

Solving Ratio Games: Algorithms and Experimental Evaluation


Bram van der Sanden, Marc Geilen, Michel Reniers, and Twan Basten

ES Reports

ISSN 1574-9517

ESR-2018-03
15 October 2018

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems



© 2018 Technische Universiteit Eindhoven, Electronic Systems.
All rights reserved.

<http://www.es.ele.tue.nl/esreports>
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Solving Ratio Games: Algorithms and Experimental Evaluation

Bram van der Sanden*, Marc Geilen*, Michel Reniers*, Twan Basten*[§]

*Eindhoven University of Technology, Eindhoven, The Netherlands

[§]TNO-ESI, Eindhoven, The Netherlands

Email: b.v.d.sanden@tue.nl, m.c.w.geilen@tue.nl, m.a.reniers@tue.nl, a.a.basten@tue.nl

Abstract

Ratio games have a range of applications in the analysis, synthesis, and verification of discrete-event systems. Since the size of these systems is often very large, there is a need for efficient algorithms to solve ratio games. The current state-of-the-art method to solve ratio games is by a reduction to (multiple) mean-payoff games. We introduce a direct solution to solve ratio games by adapting existing algorithms for mean-payoff games to ratio games. Exact algorithms are implemented for natural-number valued graphs. Numerical floating point algorithms are implemented for real-valued graphs. We compare the runtime performance of the algorithms on an extensive test set. Policy iteration turns out to be the most efficient solution in practice. It scales best to large game graphs, despite its theoretical complexity.

I. INTRODUCTION

A ratio game [1] is a two-player infinite game, played on a finite double-weighted directed graph, where each edge e has two associated non-negative weights $w_1(e)$ and $w_2(e)$. In each turn of the game, two players, Player 0 and Player 1, move a pebble on some vertex in the game over some edge to an adjacent vertex. Player 0 wants to maximize the ratio of the sums of the weights w_1 and w_2 in the limit of the infinite path, whereas Player 1 wants to minimize this ratio. Ratio games are a generalization of mean-payoff games [2], where the value of a game is given by the average per move of a single edge weight. If all edge weights $w_2(e)$ have value 1, then the ratio game reduces to a mean-payoff game.

Ratio games can be used to synthesize controllers for discrete-event systems, where cost and productivity vary for different events. Consider for example a manufacturing system that takes unprocessed products, executes a number of operations, and outputs finished products. We want to synthesize a controller that in the long run achieves the highest possible ratio of finished products with respect to the total processing time. This controller chooses the best possible action at each system state. To find the highest ratio that can be achieved, we have to look at periodic executions of the system, found in the cycles of the game graph. In partially-controllable systems, the environment may disturb this periodic execution, in which case the controller has to respond. The environment might for instance influence the duration of operations or the time when products enter the machine. In this setting a system controller is one player, and the environment is its adversary. A controller strategy is needed that maximizes the cycle ratio, given worst-case environmental influences. Synthesis of this controller corresponds to finding an optimal strategy in the ratio game.

Another application of ratio games is the synthesis of controllers for robust reactive systems [1]. These controllers are not only correct with respect to the given specification, but also robust in case of violations of the assumptions made in the specification. If assumptions are violated temporarily, the system must recover to normal operation with as few errors as possible. Robustness is expressed as the ratio of the system error (violation of safety assumptions) to the environment error (violation of assumptions). An optimal controller in a ratio game minimizes this ratio.

The current state-of-the-art method to solve ratio games is the algorithm by Bloem et al. [1], [7]. This algorithm finds an optimal strategy by reduction to a series of mean-payoff games, that are solved using the algorithms from Zwick-Paterson [3] (Fig. 1 ZP conversion). This approach is however very inefficient for larger graphs. Recently, more efficient algorithms have been introduced to solve mean-payoff games [4], [5], [6]. In this paper, we introduce two new algorithms to solve ratio games, by adapting algorithms for mean-payoff games to ratio games (highlighted in green in Fig. 1). The first algorithm is a reduction from ratio games to energy games (Fig. 1 EG conversion), adapted from [6], using the Value-Iteration algorithm [6]. This algorithm has a lower worst-case complexity bound than the ZP conversion algorithm. The second algorithm is an adapted version of the policy-iteration algorithm (Fig. 1 Policy Iteration) for mean-payoff games described by Chaloupka [4], [5]. The policy-iteration algorithm has a higher worst-case complexity than the ZP conversion algorithm, but it directly solves ratio games and turns out to be much faster on most practical cases.

In real-world applications, execution times and costs are often not natural numbers, but rather real numbers. Therefore, we adapt all algorithms for ratio games also for real numbers, implemented with floating point representations. This extension is also a novel contribution with respect to solving mean-payoff games, where typically only natural numbers are considered. The algorithms using real numbers fit more naturally to the problem domain, but yield an approximate solution, due to the rounding errors in floating point arithmetic. We derive an upper bound on the difference between

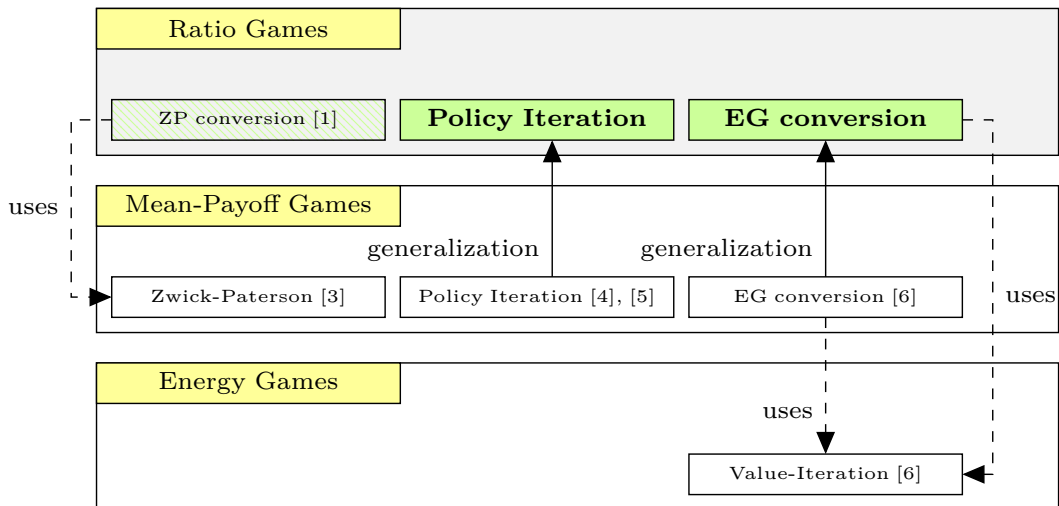


Fig. 1. Games and conversions used for solving ratio games. ZP conversion is an existing algorithm, extended for real-valued weights, and Policy Iteration and EG conversion shown in green are new algorithms, both for natural-valued and real-valued weights.

the computed ratio values and the optimal ratio values. In contrast, the algorithms for natural numbers provide an exact answer. All the developed algorithms are evaluated on an extensive test set, which includes synthetic game graphs and game graphs that relate to throughput optimization in manufacturing systems.

In the remainder, we first introduce game graphs, strategies, and ratio games. Then, we describe the algorithms to solve ratio games and give complexity bounds. The algorithms are compared by experimental evaluation. The experiments show that the policy-iteration algorithm has the best average performance. The same observations were made in the related field of parity games. These games can also be solved by policy-iteration algorithms, which perform better than value-iteration algorithms in practice [8], [9]. Despite having better complexity bounds, value-iteration algorithms often display the worst-case complexity on practical examples [8]. The appendix contains the proofs of the complexity bounds and pseudo code of all algorithms that are evaluated.¹

II. PRELIMINARIES

We define game graphs and strategies adhering to the notation of [6], [1], [7].

Weighted graphs: A *weighted graph* G is a tuple (V, E, w_1, \dots, w_n) , consisting of a finite set V of vertices, a set $E \subseteq V \times V$ of edges, and weight functions $w_i : E \rightarrow \mathbb{W}$, where \mathbb{W} can be \mathbb{N}, \mathbb{Z} , or \mathbb{R} depending on the type of game, assigning weights to edges. The codomains of the weight functions will be clear from the context. We assume that weighted graphs are *total*, which means that for all $v \in V$, there exists a $v' \in V$ such that $(v, v') \in E$.

A finite *path* p in G is a sequence of at least two vertices $v_0 v_1 \dots v_n$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$, provided $n \geq 1$. A *cycle* in G is a finite path $p = v_0 v_1 \dots v_n$ such that $v_0 = v_n$. A cycle $v_0 v_1 \dots v_n$ is *reachable* from v in G , if there exists a path $u_0 u_1 \dots u_m$ in G such that $u_0 = v$ and $u_m = v_0$. A path $v_0 v_1 \dots v_n$ in G is *acyclic*, if $v_i \neq v_j$ for all $0 \leq i < j \leq n$. The weight of a path $v_0 v_1 \dots v_n$ with respect to weight function w_i is given by $\sum_{j=0}^{n-1} w_i(v_j, v_{j+1})$. A cycle in G is *nonnegative*, if its weight is at least 0. The maximal weight W in a weighted graph is defined by $W = \max\{w_i(e) \mid e \in E, i \in \{1, 2\}\}$.

Game graphs: A *game graph* Γ is a tuple $(V, E, w_1, \dots, w_n, \langle V_0, V_1 \rangle)$, where $G^\Gamma = (V, E, w_1, \dots, w_n)$ is a weighted graph and $\langle V_0, V_1 \rangle$ is a partitioning of V into the set V_0 of Player-0, and the set V_1 of Player-1 vertices. An *infinite game* is played by two players who move a pebble along the edges of the graph G for infinitely many rounds. In each round, if the pebble is on some vertex $v \in V_i$, it is moved by player i along an edge $(v, v') \in E$ to a new vertex $v' \in V_j$, where it is player j 's turn, which may be the same player or the opponent. Let V^* (resp. V^ω) denote the set of finite (resp. infinite) sequences over V . A *play* $\pi = v_0 v_1 \dots \in V^\omega$ is an infinite sequence of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. A *strategy* for player i , is a function $\sigma : V^* V_i \rightarrow V$, such that for all finite paths $v_0 v_1 \dots v_n$ with $v_0 \dots v_{n-1} \in V^*$ and $v_n \in V_i$, we have $(v_n, \sigma(v_0 v_1 \dots v_n)) \in E$. We denote by Σ_i the set of strategies for player i . A strategy σ for player i is *memoryless* if $\sigma(pv) = \sigma(p'v)$ for all sequences p and p' and vertex v . We denote by Σ_i^M the set of memoryless strategies of player i . A play $v_0 v_1 \dots$ is *consistent* with strategy σ for player i , if $v_{j+1} = \sigma(v_0 v_1 \dots v_j)$ for all $j \geq 0$, where $v_j \in V_i$. Given an initial vertex $v \in V$, the *outcome* of two strategies $\sigma_0 \in \Sigma_0$ and $\sigma_1 \in \Sigma_1$ in v is the (unique) play $\text{outcome}(v, \sigma_0, \sigma_1)$, that starts in v and is consistent with both σ_0 and σ_1 . Given memoryless strategy $\sigma_i \in \Sigma_i^M$ for player i , we denote by $G_{\sigma_i}^\Gamma = (V, E_{\sigma_i}, w_1 \dots w_n)$ the weighted graph obtained by removing all

¹On acceptance of the paper, the models and code will become available online.

edges $(v, v') \in E$, such that $v \in V_i$ and $v' \neq \sigma_i(v)$. Given memoryless strategies $\sigma_0 \in \Sigma_0^M$ and $\sigma_1 \in \Sigma_1^M$, we denote by $G_{\sigma_0 \cup \sigma_1}^\Gamma = (V, E_{\sigma_0 \cup \sigma_1}, w_1 \dots w_n)$ the weighted graph obtained by removing all edges $(v, v') \in E$ such that $v \in V_i$ and $v' \neq \sigma_i(v)$.

III. RATIO GAMES

We now introduce ratio games, where two players try to optimize the ratio value of the play. Player 0 wants to maximize the ratio value, and Player 1 wants to minimize the ratio value.

Definition 1 (Ratio Game [1]). *A ratio game (RG) is an infinite game played on a game graph $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$, with $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$. The ratio of a play $\pi = v_0 v_1 \dots \in V^\omega$ is the long-run average ratio of the sums of weights w_1 and w_2 , defined as follows*

$$\text{Ratio}(\pi) = \lim_{m \rightarrow \infty} \liminf_{l \rightarrow \infty} \frac{\sum_{i=m}^l w_1(v_i, v_{i+1})}{1 + \sum_{i=m}^l w_2(v_i, v_{i+1})}.$$

The value secured by strategy $\sigma_0 \in \Sigma_0$ in a vertex v is

$$\text{Val}(v, \sigma_0) = \inf_{\sigma_1 \in \Sigma_1} \text{Ratio}(\text{outcome}(v, \sigma_0, \sigma_1)),$$

and the optimal value of a vertex v in Γ is

$$\text{Val}(v) = \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \text{Ratio}(\text{outcome}(v, \sigma_0, \sigma_1)).$$

Strategy σ_0 is optimal if $\text{Val}(v) = \text{Val}(v, \sigma_0)$ for all $v \in V$. The secured and optimal value are defined analogously for strategies of Player 1.

The outer-most limit in the definition of $\text{Ratio}(\pi)$ ensures that only the average behavior in the limit of the infinite path π is relevant. The infimum limit is needed because the sequence of quotients for $l \rightarrow \infty$ can diverge for some plays. To avoid division by zero, 1 is added to the denominator of the ratio function. This does not influence the value $\text{Ratio}(\pi)$ if $\sum_{i=0}^{\infty} w_2(v_i, v_{i+1})$ is infinite. Note that a path corresponding to an infinite play always exists, since we assume the game graph to be total.

