Parity Games

Constantin Cristiana Georgiana

May 2024

Contents

1 Definitions and notations

1.1 Definition of parity games

A parity game is a two-player infinite-duration game played on a finite directed graph. It is defined as a tuple $G = (V, V_0, V_1, E, \Omega)$ where:

- V is a finite set of vertices (or nodes).
- V_0 and V_1 partition V ($V_0 \cap V_1 = \emptyset$ and $V_0 \cup V_1 = V$).
- $E \subseteq V \times V$ is the edge relation, where each vertex has at least one outgoing edge.
- $\Omega : V \rightarrow \mathbb{N}$ is the priority function that assigns a natural number (priority) to each vertex.

Here are some key terms and concepts related to parity games:

1. **Players**: There are two players, typically called Player 0 (Even) and Player 1 (Odd).
2. **Positions**: The vertices in V_0 are positions where Player 0 makes a move, and vertices in V_1 are positions where Player 1 makes a move.
3. **Moves**: A move consists of moving a token from the current vertex to an adjacent vertex along an edge in E.
4. **Play**: A play is an infinite sequence of vertices $\pi = v_0, v_1, v_2, \ldots$ such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$.
5. **Winning Condition**: Player 0 wins a play π if the highest priority occurring infinitely often in π is even. Player 1 wins if the highest priority occurring infinitely often is odd.
6. **Strategy**: A strategy for a player is a function that determines the next move based on the history of the play.
7. **Winning Strategy**: A strategy is winning for a player if all plays consistent with that strategy are winning for that player.
8. **Winning Region**: The winning region for a player is the set of vertices from which that player has a winning strategy.

1.2 Formal notation and examples

Formal Notation:

Let's formalize some of the concepts:

- A **strategy** for Player 0 is a function $\sigma : V^* \times V_0 \to V$ such that if $\sigma(v_0, \ldots, v_n) = v_{n+1}$, then $(v_n, v_{n+1}) \in E$.
- Similarly, a **strategy** for Player 1 is a function $\tau : V^* \times V_1 \to V$ with the same constraint.
- A play $\pi = v_0, v_1, v_2, \ldots$ is **consistent** with a strategy σ for Player 0 if for all i such that $v_i \in V_0$, we have $v_{i+1} = \sigma(v_0, \ldots, v_i)$.
- The **winning condition** can be formally expressed as: Player 0 wins a play $\pi = v_0, v_1, v_2, \ldots$ if $\max\{\Omega(v) \mid v \text{ appears infinitely often in } \pi\}$ is even.

Example of a Parity Game:

Consider a simple parity game with the following components:

- Vertices: $V = \{v_0, v_1, v_2, v_3\}$
- Player 0's vertices: $V_0 = \{v_0, v_2\}$
- Player 1's vertices: $V_1 = \{v_1, v_3\}$
- Edges: $E = \{(v_0, v_1), (v_1, v_2), (v_2, v_3), (v_3, v_0), (v_1, v_3)\}$
- Priorities: $\Omega(v_0) = 2, \Omega(v_1) = 1, \Omega(v_2) = 4, \Omega(v_3) = 3$

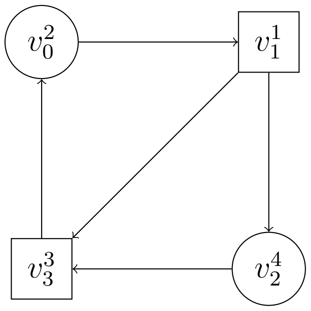

Figure 1: An example of a parity game. Circles represent vertices in V_0 (Player 0), rectangles represent vertices in V_1 (Player 1). The superscript numbers indicate the priorities assigned by Ω.

In this example, if the play starts at v_0 and both players play optimally, the resulting play would be $v_0, v_1, v_3, v_0, v_1, v_3, \ldots$. The highest priority occurring infinitely often would be 3, which is odd, so Player 1 would win this play.

Positional Determinacy:

A key property of parity games is that they are positionally determined, meaning:

- For every vertex, one of the players has a winning strategy.
- Moreover, if a player has a winning strategy from a vertex, they have a positional winning strategy, which depends only on the current vertex, not on the history of the play.

