Linear algorithms for Parity Games with the Signature of a Potential

Andrei Dumitriu

August 2022

Supervisors:
Jorge A. Pérez
Oliver Lorscheid

Abstract

Solving parity games has been an important problem in Computer Science, not only because of the applications of this infinite duration game in satisfiability checking and model checking, but also because it is one of the few problems that is both in UP and in co-UP, but not also in P. We propose two new algorithms for solving parity games, based on the newest developments in regards to mean payoff games. The first will solve the parity game using its equivalent mean payoff game. The second one will have a similar structure to the mean payoff game solver, except the winning positions will be found directly in the parity game, with the use of a strategy improvement algorithm. Both algorithms will have linear time complexity for a certain type of games, known as games with the "signature of a potential".

1 Introduction

Definition. Parity games are two-player games played on a finite graph $G = (V, E)$, with V being the finite set of vertices of the graph and $E \subseteq V \times V$ being the set of edges. The two players, P_1 and P_2, each get assigned a subset of the graph's vertices: V_1, V_2, $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$. We refer to a vertex $v \in V_1$ as being owned by P_1 and similarly for P_2. The function $\Omega = V \to \mathbb{N}^*$ is a *priority* function that assigns a natural number value to each of the vertices of the graph G. The game is played using a token that is moved from vertex to vertex on the edges of a graph, forming a sequence of vertices $v_0, v_1, v_2, ... \in V$, where $(v_0, v_1) \in E$, $(v_1, v_2) \in E$ and so on. This sequence of vertices is called the *play*. The choice of which vertex will follow a given vertex v_x is down to the owner of that vertex: if $v_x \in V_1$, the $P1$ chooses $v_y \in V$ such that $(v_x, v_y) \in E$ and similarly if $v_x \in V_2$, then P_2 makes the choice. If the play reaches a vertex with no outgoing edges, known as a *sink*, then we refer to this situation as a finite duration game, with the player that owns that vertex being declared the loser. Otherwise, in the case of an infinite duration game, the victor is decided based on the parity of the largest priority of the infinitely repeating vertices of the play: if the priority is odd, then P_1 wins, otherwise P_2 is the winner. From the rest of the paper, a parity game will be described as the following tuple: $\mathcal{P} = (V, E, V_1, V_2, \Omega)$.

It is said that P_1 and P_2 have *positional strategies* [1]. This means that the best choice for P_x at a given vertex $v \in V_x$ does not depend on the previous moves in the play. For a player P_x to have a winning strategy at v, it means that there is a play that contains v such that P_x wins the play.

This game is part of the field of scientific computing and has an important role in both satisfiability checking and model checking. Satisfiability refers to checking whether the variables of a boolean formula can be assigned values of either true or false such that the entire formula is evaluated as true. Model checking refers to verifying that a finite-state model meets a given specification.

Parity games are known to be in the UP complexity class (unambiguous non-deterministic polynomial time). Problems in this complexity class have at most one solution and a proposed solution can be verified in polynomial time. This problem is also known to be in the complement of UP, co-UP, given that the complement of the problem from the perspective of one player is the same game, but from the perspective of the other player. There have been many attempts to prove that parity games can be solved in polynomial time, however the best time complexity that has been achieved so far is, for n nodes and m distinct priorities, $O(n^{log(m)+6})$ [2].

In an upcoming paper, Marianne Akian, Stéphane Gaubert, Oliver Lorscheid, and Matthias Mnich have developed a new algorithm for mean payoff games [3]. Based on their research, we will develop two algorithms with two different approaches for solving parity games: a basic approach, which solves the parity game by transforming it into a mean payoff game and then uses Akian et al.'s algorithm [3] as is, and a strategy improvement approach, where we use the general structure and ideas of Akian et al's algorithm [3], without using the equivalent mean payoff game, and combine it with Fearnley's strategy improvement algorithm for solving parity games [4]. We will also show that for a certain subset of games, known as games with the "signature of a potential", the two algorithms provide a solution in linear time. Additionally, implementations in C of the algorithms were created and their performance was compared on a variety of parity games.

2 Solving mean payoff games

First, we need to define mean payoff games. There are multiple variations of mean payoff games, but we will go with the definition used by Akian et al. [3]. Changing from one type of mean payoff game to another is done in polynomial time [3].

2.1 Mean payoff games

Definition. Mean payoff games are another type of zero-sum games played on finite graphs $G = (V, E)$, with two players: P_- and P_+. The vertices are once again divided into two sets, each belonging to one of the players: V_-, V_+, $V_- \cap V_+ = \emptyset$, $V_- \cup V_+ = V$. Additionally, each edge is assigned a *weight* based on a function $\mu : E \to \mathbb{Z}$. At a vertex $v \in V_x$, $x \in \{-, +\}$, the playerP_x chooses an adjacent vertex $w \in V$, $(v, w) \in E$. As such, we have a finite or infinite sequence of the type (v_0, v_1), (v_1, v_2), ..., also known as the *play* $\pi = (v_0, v_1, ...)$. A finite play $\pi = (v_0, v_1, ...v_n)$ occurs if $v_n \in V_x$, $x \in \{-, +\}$ has no outgoing edges, in which case P_x loses and the other player wins. Otherwise, in the case of an infinite play, if $lim\ inf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} \mu(v_i, v_{i+1}) < 0$, then player P_- wins, whereas if $lim\ sup_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} \mu(v_i, v_{i+1}) > 0$, player P_+ wins. If neither player can reach their winning condition, then there is a draw.

2.2 Solving mean payoff games via reachability games

This subsection describes Akian et al.'s algorithm for solving parity games[3]. Note that this description is based on a draft of the paper and as such there may be some minor differences between what will be said here and the final, published version of the paper.

K-obsolete edges. An edge (v, w), with $v \in V_x$ is a k-obsolete edge if after P_x chooses this edge, P_y has a position strategy such that no matter what other choices P_x makes, the play returns to the vertex v, forming a cycle of maximum length k with a payoff favorable to P_y. This means that it is never advantageous for P_x to play that edge, so they would never choose it in favor of other options. As such, we can remove the edge from the graph without the outcome of the game being affected.

In order to determine if the vertex $v \in V_x$ has outgoing edges that are k-obsolete, we use the following functions:

- $\epsilon : V \to \{-1, 1\}$, $\epsilon(v) = \begin{cases} -1 & if\ v \in V_-, \\ 1 & if\ v \in V_+, \end{cases}$
- $\eta_i : V \to \mathbb{Z}$, $\eta_i(w) = \begin{cases} \epsilon(v) * \infty & if\ i = 0,\ v \neq w, \\ \epsilon(w) * max\{\epsilon(w) * (\mu(w, u) + \eta_{i-1}(u)) \mid (w, u) \in E\} & if\ i > 0,\ v \neq w, \\ 0 & if\ w = v. \end{cases}$

Explanation. If the play of maximum length $i + 1$ that starts in the vertex $w \neq v$ does not end in v, then $\eta_i(w) = \epsilon(v) * \infty$. However, if it does end in v, then we have a play of length $j + 2$ created by the positional strategies of the two players, $(w, v_1, v_2, ..., v_j, v)$. As such, $\eta_i(w) = \mu(w, v_1) + \mu(v_j, v_2) + ... + \mu(v_j, v)$, $\forall i \geq j + 1$. P_+ wants to maximise the sum of all edge weights in a play, while P_- wants to minimize it. As such, at vertex $w \neq v$, P_+ will choose the edge (w, u_+) such that $\eta_i(w) = max\{\mu(w, u) + \eta_{i-1}(u) \mid (w, u) \in E\}$, while P_- will choose the edge (w, u_-) such that $\eta_i(w) = min\{\mu(w, u) + \eta_{i-1}(u) \mid (w, u) \in E\}$.

Identifying k-obsolete edges. We can observe that for the edge $(v, w) \in E$, $v \in V_x$, to be obsolete, $\eta_{k-1}(w) \neq \epsilon(v) * \infty$, meaning that the play from w does reach the vertex v and $\epsilon(v) * (\mu(v, w) + \eta_{k-1}(w)) < 0$, meaning that the payoff of the cycle is disadvantageous to P_x.