Theorem 2 (Optimal memoryless strategies [1]). *Ratio games have optimal memoryless strategies. Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG. For all vertices $v \in V$, we have*

$$\text{Val}(v) = \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \text{Ratio}(\text{outcome}(v, \sigma_0, \sigma_1)) = \inf_{\sigma_0 \in \Sigma_0} \sup_{\sigma_1 \in \Sigma_1} \text{Ratio}(\text{outcome}(v, \sigma_0, \sigma_1)),$$

and there exist memoryless strategies $\sigma_0 \in \Sigma_0^M$ and $\sigma_1 \in \Sigma_1^M$ such that

$$\text{Val}(v) = \text{Val}(v, \sigma_0) = \text{Val}(v, \sigma_1).$$

This means that ratio games are *memoryless determined*, i.e., memoryless strategies are sufficient for optimality, and the optimal (maximal) value that Player 0 can achieve is equal to the optimal (minimal) value that Player 1 can achieve. *Uniform* optimal strategies exist for both players, which means that a unique memoryless strategy can be used to achieve the optimal values, independent of the starting vertex. Fig. 2 shows an example of a ratio game.

Problems in Ratio Games: In this paper we consider the following two problems for a ratio game $(V, E, w_1, w_2, \langle V_0, V_1 \rangle)$:

- 1) *Value Problem.* For each vertex $v \in V$, compute the optimal value $\text{Val}(v)$.
- 2) *Optimal Strategy Synthesis Problem.* Given any vertex $v \in V$, construct an optimal uniform strategy from v for both players.

The value problem can be used to find a ratio guarantee, by inspecting the achievable ratio in each system state. This can be used to give a throughput guarantee. The optimal strategy synthesis can be used to find a controller that achieves the best possible ratio, for instance to guarantee throughput-optimality under external influences. We also define the decision problem, which is used in some approach as an ingredient to solve the value problem:

- 3) *Value Decision Problem.* Given threshold $q \in \mathbb{Q}$, for each vertex $v \in V$, decide if $\text{Val}(v) \geq q$.

In the next sections, we describe algorithms to solve the problems defined. Complexities of these algorithms are given in Table I. Proofs for the complexity values can be found in Appendix A.

IV. REDUCTION TO MEAN-PAYOFF GAMES

The current state-of-the-art method to solve ratio games is a reduction to multiple mean-payoff games [1] (Fig. 1, ZP conversion), and uses the Zwick-Paterson algorithm [3] to solve the mean-payoff games. The reduction is given only for natural-valued weights. We extend the reduction to work also for real-valued weights. The method consists of three ingredients; solving the value decision problem, solving the value problem, and solving the optimal strategy synthesis problem.

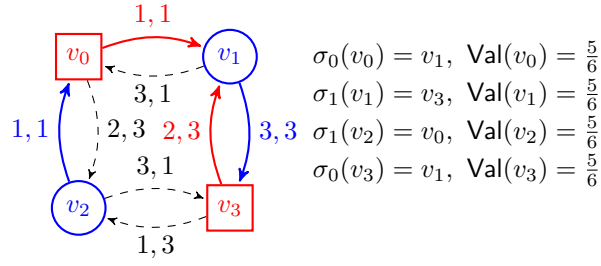


Fig. 2. Ratio game with $v_0, v_3 \in V_0$ and $v_1, v_2 \in V_1$. Each edge is annotated with the weights w_1 and w_2 . The positional optimal strategies of Players 0 and 1 are depicted with solid red and blue edges respectively. Edges that are not used by the strategy are dashed.

| \mathbb{W} | Value | Strategy |
|-----------------------|---|--|
| ZP N | $\mathcal{O}(V ^3 \cdot W^2 \cdot E \cdot \log(V \cdot W))$ | $\mathcal{O}(V ^4 \cdot W^2 \cdot \log(\frac{ E }{ V }) \cdot E \cdot \log(V \cdot W))$ |
| $\mathbb{R}_{\geq 0}$ | $\mathcal{O}(V ^2 \cdot W^2 \cdot E \cdot \rho^{-1})$ | $\mathcal{O}(V ^3 \cdot W^2 \cdot \log(\frac{ E }{ V }) \cdot E \cdot \rho^{-1})$ |
| EG N | $\mathcal{O}((\log(V) + \log(W)) \cdot V ^2 \cdot W^2 \cdot E)$ | $\mathcal{O}((\log(V) + \log(W) + \log(\rho)) \cdot V ^2 \cdot W^2 \cdot E)$ |
| $\mathbb{R}_{\geq 0}$ | $\mathcal{O}((\log(V) + \log(W) + \log(\rho)) \cdot V ^2 \cdot W^2 \cdot E)$ | $\mathcal{O}((\log(V) + \log(W) + \log(\rho)) \cdot V ^2 \cdot W^2 \cdot E)$ |
| PI N | $\mathcal{O}(V ^{10} \cdot E \cdot W^3)$ | $\mathcal{O}(V ^9 \cdot E \cdot W^3 \cdot \rho^{-2})$ |
| $\mathbb{R}_{\geq 0}$ | $\mathcal{O}(V ^{10} \cdot E \cdot W^3)$ | $\mathcal{O}(V ^9 \cdot E \cdot W^3 \cdot \rho^{-2})$ |

TABLE I

COMPLEXITIES OF THE ALGORITHMS DESCRIBED. “VALUE” STANDS FOR THE VALUE PROBLEM, AND “STRATEGY” FOR THE OPTIMAL STRATEGY SYNTHESIS PROBLEM. DERIVATIONS OF THE COMPLEXITIES ARE GIVEN IN APPENDIX A. FOR REAL-WEIGHTED GRAPHS ($\mathbb{W} = \mathbb{R}_{\geq 0}$), A PARAMETER ρ IS USED THAT DENOTES THE SMALLEST DIFFERENCE WHEN COMPARING RATIOS.

a) *Value Decision Problem:* An important ingredient in this approach is a decision procedure on vertex values in ratio games. Given a positive ratio $\frac{a}{b} \in \mathbb{Q}_{\geq 0}$ and a vertex $v \in V$, this procedure compares $\text{Val}(v)$ to $\frac{a}{b}$. First, we create a mean-payoff game $\Gamma_{MPG} = (V, E, w, \langle V_0, V_1 \rangle)$ with payoff function $w(e) = b \cdot w_1(e) - a \cdot w_2(e) \in \mathbb{Z}$ from ratio game $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$. Let $\text{Val}(v)$ be the value in ratio game Γ , and $\text{Val}_{MPG}(v)$ be the value in Γ_{MPG} (for a definition, see [6]). Now $\text{Val}(v) \geq \frac{a}{b}$ iff $\text{Val}_{MPG}(v) \geq 0$.

The crucial observation here is the following equivalence. Given a positive ratio $\frac{a}{b}$, we can compare this ratio to the cycle ratio of some cycle $c = v_0 \dots v_n$ with $n > 0$, and rewrite it to a decision on a mean-payoff value:

$$\begin{aligned} & \frac{\sum_{i=0}^{n-1} w_1(v_i, v_{i+1})}{\sum_{i=0}^{n-1} w_2(v_i, v_{i+1})} \geq \frac{a}{b} \\ \Leftrightarrow & \frac{b \sum_{i=0}^{n-1} w_1(v_i, v_{i+1}) - a \sum_{i=0}^{n-1} w_2(v_i, v_{i+1})}{b \sum_{i=0}^{n-1} w_2(v_i, v_{i+1})} \geq 0 \\ \Leftrightarrow & \frac{\sum_{i=0}^{n-1} (b \cdot w_1(v_i, v_{i+1}) - a \cdot w_2(v_i, v_{i+1}))}{n} \geq 0. \end{aligned}$$

b) *Value Problem:* Given this decision procedure, we can compute the optimal value of each vertex. We compute $\text{Val}(v)$ for each vertex $v \in V$ by a binary search on the set of possible outcomes. For natural-valued weights, we get an exact answer, and for real-valued weights, we can obtain an approximation within some desired bound ρ of the actual value.

c) *Optimal Strategy Synthesis Problem:* The optimal strategy synthesis problem can be solved using the group testing technique [3], and the procedure that computes the optimal value of each vertex. For each vertex, the algorithm iteratively eliminates half of the outgoing edges and recomputes the value of the vertex. If the value stays the same, the optimal strategy does not need to use any of the removed edges. Else, one of the edges in the removed edges is an optimal edge. Either way, we can restrict the attention to half of the outgoing edges. This procedure is repeated until an edge corresponding to an optimal strategy is found.

V. REDUCTION TO ENERGY GAMES

There are more efficient algorithms to find optimal strategies in mean-payoff games, than the group testing technique described in Section IV. Among the algorithms with the lowest complexity bounds there is a reduction to energy games (Fig. 1, EG conversion and Value-Iteration). We generalize this algorithm for ratio games (Fig. 1, EG conversion), and also introduce the reduction for real-valued weights next to natural-valued weights. The algorithm solves both the value problem and the optimal strategy synthesis problem at the same time, which improves the performance

significantly compared to the method in Section IV. This algorithm has a lower worst-case complexity bound than the policy iteration algorithm, but is slower on average.

Before describing the algorithm, we first introduce energy games, which are used to solve the decision problem. Similar to ratio games and mean-payoff games, energy games also have optimal memoryless strategies [10].

Definition 3 (Energy game). *An energy game (EG) is an infinite game played on a game graph $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$, with $w : E \rightarrow \mathbb{R}$, where the goal of Player 0 is to construct an infinite play $v_0v_1 \dots \in V^\omega$ such that for some initial credit $c \in \mathbb{R}$:*

$$c + \sum_{i=0}^j w(v_i, v_{i+1}) \geq 0 \text{ for all } j \geq 0. \quad (1)$$

Given an initial credit c , play $\pi = v_0v_1 \dots$ is winning for Player 0 if it satisfies Equation (1); otherwise it is winning for Player 1. A vertex $v \in V$ is *winning* for Player i , if there exists an initial credit c and a winning strategy for Player i from v for credit c . Player 0 essentially needs to ensure that all cycles that can be formed by Player 1 have nonnegative weight.

Energy games can be solved efficiently by finding an energy progress measure [6]. An energy progress measure is a function $f : V \rightarrow \mathbb{R}$ such that for all $(v, u) \in E$: $f(v) \geq f(u) - w(v, u)$. Value $f(v)$ is a sufficient credit to ensure that all reachable cycles from a given vertex in a graph are nonnegative. A special value \top is used to denote when the value $f(v)$ for some v becomes larger than the absolute sum of all negative-weighted edges. If $f(v) \neq \top$ on vertex v , then Player 0 has a winning strategy from v , provided an initial credit $f(v)$.

a) *Decision Problem:* The Value-Iteration algorithm [6] can be used to find an energy progress measure, thereby solving the energy game. Although the description assumes weights in \mathbb{Z} , the algorithm can be adapted for weights in \mathbb{R} . This algorithm is used as an ingredient to solve mean-payoff games in [6]. The algorithm performs a binary search on the set of possible outcomes of the game to find the value $\text{Val}(v)$ for each vertex v in the graph. This is done in terms of multiple resolutions to the decision problem, to compare the value of a vertex with a given threshold.

Consider a threshold value $\nu \in \mathbb{Z}$, and assume $n > 0$. Then, we can use the following observation to convert the decision problem for mean-payoff games to the decision problem for energy games [6]; given $v \in V$, decide if v is winning for Player 0. The crux here is the following equivalence, assuming $n > 0$:

$$\frac{\sum_{i=0}^{n-1} w_1(v_i, v_{i+1})}{n} \geq \nu \Leftrightarrow \sum_{i=0}^{n-1} (w_1(v_i, v_{i+1}) - \nu) \geq 0.$$

We generalize this approach to ratio games, using a similar equivalence. Given a ratio $\frac{a}{b}$, we can compare this ratio to the cycle ratio of some cycle v_0, \dots, v_n with $n > 0$, and rewrite it to a decision on an energy game value:

$$\frac{\sum_{i=0}^{n-1} w_1(v_i, v_{i+1})}{\sum_{i=0}^{n-1} w_2(v_i, v_{i+1})} \geq \frac{a}{b} \Leftrightarrow \sum_{i=0}^{n-1} (b \cdot w_1(v_i, v_{i+1}) - a \cdot w_2(v_i, v_{i+1})) \geq 0.$$

Solving the energy game gives an energy progress measure f . For any $v \in V$, if $f(v) \neq \top$, Player 0 has a winning strategy from v in the energy game, and it follows that $\text{Val}(v) \geq \frac{a}{b}$ in the ratio game.

b) *Value and Optimal Strategy Synthesis Problem:* The optimal strategy synthesis problem for ratio games with natural-valued weights is solved by adapting Algorithm 2 in [6]. The adapted algorithm is given in Algorithm 3 in Appendix D.

This algorithm performs a binary search on the set of possible outcomes $S = \{\frac{a}{b} \mid 0 \leq a \leq |V| \cdot W, 0 < b \leq |V| \cdot W\} \cup \{\infty\}$, to find the value of each vertex $v \in V$. Given an interval $[l, r]$, it first determines $m = (l + r)/2$, and considers the two intervals $[l, m]$ and $[m, r]$. Since m may not be a value in S , we instead consider intervals $[l, a_1]$ and $[a_2, r]$, where $a_1, a_2 \in S$. Value $a_1 \in S$ is the largest value in S satisfying $a_1 \leq m$, and $a_2 \in S$ is the smallest value in S satisfying $a_2 \geq m$. Then, we perform four reductions to decisions on energy games, to determine partition $(V_{<a_1}, V_{\geq a_1}, V_{\leq a_2}, V_{>a_2})$. This partition classifies the vertices by their payoff values for Player 0. For instance, set $V_{\geq a_1}$ contains all vertices from which Player 0 secures a payoff at least a_1 in the game. In case $v \in V_{\geq a_1} \cap V_{\leq a_2}$, we know that $\text{Val}(v) = a_1 = a_2$. The optimal strategy for $v_i \in V_i$ is given by $\sigma_i(v_i) = v_j$, where v_j is any outgoing successor vertex consistent with the progress measure. In the recursive steps, we consider the smaller subgraphs $G_{<a_1} = (V_{<a_1}, E \upharpoonright V_{<a_1})$ and $G_{>a_2} = (V_{>a_2}, E \upharpoonright V_{>a_2})$, where $E \upharpoonright U = E \cap (U \times U)$, the restriction of E to U .

