UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Master's Thesis

Infinite Games: Algorithms and Reductions

Maxime Nederkorn
Matrikelnummer: 3004376

Department of Computer Science and
Applied Cognitive Science
Faculty of Engineering
University of Duisburg-Essen

7. March 2022

Examiners:
Prof. Dr. Barbara König
Prof. Dr. Janis Voigtländer

Advisor:
Richard Eggert

Contents

1. Introduction

There are practical applications (cite) which are able to be modeled by or reduced to various games. Considering the many ways there are to solve those games, the possibility to (repeatedly) reduce them to one another and various ways to implement any of those algorithms there are many potential means to solve problems on those games. We want to see how the different ways to solve problems, the reductions between those problems and the combination of the two play out and compare to one another in a real-world implementation.

1.1. Task

The games we concern ourselves here are *Parity Games* (PG), *Mean Payoff Games* (MPG), *Energy Games* (EG), *Discounted Payoff Games* (DPG) and (stopping) *Simple Stochastic Games* (SSG). For any of those games there are multiple problems, but the ones we concern outselves here are “what is the value of vertex v under optimal strategies from both players?” and “what are the optimal strategies?”. For both of those problems, there may already be multiple algorithms to solve them for any specific game directly, some of which are generalizable to multiple of the games to some degree, such as Kleene Iteration and Strategy Iteration. Additionally we can reduce the games to another as follows:

$$\begin{tikzcd}
PG \arrow[r] & MPG \arrow[r] \arrow[d] & DPG \arrow[r] & SSG \\
& EG & &
\end{tikzcd}$$

The reductions themselves are computationally relatively simple compared to the problems themselves. Some of them, however, produce graphs that are bigger in some way ($PG \to MPG$, $DPG \to SSG$), which increases the complexity of subsequently applied algorithms, or recursively reduces to multiple sub-problems ($MPG \to EG$). Another part we have control over is how exactly the algorithms are implemented. The biggest factor here being that, due to their nature, the edge-weights can be seen as an incidence matrix and can therefore make use of matrix operations, especially for simple operations such as addition and multiplication.

1.2. Structure

The paper is structured as follows:

In Chapter 2 we will first define the different kinds of games we are concerned with and the notion of *value* as on objective for the respective games as well as (memory-less) strategies as a means to achieve those values.
In Chapter 3 we will first explain in 3.1 the idea of value and strategy iteration and then both concrete implementations of those as well as other algorithms for the respective games. In 3.2 we will then show how the different games and their underlying graphs

can be translated to one another and how the problems posed on those games are finally reduced.
In Chapter 4 we show the specifics as to how the theoretical algorithms were internally implemented and elaborate on the choices and decisions arising in the implementations of the reductions and solutions.
In Chapter 5 we show how the implementations embeds to be used to solve specific problems or to demonstrate with the GUI.
In Chaper 6 we show the kinds of graphs and problems we used to evaluate the solutions and the potential benefit of prior reduction as well as the results of the evaluation.
In Chapter 7 we interpret the results of the evaluation and give suggestions and ideas for further improvement.

2. Preliminaries

Foundational to our task are the different kinds of *Infinite Games* and how to determine the outcome of each such game. To do so, we also want a notion of *strategies* on such Infinite Games. To later reduce the games to one another, we furthermore want a definition of a *reduction* in the computational complexity sense.

2.1. Directed Graphs

Let V be a finite set of Vertices, let $E \subseteq V \times V$ be a set of Edges, then $G = (V, E)$ is a *Directed Graph.* We also define

$$\begin{aligned} \mathit{pre}\colon V &\to \mathcal{P}(V) & \mathit{pre}(v) &= \{u \in V \mid (u, v) \in E\} \\ \mathit{post}\colon V &\to \mathcal{P}(V) & \mathit{post}(v) &= \{u \in V \mid (v, u) \in E\} \end{aligned}$$

2.2. Arenas

An arena is an extension of Directed Graphs, where the set of Vertices, V, is partitioned into two disjunct subsets V_0 and V_1, respectively denoting the regions where player 0, also represented by $\Box$, and player 1, also represented by $\bigcirc$, are to play. We also require that the out-degree of every vertex is at least one, so that any play on the Arena can always be prolonged.
Formally, let (V, E) be a non-trivial Directed Graph, $V_0 \cup V_1 = V$, $V_0 \cap V_1 = \emptyset$ be a partition of V and $\forall v \in V\colon \mathit{post}(v) \neq \emptyset$, then $A = (V, (V_0, V_1), E)$ is an Arena.

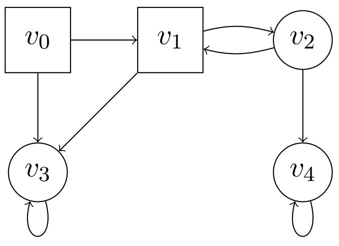


Figure 1: Arena A

Example 1:

An Arena

$$\begin{aligned} A = (&\{v_0, v_1, v_2, v_3, v_4\}, (\{v_0, v_1\}, \{v_2, v_3, v_4\}), \\ &\{(v_0, v_1), (v_0, v_3), (v_1, v_2), (v_2, v_1), \\ &(v_2, v_4), (v_4, v_4), (v_1, v_3), (v_3, v_3)\}) \end{aligned}$$

2.3. Positions, Moves

A *position*, $\pi_i = (v_0, v_1, \ldots, v_i)$, in a Game describes a finite path on the underlying graph, i.e. a position is an element of V^+. E.g. in Fig. 1 starting at v_0, (v_0, v_1, v_2, v_4) could be a play.
A *move* is an extension of a position in the graph by one more step. E.g. in Fig. 1 $(v_0, v_1, v_2) \mapsto (v_0, v_1, v_2, v_4)$ could be a move. Player i chooses the n-th move $(\ldots, v_{n-1}) \mapsto (\ldots, v_{n-1}, v_n)$ for $v_{n-1} \in V_i \in (V_0, V_1)$.

2.4. Strategies

A *strategy*, $V^* \times V_i \to V$, is a function by which player i choses the next move for any given position.
We call a strategy *memoryless*, if for any given position the next move only depends on the last vertex of the position, i.e. $V_i \to V$. We will refer to memoryless strategies of player 0 as σ and of player 1 as τ.
E.g. in Fig. 1 $\sigma = \{(v_0, v_1), (v_1, v_2)\}$ and $\tau = \{(v_2, v_1), (v_3, v_3), (v_4, v_4)\}$ could be strategies.
We call a memoryless strategy *optimal*, if it achieves the best value possible for the respective player regardless of starting vertex. There may be multiple optimal strategies for any given game.

