98. Volume 1443 of Lecture Notes in Computer Science, Springer-Verlag (1998) 628–641
15. Schewe, S., Finkbeiner, B.: Satisfiability and finite model property for the alternating-time μ-calculus. In: Proceedings of CSL '06. Volume 4207 of Lecture Notes in Computer Science, Springer-Verlag (2006) 591–605
16. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Journal of Logical Methods in Computer Science **3** (2007)
17. Schewe, S., Finkbeiner, B.: Synthesis of asynchronous systems. In: Proceedings of LOPSTR '06. Volume 4407 of Lecture Notes in Computer Science, Springer-Verlag (2006) 127–142
18. Emerson, E.A., Lei, C.: Efficient model checking in fragments of the propositional μ-calculus. In: Proceedings of LICS '86. IEEE Computer Society Press (1986) 267–278
19. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proceedings of FOCS '91. IEEE Computer Society Press (1991) 368–377

20. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic **65** (1993) 149–184
21. Browne, A., Clarke, E.M., Jha, S., Long, D.E., Marrero, W.: An improved algorithm for the evaluation of fixpoint expressions. Theoretical Computer Science **178** (1997) 237–255
22. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Information Processing Letters **68** (1998) 119–124
23. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science **200** (1998) 135–183
24. Jurdziński, M.: Small progress measures for solving parity games. In: Proceedings of STACS '00. Volume 1770 of Lecture Notes in Computer Science, Springer-Verlag (2000) 290–301
25. Ludwig, W.: A subexponential randomized algorithm for the simple stochastic game problem. Information and Computation **117** (1995) 151–155
26. Puri, A.: Theory of hybrid systems and discrete event systems. PhD thesis, Computer Science Department, University of California, Berkeley (1995)
27. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Proceedings of CAV '00. Volume 1855 of Lecture Notes in Computer Science, Springer-Verlag (2000) 202–215
28. Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Applied Mathematics **155** (2007) 210–229
29. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: Proceedings of CSL '08. Volume 5213 of Lecture Notes in Computer Science, Springer-Verlag (2008) 368–383
30. Obdržálek, J.: Fast μ-calculus model checking when tree-width is bounded. In: Proceedings of CAV '03. Volume 2725 of Lecture Notes in Computer Science, Springer-Verlag (2003) 80–92
31. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: Dag-width and parity games. In: Proceedings of STACS '06. Volume 3884 of Lecture Notes in Computer Science, Springer-Verlag (2006) 524–436
32. Schewe, S.: Solving parity games in big steps. In: Proceedings of FSTTCS '07. Volume 4805 of Lecture Notes in Computer Science, Springer-Verlag (2007) 449–460
33. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. SIAM Journal of Computing **38** (2008) 1519–1532
34. Zwick, U., Paterson, M.S.: The complexity of mean payoff games on graphs. Theoretical Computer Science **158** (1996) 343–359
35. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory **2** (1979) 109–113
36. Gurvich, V.A., Karzanov, A.V., Khachivan, L.G.: Cyclic games and an algorithm to find minimax cycle means in directed graphs. USSR Computational Mathematics and Mathematical Physics **28** (1988) 85–91
37. Friedmann, O.: A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it. In: Proceedings of LICS 2009
38. Megiddo, N., Chandrasekaran, R.: On the ε-perturbation method for avoiding degeneracy. Operations Research Letters **8** (1989) 305–308