When $w_1, w_2 \in \mathbb{R}$, we perform a binary search in S using a parameter ρ that specifies the smallest difference when comparing ratios. At each step, we split a given interval $[l, r]$ into two intervals $[l, m]$ and $[m, r]$, with $m = (l + r)/2$. Given m , we perform a reduction to a decision on an energy game, to find vertex partition $(V_{<m}, V_{\geq m})$ on all vertices in V , where $v \in V_{<m}$ if $\text{Val}(v) < m$, and $v \in V_{\geq m}$ if $\text{Val}(v) \geq m$. Then, we only consider the subgraphs $G_{<m}$ and $G_{\geq m}$. If the width of interval $[l, r]$ is smaller than ρ , we can set the value $\text{Val}(v) = (l + r)/2$ for each vertex v in the subgraph, and determine an optimal strategy.

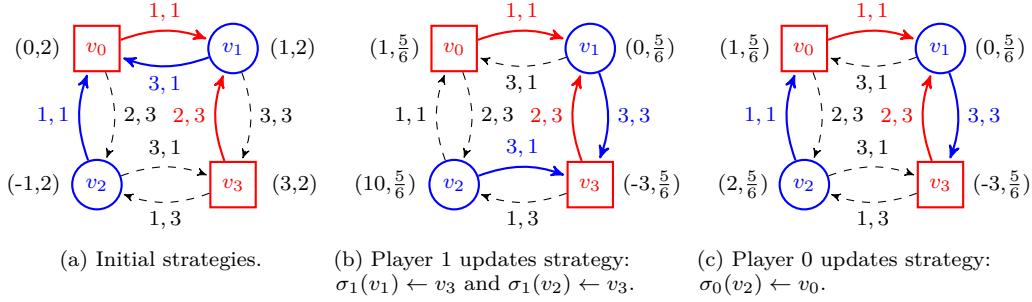


Fig. 3. Illustration of the policy-iteration algorithm. Each node is annotated with distance d and ratio r using (d, r) . Fig. 3c shows the optimal strategies.

VI. POLICY ITERATION

Dhingra and Gaubert [11] introduce a policy-iteration algorithm to compute optimal strategies for both players in mean-payoff games. The algorithm iteratively improves the strategies of both players, until both strategies become optimal. An improved algorithm is given by Chaloupka [5] for natural-valued weights, which we adapt to solve ratio games. We also extend the algorithm to work with real-valued weights.

For natural-valued weights, given vertex v , we define the ratio value $r(v) = r(v)^N / r(v)^D$, where the greatest common divisor of integers $r(v)^N$ and $r(v)^D$ is 1, to refer to the numerator $r(v)^N$ and denominator $r(v)^D$ of the ratio value $r(v)$. For real-valued weights, we define $r(v)^D = 1$, and use only the nominator to store the ratio.

The policy-iteration algorithm for ratio games starts with arbitrary memoryless strategies $\sigma_0 \in \Sigma_0^M$ and $\sigma_1 \in \Sigma_1^M$ for players 0 and 1 respectively. Given current strategy σ_0 , Player 1 finds an optimal strategy σ'_1 to be played against σ_0 , using multiple strategy improvement iterations if needed. Then, Player 0 can make a single iteration to improve its strategy to σ'_0 , after which it is the turn of Player 1 again. The convergence proof of the algorithm requires Player 1 to compute its optimal answer, which makes the policy-iteration algorithm asymmetric [12], [11].

a) Evaluation and Value Propagation: In each iteration, the strategy is first evaluated and then improved. Given current strategies σ_0 and σ_1 , we obtain graph $G_{\sigma_0 \cup \sigma_1}$, where each vertex has one outgoing edge, chosen by the strategies. Each vertex in this graph has a unique path to a unique cycle. In each such cycle, one vertex is picked as a *selected vertex*.

Evaluation produces two vectors of size $|V|$; distance vector d and ratio vector r , where $d \in \mathbb{N}^N$ and $r \in \mathbb{Q}^V$ for natural-valued weights, and $d \in \mathbb{R}^V$ and $r \in \mathbb{R}^V$ for real-valued weights. Given vertex $v \in V$, $r(v)$ is the cycle ratio of the unique cycle reached from v in $G_{\sigma_0 \cup \sigma_1}$, and $d(v)$ is the distance of v to the cycle. This distance is computed as the weight of the unique simple path from v to a selected vertex on the cycle, where each edge has a weight according to the following weight function w' : $w'(v, u) = r(u)^D \cdot w_1(v, u) - r(u)^N \cdot w_2(v, u)$. Recall that for real-valued weights $r(u)^D = 1$, and $r(u)^N$ is used to store the ratio value. The distance vector is used to select between alternatives with identical ratios. After computing d and r , the strategy is improved, which is described next.

b) Strategy Improvement Player 0: Player 0 improves strategy σ_0 , so that for each vertex $v \in V$, the unique cycle reachable from v in $G_{\sigma_0 \cup \sigma_1}$ has the maximum ratio among all cycles reachable from v in G_{σ_1} . Moreover, among cycles with identical cycle ratios, the one with the shortest distance to the cycle is selected. To improve the strategy, given a vertex $v \in V_0$, each edge $(v, u) \in E$ is checked to see whether it satisfies the *strategy improvement condition* of Player 0:

$$r(v) < r(u) \quad \vee \quad (r(v) = r(u) \wedge d(v) < d(u) + w'(v, u)).$$

If yes, then $\sigma_0(v)$ is set to u . If multiple edges satisfy the condition, one with the highest $r(u)$ -value is selected, that improves the strategy the most. Then, the values of r and d are updated and Player 1 may improve her strategy.

c) Strategy Improvement Player 1: Given vertex $v \in V_1$, each edge $(v, u) \in E$ is checked to see whether it satisfies the *strategy improvement condition* of Player 1:

$$r(v) > r(u) \quad \vee \quad (r(v) = r(u) \wedge d(v) > d(u) + w'(v, u)).$$

The strategy improvement technique of Player 1 is similar to the improvement technique of Player 0, but is repeated until there is no more improvement in the given round. If strategy σ_1 is improved, the algorithm lets Player 0 improve his strategy σ_0 again for one iteration. This process is repeated until both players cannot update their strategies. In that case strategies σ_0 and σ_1 are optimal, and $r(v) = \text{Val}(v)$ for each $v \in V$. An illustration of the policy-iteration algorithm is given in Fig. 3.

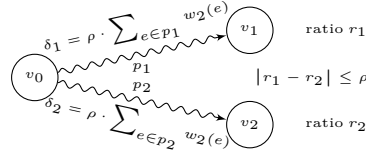


Fig. 4. The ρ relative error in the precision of the ratio values leads to a distance approximation error of δ_1 for path p_1 and δ_2 for path p_2 . The distances are therefore compared using a total relative error margin of $2 \cdot \max(\delta_1, \delta_2)$.

Implementation: The pseudo code of the policy-iteration algorithm is given in Algorithm 5. Some care needs to be taken in the implementation for real-valued weights, to ensure convergence and termination of the policy-iteration algorithm. In the implementation of this algorithm, ratio values are stored as floating-point numbers. The precision of these values is determined by machine precision μ , which gives an upper bound on the relative error due to rounding in floating point arithmetic. We use a parameter $\rho > \mu$, that specifies the upper bound on the relative error when comparing ratios. If the ratios are within this bound, we consider them equal.

In the distance calculation, we assume that the ratio is accurate up to a relative error bound of ρ . In the w' computation, this floating-point ratio is multiplied with the w_2 weights of the edges on the path. Since $r(v)^D = 1$ for the real-valued weights, this multiplication is the only source of errors in the w' computation. We introduce a variable δ that keeps track of the error in the distance approximation, illustrated in Fig. 4. When comparing the distance values of two paths p_1 and p_2 , we take into account the error margin of both distance values, δ_1 and δ_2 , and compare distances considering them equal when the difference does not exceed $2 \cdot \max(\delta_1, \delta_2)$.

Each iteration of the algorithm guarantees that we improve the strategy of both players. Upon termination of the algorithm, we know that the approximation is stable, and the difference between the computed ratio values and the optimal ratio values is bounded by $\varepsilon = 2\rho \cdot |V| \cdot \max(w_2)/mcm(w_2) + \rho \cdot |V|$, where $\max(w_2)$ is the maximal w_2 weight in the graph, and $mcm(w_2)$ is the minimum w_2 cycle mean in the graph. In practice, first a check needs to be done to ensure that $mcm(w_2) > 0$.

Theorem 4. *If a player does not update its strategy (i.e., the improvement condition does not apply to any vertex) then the strategy is ε -optimal against the standing strategy of the opponent, where $\varepsilon = 2\rho \cdot |V| \cdot \max(w_2)/mcm(w_2) + \rho \cdot |V|$, where $\max(w_2)$ is the maximal w_2 weight in the graph, and $mcm(w_2) = \min\{(1/p) \cdot \sum_{i=1}^p w_2(e_i) \mid c = (e_1, \dots, e_p) \text{ is a cycle in the graph}\}$, is the minimum w_2 cycle mean in the graph.*

Proof. We need to prove that if the current strategy of the player is not ε -optimal, then there is some vertex for which the improvement condition holds.

Assume that the *strategy* is not ε -optimal. Assume w.l.o.g. that the player is maximizing the cycle ratio (Player 0). Then there is a vertex u , such that the current strategy leads to a cycle with ratio r , while there is a reachable cycle, say c , with a ratio r' such that $r' - r > \varepsilon$.

Assume (towards a contradiction) that the *improvement condition* does not hold in any vertex of the player. Then along any path in $G_{\sigma_1}^\Gamma$ (the game graph containing for the adversary only the edges corresponding to the standing adversary strategy), the ratios $r(v)$ are ρ -non-increasing (values cannot increase by more than ρ) (otherwise the improvement condition would hold for the source vertex of such an edge). In particular, this implies that on any cycle in $G_{\sigma_1}^\Gamma$, all ratios of adjacent vertices are ρ -equal. For any vertex v reachable from the original vertex u , we know that $r(v) \leq r + \rho \cdot |V| < r' - \varepsilon + \rho \cdot |V|$. In particular we know that this holds for all vertices on cycle c . (Note that given the definition of epsilon, this is strictly smaller than r' .)

Now consider all edges on the cycle c , going backward, and starting from some vertex $v(0)$, in which the standing strategy decides to deviate from cycle c (clearly there must be such a node, otherwise the ratio $r(v)$ would be equal to r' and $r(v) < r'$). We will use $v(k)$ to denote vertex number k in this cycle, counting backward, $r(k)$ the current ratio of vertex $v(k)$, $d(k)$ the current distance of vertex $v(k)$ and $w_i(k+1, k)$ the weight i of the edge from $v(k+1)$ to $v(k)$. We denote by δ_k the error margin used to compare the distance value of $v(k)$, given the path towards the selected node in the current strategy. Since the maximum length of such a path is $|V| - 1$, we can bound δ_k by $\delta_k \leq \delta = \rho \cdot |V| \cdot \max(w_2)$.

If the edge we follow along the cycle is compliant with the current strategy, then $d(k+1) = d(k) + w'(k+1, k) = d(k) + w_1(k+1, k) - r(k)w_2(k+1, k)$. If the edge is not in the strategy, then, since the improvement condition does not hold: $d(k+1) \geq d(k) + w_1(k+1, k) - r(k)w_2(k+1, k) - 2\delta$.