Algorithm 1: Identifying k-obsolete edges from the graph of a Mean Payoff Game

Input: k, V_-, V_+, E, μ
Output: F
Function `MPGObsolete`(k, V_-, V_+, E, μ)**:**
  $F = \emptyset$;
  for $e \in \{-, +\} \wedge v \in V_e$ **do**
    for $i = 0, .., k-1$ **do**
      $\eta_i(v) = 0$; `/* Initialize values */`
    end
    for $w \in (V_-, V_+) - \{v\}$ **do**
      $\eta_0(w) = e * \infty$;
    end
    for $i = 1, .., k-1$ **do**
      for $w \in (V_-, V_+) - \{v\}$ **do**
        $\eta_i(w) = \epsilon(w) * max\{\epsilon(w) * (\mu(w, u) + \eta_{i-1}(u)) \mid (w, u) \in E\}$; `/* Compute the new iteration */`
      end
    end
    for $(v, w) \in E$ **do**
      if $e * \eta_{i-1}(w) < -e * \mu(v, w)$ **then**
        $F := F \cup \{(v, w)\}$; `/* Add all obsolete edges */`
      end
    end
  end
  return F

Strongly connected components. A subgraph with the set of vertices $X \subseteq V$ and the edge set $\{(w, u) \mid (w, u) \in E,\ w, u \in X\}$ is a strongly connected component if for all vertices $v \in X$, there is a path to all other vertices in X. Strongly connected components can be identified using a depth-first-search algorithm.

Algorithm 2: Identifying strongly connected components of a graph.

```
Input: X, E
Output: s, Y_s, ..., Y_1
Function StrConnComp(X, E):
    s := 0;
    i := 0;
    F := {(v, w) | (v, w) ∈ E, v, w ∈ X};
    while Y_1 ∪ ... ∪ Y_s ≠ X do
        if i = 0 then
            i := 1 ;                                   /* We begin a new subset of vertices */
            m_1 := 1 ;                  /* ∀x ∈ (m_i, i), vertex v_x is reachable from vertex v_i */
            v_1 := randomVertex(X − (Y_1 ∪ ... ∪ Y_s)) ;      /* Choose a random vertex not yet
             accounted for */
            j := 1;
        end
        while j ≤ i do
            for (v_j, w) ∈ F do
                F := F − {(v_j, w)} ;                          /* w is reachable from v_j */
                if ∃ k, k < m_i, w = v_k then
                    for l = k + 1, ..., i do
                        m_l := m_k ;                          /* v_{m_k} is reachable from all v_l */
                    end
                    j := m_k;
                else
                    if w ∉ {v_{m_i}, ..., v_i} then
                        i := i + 1;
                        v_i = w ;                             /* New vertex added to the list */
                        m_i := i;
                        j := i;
                    end
                end
            end
            j := j + 1;
        end
        s := s + 1 ;                  /* If j > i then there is no edge leaving v_{m_i}, ..., v_i */
        Y_s = {v_{m_i}, ..., v_i} ;                            /* New component found */
        i := m_i − 1;
        j := m_i;
    end
    return (s, Y_s, ..., Y_1)
```

Akian et al.'s algorithm[3] is not an entirely original algorithm. It instead uses Zwick and Paterson's algorithm for solving mean payoff games[5] to determine some of the winning positions of the graph for the two players (the sets of vertices W_- and W_+, where if the play reaches a vertex in W_x, P_x will win the game), as well as some of the draw positions (the set of vertices W_0, where if the play reaches a vertex in W_0, then the game ends in a draw, as neither player can reach its winning condition).

Using finite plays to examine infinite ones. For a subgraph with the set of vertices X, $n = |X|$, and the set of edges F, let $\nu(v)$ be the value of the infinite game that starts at vertex $v \in X$ and $M = max\{|\mu(v, w)| \mid (v, w) \in F\}$. We also define the function ν_k:

$$\nu_k(v):\ X \to \mathbb{Z}, \nu_k(v) = \begin{cases} max\{\mu(v,w) + \nu_{k-1}(w) \mid (v,w) \in F\} & if\ k > 0,\ v \in V_+, \\ min\{\mu(v,w) + \nu_{k-1}(w) \mid (v,w) \in F\} & if\ k > 0,\ v \in V_-, \\ 0 & if\ k = 0. \end{cases}$$

We can observe that $lim_{k\to\infty} \frac{\nu_k(v)}{k} = \nu(v)$. It can also be shown that:

$\nu(v)k - 2nM \leq \nu_k(v) \leq \nu(v)k + 2nM$.

Proof. We begin by assuming that $v \in V_+$. Ehrenfeucht and Mycielski have shown that P_+ and P_- have positional strategies in a mean payoff game[6]. Let $\pi_+ :\ V_+ \to V$ be an optimal positional strategy for player P_+ in a finite game, with at most k steps, that starts at v. If in a finite play a cycle is formed, from the fact that P_+ has an optimal positional strategy, and as such average weights of edges in that cycle is at least $\nu(v)$. Given that the cycle has at most n edges and that the weight of each edge is at least $-M$, we can say that $\nu_k(v) \geq (n-k)\nu(v) - nM$. Because $\nu(v) \leq M$, we can conclude that $\nu(v)k - 2nM \leq \nu_k(v)$. Similarly, if $v \in V_-$, we conclude that $\nu_k(v) \leq \nu(v)k + 2nM$.

Determining winning positions. Given that $\nu_k(v) \leq \nu(v)k + 2nM$, if $\nu_k(v) > 2nM$, then $2nM < \nu(v)k + 2nM$ and as such $\nu(v) > 0$. This means that in this case P_+ wins the game from the vertex v. Alternatively, if $\nu_k(v) < -2nM$ then $\nu(v) < 0$, meaning that P_- wins from the vertex v. If neither condition is meat after $k = 4n^2M + 1$, we can conclude that $\nu(v) = 0$ and that therefore the play from v results in a draw.

Explanation. An infinite play is formed by infinitely repeating the edges of a cycle. This cycle has at most n edges, given that a cycle of n edges implies that all vertices of X are part of the cycle. As such, we can say that $\nu(v)$ is a rational number with a denominator whose value is at most n. The difference between 0 and the closest rational number with a denominator of at most n is $1/n$. Therefore, if $-\frac{1}{n} < \nu(v) < \frac{1}{n}$, then $\nu(v) = 0$. From $\nu(v)k - 2nM \leq \nu_k(v) \leq \nu(v)k + 2nM$, for $k = 4n^2M + 1$, we get that:

$\frac{\nu_k(v)}{4n^2M+1} - \frac{2nM}{4n^2M+1} \leq \nu(v) \leq \frac{\nu_k(v)}{4n^2M+1} + \frac{2nM}{4n^2M+1}$.

If $-2nM \leq \nu_k(v) \leq 2nM$, we get that:

$$\frac{\nu_k(v)}{4n^2M+1} - \frac{2nM}{4n^2M+1} > \frac{\nu_k(v)}{4n^2M+1} - \frac{2nM+\frac{1}{2n}}{4n^2M+1} = \frac{\nu_k(v)}{4n^2M+1} - \frac{1}{2n} \geq \frac{-2nM}{4n^2M+1} - \frac{1}{2n} > -\frac{2nM+\frac{1}{2n}}{4n^2M+1} - \frac{1}{2n} = -\frac{1}{n},$$

and similarly that:

$\frac{\nu_k(v)}{4n^2M+1} + \frac{2nM}{4n^2M+1} < \frac{1}{n}$.

Therefore, if $-\frac{1}{n} < \nu(v) < \frac{1}{n}$ means that $\nu(v) = 0$.