2.5. Plays

A *play* describes the path of arbitrary length $\langle v_0, v_1, \ldots \rangle$ the player go trough in the process of playing the game. We refer to the play generated by applying strategies σ, τ to game G starting at $v_0 \in V$ as $\pi_{\sigma,\tau}(G, v_0) = \langle v_0, v_1, \ldots \rangle$
E.g. if we take the previous example strategies and apply them to $\pi_{\sigma,\tau}(A, v_0)$ we get the play $\langle v_0, v_1, v_2, v_1, \ldots \rangle$. The play can be arbitrarily prolonged by applying the moves $\sigma\colon v_1 \mapsto v_2$ and $\tau\colon v_2 \mapsto v_1$.

2.6. Infinite Games

Infinite Games are a category of games played by two players on a finite, directed graph. They are infinite in the sense that we require the out-degree of every vertex to be at least one. As such, regardless of the strategies chosen by the players, they never terminate.

2.6.1. Parity Games

Parity Games are played by two players, *Even* or player 0 ($\square$) and *Odd* or player 1 ($\bigcirc$). A Parity Game, $PG = (A, p)$, is played on an Arena A with a priority function $p\colon V \to \mathbb{N}_0$.

Let $\pi_{\sigma,\tau}(PG, v_0) = \langle v_0, v_1, \ldots\rangle$ be the *play* resulting from applying the strategies σ and τ to Parity Game PG. Let

$$\#_\infty(\pi_{\sigma,\tau}(PG, v_0)) = \\ \{i \in \{0, 1, \ldots, |V|\} \mid \forall j \in \mathbb{N}_0\colon \exists n \in \mathbb{N}_0\colon j < |\langle v \in \langle v_0, \ldots, v_n\rangle\colon p(v) = i\rangle|\}$$

be the set of priorities that appear arbitrarily often in the play.
If $max(\#_\infty(\pi_{\sigma,\tau}(PG, v_0))) \coloneqq v_{\sigma,\tau}(v_0)$, the value of vertex v_0 is even, then *Even* wins and vice versa. Optimal strategies for *Even*/*Odd* are those that result in the most/least starting vertices resulting in an even/odd value.

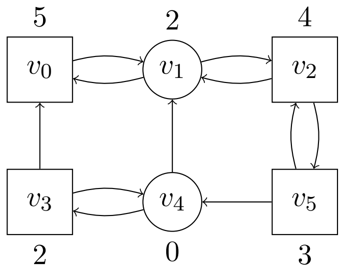


Figure 2: PG

Example 2:

Playing PG with $\sigma = \{(v_0, v_1), (v_2, v_5), (v_3, v_4), (v_5, v_2)\}$ and $\tau = \{(v_1, v_0), (v_4, v_1)\}$, the respective optimal strategies, results in play $\pi_{\sigma,\tau}(PG, v_0) = \langle v_0, v_1, v_0, ..\rangle$ for which $v_{\sigma,\tau}(v_0) = 5$ is odd and the play is therefore winning for *Odd*.

2.6.2. Mean Payoff Games

Mean Payoff Games are played by two players, *Max* or player 0 (□) and *Min* or player 1 (○). A Mean Payoff Game, $MPG = (A, w)$, is played on an Arena A and an edge-weight function $w\colon E \to \{-d, \ldots, -1, 0, 1, \ldots, d\}$, $d \in \mathbb{N}_0$.
Max aims to chose their strategy for a given play $\pi_{\sigma,\tau}(MPG, v_0) = \langle v_0, v_1, \ldots\rangle$ such as to maximize the *mean payoff*

$$\liminf_{n\to\infty} \left(\frac{1}{n} \sum_{i=0}^{n-1} w((v_i, v_{i+1})) \right).$$

We call this the *value* $v_{\sigma,\tau}(v)$ of a vertex v. Given that those values are real numbers, we will round them from here on out. Optimal strategies are those that maximize/minimize the mean payoff for a given MPG regardless of the initial vertex.

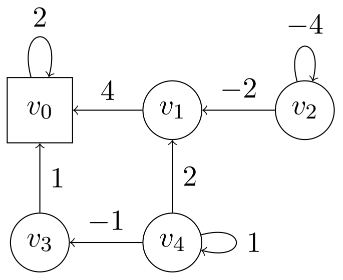


Figure 3: MPG

Example 3:

A Mean Payoff Game
$MPG = (A, \{((v_0, v_0), 2), ((v_1, v_0), 4), \ldots\})$.
Playing MPG with $\sigma = \{(v_0, v_0)\}$ and
$\tau = \{(v_1, v_0), (v_2, v_2), (v_3, v_0), (v_4, v_4)\}$, the respective optimal strategies, results in the values
$v_{\sigma,\tau}\colon v_0 \mapsto 2, v_1 \mapsto 2, v_2 \mapsto -4, v_3 \mapsto 2, v_4 \mapsto 1$

2.6.3. Energy Games

Energy Games are played by two players, *Charging* or player 0 (□) and *Depleting* or player 1 (○). An Energy Game, $EG = (A, w)$, is played on an Arena A and an edge-weight function $w\colon E \to \{-d, \dots, -1, 0, 1, \dots, d\}$, $d \in \mathbb{N}_0$.
Charging aims to chose their strategy for a given play $\pi_{\sigma,\tau}(EG, v_0) = \langle v_0, v_1, \dots\rangle$ such as to minimize the *initial credit* $c \in \mathbb{N}_0$ needed to maintain the winning condition

$$\forall k \in \mathbb{N}_0\colon \left(\sum_{i=0}^{k} w((v_i, v_{i+1}))\right) + c \geq 0.$$

We call the smallest inital credit that still maintains the winning condition for a given play the *minimum initial credit* or *value* $v_{\sigma,\tau}(v)$ of a vertex v. If no such value exists, e.g. when the vertex is of *Depleting* and has a self-loop with negative edge-weight, we write $v_{\sigma,\tau}(v) = \infty$. For any given position $\langle v_0, v_1, \dots, v_k\rangle$ in a play we call $\left(\sum_{i=0}^{k-1} w((v_i, v_{i+1}))\right) + c$ the *energy level* at that position. Keep in mind that *Charging* doesn't necessarily aim to maximise their energy level at *any* specific position but rather to maintain a non-negative energy level at *every* position.

The goal for *Charging* is to avoid getting trapped in cycles with overall negative edge-weight, as these can deplete any initial credit given, as well as to minimize the initial credit necessary to the compliant with the winning condition in all other cycles. Conversely, the goal for *Depleting* is to trap *Charging* in negative cycles or at least to maximize the initial credit necessary for *Charging* to reach a non-negative cycle. Optimal strategies are those that minimize the initial credit necessary for *Charging* and those that either deny the winning condition entirely or maximize the initial credit necessary for *Depleting.*

Example 4:

Let us reappropriate Arena A and edge-weight function w of the previous MPG example and create an Energy Game $EG = (A, 0, 4, \{((v_0, v_0), 2), ((v_1, v_0), 4), \dots\})$.
Playing EG with $\sigma = \{(v_0, v_0)\}$ and $\tau = \{(v_1, v_0), (v_2, v_2), (v_3, v_0), (v_4, v_3)\}$, the respective optimal strategies, results in the values $v_{\sigma,\tau}\colon v_0 \mapsto 0, v_1 \mapsto 0, v_2 \mapsto \infty, v_3 \mapsto 0, v_4 \mapsto 1$.