Following along the cycle c , assumed to be of length n , we reach the conclusion that

$$d(n-1) \geq d(0) + \sum_{k=0}^{n-2} w_1(k+1, k) - \sum_{k=0}^{n-2} r(k) \cdot w_2(k+1, k) - (n-1) \cdot 2\delta.$$

Note that the sums range from $k = 0$ to $k = n - 2$, i.e., the whole cycle except for the last edge from $v(0)$ to $v(n - 1)$. We can now check the improvement condition on the edge from $v(0)$ to $v(n - 1)$. The improvement condition holds only if:

$$d(0) < d(n - 1) + w_1(0, n - 1) - r(n - 1)w_2(0, n - 1) - 2\delta.$$

Hence, only if,

$$d(n - 1) > d(0) - w_1(0, n - 1) + r(n - 1)w_2(0, n - 1) + 2\delta.$$

We know that the real cycle ratio of c is $r' > r(k) + \varepsilon - \rho \cdot |V|$ for all k . Therefore:

$$\begin{aligned} \frac{\sum_{e \in c} w_1(e)}{\sum_{e \in c} w_2(e)} &> r(k) + \varepsilon - \rho \cdot |V| \text{ for all } k, \text{ and} \\ \sum_{e \in c} w_1(e) &> \sum_{e \in c} (r(\text{src}(e)) + \varepsilon - \rho \cdot |V|) \cdot w_2(e), \end{aligned}$$

where $\text{src}(e)$ is the source vertex of edge e . Thus:

$$\begin{aligned} d(n - 1) &\geq d(0) + \sum_{k=0}^{n-2} w_1(k + 1, k) - \sum_{k=0}^{n-2} r(k) \cdot w_2(k + 1, k) - (n - 1) \cdot 2\delta \\ &= d(0) + \left(\sum_{e \in c} w_1(e) - w_1(0, n - 1) \right) \\ &\quad - \left(\sum_{e \in c} r(\text{src}(e)) \cdot w_2(e) - r(n - 1) \cdot w_2(0, n - 1) \right) - (n - 1) \cdot 2\delta \\ &> d(0) + \left(\sum_{e \in c} (r(\text{src}(e)) + \varepsilon - \rho |V|) \cdot w_2(e) - w_1(0, n - 1) \right) \\ &\quad - \left(\sum_{e \in c} r(\text{src}(e)) \cdot w_2(e) - r(n - 1) \cdot w_2(0, n - 1) \right) - (n - 1) \cdot 2\delta \\ &= d(0) - w_1(0, n - 1) + r(n - 1) \cdot w_2(0, n - 1) \\ &\quad + (\varepsilon - \rho \cdot |V|) \cdot \sum_{e \in c} w_2(e) - (n - 1) \cdot 2\delta. \end{aligned}$$

In order to arrive at the required contradiction, we also need to show that $(\varepsilon - \rho \cdot |V|) \cdot \sum_{e \in c} w_2(e) - (n - 1) \cdot 2\delta \geq 2\delta$. Here, we use that $\sum_{e \in c} w_2(e) \geq n \cdot \text{mcm}(w_2)$.

$$\begin{aligned} (\varepsilon - \rho \cdot |V|) \cdot \sum_{e \in c} w_2(e) - (n - 1) \cdot 2\delta &\geq 2\delta \\ (\varepsilon - \rho \cdot |V|) \cdot \sum_{e \in c} w_2(e) &\geq n \cdot 2\delta \\ \varepsilon - \rho \cdot |V| &\geq n \cdot 2\rho \cdot |V| \cdot \max(w_2) / \sum_{e \in c} w_2(e) \\ \varepsilon &\geq 2\rho \cdot |V| \cdot \max(w_2) / \text{mcm}(w_2) + \rho \cdot |V|, \end{aligned}$$

which follows from the definition of ε . So the improvement condition must hold for node $v(0)$. This is the required contradiction. We conclude that if the improvement condition does not hold anywhere, then the strategy is ε -optimal. \square

VII. EXPERIMENTAL EVALUATION

We evaluate the runtime performance of the three algorithms on various types of graphs. All algorithms are implemented in Java, as closely as possible to the original formulation. The game graphs are stored and manipulated (for example for constructing subgraphs) using the freely available JGraphT graph library [13].

A. Experimental setup

To run the experiments, we used a computer with Linux with a 2.70GHz Intel i5-4310M CPU processor, and we allocated 4GB of heap space to Java.

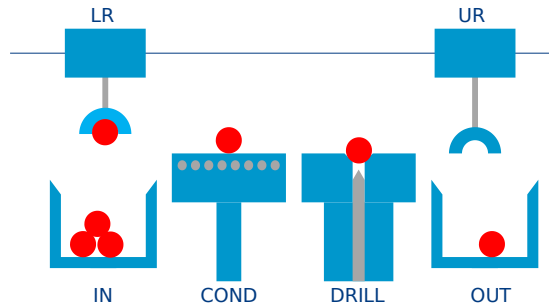


Fig. 5. Twilight system: manufacturing system example with two robots and two production stages.

B. Graph Classes

To test various classes of graphs, we used two synthetic graph generators and a set of application graphs. The same synthetic graph generators are also used by Chaloupka [5] to assess the performance of algorithms solving mean-payoff games.

a) Synthetic graphs: To generate synthetic sparse graphs, we use the graph generator SPRAND [14]. Given the number V of vertices and the edge ratio $m > 1$, SPRAND generates a random graph of V vertices and at most $V \cdot m$ edges, where $V - 1$ edges make up a Hamiltonian cycle. Each edge e on the Hamiltonian cycle satisfies $w_1(e) = 1$ and $w_2(e) = 1$ and the other edges have weights w_1 and w_2 picked uniformly at random from the interval given by $[1, W]$, given a maximal weight W . To test the algorithms, we use graphs with $m = 4$, denoted as SPRAND-4, which are also used in [5]. The vertices are randomly distributed among the two players.

To generate more structured, dense graphs, we use the graph generator TOR [15]. This family consists of 2-dimensional grids with wrap-arounds, where each vertex is connected by an edge to its top neighbor and an edge to its right neighbor. Weights are chosen uniformly at random given a range $[1, W]$.

We vary the number of vertices between 0 and 100 to check the performance on small graphs, where the weights are chosen in the interval $[1, 10]$, so $W = 10$. We also test larger graphs up to 50,000 vertices with weights from $[1, 50]$ to investigate the scalability. For each test configuration, we generate 25 random graph instances. The results show the average execution time of the algorithms over these 25 runs. The test set contains both natural-weighted graphs, and real-weighted graphs. The natural-weighted graphs can be solved by all algorithms, but the implementation using floating-point numbers yields an approximate answer. To compare the results of the numerical floating-point algorithms with the exact algorithms, we use $\rho = 1/(|V|^2)$. This ρ is the smallest possible difference between two values of different vertices.

b) Application graphs: To test realistic application graphs, we created a number of variants of the Twilight system [16] model. The Twilight manufacturing system (Fig. 5) processes balls according to a given recipe. First, a ball is loaded from the input buffer (IN) by the load robot (LR). Then, the ball is put on the conditioner (COND), to heat the ball to a desired temperature. Once heated, the ball is transported to the drill (DRILL) by either the load robot or unload robot (UR) to drill a hole in the ball. After drilling, the unload robot picks up the finished product and brings it to the output buffer (OUT).

The Twilight system is modeled using the formal modeling approach described in [16]. The model describes all resources that are part of the system and the corresponding low-level actions they provide. High-level system operations are modeled as activities. In the Twilight system, there are activities that model the transportation of a ball, and activities that model the processing steps. Requirements are formalized that express constraints on the ordering of activities. These requirements ensure that products follow the recipe, and that the robots do not collide. From this specification, a finite-state machine is constructed that captures all allowed activity orderings. By adding the timing information, a timed state space is generated.

An optimal controller for this system maximizes throughput in terms of processed products over the total executing time. To find such a controller, we transform the timed state space to a ratio game graph. Each activity is modeled as a dispatch and a set of outcomes. The controller player chooses the activity that is dispatched to the machine, and the environment player decides the outcome. Each activity has a reward value of 1 if it outputs a finished product, and 0 otherwise. It also has a delay value that indicates the added execution time. Finding an optimal controller now corresponds to finding an optimal strategy in the created ratio game.

In Tw-1, we take the Twilight system with the possibility that the unload robot needs maintenance after at most ten activities. The controller can choose the most beneficial moment to execute this maintenance. In Tw-2, the drill activity needs recalibration after at most ten activities. Again, the exact moment can be chosen. In Tw-3, the transport of a product within the system has to be assigned to a fixed robot. In Tw-4, we take Tw-3, but extend it also with an option that the robot is left unassigned.

| Name | Graph complexity | | W | Runtime (in sec) | |
|------|------------------|-----------|-----|------------------|-----------|
| | $ V $ | $ E $ | | PI-N | EG-N,ZP-N |
| Tw-1 | 53,021 | 84,305 | 70 | 2.89 | >3600 |
| Tw-2 | 45,334 | 73,260 | 100 | 1.73 | >3600 |
| Tw-3 | 121,445 | 183,124 | 70 | 4.56 | >3600 |
| Tw-4 | 980,567 | 1,515,469 | 70 | 48.85 | >3600 |

TABLE II
EXPERIMENTAL RESULTS FOR THE TWILIGHT GRAPHS.

C. Results

In the description of the experimental results, we abbreviate the policy-iteration algorithm as PI, the reduction to energy games as EG, and the Zwick-Paterson algorithm as ZP. We use N and R to indicate the algorithm for natural-valued weights and real-valued weights respectively. Fig. 6 shows the results for natural-weighted synthetic graphs. Note that we can interpret natural-valued graphs also as real-valued graphs. Figs. 6a and 6c show that ZP is the least scalable on all input games. Although the complexity bound of PI is worse than the bound of EG, in practice, PI is much faster. It scales up to 50,000 vertices with a running time of a few minutes, shown in Figs. 6b and 6d.

We also performed experiments on real-valued graphs, with $\rho = 10^{-5}$. Here we found that ZP-R and EG-R are on average slower than their counterparts ZP-N and EG-N, which can be seen in Fig. 7a and Fig. 7b. This is because of the ρ -approximation, that causes a much deeper recursion depth. When working with natural numbers, we know exactly when to stop, since the minimal distance between two values is known.

PI does not use recursion to find ratio values within ρ , but rather updates the strategies of both players till these strategies become optimal. For small ρ -values, PI-R will make the same choices as PI-N when comparing ratios. This leads to a similar runtime for different small ρ -values, shown in Fig. 6b. Therefore an ρ -value close to the machine precision can be chosen, with no additional penalty on the running time of PI, to obtain a more accurate approximation.

Table II shows the results for the Twilight examples, resembling realistic graphs. EG and ZP were unable to deal with these graphs within an hour. PI can find optimal strategies in all application graphs.

VIII. CONCLUSION

We have introduced two new algorithms to find optimal strategies for ratio games. The energy-game conversion algorithm uses a reduction to energy games, and the policy-iteration algorithm directly operates on the ratio game. The current state-of-the-art uses a reduction to mean-payoff games and the group testing technique to find an optimal strategy.

All algorithms for ratio games we present in this paper can be used on both natural numbers and real numbers. For natural numbers, the algorithms can find an exact solution, but they are not always a realistic abstraction of the problem domain. Therefore, we also designed algorithms that can deal with real numbers, which fits more closely to domains where execution times and costs are typically real numbers. This extension is also a new contribution to solve mean-payoff games with real-valued weights. We carried out an experimental study to evaluate the algorithms on both synthetic graphs and application graphs that relate to throughput optimization in manufacturing systems. In all cases, the policy-iteration algorithm turned out to be the fastest solver, outperforming the other two algorithms by several orders of magnitude.

APPENDIX

In this appendix, we derive the complexity bounds of the three algorithms presented in this paper. First, we introduce some general results about ratio games that are used in the complexity analysis of the algorithms.

A. Preliminaries

The maximal weight W of a ratio game is defined by $W = \max\{w_i(e) \mid e \in E, i \in \{1, 2\}\}$. This weight plays a role in the set of possible ratio values, and in the complexity bounds of the algorithms. For the algorithms operating on real-valued graphs, we introduce a parameter ρ , that determines the upper bound on the relative error when comparing ratios.

The following results describe the possible values for any play. They follow directly from the memoryless determinacy of ratio games, as provided by Theorem 2.

Proposition 5. *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG. For all vertices $v \in V$, and for all memoryless strategies $\sigma_0 \in \Sigma_0^M$ for Player 0, the value $\text{Val}(v, \sigma_0)$ secured by σ_0 in v is at least $\frac{a}{b}$ with $0 \leq a \leq |V| \cdot W$ and $0 < b \leq |V| \cdot W$, if and only if all cycles reachable from v in the weighted graph $G^\Gamma(\sigma_0)$ have a ratio at least $\frac{a}{b}$.*