This property is crucial for the algorithmic solutions to parity games, as it significantly reduces the search space for winning strategies.

2 Applications of parity games

2.1 Motivation for studying parity games

Parity games have emerged as a central problem in theoretical computer science and formal verification for several compelling reasons:

1. **Theoretical Significance**: Parity games occupy a unique position in complexity theory. They are in NP $\cap$ co-NP (and even in UP $\cap$ co-UP), suggesting they might be solvable in polynomial time, yet no polynomial-time algorithm has been discovered.
2. **Connection to Modal μ-calculus**: Parity games are equivalent to the model checking problem for the modal μ-calculus, a powerful temporal logic used for specifying properties of systems.
3. **Automata Theory**: Parity games are closely related to automata on infinite trees and the theory of infinite games, which are fundamental in the study of formal languages and verification.
4. **Practical Applications**: Solutions to parity games directly translate to solutions for various verification and synthesis problems in computer science.

The study of parity games has led to significant advancements in algorithm design, complexity theory, and formal methods. The quest for efficient algorithms to solve parity games has driven innovation in these fields and continues to be an active area of research.

2.2 The use of parity games in various fields

Parity games find applications in numerous areas of computer science and formal methods:

1. **Model Checking**

   Parity games are fundamental to model checking, particularly for the modal μ-calculus:

   - **System Verification**: Checking whether a system satisfies a specification expressed in the modal μ-calculus can be reduced to solving a parity game.
   - **Temporal Logic**: The model checking problem for various temporal logics, including CTL* and LTL, can be translated to parity games.

2. **Program Synthesis**

   Parity games provide a framework for automatic program synthesis:

   - **Reactive Systems**: Synthesizing controllers for reactive systems that satisfy given specifications.
   - **Correct-by-Construction**: Generating programs that are correct by construction with respect to their specifications.

3. **Automata Theory**

   Parity games are closely connected to automata on infinite objects:

   - **Determinization**: The determinization of automata on infinite words and trees.
   - **Complementation**: Constructing complement automata for automata on infinite objects.

4. **Game Theory**

   Parity games serve as a canonical example in game theory:

   - **Infinite Games**: They are a fundamental model for studying infinite-duration games.
   - **Strategy Synthesis**: Techniques for synthesizing optimal strategies in various game-theoretic contexts.

5. **Formal Verification**

   Parity games are used in formal verification tools:

- **Hardware Verification**: Verifying that hardware designs meet their specifications.
- **Software Verification**: Checking that software systems satisfy safety and liveness properties.

Practical Example in Verification

Consider the problem of verifying that a mutual exclusion protocol ensures that no two processes can be in their critical sections simultaneously. This safety property can be expressed in the modal μ-calculus and verified by solving the corresponding parity game.

The verification process would involve:

1. Modeling the protocol as a transition system.
2. Expressing the mutual exclusion property in the modal μ-calculus.
3. Constructing a parity game from the model and the property.
4. Solving the parity game to determine if the property holds.

If Player 0 (the verifier) has a winning strategy in the resulting parity game, then the protocol satisfies the mutual exclusion property. Otherwise, Player 1's winning strategy corresponds to a counterexample showing how the property can be violated.

In summary, parity games serve as a powerful mathematical framework that connects various areas of theoretical computer science and enables practical applications in verification, synthesis, and analysis of complex systems.

3 Studying the complexity of the problem

3.1 Classification of the problem in complexity theory

The complexity of parity games is a fascinating topic in theoretical computer science due to its unique position within the landscape of computational complexity.

Complexity Class Membership

Parity games exhibit several interesting complexity-theoretic properties:

1. **NP $\cap$ co-NP**: Parity games are in both NP and co-NP. This means that both a positive answer ("Player 0 wins from vertex v") and a negative answer ("Player 1 wins from vertex v") can be verified in polynomial time given the appropriate certificate (a winning strategy for the respective player).

2. **UP ∩ co-UP**: Even more restrictively, parity games are in UP ∩ co-UP (Unique Polynomial time), which means that for any instance, there is at most one accepting certificate that can be verified in polynomial time.