Algorithm 3: Solving the mean payoff game on the subgraph with the vertices in X

Input: X, V_-, V_+, E, μ
Output: Y_-, Y_+, Y_0
Function `StrConnComp`(X, V_-, V_+, E, μ)**:**
  $Y_- = Y_+ = Y_0 := \emptyset$;
  $C := \{(v, w) \in E \mid v, w \in X\}$; `/* The set of all edges between the vertices of the subgraph */`
  if $C = \emptyset \wedge X \subset V_-$ **then**
    $Y_+ := X$; `/* Only one vertex in the subgraph. */`
  end
  if $C = \emptyset \wedge X \subset V_+$ **then**
    $Y_- := X$;
  end
  if $C \neq \emptyset$ **then**
    $n := \#X$;
    $M := max\{|\mu(v, w)| \mid (v, w) \in C\}$;
    for $v \in V_x$ **do**
      $\nu_0(v) := 0$;
    end
    for $k = 1, ..., 4n^2M + 1$ **do**
      for $v \in X$ **do**
        if $v \in V_-$ **then**
          $\nu_k(v) := min\{\mu(v, w) + \nu_{k-1}(w) \mid (v, w) \in C\}$;
        end
        if $v \in V_+$ **then**
          $\nu_k(v) := max\{\mu(v, w) + \nu_{k-1}(w) \mid (v, w) \in C\}$;
        end
        if $\nu_k(v) < -2nM$ **then**
          $Y_- := Y_- \cup \{v\}$;
        end
        if $\nu_k(v) > 2nM$ **then**
          $Y_+ := Y_+ \cup \{v\}$;
        end
      end
    end
    $Y_0 := X - (Y_- \cup Y_+)$;
  end
  return (Y_-, Y_+, Y_0)

Reachability games. Reachability games are zero-sum games played on a graph $G = (V, E)$ by two players: P_1, P_2. The two players each get assigned a subset of vertices:V_1, V_2; $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$. At vertex $v \in V_1$, P_1 chooses an adjacent vertex $w \in V$, $(v, w) \in E$, while at vertex $v \in V_2$, P_2 makes that choice. As such, as they make their choices, the two players form a list of vertices, $(v_0, v_1, v_2...)$, known as the play, with $(v_0, v_1) \in E$, $(v_1, v_2) \in E$ and so on. Let X be a subset of vertices, known as the winning set. If the play reaches at least one vertex from X, then P_1 wins. Otherwise, P_2 is the winner.

Discovering additional winning positions. If $v \in W_x$, meaning that it is a winning position for P_x in a mean payoff game, then for $w \in V_x$, $(w, v) \in E$, w is also a winning position. Additionally, if $w \in V_{-x}$ and $\forall (w, u) \in E$, $u \in W_x$, then $w \in W_x$. As such, we can determine additional winning positions by playing the reachability game on the graph G with W_x as the winning set. We can also use the reachability game to determine additional positions where each of the two players can force a draw.

Algorithm 4: Identifying all vertices from which P_e can reach X

Input: X, e, V_-, V_+, E
Output: X
Function `Reach`(X, e, V_-, V_+, E)**:**
  for $(v, w) \in E \ \wedge \ v \notin X \ \wedge \ w \in X$ **do**
    if $v \in V_e \ \vee \ \forall (v, u) \in E, \ u \in X$ **then**
      $X := X \cup \{v\}$;
    end
  end
  return X

The main routine. First we repeatedly eliminate all k-obsolete edges from the graph, with $k = \#V$. Because removing k-obsolete edges may reveal additional k-obsolete edges[3], we repeat the process until there are no more k-obsolete edges found. After that, we recursively split the graph into its strongly connected components, and for each component, we use Zwick and Paterson's algorithm[5] to determine its winning positions. After that, we use the algorithm for solving reachability games to determine additional draw positions and then additional winning positions for each of the players. Once we have determined a winner or a draw for the plays that start at all the vertices in the graph, we have completed the algorithm.

Algorithm 5: Solving Mean Payoff Games via Reachability Games

Input: V_-, V_+, E, μ
Output: W_-, W_+
Function `MPGSolver`(V_-, V_+, E, μ)**:**
  $F := MPGObsolete(\#V_- + \#V_+, V_-, V_+, E, \mu)$;
  while $F \neq \emptyset$ **do**
    $E := E - F$; /* Remove obsolete edges */
    $F := MPGObsolete(\#V_- + \#V_+, V_-, V_+, E, \mu)$;
  end
  $W_- = W_+ = W_0 = \emptyset$; /* Initialize the winning sets */
  if V=$\emptyset$ **then**
    $r := 0$;
  else
    $r := 1$;
  end
  $X_1 := V$;
  while $r > 0$ **do**
    $X := X_r - (W_- \cup W_+ \cup W_0)$; /* Remove any vertices with an already decided winner/draw from the subgraph */
    $(s, X_r, ..., X_{r+s-1}) := StrConnComp(X, E)$; /* Break down the subgraph into its strongly connected components */
    $r := r + s - 1$;
    $(Y_-, Y_+, Y_0) := MPGSubgraphSolver(X_r, V_-, V_+, E, \mu)$; /* Solve the MPG for this subgraph */
    $W_0 := W_0 \cup Y_0$;
    for $e \in \{-, +\}$ **do**
      $F := \{(v, w) \mid (v, w) \in E,\ v \in Y_e, w \in Y_e \cup W_0\}$;
      $W_0 := Reach(W_0, -e, V_-, V_+, F)\}$; /* Discover additional draws */
      if $Y_e \cap W_0 = \emptyset$ **then**
        $W_e := W_e \cup Y_e$; /* Add the winning positions */
      end
    end
    $W_- := Reach(W_-, -, V_-, V_+, E)$; /* Solve the reachability game for the 2 players */
    $W_2 := Reach(W_+, 2, V_-, V_+, E)$;
    while $r > 0 \wedge X_r \subset W_- \cup W_+ \cup W_0$ **do**
      $r := r - 1$; /* If this subgraph has already been solved, we move on to the next one */
    end
  end
  return W_-, W_+, W_0

3 The basic approach and the signature of a potential

3.1 Parity games as special cases of mean payoff games

It is a known fact that parity games are simply a special case of mean payoff games. For every parity game, there is an equivalent mean payoff game, meaning that the mean payoff game is played on the same graph, with the subsets of vertices $V_- = V_1$ and $V_+ = V_2$ and the players are the same between the two games, with the same optimal positional strategies: $P_- = P_1, P_+ = P_2$. In order to ensure that, we need to construct a weight function μ for the mean payoff game where a favorable vertex for P_1 in the parity game leads to a favorable edge for P_- and similarly for P_2 and P_+.

Equivalent mean payoff game. In order to construct the mean payoff game equivalent to a parity game, the weight function will be $\mu(v,w) = (-n)^{\Omega(v)}, (v,w) \in E$. Because $n > 0$, for a vertex $v \in V$ with $\Omega(v)\ mod\ 2 = 1$, every outgoing edge $(v,w) \in E$ has $\mu(v,w) < 0$, meaning that from the vertex v, every outgoing edge decreases the average of weights, while for a vertex $v \in V,\ \Omega(v)\ mod\ 2 = 0,\ \mu(v,w) > 0,\ \forall (v,w) \in E$, which means that every outgoing edge of v would increase the average of weights. Additionally, for this weight function, we can observe that for a cycle containing the nodes $\{v_0, v_1, ..., v_x\}$ and $M = max\{\Omega(v) \mid v \in \{v_0, v_1, ..., v_x\}\}$, we have $|(-n)^M| > \sum\{|(-n)^{\Omega(v)} \mid v \in \{v_0, v_1, ..., v_x\}\} - |(-n)^M|$, meaning that the sign of $(-n)^M$ determines the payoff.

3.2 The algorithm

Based on this idea that every parity game has an equivalent mean payoff game, the simplest way to use Akian et al.'s algorithm [3] to solve parity games is to transform the parity game into its equivalent mean payoff game and then solve it in the manner discussed above.

Algorithm 6: Solving Parity Games via their equivalent Mean Payoff Game

Input: V_1, V_2, E, Ω
Output: W_1, W_2
Function `PGSolver1`(V_1, V_2, E, Ω)**:**
  for $v \in V$ **do**
    for $(v,w) \in E$ **do**
      $\mu(v,w) = (-n)^{\Omega(v)}$;
    end
  end
  $(W1, W2, W0) := MPGSolver(V_1, V_2, E, \mu)$; `/* Akian et al. main function */`
  return $(W1, W2)$; `/* Draws not possible in PG */`

3.3 The signature of a potential for mean payoff games

In their work, Akian et al. identified that their algorithm has its best case when the graph has the signature of a potential $\psi : V \to \mathbb{Z}$ [3]. This means that there exists a function $\psi : V \to \mathbb{Z}$ such that:

(A0) $\mu(v,w) = \mu(w,v) = \psi(v) - \psi(w) = 0$, if $\epsilon(v) = \epsilon(w),\ (v,w) \in E$,
(A1) $sign(\psi(v) - \psi(w)) = -1$, if $\epsilon(v) \neq \epsilon(w),\ (v,w) \in E,\ (w,v) \notin E$,
(A2) $sign(\epsilon(v)\mu(v,w) - \epsilon(w)\mu(w,v)) = sign(\psi(v) - \psi(w))$, if $\epsilon(v) \neq \epsilon(w),\ \{(v,w),(w,v)\} \subset E$,
with:

$$\epsilon : V \to \{-1, 1\},\ \epsilon(v) = \begin{cases} -1 & if\ v \in V_-, \\ 1 & if\ v \in V_+, \end{cases}$$

$$sign : \mathbb{Z} \to \{0, \pm 1\},\ sign(x) = \begin{cases} -1 & if\ x < 0, \\ 0 & if\ x = 0, \\ 1 & if\ x > 0. \end{cases}$$

Linear time. In Akian et al.'s paper[3], it is shown that the winning positions of a mean payoff game with the signature of a potential are the same as the winning positions of its derived reachability game. Given that the reachability game can be solved in linear time, $|V| + |E|$[3], then the mean payoff game can also be solved in $|V| + |E|$ time.