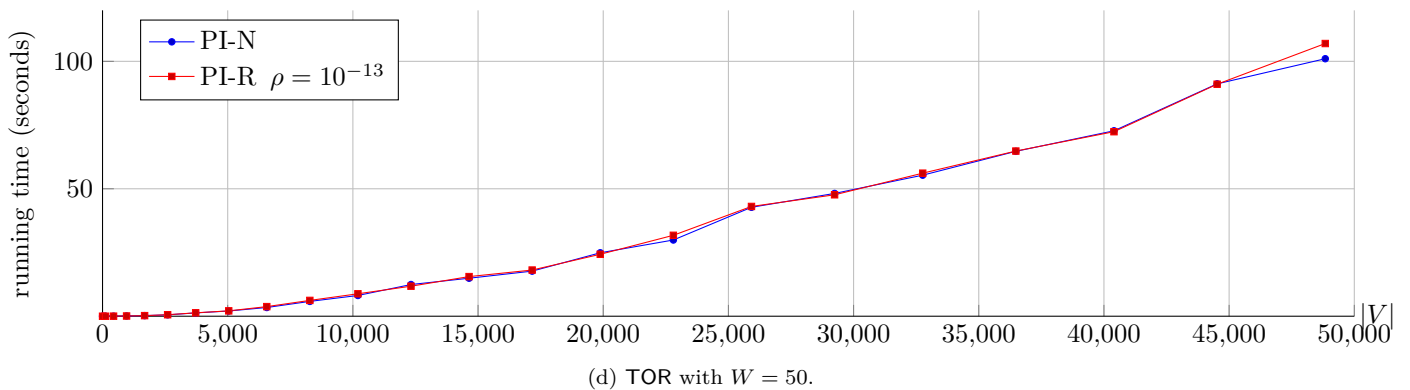
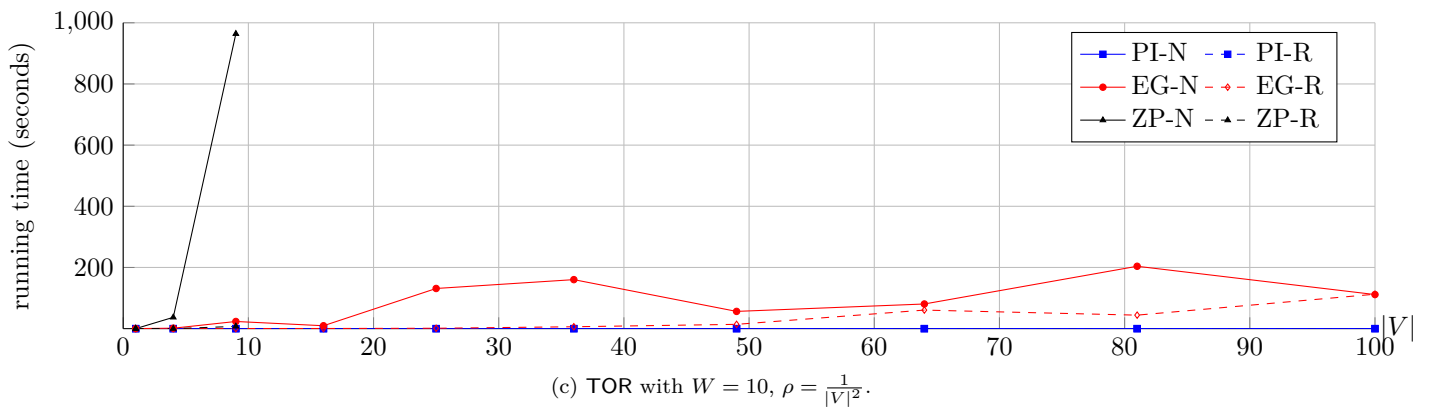
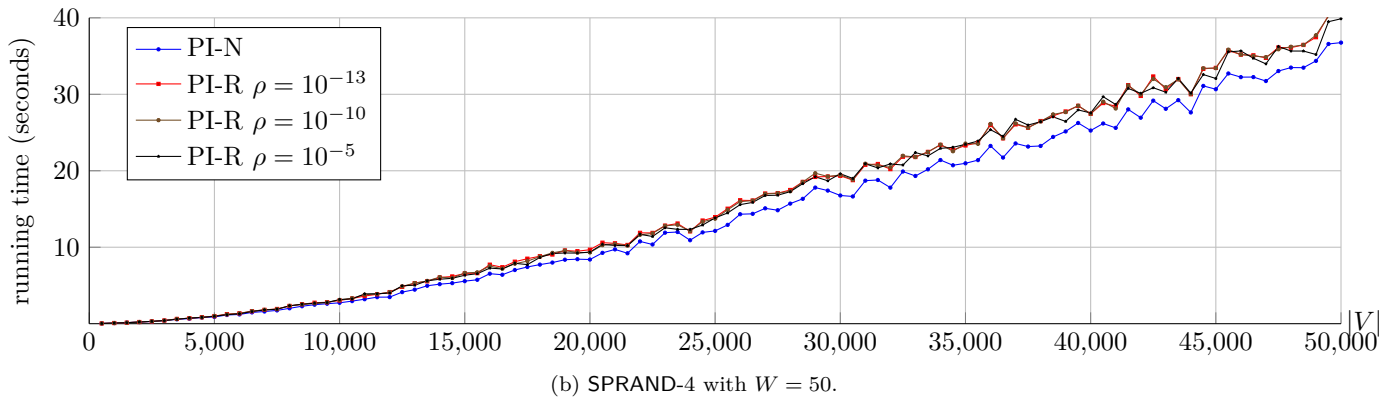
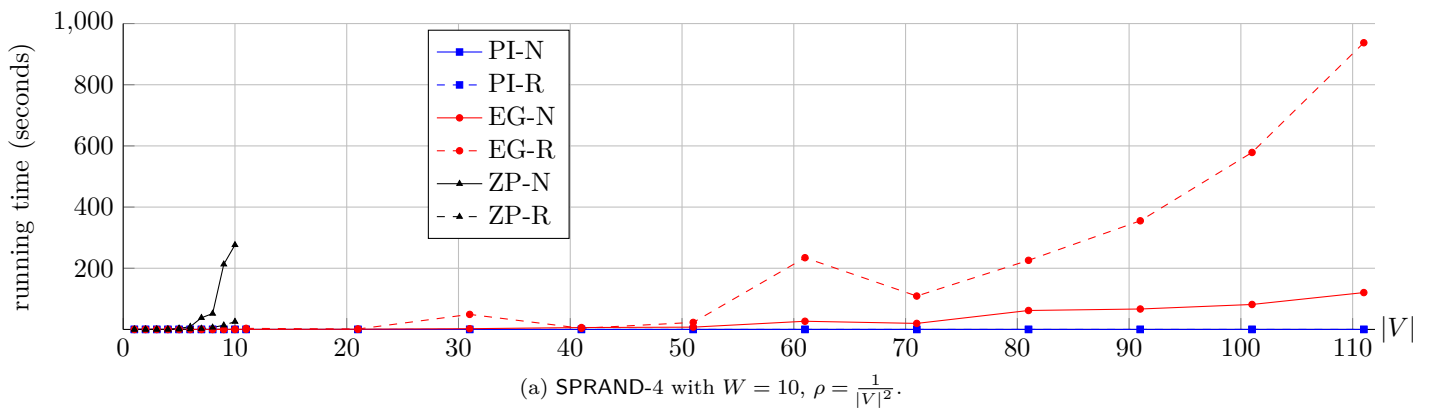


Fig. 6. Experimental results of running the algorithms on natural-weighted synthetic graphs.

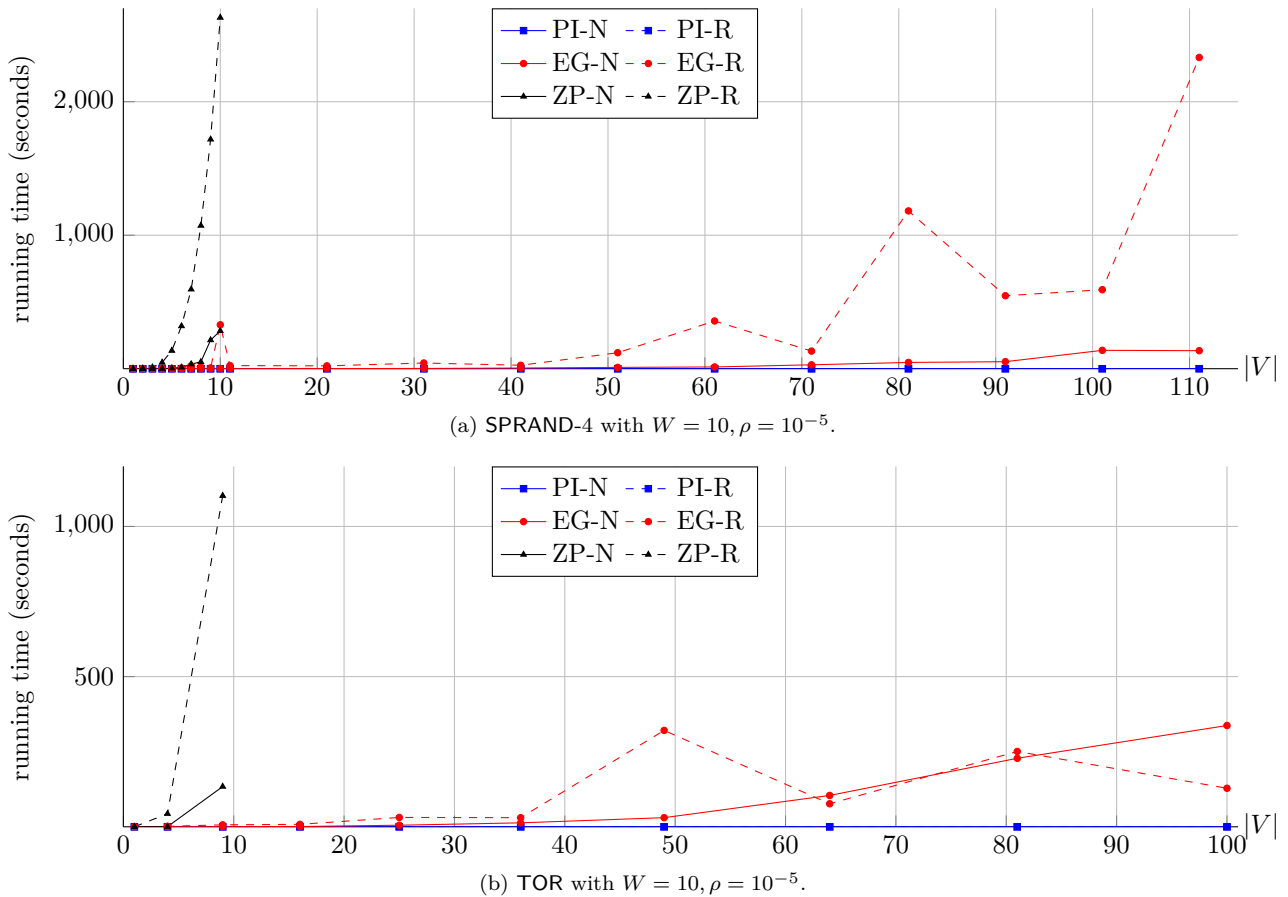


Fig. 7. Experimental results on real-weighted synthetic graphs, where $\rho = 10^{-5}$.

Proposition 6 ([1]). *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG. The outcome $\text{Val}(\pi)$ of any play π of Γ is in the value set $S = \{\frac{a}{b} \mid 0 \leq a \leq |V| \cdot W, 0 < b \leq |V| \cdot W\} \cup \{\infty\}$. If $w_1, w_2 : E \rightarrow \mathbb{N}$, this set is finite and has size $|S| = \mathcal{O}(|V|^2 \cdot W^2)$.*

The outcome $\text{Val}(\pi)$ only considers the weights in the limit, so only cycles in the play are relevant, which is also reflected in Proposition 5. The cycle length can vary from 1 to $|V|$, and the costs w_1 and w_2 per edge from 0 to W , so the costs per cycle vary from 0 to $|V| \cdot W$, for both the numerator and the denominator of the outcome $\text{Val}(\pi)$. If denominator $b = 0$, $\text{Val}(\pi) = \infty$. In the remainder of this section, we describe the complexity bounds of the algorithms presented in this paper.

B. Reduction to Mean-Payoff Games

Lemma 7 (Value Decision Problem [1]). *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with maximal weight W . Given a ratio $\frac{a}{b}$ with $0 \leq a \leq |V| \cdot W$ and $0 < b \leq |V| \cdot W$, we can decide whether a state $v \in V$ has value $\text{Val}(v) = \frac{a}{b}$, $\text{Val}(v) < \frac{a}{b}$ or $\text{Val}(v) > \frac{a}{b}$ by a reduction to a decision for a mean-payoff game. If $w_1, w_2 : E \rightarrow \mathbb{N}$, this can be done in $\mathcal{O}(|V|^3 \cdot W^2 \cdot |E|)$ time, and if $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$ in $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E| \cdot \rho^{-1})$.*

Proof. (Sketch) Create a mean-payoff game $\Gamma_{MPG} = (V, E, w, \langle V_0, V_1 \rangle)$ with payoff function $w(e) = b \cdot w_1(e) - a \cdot w_2(e)$. Let $\text{Val}(v)$ be the value in Γ , and $\text{Val}_{MPG}(v)$ be the value in Γ_{MPG} . Now, $\text{Val}(v) \leq \frac{a}{b}$ implies $\text{Val}_{MPG}(v) \leq 0$, and $\text{Val}(v) \geq \frac{a}{b}$ implies $\text{Val}_{MPG}(v) \geq 0$.

The decision in the mean-payoff game for natural weights can be made in $\mathcal{O}(|V|^2 \cdot W' \cdot |E|)$ [3], where W' is the maximal weight in the mean-payoff game. Since, $W' \leq b \cdot W \leq |V| \cdot W^2$, we get a complexity bound of $\mathcal{O}(|V|^3 \cdot W^2 \cdot |E|)$. In the derivation, the observation is used that the distance between the threshold 0 and the closest rational number with a denominator of size at most n is $1/n$.

For real-valued weights, we determine an estimate v_k on the value that is within ρ of the actual value in $\mathcal{O}(k \cdot |E|)$, using Theorem 2.1 in [3], and setting $k = 2 \cdot |V| \cdot W' \cdot \rho^{-1}$. This yields a complexity bound of $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E| \cdot \rho^{-1})$. Here, the closest distance is determined by ρ , instead of the fixed value $1/n$. \square

Theorem 8 (Value Problem [1]). *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with maximal weight W . $\text{Val}(v)$ for each state $v \in V$ can be computed in $\mathcal{O}(|V|^3 \cdot W^2 \cdot |E| \cdot \log(|V| \cdot W))$ if $w_1, w_2 : E \rightarrow \mathbb{N}$, and in $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E| \cdot \rho^{-1})$ if $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$.*

Proof. For natural weights, use the decision problem from Lemma 7 to perform a binary search on the possible values $S \setminus \{\infty\}$. If the ratio is greater than $|S| \cdot W$, it is infinite. For real-valued weights, use the result of Lemma 7, which gives an outcome within ρ of the actual value. \square

Theorem 9 (Optimal Strategy Synthesis Problem [1]). *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with weights $w_1, w_2 : E \rightarrow \mathbb{N}$ and maximal weight W . Optimal memoryless strategies for both players can be found in $\mathcal{O}\left(|V|^4 \cdot W^2 \cdot \log\left(\frac{|E|}{|V|}\right) \cdot |E| \cdot \log(|V| \cdot W)\right)$.*

Proof. Using Proposition 6 in a binary search over values $S \setminus \{\infty\}$. The value of each state $v \in V$, can be computed in $\mathcal{O}(|V|^3 \cdot W^2 \cdot |E| \cdot \log(|V| \cdot W))$. The group testing technique can be applied to compute optimal positional strategies with $\mathcal{O}(|V| \cdot \log\left(\frac{|E|}{|V|}\right))$ value computations. \square

Theorem 10. *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with weights $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$ and maximal weight W . Optimal memoryless strategies for both players can be found in $\mathcal{O}\left(|V|^3 \cdot W^2 \cdot \log\left(\frac{|E|}{|V|}\right) \cdot |E| \cdot \rho^{-1}\right)$.*

Proof. Using Lemma 7, the value of a vertex $v \in V$ can be computed in $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E| \cdot \rho^{-1})$ if $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$. The group testing technique of [3] can be used to compute optimal positional strategies with $\mathcal{O}\left(|V| \cdot \log\left(\frac{|E|}{|V|}\right)\right)$ value computations. \square

C. Reduction to Energy Games

Lemma 11 (Value Decision Problem). *Let $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with maximal weight W and $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$. Given a ratio $\frac{a}{b}$ with $0 \leq a \leq |V| \cdot W$ and $0 < b \leq |V| \cdot W$, we can decide whether a state $v \in V$ has value $\text{Val}(v) = \frac{a}{b}$, $\text{Val}(v) < \frac{a}{b}$ or $\text{Val}(v) > \frac{a}{b}$ by a reduction to a decision for an energy game in $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E|)$ time.*

Proof. (Sketch) Create an energy game $\Gamma_{EG} = (V, E, w, \langle V_0, V_1 \rangle)$ with edge weights $w(e) = b \cdot w_1(e) - a \cdot w_2(e)$. Let $\text{Val}(v)$ be the value in Γ , and $\text{Val}_{EG}(v)$ be the value in Γ_{EG} . Now, $\text{Val}(v) \leq \frac{a}{b}$ implies $\text{Val}_{EG}(v) \leq 0$, and $\text{Val}(v) \geq \frac{a}{b}$ implies $\text{Val}_{EG}(v) \geq 0$.