2.6.4. Discounted Payoff Games

Discounted Payoff Games are played by two players, *Max* or player 0 (□) and *Min* or player 1 (○). A Discounted Payoff Game, $DPG = (A, w, \lambda)$, is played on an Arena A, an edge-weight function $w\colon E \to \{-d, \dots, -1, 0, 1, \dots, d\}$, $d \in \mathbb{N}_0$ and a discount factor $0 < \lambda < 1$.
Max aims to chose their strategy for a given play $\pi_{\sigma,\tau}(DPG, v_0) = \langle v_0, v_1, \dots\rangle$ such as to maximize the *discounted payoff*

$$(1 - \lambda)\left(\sum_{i=0}^{\infty} \lambda^i \cdot w((v_i, v_{i+1}))\right).$$

Again we call this the *value* $v_{\sigma,\tau}(v)$ of a vertex v, we will round them from here on out. Optimal strategies are those that maximize/minimize the discounted payoff for a given MPG regardless of the initial vertex.

Example 5:

Let us again reappropriate Arena A and edge-weight function w and create a Discounted Payoff Game $DPG = (A, \{((v_0, v_0), 2), ((v_1, v_0), 4), \dots\}, 0.95)$.
Playing DPG with $\sigma = \{(v_0, v_0)\}$ and $\tau = \{(v_1, v_0), (v_2, v_2), (v_3, v_0), (v_4, v_4)\}$, the respective optimal strategies, results in the values $v_{\sigma,\tau}\colon v_0 \mapsto 2, v_1 \mapsto 2.1, v_2 \mapsto -4, v_3 \mapsto 1.95, v_4 \mapsto 1$.

2.7. Simple Stochastic Games

Simple Stochastic Games are played by two players, *Max* or player 0 ($\square$) and *Min* or player 1 ($\bigcirc$). A Simple Stochastic Game, $SSG = (G, (V_0, V_1, V_2), \mathtt{0}, \mathtt{1}, p)$, is played on a Directed Graph G, with a partition (V_0, V_1, V_2), two sink vertices $\mathtt{0}, \mathtt{1}$ and a probability function $p\colon V_2 \to (V \to [0,1])$. We require that the probabilites of all outgoing edges of each V_2 vertex sum to 1: $\forall v \in V_2\colon \sum_{u \in V} p(v)(u) = 1$ and that $\forall (u,v) \in (V_2 \times V) \setminus E\colon p_u(v) = 0$. On vertices $v \in V_2$, also called *Average* or *Random* ($\Diamond$) vertices, the next vertex gets chosen according to the probability function $p_v\colon V \to [0,1]$ rather than a player. We also require, like for Arenas, that the out-degree of every vertex is at least one, with the exception of $\mathtt{0}, \mathtt{1}$, for which it is zero.
Max wins if the $\mathtt{1}$ sink is reached, *Min* wins if the $\mathtt{0}$ sink is reached or the game doesn't terminate. Since the result of a play $\pi_{\sigma,\tau}(SSG, v_0)$ can be probablistic, it is assigned a probabilty to reach the $\mathtt{1}$ sink rather than a fixed value. We display this probability as a rounded real number. We say that a SSG is *stopping* if for every possible position in a play, so given two fixed strategies σ, τ, there is a path to one of the sink vertices. All SSGs we will consider will be stopping-SSGs. Since stopping-SSGs terminate, they are not Infinite Games. For simplicity, we will still include SSGs under the *Infinite Games* label from here on out. The goal for *Max* in a stopping-SSG is to maximize the chance to reach the $\mathtt{1}$ vertex, the goal for *Min* to reach the $\mathtt{0}$ vertex. Optimal strategies are those that maximize the chance to reach the desired sink-vertex of the respective player regardless of the initial vertex.

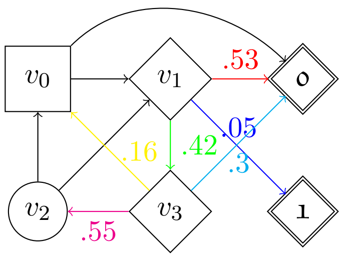


Figure 4: SSG

Example 6:

A Simple Stochastic Game

$$SSG = (G, (\{v_0\}, \{v_2\}, \{v_1, v_3\}), \mathtt{0}, \mathtt{1},$$
$$\{(v_1, \{(\mathtt{0}, .53), (\mathtt{1}, .05), (v_3, .42)\}),$$
$$(v_3, \{(\mathtt{0}, .3), (v_0, .16), (v_2, .55)\})\}).$$

Playing SSG with $\sigma = \{(v_0, v_1)\}$ and $\tau = \{(v_2, v_0)\}$, the respective optimal strategies, results in the values $v_{\sigma,\tau}\colon v_0 \mapsto .07, v_1 \mapsto .07, v_2 \mapsto .07, v_3 \mapsto .05, \mathtt{0} \mapsto 0, \mathtt{1} \mapsto 1$.

2.8. Reductions

A *reduction* describes an algorithm with which we can transform (reduce) one class of problems to another. For our purposes, we choose target problem classes that are already solved. Aside from reducing to an already solved problem class and therefore, per definition, also solving the original problem class, reducing and subsequent solving of instances of the original problem may be faster than any attempt to directly solve the original problem without reducing to another intermediary infinite game problem.
Formally, we express our problem classes as formal languages A and B over the alphabets Σ^* and Γ^*. If f is totally computable and

$$f\colon \Sigma^* \to \Gamma^*, A \subseteq \Sigma^*, B \subseteq \Gamma^*$$
$$\forall w \in \Sigma^*\colon w \in A \iff f(w) \in B$$

then f is a reduction from Σ^* to Γ^*.
In practice our formal languages A and B will be some instances of types of infinite games together with statements about such games, such as the values under perfect play or perfect strategies. The reduction will then draw an equivalence to the other infinite game, e.g. if $w =$ "$v(v_u) = 5$ in G" is a word in A then $f(w) =$ "$v(v_v) = 16$ in H" is a word in B.

3. Solutions and Reductions in Theory

Before we go into the specifics of the implementation of the algorithms used to solve the problems and the reductions we first want to comprehensively explain the theory behind each of them.

3.1. Solutions in Theory

One of the key parts of our work is the solving of problems on our games, be it in conjunction with a reduction or not. First we shall explain the general idea of Kleene Iteration and Strategy Iteration, as they can be generalized and used to solve multiple of the problems posed. Then we will explain the specifics of each implementation of those as well as unique algorithms used to solve each of the games we are interested in.