Explanation. First, we show that if a mean payoff game has the signature of a potential, then it still has the signature of a potential, even after all of the k-obsolete edges are removed. We

do this by looking at E', the set of edges that results from removing the 2-obsolete edges of E, extended by $\mu'(v,w) = -\epsilon(v)\infty,\ \forall (v,w) \in V \times V$:

$$(v,w) \in E,\ \mu(v,w) = \begin{cases} \mu(v,w) & if\ (v,w) \in E, \\ -\epsilon(v)\infty & if\ (v,w) \notin E. \end{cases}$$

For $(v,w) \in E$ and $(w,v) \in E - E'$, we can observe that for $\epsilon(v) \neq \epsilon(w)$, $sign(\epsilon(w)(\mu(w,v)+\mu(v,w))) = -1$ and as such $sign(\mu(w,v) + \mu(v,w)) = -\epsilon(w)$. This means that $sign(\epsilon(v)(\mu(v,w) + \mu(w,v))) = 1 = sign(\psi(v) - \psi(w))$.

In a strongly connected subgraph of a mean payoff game with the signature of a potential, $X \subset V$, we can form a path $(v_0, ..., v_n, v_0)$, with $v_0, ..., v_n \in X$. For $\epsilon(v) \neq \epsilon(w),\ \forall (v,w) \in E;\ v, w \in v_0, ..., v_n, v_0)$, $0 \leq \psi(v_0) - \psi(v_1)$, we have that $0 \leq \psi(v_1) - \psi(v_2)$,...,$0 \leq \psi(v_n) - \psi(v_0)$. As such, $\psi(v_0) \geq \psi(v_1) \geq ... \geq \psi(v_n) \geq \psi(v_0)$. This means that $\psi(v_0) = ... = \psi(v_n)$. Therefore, $\forall (v,w) \in E$, $\epsilon(v) \neq \epsilon(w)$, $\psi(v) = \psi(w)$ and as such $\mu(v,w) + \mu(w,v) = 0$.

Based on this, $\forall (v,w) \in E'$:

1. If $\epsilon(v) = \epsilon(w)$, $\mu(w,v) = 0$,
2. If $\epsilon(v) \neq \epsilon(w)$, $\mu(v,w) + \mu(w,v) = 0$.

Let Z_- and Z_+ be the set of sinks in $G' = (V, E')$, $Z_- \subset V_-$, $Z_+ \subset V_+$. If from a vertex $v \in V_-$, P_- has a positional strategy for reaching a vertex in the set Z_+, then v is a winning position for P_-. Similarly, if from a vertex $w \in V_+$, P_+ can reach a vertex in the set Z_-, then w is a winning position for P_-.

If however, the play does not reach a vertex in either Z_- or Z_+, then it is an infinite play. The payoff of this play will be equal to 0, given that $\mu(v,w) + \mu(w,v) = 0$, $\forall (v,w) \in E'$, $\epsilon(v) \neq \epsilon(w)$, or $\mu(v,w) = 0$, $\forall (v,w) \in E'$, $\epsilon(v) = \epsilon(w)$. As such, this play results in a draw.

Therefore, we can say that $W_- := Reach(Z_+, -, V_-, V_+, E)$, $W_+ := Reach(Z_-, +, V_-, V_+, E)$, $W_0 = V - (W_- \cup W_+)$, where W_- and W_+ are the sets of winning positions for each of the two players, while W_0 is the set of draw positions.

Because the reachability game is linear in $|V| + |E|$, we can therefore conclude that mean payoff games with the signature of a potential are solvable in linear time.

3.4 The signature of a potential for parity games

Definition A parity game has the signature of a potential $\psi : V \to \mathbb{Z}$ if its equivalent mean payoff game has the signature of a potential $\psi : V \to \mathbb{Z}$.

We will be using the following helper functions:

- $\epsilon : V \to \{-1, 1\}$, $\epsilon(v) = \begin{cases} -1 & if\ v \in V_1, \\ 1 & if\ v \in V_2, \end{cases}$
- $\theta : V \to \{-1, 1\}$, $\theta(v) = \begin{cases} -1 & if\ \Omega(v)\ mod\ 2 = 1, \\ 1 & if\ \Omega(v)\ mod\ 2 = 0. \end{cases}$

Theorem If the parity game $\mathcal{P} = (V, E, V_1, V_2, \Omega)$ abides by the following conditions, then it is a parity game with the signature of a potential and can be solved in $|V| + |E|$ linear time:

(P0) $\forall (v,w) \in E,\ \epsilon(v) \neq \epsilon(w)$,
(P1) If $(v,w) \in E$, but $(w,v) \notin E$, then $\epsilon(v) * sign(\theta(v)\Omega(v) + \theta(w)\Omega(w)) = -1$.

Proof. We show that for the parity game that abides by the conditions outlined above, its equivalent mean payoff game has the signature of a potential. As previously established, for mean payoff games obtained from parity games, $\mu(v, w) = (-n)^{\Omega(v)}$. For (A0) to be true, it is obvious that $\mu(v, w) = 0$ which is impossible, as $n > 0$. For determining how the conditions (A1) and (A2) apply to parity games, $\psi(v) = \epsilon(v)\theta(v)\Omega(v)$ was used. (A2) is always true with this potential function and (P1) was derived from (A1).

Because turning a parity game into a mean payoff game is linear in $|E|$, if the mean payoff game can be solved in $|V| + |E|$, then a parity game with the signature of a potential can be solved in $|V| + |E|$.

Bipartite and symmetric graphs. Berwanger and Serre[7] have previously proven that games played on bipartite and symmetric graphs can be solved in linear time: $|V| + |E|$. In this case, a bipartite graph means that $\forall(v, w) \in E$, $v \in V_1$ and $w \in V_2$ or $v \in V_2$ and $v \in V_1$. For a graph to be symmetric, $\forall(v, w) \in E, (w, v) \in E$. Evidently, bipartite and symmetric graphs have the signature of a potential, as they the conditions set above. However, our research shows that the set of parity games that are solvable in linear time is not limited to bipartite and symmetric games, as is made evident by (P1).

4 The strategy improvement approach

In addition to the trivial modifications we made to to the mean payoff game algorithm to turn it into one for parity games, we are also looking to develop an algorithm that works similarly to Akian et al.'s[3], but without transforming the parity into a mean payoff game in the process. Therefore, this new algorithm would have the same general structure of the original, but modified to specifically fit parity games.

4.1 The main routine

As previously stated, we want this algorithm to be modeled after the one for mean payoff games. This means that we maintain the same strategy of first eliminating outgoing edges from each vertex that lead to undesirable cycles for the player that owns that vertex, then we break down the graph into its strongly connected components, and for each of those we determine the winning position for each player, ending in using a reachability game to determine the outcome of the game from each vertex.

The modifications. Because we use the same general steps, the main routine is very similar to the original, but for the lack of the W_0 set and the commands that relate to it. That is because a tie is impossible in parity games, given that the infinitely repeating maximum value is always either even or uneven. As such, searching for positions that lead to a tie is useless.