The crucial observation here is the following equivalence. Given a ratio $\frac{a}{b}$, we can compare this ratio to the cycle ratio of some cycle $c = v_i, \dots, v_n$ with $n > 0$, and rewrite it to a decision on an energy game value:

$$\frac{\sum_{i=0}^{n-1} w_1(v_i, v_{i+1})}{\sum_{i=0}^{n-1} w_2(v_i, v_{i+1})} \geq \frac{a}{b} \Leftrightarrow \sum_{i=0}^{n-1} (b \cdot w_1(v_i, v_{i+1}) - a \cdot w_2(v_i, v_{i+1})) \geq 0.$$

The decision in the energy game can be made in $\mathcal{O}(|V| \cdot W' \cdot |E|)$ [3], where W' is the maximal weight in the energy game. Since, $W' \leq b \cdot W \leq |V| \cdot W^2$, we get a complexity bound of $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E|)$. \square

Theorem 12 (Value and Optimal Strategy Synthesis Problem). *Let $\Gamma_{RG} = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with weights $w_1, w_2 : E \rightarrow \mathbb{N}$ and maximal weight W . Optimal memoryless strategies for both players can be found in $\mathcal{O}((\log(|V|) + \log(W)) \cdot |V|^2 \cdot W^2 \cdot |E|)$.*

Proof. The maximal distance between two values in S is $|V| \cdot W - (1/(|V| \cdot W))$, and the minimal distance is $1/((V-1) \cdot V)$. Therefore, the height of the tree of recursive calls is bounded by $\mathcal{O}\left(\log\left(\frac{(|V| \cdot W - (1/(|V| \cdot W)))}{1/((V-1) \cdot V)}\right)\right) \leq \mathcal{O}(\log(|V| \cdot W))$.

Since each reduction to a decision on an energy game is bounded by $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E|)$, we obtain a running time of $\mathcal{O}((\log(|V|) + \log(W)) \cdot |V|^2 \cdot W^2 \cdot |E|)$. \square

Theorem 13. *Let $\Gamma_{RG} = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with weights $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$ and maximal weight W . Optimal memoryless strategies for both players can be found in $\mathcal{O}((\log(|V|) + \log(W) + \log(\rho)) \cdot |V|^2 \cdot W^2 \cdot |E|)$.*

Proof. The maximal distance between two values in S is bounded by $|V| \cdot W$, and the minimal distance considered is $\rho \leq 1$. Therefore, the height of the tree of recursive calls is bounded by $\mathcal{O}\left(\log\left(\frac{|V| \cdot W}{\rho}\right)\right) = \mathcal{O}(\log(|V| \cdot W \cdot \rho^{-1}))$.

Since each reduction to a decision on an energy game is bounded by $\mathcal{O}(|V|^2 \cdot W^2 \cdot |E|)$, we obtain a running time of $\mathcal{O}((\log(|V|) + \log(W) + \log(\rho^{-1})) \cdot |V|^2 \cdot W^2 \cdot |E|)$. \square

D. Policy Iteration

Theorem 14 (Value and Optimal Strategy Synthesis Problem). *Let $\Gamma_{RG} = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with weights $w_1, w_2 : E \rightarrow \mathbb{N}$ and maximal weight W . Optimal memoryless strategies for both players can be found in $\mathcal{O}(|V|^{14} \cdot |E| \cdot |W|^4)$.*

Proof. This bound is calculated following the derivation in [5]. We first derive upper bounds on the values on the vectors r and d . For ratio values, $r_i(v) \leq |V| \cdot W$ for $v \in V$. The distances are computed by the weight function $w'(u, w) = r(v)^D \cdot w_1(u, w) - r(v)^N \cdot w_2(u, w)$. Given the maximal length $|V| - 1$ of a simple path, the maximal weight of this path is bounded by $(|V| - 1) \cdot (|V| \cdot W) \cdot W \leq |V|^2 \cdot W^2$. Value $d_i(v)$ for some $v \in V$ is increased only if the ratios of two cycles were the same. The number of times a new cycle is formed with the same ratio value is bounded by $|V|$ [4]. So $d_i(v) \leq |V|^3 \cdot W^2$ for $v \in V$.

In each iteration of PI-N, either r_i increases or d_i increases. If r_i increases, it does so by at least $1/(|V| \cdot (|V| - 1))$. Given the maximum ratio value of $|V| \cdot W$, and the fact that we have $|V|$ $r_i(v)$ -values, the number of iterations in which r_i increases is $\mathcal{O}(|V|^4 \cdot W)$. If r_i does not increase, d_i increases, and it must increase by at least $1/(|V| \cdot W)$. Therefore, the number of iterations between two increases of r_i is always in $\mathcal{O}(|V|^5 \cdot W^2)$. Together, the number of iterations of the while loop in PI-N is in $\mathcal{O}(|V|^9 \cdot W^3)$.

In MCT, the number of iterations where r_i decreases is $\mathcal{O}(|V|^4 \cdot W)$. The number of iterations where d_i decreases is always $\mathcal{O}(|V|)$. In this situation, the algorithm computes minimum-weight paths in $|V|$ passes over the edges. So the complexity of MCT is $\mathcal{O}(|V|^5 \cdot |E| \cdot W)$. Combined, the complexity of PI-N is $\mathcal{O}(|V|^{14} \cdot |E| \cdot W^4)$. \square

Theorem 15. *Let $\Gamma_{RG} = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ be a RG with weights $w_1, w_2 : E \rightarrow \mathbb{R}_{\geq 0}$ and maximal weight W . Optimal memoryless strategies for both players can be found in $\mathcal{O}(|V|^8 \cdot |E| \cdot |W|^3 \cdot \rho^{-3})$.*

Proof. This bound is calculated following the derivation in [5]. We use the same bounds on vector entries in r and d as in the proof of Theorem 14. For the ratio values $r_i(v) \leq |V| \cdot W$ for $v \in V$. For the distance values $d_i(v) \leq |V|^3 \cdot W^2$ for $v \in V$.

In each iteration of PI-R, either r_i increases or d_i increases. If r_i increases, it does so by at least ρ . Given the maximum ratio value of $|V| \cdot W$, and the fact that we have $|V|$ $r_i(v)$ -values, the number of iterations in which r_i increases is $\mathcal{O}(|V|^2 \cdot W \cdot \rho^{-1})$. If r_i does not increase, d_i increases, and it must increase by at least ρ . Therefore, the number of iterations between two increases of r_i is always in $\mathcal{O}(|V|^3 \cdot W \cdot \rho^{-2})$. Together, the number of iterations of the while loop in PI-N is in $\mathcal{O}(|V|^5 \cdot W^2 \cdot \rho^{-2})$.

In MCT, the number of iterations where r_i decreases is $\mathcal{O}(|V|^2 \cdot W \cdot \rho^{-1})$. The number of iterations where d_i decreases is always $\mathcal{O}(|V|)$. In this situation, the algorithm computes minimum-weight paths in $|V|$ passes over the edges. So the complexity of MCT is $\mathcal{O}(|V|^3 \cdot |E| \cdot W \cdot \rho^{-1})$. Combined, the complexity of PI-R is $\mathcal{O}(|V|^8 \cdot |E| \cdot W^3 \cdot \rho^{-3})$. \square

Algorithm 1. Zwick-Paterson for graphs with $w_1, w_2 \in \mathbb{N}$ (ZP-N).

```

1: proc GETSTRATEGY( $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ )
2:    $val \leftarrow \text{COMPUTEVALUES}(\Gamma)$ 
3:    $E_{out}(v) \leftarrow$  list of all outgoing edges of  $v$ 
4:    $s \leftarrow |E_{out}(v)|$ 
5:   foreach  $v \in V$  do
6:      $S(v) \leftarrow \text{FINDOUTGOINGEDGE}(\Gamma, v, val(v), E_{out}(v), 1, s, \rho)$ 
7:   return  $S$   $\triangleright S : V \rightarrow V$  is an optimal strategy
1: proc FINDOUTGOINGEDGE( $\Gamma, v, val, edges, l, r$ )
2:   if  $l = r$  then
3:     return  $edges[l]$ 
4:   else
5:      $middle \leftarrow \lfloor (l + r)/2 \rfloor$ 
6:      $E^{sub} \leftarrow \Gamma(E) \cap \{edges[i] \mid l \leq i \leq middle\}$ 
7:      $\Gamma^{sub} \leftarrow \text{SUBGRAPH}(\Gamma, E^{sub})$ 
8:      $values \leftarrow \text{COMPUTEVALUES}(\Gamma^{sub})$ 
9:     if  $values(v) = val$  then
10:       $\triangleright$  there is a positional optimal strategy
      that does not use any of the
      removed edges
11:       $\text{FINDOUTGOINGEDGE}(\Gamma, v, val, edges, l, middle)$ 
12:     else
13:       $\text{FINDOUTGOINGEDGE}(\Gamma, v, val, edges, middle, r)$ 
1: proc COMPUTEVALUES( $\Gamma$ )
2:   foreach  $v \in V$  do
3:      $val(v) \leftarrow \text{COMPUTEVALUE}(\Gamma, v, 0, V \cdot W)$ 
4:   return  $val$   $\triangleright val(v)$  is the optimal value for

```

```

 $v \in V$ 
1: proc COMPUTEVALUE( $\Gamma, v, \frac{l^N}{l^D}, \frac{r^N}{r^D}$ )
2:    $\frac{m^N}{m^D} \leftarrow \frac{1}{2} \left( \frac{l^N}{l^D} + \frac{r^N}{r^D} \right)$   $\triangleright$  split range
3:    $m_1 \leftarrow \frac{a_1}{b_1} \leftarrow \max\{\frac{a}{b} \mid 1 \leq a, b \leq |V| \cdot W$ 
 $\wedge \frac{l^N}{l^D} \leq middle\}$ 
4:    $m_2 \leftarrow \frac{a_2}{b_2} \leftarrow \min\{\frac{a}{b} \mid 1 \leq a, b \leq |V| \cdot W$ 
 $\wedge \frac{r^N}{r^D} \geq \frac{a}{b} \geq middle\}$ 
5:    $\Gamma_{MPG} = (V, E, a_2 \cdot w_1 - a_1 \cdot w_2, \langle V_0, V_1 \rangle)$ 
6:    $(V_{<m_1}, V_{=m_1}, V_{>m_1}) \leftarrow \text{THREEWAYPARTITION}(\Gamma_{MPG}, m_1)$ 
7:   if  $v \in V_{=m_1}$  then
8:     return  $m_1$ 
9:   else if  $v \in V_{<m_1}$  then
10:      $\text{COMPUTEVALUE}(\Gamma, v, |V|, \frac{l^N}{l^D}, m_1)$ 
11:   else  $\triangleright v \in V_{>m_1}$ 
12:      $\text{COMPUTEVALUE}(\Gamma, v, |V|, m_2, \frac{r^N}{r^D})$ 
1: proc THREEWAYPARTITION( $\Gamma_{MPG} = (V, E, w, \langle V_0, V_1 \rangle), \nu$ )
2:    $k \leftarrow 4 \cdot |V|^3 \cdot |W|$ 
3:    $est \leftarrow \text{COMPUTEESTIMATES}(\Gamma_{MPG}, k)$ 
4:   if  $V \leq 2$  then  $\triangleright$  avoid division by zero
5:      $\delta \leftarrow 0$ 
6:   else
7:      $\delta \leftarrow 1/(2 \cdot |V| \cdot |V - 1|)$ 
8:    $smaller \leftarrow \{v \mid v \in V \wedge est(v) < \nu - \delta\}$ 
9:    $equal \leftarrow \{v \mid v \in V \wedge \nu - \delta \leq est(v) \leq \nu + \delta\}$ 
10:   $larger \leftarrow \{v \mid v \in V \wedge est(v) > \nu + \delta\}$ 
11:  return  $(smaller, equal, larger)$ 