3.1.1. Kleene Iteration

Kleene

3.1.2. Strategy Iteration

3.1.3. PGs

3.1.4. MPGs

3.1.5. DPGs

3.1.6. EGs

3.1.7. SSGs

3.2. Reductions in Theory

3.2.1. PGs to MPGs

The reduction from PGs to MPGs is based on the idea of [Jur98]. For a given $PG = (A, p)$ we transform into a $MPG = (A, w)$, with the underlying arena A being the same and the weights w of MPG based directly on the priority function p of PG. Optimal strategies in MPG translate one to one to PG and the vertex v having value ≥ 0 in MPG translates to it being winning for *Even* in PG. Recall that for both PGs and $MPGs$ winning strategies can be memoryless.

Consider any play $\pi_{\sigma,\tau}$ in A with $n = |V|$. Given that A is finite and $\pi_{\sigma,\tau}$ can continue infinitely, any such play, regardless of starting vertex v_0 will necessarily have a repeat vertex after a path of length $k \leq n$ and will then, by virtue of the strategies being memoryless, indefinitely continue in a cycle of length $l \leq n - k$.

From the definition of the value of a PG it follows directly that only the priorities of the nodes in the cycle matter for the outcome to the game, specifically the highest priority. For MPGs we have:

$$\liminf_{n\to\infty}\left(\frac{1}{n}\sum_{i=0}^{n-1} w((v_i, v_{i+1}))\right) =$$

$$\liminf_{n\to\infty}\left(\frac{1}{n}\sum_{i=k}^{n-1} w((v_i, v_{i+1}))\right) + \liminf_{n\to\infty}\left(\frac{1}{n}\sum_{i=0}^{k-1} w((v_i, v_{i+1}))\right) =$$

$$\liminf_{n\to\infty}\left(\frac{1}{n}\sum_{i=0}^{n-1} w((v_{i+k}, v_{i+1+k}))\right) =$$

$$\liminf_{n\to\infty}\left(\frac{1}{n}\sum_{i=0}^{n-1}\frac{1}{l}\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))\right) =$$

$$\frac{1}{l}\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))$$

So again, the only edge-weights that matter for the value of a play are those that are part of the terminal cycle.

To equate the values of the games we want to set the edge-weights in such a way that the outgoing edge of any vertex of a given cycle alone can set the parity of the value of the game if and only if the priority of the vertex is the highest in the cycle.
[Jur98] does this by setting $w((u,v)) = (-n)^{p(u)}$. This way, the absolute weight of the outgoing edge of the vertex with highest priority is always higher than any possible make-up of the rest of the cycle, i.e. $|(-n)^{p(u)}| > (n-1)|(-n)^{p(u)-1}|$.
This, however, is unsatisfying since the complexity of solving MPGs with the algorithms we use is dependent on d, the maximum absolute weight in the MPG. We therefore want to be conservative with the edge-weights and try to minimize them.
To that end, we set our weights as follows:

$$w((u,v)) = (-1)^{p(u)} \cdot (sum\langle z \in \{|w((x,y))| \mid (x,y) \in E\} \mid x \in {V_u}^1\rangle + 1^{p(u)\ mod\ 2})$$

This way the vertex with highest priority has an outgoing edge which, per definition, dominates the value of the MPG since it has a higher absolute weight than all vertices of lower priority of the opponent, regardless of their counterstrategy τ. Thus any play on PG whose value is *Even/Odd* will have a value of $\geq 0/\leq -1$ when translated to an MPG in such a way and any optimal strategy in *PG* will accordingly be an optimal strategy in *MPG*. Conversely, any MPG that will be reduced-to in such a way will have an edge in it's terminal cycle whose absolute weight will be greater than the sum of all the edges of the opponent in said cycle. The vertex said greatest edge emanates from being exactly equivalent to the vertex with highest priority in the original PG. Thus any play on such MPG whose value, or equivalently whose greatest edge, is $\geq 0/\leq -1$ will have *Even/Odd* value in the original MPG and any optimal strategy in MPG will be a optimal strategy in PG.

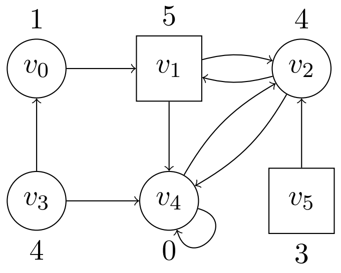
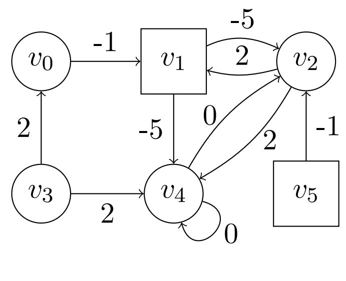


Figure 5: PG

Example 7: We translate *PG* to *MPG*.
The edge-weights are successively built as follows:

$u \in V$	$p(u)$	V_u	2	$w((u,\_))$
v_4	0	$\emptyset$	$\emptyset$	0
v_0	1	$\{v_4\}$	$\langle 0\rangle$	-1
v_5	3	$\{v_4\}$	$\langle 0\rangle$	-1
v_2, v_3	4	$\{v_0, v_5\}$	$\langle 1,1\rangle$	2
v_1	5	$\{v_2, v_3, v_4\}$	$\langle 2,2,0\rangle$	-5



Here we can limit d to 5 instead of $|(-6)^5| = 7776$.

Figure 6: MPG

[1]$V_u = \{v \in V \mid p(x) < p(u) \land (p(x) \neq p(u) \bmod 2)\}$ and therefore $V_u = \emptyset$ for $u = min\{p(v) \mid v \in V\}$
[2]$\langle z \in \{|w((x,y))| \mid (x,y) \in E\} \mid x \in V_u\rangle$

3.2.2. MPGs to DPGs

The reduction from MPGs to DPGs is the one described in [ZP96]. The underlying Arena A and edge-weights w stay the same. Recall from 3.2.1 that a play in an Arena eventually results in a repeat vertex after a path of length $k \leq n$ and then indefinitely continues in a cycle of length $l \leq n - k$. The value of such play being:

$$\frac{1}{l}\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))$$

For DPGs we have the value being:

$$\begin{aligned}
v(v_0) = \ & (1-\lambda)\left(\sum_{i=0}^{\infty} \lambda^i \cdot w((v_i, v_{i+1}))\right) = \\
& (1-\lambda)\left(\sum_{i=k}^{\infty} \lambda^i \cdot w((v_i, v_{i+1}))\right) + (1-\lambda)\left(\sum_{i=0}^{k-1} \lambda^i \cdot w((v_i, v_{i+1}))\right) \\
& (1-\lambda)\left(\sum_{i=0}^{\infty} \lambda^{i+k} \cdot w((v_{i+k}, v_{i+1+k}))\right) + (1-\lambda)\left(\sum_{i=0}^{k-1} \lambda^i \cdot w((v_i, v_{i+1}))\right) \\
& (1-\lambda)\lambda^k\left(\sum_{i=0}^{\infty} \lambda^{il} \cdot \sum_{j=0}^{l-1} \lambda^j \cdot w((v_{j+k}, v_{j+1+k}))\right) + (1-\lambda)\left(\sum_{i=0}^{k-1} \lambda^i \cdot w((v_i, v_{i+1}))\right) \\
& \frac{(1-\lambda)\lambda^k}{1-\lambda^l}\left(\sum_{j=0}^{l-1} \lambda^j \cdot w((v_{j+k}, v_{j+1+k}))\right) + (1-\lambda)\left(\sum_{i=0}^{k-1} \lambda^i \cdot w((v_i, v_{i+1}))\right) \quad (1)
\end{aligned}$$