Algorithm 7: Solving Parity Games via Reachability Games

Input: V_1, V_2, E, Ω
Output: W_1, W_2
Function `PGSolver2(`V_1, V_2, E, Ω`)`**:**
  $F := PGObsolete(\#V_1 + \#V_2, V_1, V_2, E, \Omega)$; `/* Determine obsolete edges */`
  while $F \neq \emptyset$ **do**
    $E := E - F$; `/* Remove obsolete edges */`
    $F := PGObsolete(\#V_1 + \#V_2, V_1, V_2, E, \Omega)$; `/* Determine new obsolete edges */`
  end
  $W_1 = W_2 = \emptyset$;
  if V=$\emptyset$ **then**
    $r := 0$;
  else
    $r := 1$;
  end
  $X_1 := V$;
  while $r > 0$ **do**
    $X := X_r - (W_1 \cup W_2)$;
    $(s, X_r, ..., X_{r+s-1}) := StrConnComp(X, E)$; `/* No changes to this function */`
    $r := r + s - 1$;
    $(Y_1, Y_2) := PGSubgraphSolver(X_r, V_1, V_2, E, \Omega)$;
    $W_1 := W_1 \cup Y_1$;
    $W_2 := W_2 \cup Y_2$; `/* We don't need to check for draws, not possible in PG */`
    $W_1 := Reach(W_1, 1, V_1, V_2, E)$; `/* No changes to this function */`
    $W_2 := Reach(W_2, 2, V_1, V_2, E)$;
    while $r > 0 \ \wedge \ X_r \subset W_1 W_2$ **do**
      $r := r - 1$;
    end
  end
  return W_1, W_2

The functions $StrConnComp$ and $Reach$ require no modifications from how they are defined in the algorithm for mean payoff games.

4.2 Eliminating k-obsolete edges

Definition. A k-obsolete edge in a parity game is an outgoing edge of $v \in V_x$, such that by playing it, no matter what other choices P_x makes, P_y has a positional strategy through which the play returns to v, forming a cycle of maximum length k, where the parity of the maximum priority in this play corresponds to that of the player P_y, meaning that it is never advantageous for P_x to play that edge. Therefore, that edge can be eliminated from the graph without the outcome of the game being affected; $\{P_x, P_y\} = \{P_1, P_2\}$.

Evaluating a given play. Before we focus on how the two players make their choices, we look at how we determine the maximum priority of a given play and whether that play ends in v or not, where v is the vertex whose outgoing edges are verified for being k-obsolete. For a given play $(v_i, ..., v_0)$ obtained from the positional strategies of both players, we have:

$$\eta'_i : V \to \mathbb{N} \cup \{\infty\},\ i = 0, ..., k-1,\ \eta'_i(v_i) = \begin{cases} max\{\Omega(v_i), \Omega(v_{i-1}), ..., \Omega(v_1), \eta'_0(v_0)\} & if\ i > 0,\ v_i \neq v,\ \{u \mid (v_i, u) \in E\} \neq \emptyset, \\ \infty & if\ i > 0,\ v_i \neq v,\ \{u \mid (v_i, u) \in E\} \neq \emptyset, \\ \infty & if\ i = 0,\ v_0 \neq v, \\ 0 & if\ v_i = v. \end{cases}$$

Explanation. In the case that the play does not end in v ($v_0 \neq v$), we need a value for $\eta'_0(v_0)$

that is so large that $\eta'_i(v_i) = \eta'_0(v_0)$, indicating that the path does not lead to v. As such, we choose $\eta'_0(w) = \infty$, $\forall w \in V - \{v\}$. Meanwhile, if $v_0 = v$, we need a value for $\eta'_0(v)$ such that $\eta'_i(v_i)$ will always be equal to the maximum priority of the vertices. We choose $\eta'_0(v) = 0$, as the priorities of vertices are natural numbers. This will be the value associated with v in all iterations: $\eta'_i(v) = 0,\ i = 0, ..., k-1$.

How the players make choices. At a given vertex $w \in V_z$, $w \neq v$, $z \in \{1, 2\}$, P_z, chooses the vertex $u' \in \{u \mid (w, u) \in E\}$. Since the players play optimally, they will choose the best option available to them. We can rank their options for the vertex u', from best to worst, in the following order:

1. $\eta'_i(u') = \infty$, if $P_z = P_x$ and $\infty \in \{\eta'_i(u) \mid (w, u) \in E\}$ - the priority of P_x is to avoid any plays that lead to v. For the edge $(v, t) \in E$ to be k-obsolete, in the play that starts at t, P_x should only be able to choose paths that lead to v.
2. $\eta'_i(u') = max\{\eta'_i(u) \mid (w, u) \in E,\ \eta'_i(u) \neq \infty,\ \eta'_i(u)\ mod\ 2 = z\ mod\ 2\}$, if $\{\eta'_i(u) \mid (w, u) \in E,\ \eta'_i(u) \neq \infty,\ \eta'_i(u)\ mod\ 2 = z\ mod\ 2\} \neq \emptyset$ - there are plays that lead to v with the maximum of the preferred parity of P_z, and they choose the play with the largest of those values.
3. $\eta'_i(u') = min\{\eta'_i(u) \mid (w, u) \in E,\ \eta'_i(u) \neq \infty,\ \eta'_i(u)\ mod\ 2 \neq z\ mod\ 2\}$, if $\{\eta'_i(u) \mid (w, u) \in E,\ \eta'_i(u) \neq \infty,\ \eta'_i(u)\ mod\ 2 \neq z\ mod\ 2\} \neq \emptyset$ - this means that the only available plays that lead to v have values of the opposite parity to z, so the best choice is to select the smallest of these values.
4. $\eta'_i(u') = \infty$, if $P_z = P_y$ and $\infty \in \{\eta'_i(u) \mid (w, u) \in E\}$ - in verifying if the edge $(v, t) \in E$ is obsolete, P_y must be able to force the play to return to v. However, at this vertex w, none of the plays available to P_y lead to v.

If none of these options are available, it simply means that there is no outgoing edge from w.

Function definition. Now that we covered the positional strategies of the players, we can finally give a full definition for the function η'_i:

$$\eta'_i : V \to \mathbb{N} \cup \{\infty\},\ i = 0, ..., k-1,$$

$$\eta'_i(w) = \begin{cases} max\{\Omega(w),\ \epsilon(w) * \phi(max\{\epsilon(w) * \phi(\eta'_{i-1}(u), v) \mid (w, u) \in E\}, v)\} & if\ i \neq 0,\ w \neq v,\ \{u \mid (w, u) \in E\} \neq \emptyset, \\ \infty & if\ i \neq 0,\ w \neq v,\ \{u \mid (w, u) \in E\} = \emptyset, \\ \infty & if\ i = 0,\ w \neq v, \\ 0 & if\ w = v, \end{cases},$$

where $\phi : (\mathbb{Z} \cup \{\pm\infty\}) \times V \to \mathbb{Z} \cup \{\pm\infty\}$, $\phi(n, t) = \begin{cases} \epsilon(t) * n & if\ n = \pm\infty, \\ -n & if\ n \neq \pm\infty,\ n\ mod\ 2 = 1, \\ n & if\ n \neq \pm\infty,\ n\ mod\ 2 = 0. \end{cases}$

Explanation. The cases where $i = 0$ and $w = v$ have already been covered above, so we will now focus on the other two cases.If w has no outgoing edges, then the play that starts at w can only contain itself and not v, so the value of $\eta'_i(w)$ should be ∞. Looking at the case where w has outgoing edges. For $a, b, c, d \in \mathbb{N}$, $a < b$, $c < d$, we have:

$\epsilon(w) * \phi(2b, v) < \epsilon(w) * \phi(2a, v) \leq 0 \leq \epsilon(w) * \phi(2c + 1, v) < \epsilon(w) * \phi(2d + 1, v)$ if $w \in V_1$,

$\epsilon(w) * \phi(2d + 1, v) < \epsilon(w) * \phi(2c + 1, v) \leq 0 \leq \epsilon(w) * \phi(2a, v) < \epsilon(w) * \phi(2b, v)$ if $w \in V_2$.

For $w \in V_x$, $\epsilon(w) * \phi(\infty, v) = \infty$, while for $w \in V_y$, $\epsilon(w) * \phi(\infty, v) = -\infty$.

This means that $max\{\epsilon(w) * \phi(\eta'_{i-1}(u), v) \mid (w, u) \in E\} = \epsilon(w) * \phi(\eta'_i(u'), v)$.

Finally, we can observe that $\epsilon(w) * \phi(\epsilon(w) * \phi(\eta'_i(u'), v), v) = \eta'_i(u')$.

Proposition. Once the values of $\eta'_{k-1}(u)$, $\forall u \in V$ have been determined, for an edge $(v, w) \in E$ to be k-obsolete, it needs to abide by the following condition:
$\eta'_{k-1}(w) \neq \infty \wedge ((max\{\eta'_{k-1}(w), \Omega(v)\}\ mod\ 2 = 1 \wedge v \in V_2) \vee (max\{\eta'_{k-1}(w), \Omega(v)\}\ mod\ 2 = 0 \wedge v \in V_1))$

Explanation. For the edge (v, w) to be k-obsolete, the play of maximum k vertices that starts at w needs to end in v ($\eta'_{k-1}(w) \neq \infty$) and the maximum priority in the play is of the parity associated with P_y.