```

Algorithm 2. Zwick-Paterson for graphs with $w_1, w_2 \in \mathbb{R}$ (ZP-R).

```

1: proc GETSTRATEGY( $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle, \rho)$ )
2:    $val \leftarrow \text{COMPUTEVALUES}(\Gamma, \rho)$ 
3:    $E_{out}(v) \leftarrow$  list of all outgoing edges of  $v$ 
4:    $s \leftarrow |E_{out}(v)|$ 
5:   foreach  $v \in V$  do
6:      $S(v) \leftarrow \text{FINDOUTGOINGEDGE}(\Gamma, v, val(v), E_{out}(v), 1, s, \rho)$ ;
7:   return  $S \triangleright S : V \rightarrow V$  is an optimal strategy
1: proc FINDOUTGOINGEDGE( $\Gamma, v, val, edges, l, r, \rho$ )
2:   if  $l = r$  then
3:     return  $edges[l]$ 
4:   else
5:      $middle \leftarrow \lfloor (l + r)/2 \rfloor$ 
6:      $E^{sub} \leftarrow \Gamma(E) \cap \{edges[i] \mid l \leq i \leq middle\}$ 
7:      $\Gamma^{sub} \leftarrow \text{SUBGRAPH}(\Gamma, E^{sub})$ 
8:      $values \leftarrow \text{COMPUTEVALUES}(\Gamma^{sub}, \rho)$ 
9:     if  $(values(v) - val) < \rho$  then
10:        $\triangleright$  there is a positional optimal strategy
           that does not use any of the
           removed edges
11:        $\text{FINDOUTGOINGEDGE}(\Gamma, v, val, edges, l, middle)$ 
12:     else
13:        $\text{FINDOUTGOINGEDGE}(\Gamma, v, val, edges, middle, r)$ 
1: proc COMPUTEVALUES( $\Gamma, \rho$ )
2:   foreach  $v \in V$  do
3:      $val(v) \leftarrow \text{COMPUTEVALUE}(\Gamma, v, 0, V \cdot W, \rho)$ 
4:   return  $val \triangleright val(v)$  is the optimal value for
            $v \in V$ 
1: proc COMPUTEVALUE( $\Gamma, v, l, r, \rho$ )
2:    $m \leftarrow (l + r)/2$   $\triangleright$  split range
3:    $\Gamma_{MPG} = (V, E, a_2 \cdot w_1 - a_1 \cdot w_2, \langle V_0, V_1 \rangle)$ 
4:    $(V_{< m}, V_{= m}, V_{> m}) \leftarrow \text{THREEWAYPARTITION}(\Gamma_{MPG}, m, \rho)$ 
5:   if  $v \in V_{= m}$  then
6:     return  $m$ 
7:   else if  $v \in V_{< m}$  then
8:      $\text{COMPUTEVALUE}(\Gamma, v, |V|, l, m, \rho)$ 
9:   else  $\triangleright v \in V_{> m}$ 
10:     $\text{COMPUTEVALUE}(\Gamma, v, |V|, m, r, \rho)$ 
1: proc THREEWAYPARTITION( $\Gamma_{MPG} = (V, E, w, \langle V_0, V_1 \rangle, \nu, \rho)$ )
2:    $k \leftarrow \lfloor 2 \cdot |V| \cdot |W| / \rho \rfloor$ 
3:    $est \leftarrow \text{COMPUTEESTIMATES}(\Gamma_{MPG}, k)$ 
4:    $\delta \leftarrow \rho$ 
5:    $smaller \leftarrow \{v \mid v \in V \wedge est(v) < \nu - \delta\}$ 
6:    $equal \leftarrow \{v \mid v \in V \wedge \nu - \delta \leq est(v) \leq \nu + \delta\}$ 
7:    $larger \leftarrow \{v \mid v \in V \wedge est(v) > \nu + \delta\}$ 
8:   return  $(smaller, equal, larger)$ 
1: proc COMPUTEESTIMATES( $\Gamma_{MPG} = (V, E, w, \langle V_0, V_1 \rangle, k)$ )
2:   foreach  $v \in V$  do
3:      $val_0(v) \leftarrow 0$ 
4:     for  $i \leftarrow 1$  to  $k$  do
5:       foreach  $v \in V$  do
6:         if  $v \in V_0$  then
7:            $val_k(v) = \max_{v, u \in E} \{w(v, u) + val_{k-1}(u)\}$ 
8:         else  $\triangleright v \in V_1$ 
9:            $val_k(v) = \min_{v, u \in E} \{w(v, u) + val_{k-1}(u)\}$ 
10:    foreach  $v \in V$  do  $\triangleright$  compute estimates
11:      if  $k = 0$  then
12:         $est(v) \leftarrow val_k(v)$ 
13:      else
14:         $est(v) \leftarrow val_k(v)/k$ 
15:    return  $est$ 

```

Algorithm 3. Energy Game reduction for graphs with $w_1, w_2 \in \mathbb{R}$ (EG-R).

```

1: proc EG-R( $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle, f_l, l, r)$ )
2:   require  $w_1, w_2 \in \mathbb{R}$ 
3:    $m \leftarrow (l + r)/2$   $\triangleright$  split range
4:   if  $|r - l| < \rho$  then
5:     foreach  $v \in V$  do
6:        $val(v) \leftarrow l$ 
7:       if  $v \in V_0$  then
8:          $S(v) \leftarrow \min\{f_l(u) \mid (v, u) \in E\}$ 
9:       else
10:         $S(v) \leftarrow \max\{f_l(u) \mid (v, u) \in E\}$ 
11:      return  $(S, val)$ 
12:   else
13:      $\triangleright$  determine  $V_{\geq m}(f_m(v) \neq \top)$ ,
            $V_{< m}(f_m(v) = \top)$ 
14:      $f_m \leftarrow \text{SOLVEEG}(\Gamma_{EG} = (V, E, w_1 - m \cdot w_2, \langle V_0, V_1 \rangle))$ 
15:      $V_{< m} \leftarrow \{v \mid f_m(v) = \top\}$ 
16:      $V_{\geq m} \leftarrow \{v \mid f_m(v) \neq \top\}$ 
17:      $\triangleright$  recursive calls
18:      $\Gamma^< \leftarrow (V_{< m}, E \upharpoonright V_{< m}, w_1 \upharpoonright V_{< m}, w_2 \upharpoonright V_{< m}, \langle V_0 \cap V_{< m}, V_1 \cap V_{< m} \rangle)$ 
19:      $(S_l, val_l) \leftarrow \text{EG-R}(\Gamma^<, f_l, l, m)$ 
20:      $\Gamma^{\geq} \leftarrow (V_{\geq m}, E \upharpoonright V_{\geq m}, w_1 \upharpoonright V_{\geq m}, w_2 \upharpoonright V_{\geq m}, \langle V_0 \cap V_{\geq m}, V_1 \cap V_{\geq m} \rangle)$ 
21:      $(S_r, val_r) \leftarrow \text{EG-R}(\Gamma^{\geq}, f_m, m, r)$ 
22:     return  $(S_l \cup S_r, val_l \cup val_r)$ 
23:   EG-R( $(V, E, w_1, w_2, \langle V_0, V_1 \rangle, f_l, l, r, \mathbf{0}, \mathbf{0}, W)$ )
            $\triangleright (\forall v \in V : \mathbf{0}(v) = \mathbf{0})$ 
1: proc SOLVEEG( $\Gamma_{EG} = (V, E, w, \langle V_0, V_1 \rangle)$ )
2:    $\triangleright$  see pseudo code in [6]

```

Algorithm 4. Energy Game reduction for graphs with $w_1, w_2 \in \mathbb{N}$ (EG-N).

```

1: proc EG-N( $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle), \frac{l^N}{l^D}, \frac{r^N}{r^D}$ )
2:   require  $w_1, w_2 \in \mathbb{N}$ 
3:    $\frac{m^N}{m^D} \leftarrow \frac{1}{2} \left( \frac{l^N}{l^D} + \frac{r^N}{r^D} \right)$                                 ▷ split range
4:    $m_1 \leftarrow \frac{a_1}{b_1} \leftarrow \max \left\{ \frac{a}{b} \mid 1 \leq a, b \leq |V| \cdot W \right.$ 
       $\left. \wedge \frac{l^N}{l^D} \leq \frac{m^N}{m^D} \right\}$ 
5:    $m_2 \leftarrow \frac{b_2}{b_2^N} \leftarrow \min \left\{ \frac{a}{b} \mid 1 \leq a, b \leq |V| \cdot W \right.$ 
       $\left. \wedge \frac{r^N}{r^D} \geq \frac{a}{b} \geq \frac{m^N}{m^D} \right\}$ 
6:   ▷ determine  $V_{>a_1} (f(v) \neq \top), V_{<a_1} (f(v) = \top)$ 
7:    $f_1 \leftarrow \text{SOLVEEG}(\Gamma_{EG}^1 = (V, E,$ 
       $b_1 \cdot w_1 - a_1 \cdot w_2, \langle V_0, V_1 \rangle))$ 
8:   ▷ determine  $V_{<a_1}, V_{>a_1}$ 
9:    $f_2 \leftarrow \text{SOLVEEG}(\Gamma_{EG}^2 = (V, E,$ 
       $a_1 \cdot w_2 - b_1 \cdot w_1, \langle V_1, V_0 \rangle))$ 
10:  ▷ determine  $V_{>a_2}, V_{<a_2}$ 
11:   $f_3 \leftarrow \text{SOLVEEG}(\Gamma_{EG}^3 = (V, E,$ 
       $b_2 \cdot w_1 - a_2 \cdot w_2, \langle V_0, V_1 \rangle))$ 
12:  ▷ determine  $V_{<a_2}, V_{>a_2}$ 
13:   $f_4 \leftarrow \text{SOLVEEG}(\Gamma_{EG}^4 = (V, E,$ 
       $a_2 \cdot w_2 - b_2 \cdot w_1, \langle V_1, V_0 \rangle))$ 
14:  foreach  $v \in V$  do
15:    if  $f_1(v) \neq \top \wedge f_2(v) \neq \top$  then
16:       $val(v) \leftarrow a_1$ 
17:    if  $v \in V_0$  then
18:       $S(v) \leftarrow \min \{ f_1(u) \mid (v, u) \in E \}$ 
19:    else
20:       $S(v) \leftarrow \max \{ f_1(u) \mid (v, u) \in E \}$ 
21:    if  $f_3(v) \neq \top \wedge f_4(v) \neq \top$  then
22:       $val(v) \leftarrow a_2$ 
23:      if  $v \in V_0$  then
24:         $S(v) \leftarrow \min \{ f_3(u) \mid (v, u) \in E \}$ 
25:      else
26:         $S(v) \leftarrow \max \{ f_3(u) \mid (v, u) \in E \}$ 
27:       $V_{<a_1} \leftarrow \{ v \mid f_1(v) = \top \}$ 
28:       $V_{>a_2} \leftarrow \{ v \mid f_4(v) = \top \}$ 
29:      ▷ recursive calls
30:       $\Gamma_{RG}^{sub1} \leftarrow (V_{<a_1}, E \upharpoonright V_{<a_1}, w \upharpoonright V_{<a_1},$ 
         $\langle V_0 \cap V_{<a_1}, V_1 \cap V_{<a_1} \rangle)$ 
31:      EG-N( $\Gamma_{RG}^{sub1}, \frac{l^N}{l^D}, a_1$ )
32:       $\Gamma_{RG}^{sub2} \leftarrow (V_{>a_2}, E \upharpoonright V_{>a_2}, w \upharpoonright V_{>a_2},$ 
         $\langle V_0 \cap V_{>a_2}, V_1 \cap V_{>a_2} \rangle)$ 
33:      EG-N( $\Gamma_{RG}^{sub2}, a_2, \frac{r^N}{r^D}$ )
1: proc SOLVEEG( $\Gamma_{EG} = (V, E, w, \langle V_0, V_1 \rangle)$ )
2:   ▷ see pseudo code in [6]