3. **PLS (Polynomial Local Search)**: Parity games also belong to the complexity class PLS, which characterizes problems where a locally optimal solution can be found by iterative improvement.

This places parity games in a rare category of problems that are likely in P (polynomial time) but for which no polynomial-time algorithm has been proven. Other problems in this category include factoring integers, discrete logarithm, and stochastic games.

Parity Games and P vs. NP

The status of parity games has implications for the P vs. NP question:

- If parity games are in P, it would not resolve the P vs. NP question, but it would provide insights into the structure of problems in NP ∩ co-NP.
- If parity games are proven to be NP-hard, it would imply NP = co-NP, which is widely believed to be false. This makes it unlikely that parity games are NP-complete.

Reductions and Equivalences

Parity games are polynomially equivalent to several other important problems:

- **Modal μ-calculus Model Checking**: Determining whether a given structure satisfies a formula in the modal μ-calculus.
- **Boolean μ-calculus Evaluation**: Evaluating a Boolean formula with least and greatest fixed-point operators.
- **Determinization of Automata**: Converting nondeterministic parity automata to deterministic ones.

These equivalences highlight the central role of parity games in theoretical computer science and formal verification.

3.2 The quasi-polynomial breakthrough

In 2017, a significant breakthrough in the complexity of parity games was achieved when Calude et al. presented the first quasi-polynomial time algorithm for solving parity games. This was a major advancement over the previous best algorithms, which had sub-exponential time complexity.

Quasi-polynomial Complexity

A quasi-polynomial time algorithm has a running time of $n^{O(\log n)}$, where n is the size of the input. This is significantly better than exponential time (2^n) but not as efficient as polynomial time (n^c for some constant c).

The algorithm by Calude et al. has a time complexity of approximately $n^{O(\log d)}$, where n is the number of vertices and d is the number of different priorities in the game.

Key Insights of the Quasi-polynomial Algorithm

The breakthrough algorithm is based on several key insights:

1. **Separating Automata**: The algorithm uses the concept of separating automata, which can distinguish between plays that are winning for Player 0 and those winning for Player 1.

2. **Succinct Witnesses**: It shows that winning strategies can be represented by succinct witnesses of quasi-polynomial size.

3. **Progress Measure**: The algorithm employs a novel progress measure that tracks the evolution of the game in a more efficient manner than previous approaches.

Subsequent Improvements

Following the initial breakthrough, several researchers have proposed refinements and alternative quasi-polynomial algorithms:

- Jurdziński and Lazić (2017) developed a quasi-polynomial algorithm based on succinct progress measures.

- Lehtinen (2018) presented an approach using register games, which provides a different perspective on the quasi-polynomial solution.

- Parys (2019) simplified the original algorithm and improved its practical performance.

Implications of the Breakthrough

The quasi-polynomial algorithm for parity games has several important implications:

1. It significantly narrows the gap between the upper and lower bounds for the complexity of parity games.

2. It suggests that a polynomial-time algorithm might be achievable, as quasi-polynomial algorithms often precede polynomial ones in the history of algorithm development.

3. It provides new techniques and insights that have been applied to related problems in formal verification and automata theory.

Despite this breakthrough, the question of whether parity games can be solved in polynomial time remains open and continues to be an active area of research.

4 Algorithms

4.1 Recursive algorithm (Zielonka)

The recursive algorithm for solving parity games, developed by Wiesław Zielonka, is one of the classical approaches to this problem. Despite its worst-case exponential time complexity, it often performs well in practice and serves as the foundation for many subsequent algorithms.

Algorithm Description

The recursive algorithm is based on the concept of attractors and the properties of the parity winning condition:

Key Components

1. **Attractor Computation**: For a set of vertices U and a player i, the attractor $\text{Attractor}_i(G, U)$ is the set of vertices from which player i can force a visit to U. It is computed iteratively:

   - Start with $A_0 = U$
   - For each $k \geq 0$, compute A_{k+1} by adding to A_k all vertices $v \in V_i$ that have at least one edge to A_k, and all vertices $v \in V_{1-i}$ that have all edges leading to A_k.
   - The process continues until a fixed point is reached, which is the attractor.