As such, the algorithm for identifying k-obsolete edges is the following:

Algorithm 8: Removing k-obsolete edges from the graph of a Parity Game

Input: k, V_1, V_2, E, Ω
Output: F
Function `PGObsolete`(k, V_1, V_2, E, Ω)**:**
  $F = \emptyset$;
  for $e \in \{1, 2\} \wedge v \in V_e$ **do**
    for $i = 0, .., k-1$ **do**
      $\eta'_i(v) = 0$; `/* Initialize values */`
    end
    for $w \in (V_1 \cup V_2) - \{v\}$ **do**
      $\eta'_0(w) = \infty$;
    end
    for $i = 1, .., k-1$ **do**
      for $w \in (V_1 \cup V_2) - \{v\}$ **do**
        if $\{u \mid (w, u) \in E\} = \emptyset$ **then**
          $\eta'_i(w) = \infty$; `/* No outgoing edges */`
        else
          $\eta'_i(w) = max\{\Omega(w),\ \epsilon(w) * \phi(max\{\epsilon(w) * \phi(\eta'_{i-1}(u), v) \mid (w, u) \in E\}, v)\}$;
          `/* Compute the new iteration */`
        end
      end
    end
    for $(v, w) \in E$ **do**
      if $\eta'_{k-1}(w) \neq \infty \wedge max(\eta'_{k-1}(w), \Omega(v))\ mod\ 2 \neq e\ mod\ 2$ **then**
        $F := F \cup \{(v, w)\}$; `/* Add all obsolete edges */`
      end
    end
  end
  return F

4.3 Solving the subgraphs

Strategy Improvement Algorithm. Just like Akian et al.[3] used an already existing algorithm, namely Zwick and Paterson's[5], as a foundation for their mean payoff game solver, we should also select a parity game solver, used in determining the winning positions for each of the subgraphs we obtain by splitting the graph into its strongly connected components. For this purpose, we will be using John Fearnley's *strategy improvement algorithm*, with the *greedy all-switches rule*[4]. The rest of the subsection will focus on how this algorithm functions.

A positional strategy for P_1 is a function $\tau : V_1 \to V$, for which $(v, \tau(v)) \in E$, while one for P_2 is a function $\sigma : V_2 \to V$, where $(v, \sigma(v)) \in E$. The sets of all positional strategies for both P_1 and P_2 are referred to as Σ_1 and Σ_2 respectively. Given the strategies $\tau \in \Sigma_1$ and $\sigma \in \Sigma_2$, from a given

vertex v_0, there is a unique play $Play(v_0, \tau, \sigma) = v_0, v_1, ...$, where for $v_i \in V_1$, $v_{i+1} = \tau(v_i)$, while for $v_i \in V_2$, $v_{i+1} = \sigma(v_i)$, $i \in \mathbb{N}$. For an infinitely long play $Play(v, \tau, \sigma)$, if the largest infinitely occurring priority of the play is an odd number, we say that τ is a *winning strategy* for the vertex v, for every $\sigma \in \Sigma_2$. Similarly, if the if the largest infinitely occurring priority of the play is an even number, we say that σ is a winning strategy for the vertex v, for every $\tau \in \Sigma_1$.

Strategy improvement algorithms need to select one player for which to repeatedly improve their strategy. In this subsection, just like in the original paper, that player will be P_2.

Note that this algorithm assumes that the graph has no sinks, meaning that all vertices have outgoing edges. Given that we only apply this algorithm to strongly connected components, for a subgraph with at least one edge, that will always be the case.

Graph modifications. At the beginning of the algorithm, the graph is changed by having a new vertex s added to it. This vertex is a sink, so it has no outgoing edges. Additionally, the graph also receives some new edges: $E = E \cup \{(v, s) \mid \forall v \in V_2\}$. Reaching a sink leads to the play being finite, meaning that, at any point, P_2 can end the game by choosing to move the token to s.

Admissible strategy. An *admissible strategy* $\sigma \in \Sigma_2$ is either a winning strategy, or the play $Play(v, \tau, \sigma)$ ends in the sink s. For P_2, the algorithm will consider only admissible strategies.

Valuation. A *valuation* is a measure of how good the pair of strategies σ, τ is, given the starting vertex. In this algorithm, a valuation will be of the form $P \to \mathbb{Z}$, where P is the array of all priority values that occur in the graph, and will count the number of occurrences of each priority in a given finite play. The set of all functions of these functions is referred to as $Vals_G$. Therefore, for $L_v \in Vals_G$, $L_v(p) = \#\{w \mid w \in Play(v, \tau, \sigma),\ w \neq s,\ \Omega(w) = p\}$. We also define $\top$ as the valuation of an infinite play with an even maximum priority and $\dashv$, he valuation of an infinite play with an odd maximum priority. For the vertex v, a strategy $\tau \in \Sigma_1$ and an admissible strategy $\sigma \in \Sigma_2$, we have the *valuation function*:

$$Val^{\tau,\sigma} : V \to Vals_G \cup \{\top\}, Val^{\tau,\sigma}(v) = \begin{cases} \top & if\ Play(v, \tau, \sigma)\ is\ infinite\ with\ an\ even\ maximum\ priority, \\ \dashv & if\ Play(v, \tau, \sigma)\ is\ infinite\ with\ an\ odd\ maximum\ priority, \\ L_v & if\ Play(v, \tau, \sigma)\ is\ finite. \end{cases}$$

Note that because σ is an admissible strategy, $\sigma(v) \neq \dashv$ for $v \in V_2$. Also note that if $Val^{\tau,\sigma}(v) = \top$, that means that P_2 is the winner at vertex v.

Additionally, the operators $\sqsubseteq$ and $\sqsubset$ are introduced. For every $L \in Vals_G$, $L \sqsubseteq \top$ is true, and so is $\dashv \sqsubseteq L$. For $L_x, L_y \in Vals_G$, if $L_x = L_y$, $L_x \sqsubseteq L_y$ is true, as well as $L_y \sqsubseteq L_x$. However, if $L_x \neq L_y$, then for $p = max\{\Omega(v) \mid v \in V,\ L_x(\Omega(v)) \neq L_y(\Omega(v))\}$, if $L_x \sqsubset L_y$, then p is even and $L_x(p) < L_y(p)$, or p is odd and $L_x(p) > L_y(p)$.

Best response. The *best response* to an admissible strategy $\sigma \in \Sigma_2$ is the strategy $br(\sigma) \in \Sigma_1$ that minimizes the valuation of each vertex. Strategy improvement for an admissible strategy σ only works if P_1's strategy is the best response, so we can define $Val^\sigma = Val^{br(\sigma),\sigma}$.

Switchable set. We define an edge $(v, u) \in E$ to be *switchable* if $\sigma(v) \neq u$ and $Val^\sigma(\sigma(v)) \sqsubset Val^\sigma(u)$. A *switchable set* $S \subseteq E$ is a set of edges, such that for any pair of edges $(v, w), (u, t) \in E$, $v \neq u$. This means that the switchable set contains at most one outgoing edge for each of the vertices of the graph. For a switchable set S and the strategy σ, we can create a new strategy $\sigma[S]$:

$$\sigma[S] : V_2 \to V,\ \sigma[S](v) = \begin{cases} w & if\ (v, w) \in S, \\ \sigma(v) & if\ \{(v, w) \mid (v, w) \in E\} \cap S = \emptyset \end{cases}.$$

How a strategy is improved. If S is a switchable set with only switchable edges in it, then

$Val^{\sigma}(v) \sqsubseteq Val^{\sigma[S]}(v)$, $\forall v \in V_2$, and there are one or more vertices $w \in V_2$ for which $Val^{\sigma}(w) \sqsubset Val^{\sigma[S]}(w)$. This means that the strategy $\sigma[S]$ is an improvement over σ. The way the switchable set with only switchable edges is found is by using the *greedy all-switches rule*, meaning that for every vertex that has outgoing switchable edges, the algorithm chooses the switchable edge (v, w) that maximises $Val^{\sigma}(w)$ under the $\sqsubseteq$ ordering. The strategy improvement algorithm keeps iterating on the strategy for P_2 until no more improvements can be found. Because there is a strict $\sqsubset$ ordering, this means that the algorithm can not repeat strategies, so it must eventually terminate.