Now if we let $\lambda \to 1^-$ we get:

$$\begin{aligned}
& \lim_{\lambda \to 1^-}\left(\frac{(1-\lambda)\lambda^k}{1-\lambda^l}\left(\sum_{j=0}^{l-1} \lambda^j \cdot w((v_{j+k}, v_{j+1+k}))\right) + (1-\lambda)\left(\sum_{i=0}^{k-1} \lambda^i \cdot w((v_i, v_{i+1}))\right)\right) = \\
& \lim_{\lambda \to 1^-}\left(\frac{1-\lambda}{1-\lambda^l}\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))\right) = \\
& \frac{1}{l}\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))
\end{aligned}$$

Since we can't actually let λ be 1, we have to use a λ that is sufficiently large enough as to enable us to truncate to the nearest rational number as was done in 3.1.4.1. We can take (2) and rescale all the weights by W, such that all weights are ≥ 0:

$$v(v_0) + W =$$

$$\frac{(1-\lambda)\lambda^k}{1-\lambda^l}\left(\sum_{j=0}^{l-1}\lambda^j \cdot (w((v_{j+k}, v_{j+1+k})) + W)\right) + (1-\lambda)\left(\sum_{i=0}^{k-1}\lambda^i \cdot (w((v_i, v_{i+1})) + W)\right) \geq$$

$$\frac{(1-\lambda)\lambda^k}{1-\lambda^l}\left(\sum_{j=0}^{l-1}\lambda^j \cdot (w((v_{j+k}, v_{j+1+k})) + W)\right) \geq$$

$$\frac{(1-\lambda)\lambda^{k+l-1}}{1-\lambda^l}\left(\sum_{j=0}^{l-1}(w((v_{j+k}, v_{j+1+k})) + W)\right) \geq$$

$$(1-n(1-\lambda))\left(\sum_{j=0}^{l-1}(w((v_{j+k}, v_{j+1+k})) + W)\right) \geq$$

$$(1-n(1-\lambda))\left(W + \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))\right)$$

Now if we revert the scaling we arrive at:

$$v(v_0) + W \geq (1-n(1-\lambda))\left(W + \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))\right) \Leftrightarrow$$

$$v(v_0) + W \geq W + \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k})) - n(1-\lambda)\left(W + \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))\right) \Leftrightarrow$$

$$v(v_0) \geq \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k})) - n(1-\lambda)\left(W + \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k}))\right) \geq$$

$$\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k})) - n(1-\lambda) \cdot 2W$$

Similarly, we can rescale by -W, such that all weights are ≤ 0 and in total we arrive at:

$$\sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k})) + n(1-\lambda) \cdot 2W \geq v(v_0) \geq \sum_{j=0}^{l-1} w((v_{j+k}, v_{j+1+k})) - n(1-\lambda) \cdot 2W$$

From 3.1.4 we know that the minimum distance between two valid values of a MPG is $\frac{1}{n(n-1)}$, therefore we want to set λ in such a way as to constrict the size of the just derived interval to:

$$|2n(1-\lambda) \cdot 2W| \leq \frac{1}{n(n-1)}$$

We get this precision with $\lambda = 1 - \frac{1}{4n^3W}$. If we reduce a MPG to a DPG with such λ, we can deduce the value for a given vertex of the original MPG by taking the corresponding value v in the DPG and rounding to the unique rational number, with denominator at most n, in the closed interval $v \pm \frac{1}{2n(n-1)}$.

3.2.3. MPGs to EGs

The reduction from MPGs to EGs is the one described in [BCD^{+}11].
If we play an EG on A, w of a MPG, any vertex for which the value is $v_{\sigma,\tau}(v) \neq \infty$ is equal to a vertex in the MPG whose value is $v_{\sigma,\tau}(v) \geq 0$, since the player can force a terminal cycle whose edge-weights sum up to ≥ 0. We can use this dichotomy together with the fact that we can rescale the edge-weights of MPGs, allowing us to precisely determine any value of the MPG in the EG, to repeatedly solve rescaled MPGs and restrict the possible values until only one possible solution remains.

For a given set of strategies σ, τ, any move will land on a vertex with the same value as the previous since the play ultimately always lands in the same terminal cycle with the same mean weight. This means that, for a fixed set of strategies, vertices with different values will necessarily be disconnected if one removes the edges that aren't in play under σ, τ and the values of all vertices stay the same if those edges are removed.
What this allows us to do is that after every dichotomy wrt. the values, where we split the set of vertices into three sets, one with value $> \nu$, one with value $< \nu$ and one with known value ν. We can restrict subsequent dichotomies to disjoint subsets of the graph, since we know that vertices with different value, as shown to be the case by the dichotomy, have to be disconnected if unused edges are removed.

3.2.4. DPGs to SSGs

4. Solutions and Reductions in Practice

4.1. Solutions in Practice

4.2. Reductions in Practice

5. Implementation

6. Evaluation

Our goal was to compare the different algorithms used to solve problems, especially under reduction to other games, and to see how they compare. To that end we generated random games with multiple parameters and benchmarked all methods we had to solve the respective problems. For any combination of problem, parameters for generation and algorithm used to solve the problem we have $N = 100$ and the times shown repesent the average time of those 100 runs. For their generation, all games have a parameter $n = |V|$ that measures the size of the graph by means of the number of vertices and a parameter p which describes the probability that any possible edge $e \in (V \times V)$ is part of the generated game, meaning the expected out-degree of every vertex is $|V| \cdot p$. Since we require for all games that each vertex has outdegree ≥ 1, we consequently require that $\frac{1}{|V|} \leq p \leq 1$. We then assign each vertex one random edge and all other potential outgoing edges of that vertex have probability $\frac{(p \cdot |V|)-1}{|V|-1}$ of existing. p can be understood as a measure of how connected a graph is. In fact, towards the extreme ends, for $p = \frac{1}{|V|}$ and sufficiently large $|V|$ one would expect a disconnected graph on one end and for $p = 1$ a complete graph on the other end. Each vertex is randomly assigned to a player with equal probability, including the "Random" player in SSGs. Games may have additional parameters that are described seperately for each game. The parameters were chosen in a way that they intuitively represent extremes towards each end and then some reasonable middle ground. For all cases, the runtimes are either weakly dependent on p other parameters or they scale monotonously wrt. to those parameters. One can therefore think of intermediate parameterizations as some interpolation between the shown values. For readability we only show some algorithms here. Those that work without reduction to another game and any other that are of interest. The full results are in the appendix.