```

Algorithm 5. Policy Iteration for graphs with $w_1, w_2 \in \mathbb{N}$ (PI-N), based on [5].

```

1: proc PI-N( $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ )
2:   ensure:  $\sigma_i : V_i \rightarrow V$  is an optimal strategy
      for player  $i$ ,  $r_i(v)$  is the optimal value
      for  $v \in V$ .
3:    $i \leftarrow -1$ 
4:    $improvement \leftarrow true$ 
5:    $d_{-1} \leftarrow 0^V$ 
6:    $r_{-1} \leftarrow -\infty^V$ 
7:    $\sigma_0^0 \leftarrow$  arbitrary strategy of player 0
8:    $\sigma_1^0 \leftarrow$  arbitrary strategy of player 1
9:   while  $improvement$  do
10:     $i := i + 1$ 
11:     $\triangleright$  improve positional strategy of player 1,
      update vectors.
12:     $(d_i, r_i, \sigma_1^{i+1}) \leftarrow$  MCT( $\Gamma, \sigma_0^i, \sigma_1^i, d_{i-1}, r_{i-1}$ )
13:     $\sigma_0^{i+1} \leftarrow \sigma_0^i$ 
14:     $improvement \leftarrow false$ 
15:     $\triangleright$  improve positional strategy of player 0.
16:    foreach  $(v, u) \in E \cap V_0 \times V$  do
17:      if  $r_i(v) < r_i(u) \vee (r_i(v) = r_i(u) \wedge$ 
       $d_i(v) < d_i(u) + r_i(u)^D \cdot w_1(v, u) -$ 
       $r_i(u)^N \cdot w_2(v, u))$  then
18:         $\sigma_0^{i+1}(v) \leftarrow u$ 
19:         $improvement \leftarrow true$ 
20:    return  $(r_i, \sigma_0^i, \sigma_1^i)$ 

1: proc MCT( $\Gamma, \sigma_0^i, \sigma_1^i, d_{i-1}, r_{i-1}$ )
2:    $\triangleright$  adapted from Cochet-Terrasson et al. [4]
3:    $t \leftarrow -1$ 
4:    $improvement \leftarrow true$ 
5:    $\sigma_0^{i,0} \leftarrow \sigma_0^i$ 
6:   while  $improvement$  do
7:     $t \leftarrow t + 1$ 
8:     $(d_{i,t}, r_{i,t}) \leftarrow$  EVALUATESTRATEGY( $G_{\sigma_0^i \cup \sigma_1^i, t}, d_{i-1}, r_{i-1}$ )
9:     $\sigma_1^{i,t+1} \leftarrow \sigma_1^{i,t}$ 
10:    $improvement \leftarrow false$ 
11:   foreach  $(v, u) \in E \cap V_1 \times V$  do
12:      $\triangleright$  improve strategy of player 1
13:     if  $r_{i,t}(v) > r_{i,t}(u) \vee (r_{i,t}(v) = r_{i,t}(u) \wedge$ 
       $d_{i,t}(v) > d_{i,t}(u) + r_{i,t}(u)^D \cdot w_1(v, u) -$ 
       $r_{i,t}(u)^N \cdot w_2(v, u))$  then
14:        $\sigma_1^{i,t+1}(v) \leftarrow u$ 
15:        $improvement \leftarrow true$ 
16:   return  $(d_{i,t}, r_{i,t}, \sigma_1^{i,t})$ 

1: proc EVALUATESTRATEGY( $G, d_{i-1}, r_{i-1}$ )
2:    $\triangleright$  adapted from [4]
3:    $(V_s, r_{i,t}) \leftarrow$ 
      FINDCYCLESINGRAPH( $G_{\sigma_0^i \cup \sigma_1^i, t}$ )
4:    $(d_{i,t}, r_{i,t}) \leftarrow$ 
      COMPUTEDISTANCES( $G, V_s, r_{i,t}, d_{i-1}, r_{i-1}$ )
5:   return  $(d_{i,t}, r_{i,t})$ 

1: proc FINDCYCLESINGRAPH( $G_{\sigma_0^i \cup \sigma_1^i, t}$ )
2:    $S \leftarrow \emptyset$ 
3:   foreach  $v \in V$  do
4:      $visited(v) \leftarrow \perp$ 
5:   foreach  $v \in V$  do
6:     if  $visited(v) = \perp$  then
7:        $u \leftarrow v$ 
8:       while  $visited(u) = \perp$  do
9:          $visited(u) \leftarrow v$ 
10:         $u \leftarrow successor(u)$ 
11:       if  $visited(u) = v$  then
12:          $\triangleright$  cycle found; find smallest vertex
      and cycle ratio.
13:         $v_s \leftarrow u$ 
14:         $x \leftarrow successor(u)$ 
15:         $cw_1 \leftarrow w_1(u, successor(u))$ 
16:         $cw_2 \leftarrow w_2(u, successor(u))$ 
17:        while  $x \neq u$  do
18:          if  $x < v_s$  then
19:             $v_s \leftarrow x$ 
20:             $cw_1 \leftarrow cw_1 + w_1(u, successor(x))$ 
21:             $cw_2 \leftarrow cw_2 + w_2(u, successor(x))$ 
22:             $x \leftarrow successor(x)$ 
23:             $r_{i,t}(v_s) \leftarrow cw_1 / cw_2$ 
24:             $S \leftarrow S \cup \{v_s\}$ 
25:   return  $(S, r_{i,t})$ 

1: proc COMPUTEDISTANCES( $G_{\sigma_0^i \cup \sigma_1^i, t}, S, r_{i,t}, d_{i-1}, r_{i-1}$ )
2:   foreach  $v \in V$  do
3:      $visited(v) \leftarrow false$ 
4:   foreach  $u \in S$  do
5:     if  $r_{i-1}(u) = r_{i,t}(u)$  then
6:        $d_{i,t}(u) \leftarrow d_{i-1}(u)$ 
7:     else
8:        $d_{i,t}(u) \leftarrow 0$ 
9:      $visited(u) \leftarrow true$ 
10:  foreach  $v \in V$  do
11:    if  $\neg visited(v)$  then
12:       $u \leftarrow v$ 
13:      while  $\neg visited(u)$  do
14:         $visited(u) \leftarrow true$ 
15:         $s.push(u)$ 
16:         $u \leftarrow successor(u)$ 
17:      while  $\neg s.empty()$  do
18:         $x \leftarrow s.pop()$ 
19:         $r_{i,t}(x) \leftarrow r_{i,t}(u)$ 
20:         $d_{i,t}(x) \leftarrow d_{i,t}(u) + r_{i,t}(u)^D \cdot$ 
       $w_1(x, u) - r_{i,t}(u)^N \cdot w_2(x, u)$ 
21:         $u \leftarrow x$ 
22:   return  $(d_{i,t}, r_{i,t})$ 

```

Algorithm 6. Policy Iteration for graphs with $w_1, w_2 \in \mathbb{R}$ (PI-R), based on [5].

```

1: proc PI-R( $\Gamma = (V, E, w_1, w_2, \langle V_0, V_1 \rangle)$ )
2:   ensure:  $\sigma_i : V_i \rightarrow V$  is an optimal strategy
      for player  $i$ ,  $r_i(v)$  is the optimal value
      for  $v \in V$ .
3:    $i \leftarrow -1$ 
4:    $improvement \leftarrow true$ 
5:    $d_{-1} \leftarrow 0^V$ 
6:    $\omega_{-1} \leftarrow 0^V$ 
7:    $r_{-1} \leftarrow -\infty^V$ 
8:    $\sigma_0^0 \leftarrow$  arbitrary strategy of player 0
9:    $\sigma_1^0 \leftarrow$  arbitrary strategy of player 1
10:  while  $improvement$  do
11:     $i := i + 1$ 
12:     $\triangleright$  improve strategy of player 1,
      update vectors.
13:     $(d_i, r_i, \sigma_1^{i+1}, \omega_i) \leftarrow$  MCT(
       $\Gamma, \sigma_0^i, \sigma_1^i, d_{i-1}, r_{i-1}, \omega_{i-1}, \rho$ )
14:     $\sigma_0^{i+1} \leftarrow \sigma_0^i$ 
15:     $improvement \leftarrow false$ 
16:     $\triangleright$  Improve positional strategy of player 0.
17:    foreach  $(v, u) \in E \cap V_0 \times V$  do
18:       $\delta \leftarrow \max(\omega_i(v), \omega_i(u) + w_2(v, u)) \cdot \rho$ 
19:      if  $r_i(v) <_\rho r_i(u) \vee (r_i(v) \equiv_\rho r_i(u) \wedge$ 
       $d_i(v) <_{2\delta} d_i(u) + r_i(u)^D \cdot$ 
       $w_1(v, u) - r_i(u)^N \cdot w_2(v, u))$  then
20:         $\sigma_0^{i+1}(v) \leftarrow u$ 
21:         $improvement \leftarrow true$ 
22:    return  $(r_i, \sigma_0^i, \sigma_1^i)$ 

1: proc MCT( $\Gamma, \sigma_0^i, \sigma_1^i, d_{i-1}, r_{i-1}, \omega_{i-1}, \rho$ )
2:   $\triangleright$  adapted from Cochet-Terrasson et al. [4]
3:   $t \leftarrow -1$ 
4:   $improvement \leftarrow true$ 
5:   $\sigma_0^{i,0} \leftarrow \sigma_0^i$ 
6:  while  $improvement$  do
7:     $t \leftarrow t + 1$ 
8:     $(d_{i,t}, r_{i,t}, \omega_{i,t}) \leftarrow$  EVALUATESTRATEGY(
       $G_{\sigma_0^i \cup \sigma_1^i, t}, d_{i-1}, r_{i-1}, \omega_{i-1}, \rho$ )
9:     $\sigma_1^{i,t+1} \leftarrow \sigma_1^{i,t}$ 
10:    $improvement \leftarrow false$ 
11:   foreach  $(v, u) \in E \cap V_1 \times V$  do
12:      $\triangleright$  improve strategy of player 1
13:      $\delta \leftarrow \max(\omega_{i,t}(v), \omega_{i,t}(u) + w_2(v, u)) \cdot \rho$ 
14:     if  $r_{i,t}(v) >_\rho r_{i,t}(u) \vee (r_{i,t}(v) \equiv_\rho$ 
       $r_{i,t}(u) \wedge d_{i,t}(v) >_{2\delta} d_{i,t}(u) +$ 
       $r_{i,t}(u)^D \cdot w_1(v, u) - r_{i,t}(u)^N \cdot w_2(v, u))$ 
      then
15:        $\sigma_1^{i,t+1}(v) \leftarrow u$ 
16:        $improvement \leftarrow true$ 
17:   return  $(d_{i,t}, r_{i,t}, \sigma_1^{i,t}, \omega_{i,t})$ 

1: proc EVALUATESTRATEGY( $G, d_{i-1}, r_{i-1}, \omega_{i-1}, \rho$ )
2:   $\triangleright$  adapted from [4]
3:   $(V_s, r_{i,t}) \leftarrow$ 
      FINDCYCLESINGRAPH( $G_{\sigma_0^i \cup \sigma_1^i, t}$ )
4:   $(d_{i,t}, r_{i,t}, \omega_{i,t}) \leftarrow$  COMPUTEDISTANCES(
       $G, V_s, r_{i,t}, d_{i-1}, r_{i-1}, \omega_{i-1}, \rho$ )
5:  return  $(d_{i,t}, r_{i,t}, \omega_{i,t})$ 

1: proc COMPUTEDISTANCES(
       $G_{\sigma_0^i \cup \sigma_1^i, t}, S, r_{i,t}, d_{i-1}, r_{i-1}, \omega_{i-1}, \rho$ )
2:  foreach  $v \in V$  do
3:     $visited(v) \leftarrow false$ 
4:  foreach  $u \in S$  do
5:    if  $r_{i-1}(u) \equiv_\rho r_{i,t}(u)$  then
6:       $d_{i,t}(u) \leftarrow d_{i-1}(u)$ 
7:       $\omega_{i,t}(u) \leftarrow \omega_{i-1}(u)$ 
8:    else
9:       $d_{i,t}(u) \leftarrow 0$ 
10:      $\omega_{i,t}(u) \leftarrow 0$ 
11:      $visited(u) \leftarrow true$ 
12:  foreach  $v \in V$  do
13:    if  $\neg visited(v)$  then
14:       $u \leftarrow v$ 
15:      while  $\neg visited(u)$  do
16:         $visited(u) \leftarrow true$ 
17:         $s.push(u)$ 
18:         $u \leftarrow successor(u)$ 
19:      while  $\neg s.empty()$  do
20:         $x \leftarrow s.pop()$ 
21:         $r_{i,t}(x) \leftarrow r_{i,t}(u)$ 
22:         $d_{i,t}(x) \leftarrow d_{i,t}(u) + r_{i,t}(u)^D \cdot$ 
           $w_1(x, u) - r_{i,t}(u)^N \cdot w_2(x, u)$ 
23:         $\omega_{i,t}(x) \leftarrow \omega_{i,t}(u) + w_2(x, u)$ 
24:         $u \leftarrow x$ 
25:      return  $(d_{i,t}, r_{i,t})$ 

```

ACKNOWLEDGMENTS

This research is supported by the Dutch Technology Foundation NWO-TTW, carried out as part of the Robust Cyber-Physical Systems (RCPS) program, project number 12694.

REFERENCES

- [1] R. Bloem, K. Greimel, T. A. Henzinger, and B. Jobstmann, "Synthesizing robust systems," in *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*. Austin, TX, USA: IEEE, nov 2009, pp. 85–92.
- [2] A. Ehrenfeucht and J. Mycielski, "Positional strategies for mean payoff games," *International Journal of Game Theory*, vol. 8, no. 2, pp. 109–113, 1979.
- [3] U. Zwick and M. Paterson, "The complexity of mean payoff games on graphs," *Theoretical Computer Science*, vol. 158, no. 1-2, pp. 343–359, 1996.
- [4] J. Chaloupka, *Parallel Algorithms for Mean-Payoff Games: An Experimental Evaluation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 599–610. [Online]. Available: https://doi.org/10.1007/978-3-642-04128-0_54
- [5] —, "Algorithms for mean-payoff and energy games," Ph.D. dissertation, Masarykova univerzita, Fakulta informatiky, Brno, 2011.
- [6] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J. F. Raskin, "Faster algorithms for mean-payoff games," *Formal Methods in System Design*, vol. 38, no. 2, pp. 97–118, 2011.
- [7] R. Bloem, K. Chatterjee, K. Greimel, T. A. Henzinger, G. Hofferek, B. Jobstmann, B. Könighofer, and R. Könighofer, "Synthesizing robust systems," *Acta Informatica*, 2013.
- [8] S. Schewe, "An optimal strategy improvement algorithm for solving parity and payoff games," in *Computer Science Logic: 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, M. Kaminski and S. Martini, Eds. Berlin, Heidelberg: Springer, 2008, pp. 369–384. [Online]. Available: https://doi.org/10.1007/978-3-540-87531-4_27
- [9] J. Fearnley, *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*. Springer International Publishing, 2017, ch. Efficient Parallel Strategy Improvement for Parity Games, pp. 137–154. [Online]. Available: https://doi.org/10.1007/978-3-319-63390-9_8

- [10] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba, "Infinite runs in weighted timed automata with energy constraints," in *Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08)*, ser. Lecture Notes in Computer Science, F. Cassez and C. Jard, Eds., vol. 5215. Saint-Malo, France: Springer, sep 2008, pp. 33–47.
- [11] V. Dhingra and S. Gaubert, "How to solve large scale deterministic games with mean payoff by policy iteration," in *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2006, Pisa, Italy, October 11-13, 2006*, 2006, p. 12.
- [12] 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*. American Mathematical Society, 1993, pp. 51–73.
- [13] B. Naveh *et al.* (2017) JGraphT. [Online]. Available: <http://jgrapht.sourceforge.net>
- [14] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," in *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, 1994, pp. 516–525.
- [15] B. V. Cherkassky and A. V. Goldberg, "Negative-cycle detection algorithms," *Mathematical Programming*, vol. 85, no. 2, pp. 277–311, Jun 1999. [Online]. Available: <https://doi.org/10.1007/s101070050058>
- [16] B. van der Sanden, J. Bastos, J. Voeten, M. Geilen, M. A. Reniers, T. Basten, J. Jacobs, and R. R. H. Schiffelers, "Compositional specification of functionality and timing of manufacturing systems," in *2016 Forum on Specification and Design Languages, FDL 2016, Bremen, Germany, September 14-16, 2016*, R. Drechsler and R. Wille, Eds. IEEE, 2016, pp. 1–8. [Online]. Available: <https://doi.org/10.1109/FDL.2016.7880372>