How the algorithm works. First, we create σ_{init}, which is the strategy in which P_2 uses just the edges from all the vertices in V_2 to the sink s. This will be the initial value of σ. Next, we compute the best response. We initially set τ to an arbitrary strategy. Next, we identify *odd-switchable* edges $(v, w) \in E$ where $Val^{\tau,\sigma}(w) \sqsubset Val^{\tau,\sigma}(\tau(v))$. The algorithm repeatedly switches odd-switchable edges until there are none left, creating the best response. After that, we repeatedly find the switchable edges for σ, and switch them in until there are no more switchable edges. The vertices from which P_2 wins are those which form an infinite length path where the maximum infinitely repeating priority is even, or, in other words: $W_2 = \{v \mid v \in V,\ Val^{\tau,\sigma}(v) = \top\}$. Conversely, $W_1 = \{v \mid v \in V,\ Val^{\tau,\sigma}(v) \neq \top\}$.

Additionally, because we apply this algorithm to subgraphs, we first check if the subgraph in question has no edges. Solving this subgraph is trivial. Because of this step, this algorithm has a linear complexity for games with the signature of potential. The proof from the previous section applies here as well.

Algorithm 9: The algorithm for determining the switchable set with odd-switchable edges

Input: $X, V_1, V_2, C, \Omega, Val^{\tau,\sigma}, \tau$
Output: F
Function $Switchable_1(X, V_1, V_2, C, \Omega, Val^{\tau,\sigma}, \tau)$**:**
    $S_1 = \emptyset$;
    for $v \in X \cap V_1$ **do**
        $u = \sigma(v)$;
        for $(v, w) \in C$ **do**
            if $Val^{\tau,\sigma}(u) \sqsubset Val^{\tau,\sigma}(w)$ **then**
                $u = w$;
            end
        end
        if $u \neq \sigma(v)$ **then**
            $S_1 = S_1 \cup (v, u)$; /* (v, u) is the best odd-switchable edge at v for P_1 */
        end
    end
    return S_1 ; /* The switchable set of odd-switchable edges */

Algorithm 10: The algorithm for determining the switchable set with switchable edges

Input: $X, V_1, V_2, C, \Omega, Val^{\tau,\sigma}, \sigma$
Output: F
Function $Switchable_2(X, V_1, V_2, C, \Omega, Val^{\tau,\sigma}, \sigma)$**:**
    $S_2 = \emptyset$;
    for $v \in X \cap V_1$ **do**
        $u = \tau(v)$;
        for $(v, w) \in C$ **do**
            if $Val^{\tau,\sigma}(w) \sqsubset Val^{\tau,\sigma}(u)$ **then**
                $u = w$;
            end
        end
        $Play(-1, \tau, \sigma) = \emptyset$;
        if $Val^{\tau,\sigma}(u) \sqsubset L_s$ **then**
            $u = s$; /* The best switchable edge is the edge between v and the sink */
        end
        if $u \neq \tau(v)$ **then**
            $S_2 = S_2 \cup (v, u)$; /* (v, u) is the best switchable edge at v for P_2 */
        end
    end
    return S_2 ; /* The switchable set of switchable edges */

Algorithm 11: The strategy improvement algorithm for solving parity games

```
Input: X, V_1, V_2, E, Ω
Output: F
Function PGSubgraphSolver(X, V_1, V_2, E, Ω):
    Y_1 = Y_2 := ∅;
    C := {(v, w) ∈ E | v, w ∈ X} ;      /* The set of all edges between the vertices of the
     subgraph */
    if C = ∅ ∧ X ⊂ V_1 then
        Y_2 := X ;                                  /* Only one vertex in the subgraph.  */
    end
    if C = ∅ ∧ X ⊂ V_2 then
        Y_1 := X;
    end
    if C ≠ ∅ then
        σ := σ_init(X, V_2, C) ;                                  /* Initialize strategy */
        τ := arbitraryStrat(X, V_1, C) ;                /* Initialize with random strategy */
        S_2 = S_1 := ∅;
        repeat
            repeat
                for v ∈ X do
                    if v ∈ V_1 then
                        w = τ(v);
                    else
                        w = σ(v);
                    end
                    while v ≠ w ∧ w ∈ X do
                        if w ∈ V_1 then
                            w = τ(w);
                        else
                            w = σ(w);
                        end
                    end
                    if w ∉ X then
                        Val^{τ,σ}(v) = L_v ;                    /* Finite play, ends in sink */
                    else
                        if max{Play(v, τ, σ)} mod 2 = 0 then
                            Val^{τ,σ}(v) = ⊤ ;                                /* Infinite play */
                        else
                            Val^{τ,σ}(v) =⊣;
                        end
                    end
                end
                S_1 := Switchable_1(X, V_1, V_2, C, Ω, Val^{τ,σ}, τ) ;   /* Compute the switchable set */
                τ = τ[S_1] ;                                                /* Switch edges */
            until S_1 = ∅;
            S_2 := Switchable_2(X, V_1, V_2, C, Ω, Val^{τ,σ}, σ) ;      /* Compute the switchable set */
            σ = σ[S_2] ;                                                    /* Switch edges */
        until S_2 = ∅;
        for v ∈ C do
            if Val^{τ,σ}(v) = ⊤ then
                Y_2 = Y_2 ∪ {v};
            end
        end
        Y_1 = X − Y_2;
    end
    return (Y_1, Y_2);
```

5 Comparing the algorithms

Implementation. In order to test the performance of the algorithms outlined above, the following algorithms were implemented in C:

- $pgSolver1$, an implementation of the basic approach, where the parity game is first translated into a mean payoff game
- $pgSolver2$, an implementation of the strategy improvement approach, where we use Fearnley's strategy improvement algorithm[4] to solve subgraphs and then solve the entire game using the equivalent reachability game
- $pgSolver3$, a sequential implementation of Fearnley's strategy improvement algorithm[4] used on the entire graph of the game
- $Test\ Generator$, a tool that generates 15 random parity games and 15 random, bipartite, symmetric parity games. All generated graphs have at most 1000 nodes, no sinks and the maximum priority of any one node is 10.

Used tests. In addition to the 30 test cases generated by our tool, 15 more tests were taken from Keiren's set of benchmark tests[8]. These sets confirm to the same restrictions as the others: a maximum of 1000 nodes, the maximum priority of any node is 10 and there are no sinks. These tests were not hand-picked, instead, the tests used were simply the first 40 files that fit the criteria which were opened with the C $readdir()$ function.

The tests were carried out on a Dell laptop with an Intel Core i7-9750H CPU, clocked at 2.60 GHz, with 12 cores and 16GB of ram.

After using the algorithms to solve the parity games, our program stores the execution times of each algorithm, along with information about each individual game in a .xlsx file. This file, along with the program and the test cases are publicly available[1]. All information on how to use the programs that were created for this experiment is found in the README.md file found in the repository.
Note that the algorithm $pgSolver1$ was only tested on parity games that either had the maximum priority $M \leq 2$ or $M \leq 4$, with the number of nodes $n \leq 300$. This is because the complexity of the subgraph solver loop in this algorithm is n^{M+2}, which not only would lead to extremely long execution times, but also would lead to a technical error, as this implementation needs that value to fit in the $long\ long$ data type, which has the range $(9,223,372,036,854,775,807,\ +9,223,372,036,854,775,807)$.

Table 1. This table showcases the information on 6 of the test that were run from Keiren's benchmarks [8]. From here, it is obvious how much the execution time of pgSolver1 can vary. Even though the other 2 tests shown here had more than 100 vertices over the that particular game, the fact that the maximum priority was equal to 2, combined with its internal structure, lead to an execution time of 5 minutes. The nature of the subgraphs that are obtained from the original game greatly affect the execution time, as they influence how often the loop in $mpgSubgraphSolver$ is used. Given the time complexity of that loop, it is the reason why it is possible to have such long execution times on such small graphs.

We can also already notice how efficient $pgSolver3$ is. In Fearnley's experimental results [4], the sequential implementation of the strategy improvement algorithm could solve tests from Keiren's set[8] with even $500,000$ vertices in less than 1 second.

[1] https://github.com/Vortex1711/PG-Reachability-Benchmakrs

Test	Vertices	Max Priority	pgSolver1 time (s)	pgSolver2 time (s)	pgSolver3 time (s)
StarNesterk=1_n=2.gm	336	1	1	1	0
ABP(BW)datasize=4_infinitely_often_receive_d1.gm	196	2	3041	1	0
StarNesterk=1_n=2.gm	336	1	1	1	0
Pardatasize=8_infinitely_often_read_write.gm	414	2	!	6	0
Pardatasize=4_infinitely_often_receive_for_all_d.gm	734	2	!	23	0
randomgame(1000,_10,_1,_20)id=0.gm	1000	10	!	196	0

Table 1: Results on some of Keiren's tests. ! marks that the algorithm was not executed for that test.