6.1. PGs

For Parity games, the additional parameter is w, which describes the range over which the priorities are handed out. Similar to the ownership of the vertices, the priority of vertices is randomly chosen from the possible range with equal chance for each. In theory as well as in this implementation, for the purpose of solving the problems one can rescale the range of the priorities to $\{0, 1, ...|V|\}$ or smaller without affecting the values or perfect strategies of the game. A higher w however decreases the overlap of priorities between vertices on generation, which has a minor impact on the runtime of Zielonka's algorithm.

In theory, Strategy Iteration via DPGs ought to be a contender for large enough n for both the value and stratey problem. The reason for why it isn't depends on the reduction from MPGs to DPGs and is laid out in the subsection for MPGs.

Strategy Iteration from above via EGs can outperform Zielonka's algorithm for large $|V|$ and reasonably dense graphs. However, the break-even point is only reached for such large $|V|$ and consequently such long runtimes that gathering statistically significant runtime data is impractical, it can however be confirmed for individual instances. The exact inflection points for the parameterization when the Strategy Iteration starts outperforming Zielonka's algorithm are therefore not known.

6.1.1. Value Problem of PGs

For solving the Value Problem of PGs the result are very clear. For all cases tested, Zielonka's Algorithm is straight up the fastest. It's simplicity demands very little overhead and as such small graphs are solved near instantaneously. It's recursive nature allows it to scale very well to big graphs as well. Less connectivity, as shown by smaller p, mean the attractors as laid out in the algorithm tend to not reach as far – meaning more levels of recursion are needed to cover the entire graph. Similargly, bigger w means that the winning regions that act as a crystallisation point for the attracters tend to start out smaller and thus form smaller attractors.

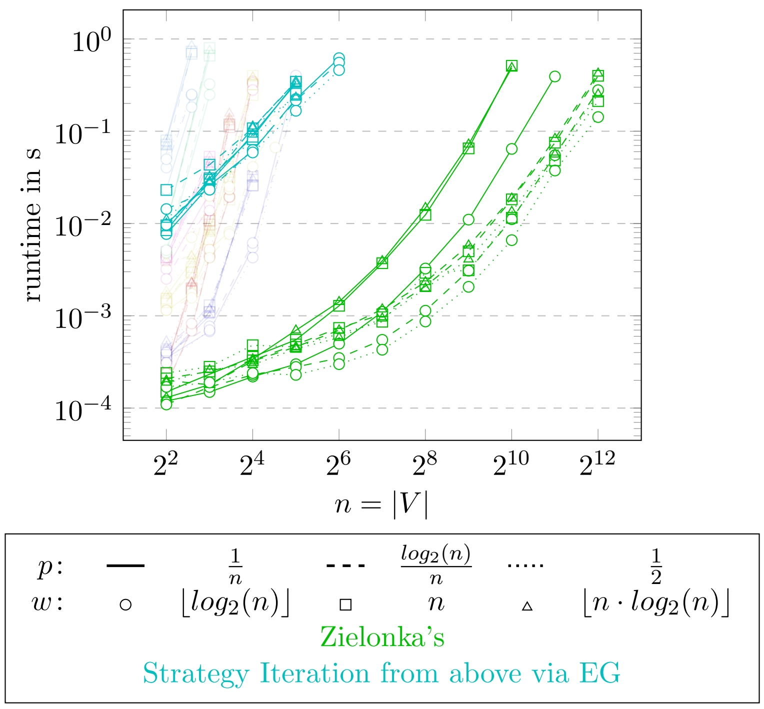


Figure 7: Solving the value problem of PGs

6.1.2. Strategy Problem of PGs

Unfortunately our naive approach to finding optimal strategies via Zielonka's algorithm is $|V| \cdot log_2(\frac{|E|}{|V|})$ times slower than finding the values alone, meaning the big lead it had in finding the values of games is diminished significantly, bringing it more in line with other algorithms. Depending on the exact p and w, the BCDGR algorithm via EGs, which has similarly low overhead to Zielonka's algorithm, but doesn't require repeated invokation to deliver optimal stragies, has better runtimes for reasonably small graphs. However, since it scales worse, such advantage is quickly lost with growing $|V|$. After that, Zielonka's algorithm, even with the malus it suffer for finding optimal strategies, is the fastest available method until Strategy Iteration from above via EGs takes over.

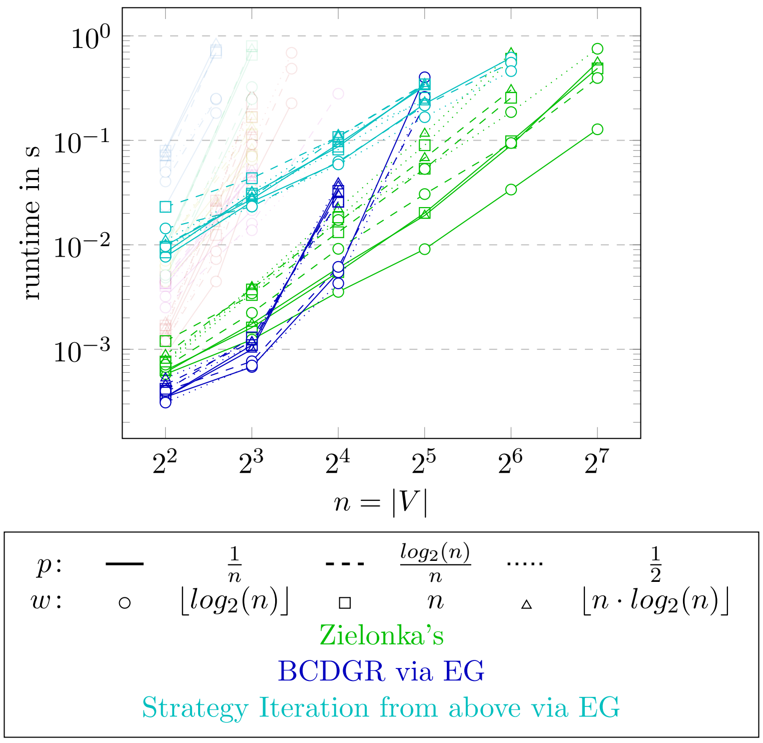


Figure 8: Solving the strategy problem of PGs

6.2. MPGs

As with Parity games, the additional parameter is w describing the range over which the priorities are handed out and for our random generation they are once again randomly uniformly distributed over their range. Unlike with PGs, in this case the meaning of w cannot be scaled down to make subsequently applied algorithms faster. As a result we see some algorithms scale strongly in response to changing w across it's range and even beyond.

In theory, Strategy Iteration via DPGs is faster than other algorithms for large enough n. In practise however this isn't easily achieved. The method by which MPGs are reduced to DPGs results in discount factors very close to 1. So close in fact, that for big enough MPGs the difference $1 - \lambda$ approaches the limit of the precision standard floating point implementations can represent [75419]. As a result, standard linear program solvers are unable to solve the linear programs that are part of the strategy iteration with adequate precision. One could conceive a linear program solver that works with larger floats that enable higher precision. However this only raises the ceiling of what is solveable marginally. It also eventually forces the floats to exceed standard register sizes, meaning the floating point arithmetic fundamental to the linear program solver would have to resort to emulating floating point arithmetic for larger floats, which would take more clock cycles per operation and ultimately longer runtimes, undermining the usefullness of the approach in the first place. Effectively this mean that, even if this approach is possible, it is never pratically viable for MPGs reduced to DPGs.

6.2.1. Value Problem of MPGs

Unlike for PGs the native approach used here, Zwick and Paterson's algorithm isn't a clear winner. Intuitively the $k = 4 \cdot |V|^3 \cdot d$ iterations the algorithm goes through to guarantee correcteness are way in excess of what's practically needed. This means it's only competetive for graphs with very small $|V|^3 \cdot d$. As with PGs, solving with BCDGR via EG offers low overhead and thus reasonably fast runtimes for smaller graphs, but again as with PGs the approach scales mediocrely wrt. $|V|$.
As with PGs, Strategy Iteration from above via EGs eventually outperforms BCDGR via EG for some parameterizations. *Unlike* with PGs, we don't suffer from an as large d any more, meaning the approach becomes competitive much sooner. The exact inflection point depend on the specific MPG in question, with a relatively large overlap where either may be superior.

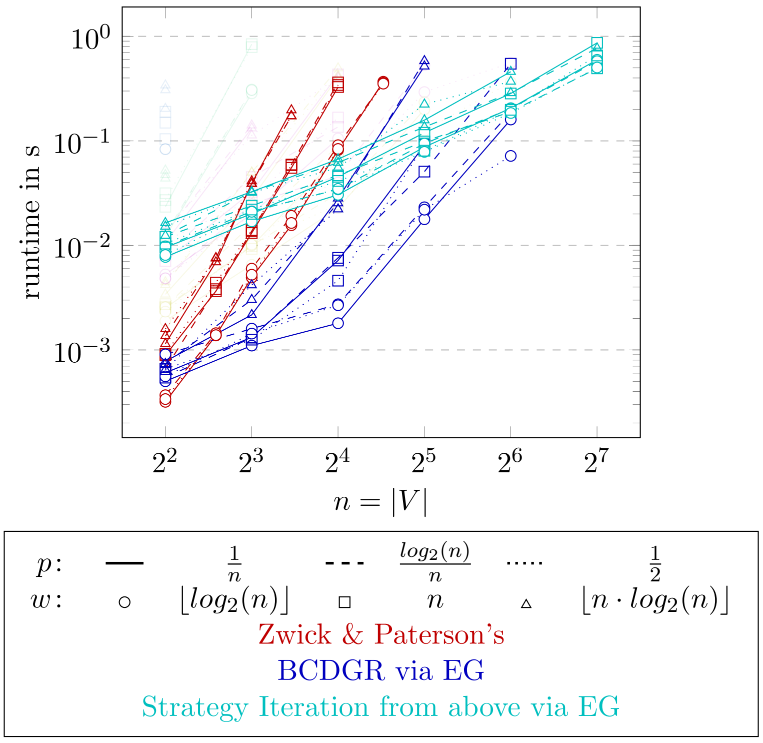


Figure 9: Solving the value problem of MPGs

6.2.2. Strategy Problem of MPGs

As with Zielonka's algorithm on PGs, Zwick and Paterson's algorithm for optimal strategies has an additional cost of $|V| \cdot log_2(\frac{|E|}{|V|})$ over finding the values of the gmae.