Figure 1. This chart shows the execution time of all 3 algorithms across all 45 tests, with the exception of the test where $pgSolver1$ had an execution time of 5 minutes, in order to properly showcase the rest of the data.
Here, we can see that the execution times of the $pgSolver2$ algorithm greatly increase with the number of nodes in the graph. The average execution time of this algorithm for tests with at least 500 nodes was 68.68s.

Figure 1: Execution time of all algorithms across almost all test cases.

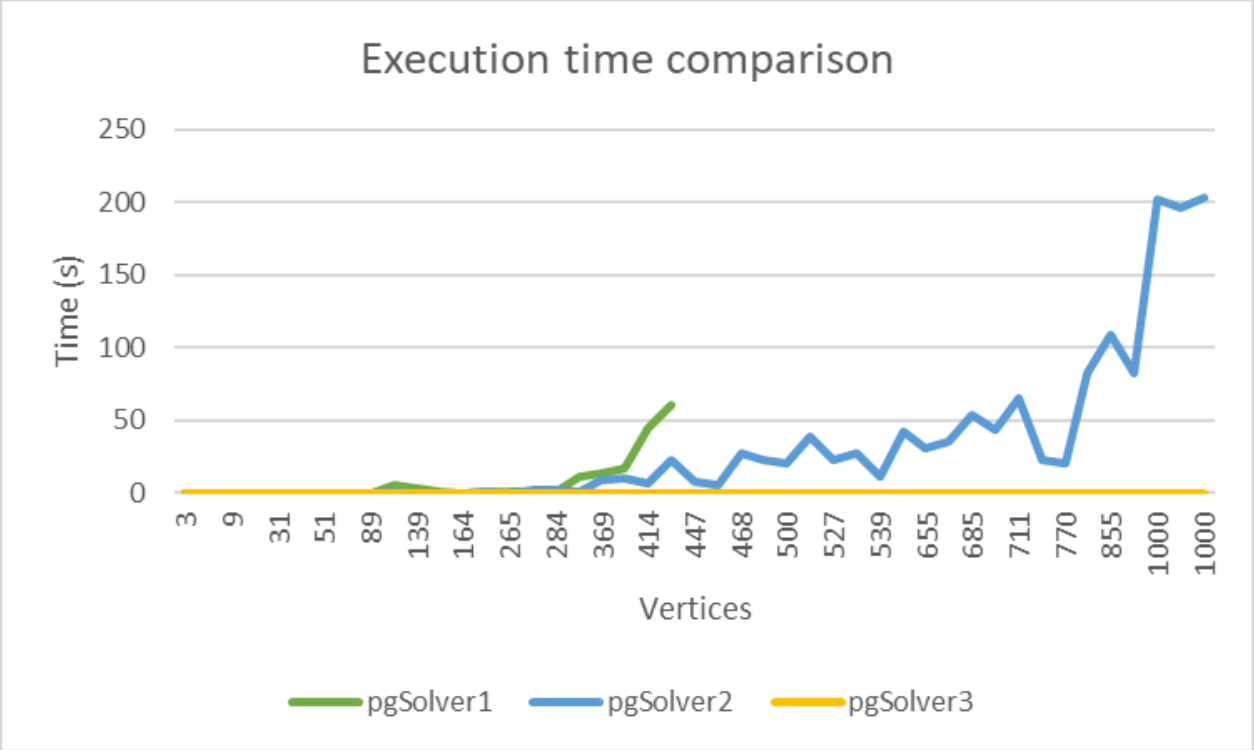
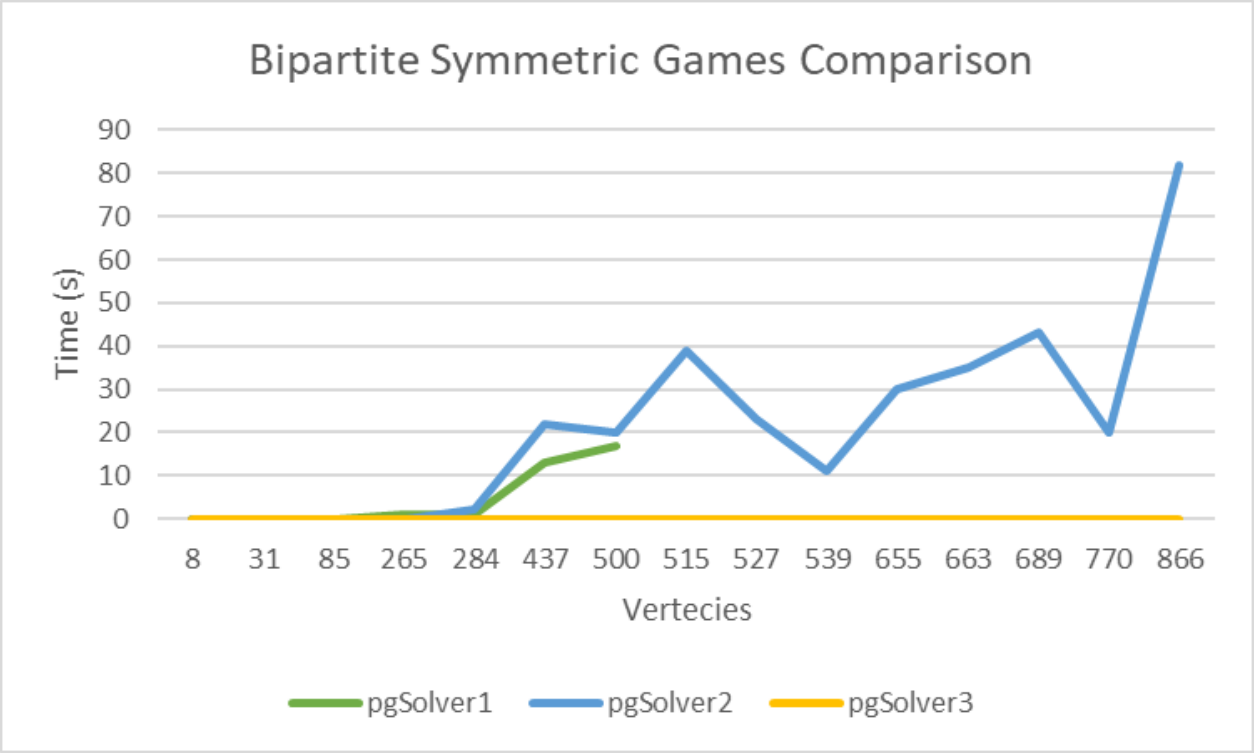


Figure 2. This chart shows the execution time of all 3 algorithms on the set of random bipartite, symmetric parity games that we generated. Considering that these games are solved in linear time, as they are games with the signature of a potential, we see a drastic decrease in execution time. In this particular subset, average of the execution times for $pgSolver2$ in parity games with over 500 nodes is just 33.66s

Figure 2: Execution time of all algorithms for bipartite, symmetric parity games.



6 Conclusion

From the experimental results, we can conclude that while we achieved our goal of adapting Akian et al.'s algorithm [3] to solve parity games, and for a particular set of games, known as parity games with the signature of a potential, even provide the solution of the game in linear time complexity, these algorithms are not very efficient in practice.Through future research, perhaps a better performing algorithm could be discovered that uses the same structure as the algorithms we focused on here, but uses a different parity game solver for determining the winners in subgraphs.

References

[1] Oliver Friedmann and Martin Lange. The pgsolver collection of parity game solvers. *University of Munich*, pages 4–6, 2009.

[2] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022.

[3] M. Akian, S. Gaubert, Lorscheid O., and Mnich M. Mean payoff games with the signature of a potential, 2022. Unpublished.

[4] John Fearnley. Efficient parallel strategy improvement for parity games. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 137–154, Cham, 2017. Springer International Publishing.

[5] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343–359, 1996.

[6] Ehrenfeucht A. and Mycielski J. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

[7] D. Berwanger and O. Serre. Parity games on undirected graphs. *Information Processing Letters*, 112(23):928–932, 2018.

[8] Jeroen Keiren. Benchmarks for parity games. volume 9392, 07 2014.