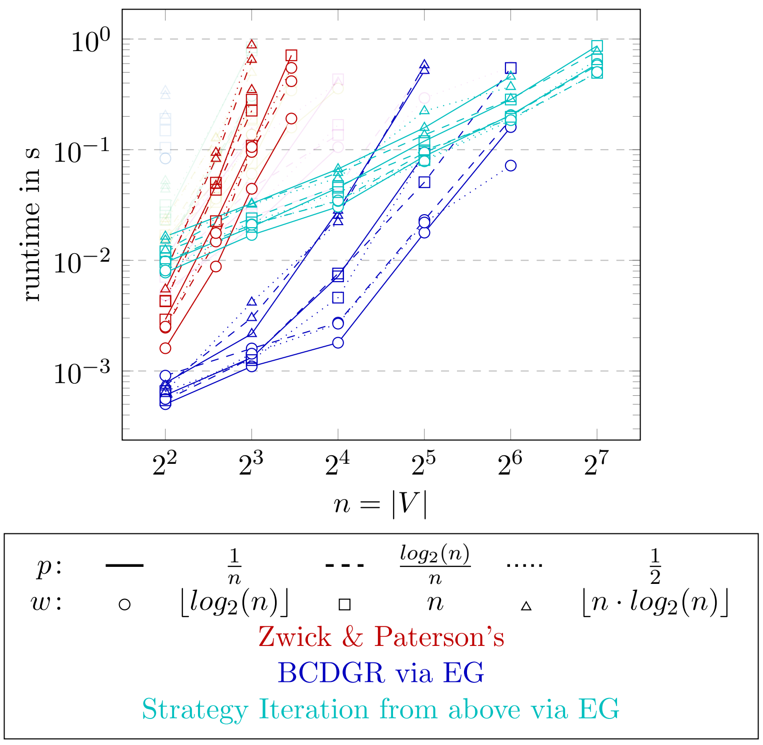


Figure 10: Solving the strategy problem of MPGs

6.3. EGs

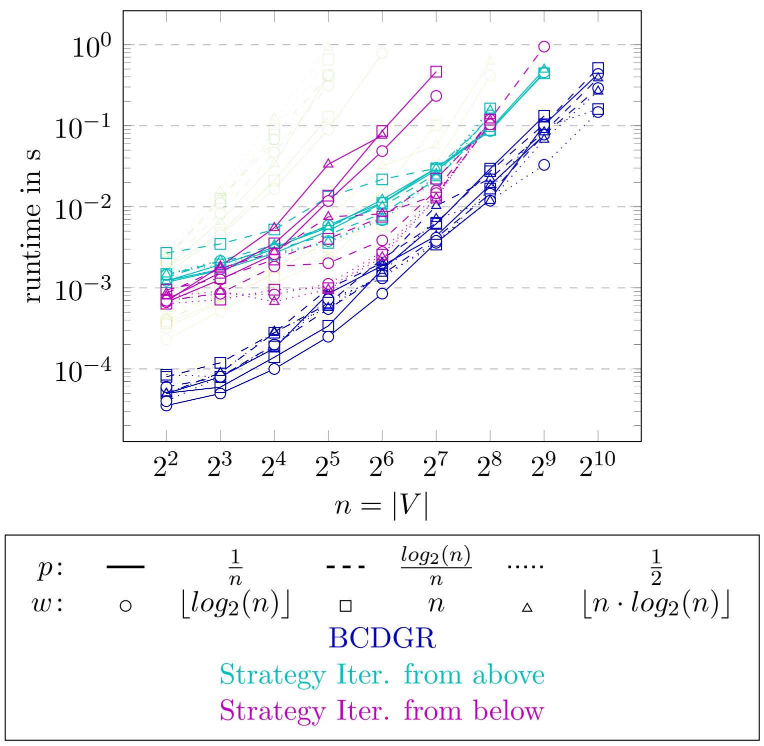


Figure 11: Solving both EG problems

6.4. DPGs

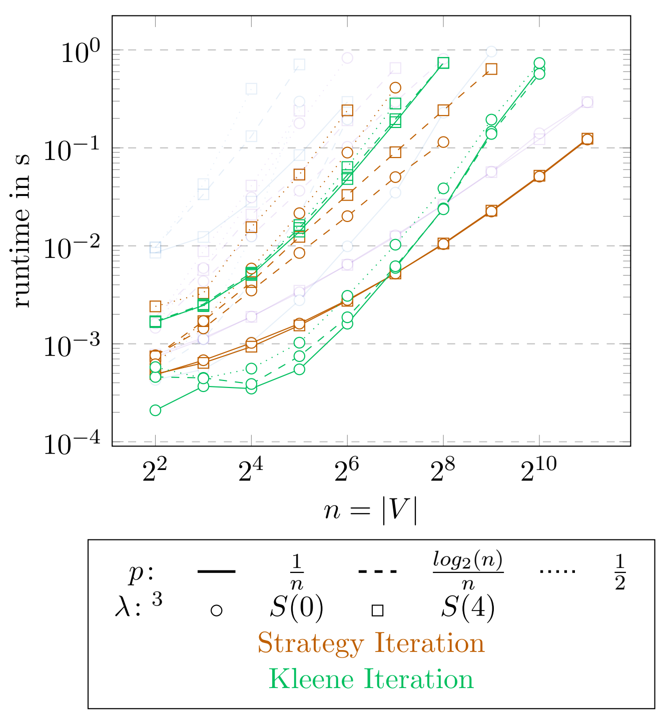


Figure 12: Solving both DPG problems

6.5. SSGs

sinks not included in p no w

[3] $S(x) = \frac{1}{1+e^{-x}}$

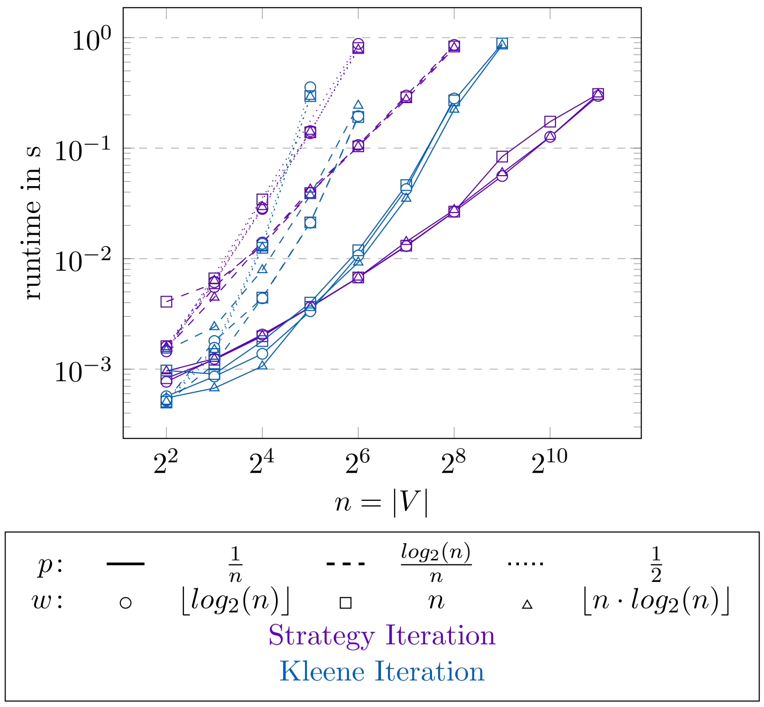


Figure 13: Solving both stopping-SSG problems

7. Conclusion and further approaches

A. Versicherung an Eides Statt

Ich versichere an Eides statt durch meine untenstehende Unterschrift,

- dass ich die vorliegende Arbeit - mit Ausnahme der Anleitung durch die Betreuer - selbstständig ohne fremde Hilfe angefertigt habe und
- dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und
- dass ich ausschließlich die angegebenen Quellen (Literatur, Internetseiten, sonstige Hilfsmittel) verwendet habe und
- dass ich alle entsprechenden Angaben nach bestem Wissen und Gewissen vorgenommen habe, dass sie der Wahrheit entsprechen und dass ich nichts verschwiegen habe.

Mir ist bekannt, dass eine falsche Versicherung an Eides Statt nach § 156 und nach § 163 Abs. 1 des Strafgesetzbuches mit Freiheitsstrafe oder Geldstrafe bestraft wird.

Ort, Datum	Unterschrift

References

[75419] IEEE Standard for Floating-Point Arithmetic. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), S. 1–84. `http://dx.doi.org/10.1109/IEEESTD.2019.8766229`. – DOI 10.1109/IEEESTD.2019.8766229

[BCD$^+$11] Brim, L. ; Chaloupka, J. ; Doyen, L. ; Gentilini, R. ; Raskin, J. F.: Faster algorithms for mean-payoff games. In: *Formal Methods in System Design* 38 (2011), Apr, Nr. 2, 97-118. `http://dx.doi.org/10.1007/s10703-010-0105-x`. – DOI 10.1007/s10703–010–0105–x. – ISSN 1572–8102

[Jur98] Jurdzinski, Marcin: Deciding the Winner in Parity Games is in UP \cap co-Up. In: *Inf. Process. Lett.* 68 (1998), Nr. 3, 119–124. `http://dx.doi.org/10.1016/S0020-0190(98)00150-1`. – DOI 10.1016/S0020–0190(98)00150–1

[ZP96] Zwick, Uri ; Paterson, Mike: The complexity of mean payoff games on graphs. In: *Theoretical Computer Science* 158 (1996), Nr. 1, 343-359. `http://dx.doi.org/https://doi.org/10.1016/0304-3975(95)00188-3`. – DOI https://doi.org/10.1016/0304–3975(95)00188–3. – ISSN 0304–3975