2. **Recursive Structure**: The algorithm recursively solves smaller subgames obtained by removing attractors, and combines the results to determine the winning regions of the original game.

Algorithm 1 Zielonka's Recursive Algorithm

```
function Solve($G = (V, V_0, V_1, E, \Omega)$)
    if $V = \emptyset$ then return $(\emptyset, \emptyset)$          ▷ Return empty winning regions
    end if
    $p \leftarrow \max\{\Omega(v) \mid v \in V\}$          ▷ Highest priority in the game
    $i \leftarrow p \bmod 2$          ▷ Player who loses if priority $p$ occurs infinitely often
    $U \leftarrow \{v \in V \mid \Omega(v) = p\}$          ▷ Vertices with highest priority
    $A \leftarrow \text{Attractor}_i(G, U)$          ▷ Attractor of $U$ for Player $i$
    $G' \leftarrow G \setminus A$          ▷ Remove attractor from the game
    $(W'_0, W'_1) \leftarrow \text{Solve}(G')$          ▷ Recursively solve the smaller game
    if $W'_{1-i} = \emptyset$ then
        $W_i \leftarrow V$          ▷ Player $i$ wins everywhere
        $W_{1-i} \leftarrow \emptyset$
    else
        $B \leftarrow \text{Attractor}_{1-i}(G, W'_{1-i})$          ▷ Attractor of opponent's winning region
        $G'' \leftarrow G \setminus B$          ▷ Remove this attractor
        $(W''_0, W''_1) \leftarrow \text{Solve}(G'')$          ▷ Recursively solve again
        $W_i \leftarrow W''_i$          ▷ Update winning regions
        $W_{1-i} \leftarrow W''_{1-i} \cup B$
    end if
    return $(W_0, W_1)$
end function
```

Complexity Analysis

The worst-case time complexity of Zielonka's algorithm is $O(n^d)$, where n is the number of vertices and d is the number of different priorities. This is because:

- In the worst case, each recursive call removes only a small number of vertices.
- The depth of recursion can be proportional to the number of different priorities.
- Each recursive call involves attractor computations, which take $O(m)$ time, where m is the number of edges.

Despite this exponential worst-case complexity, the algorithm often performs well in practice, especially for games with a small number of priorities or specific structural properties.

4.2 Small progress measures (Jurdziński)

The small progress measures algorithm, developed by Marcin Jurdziński, is another classical approach to solving parity games. It uses a different paradigm compared to the recursive algorithm, focusing on a fixed-point computation of a ranking function.

Algorithm Concept

The algorithm is based on the notion of progress measures, which are functions that map vertices to tuples of natural numbers. These tuples represent a form of ranking that captures the ability of Player 0 to win from each vertex.

Progress Measures

A progress measure is a function $\mu : V \to M_\Omega$, where M_Ω is a set of tuples $(m_1, m_2, \ldots, m_d)$ with specific constraints:

- d is the number of odd priorities in the game.
- For each odd priority p, the corresponding component m_p is bounded by the number of vertices with priority p.
- The tuples are ordered lexicographically.

The progress measure satisfies certain local conditions that ensure it correctly identifies the winning regions.

Algorithm Description

Algorithm 2 Small Progress Measures Algorithm

```
1:  function SolveWithProgressMeasures($G = (V, V_0, V_1, E, \Omega)$)
2:      Initialize $\mu(v) = (0, 0, \ldots, 0)$ for all $v \in V$
3:      while $\mu$ is not a fixed point do
4:          for each $v \in V$ do
5:              if $v \in V_0$ then
6:                  $\mu(v) \leftarrow \min\{\text{Lift}(v, w, \mu) \mid (v, w) \in E\}$
7:              else
8:                  $\mu(v) \leftarrow \max\{\text{Lift}(v, w, \mu) \mid (v, w) \in E\}$
9:              end if
10:         end for
11:     end while
12:     $W_0 \leftarrow \{v \in V \mid \mu(v) \neq \top\}$        ▷ Vertices with finite measure
13:     $W_1 \leftarrow V \setminus W_0$                          ▷ Vertices with infinite measure
          return $(W_0, W_1)$
14: end function
```

The Lift operation updates the progress measure based on the priority of the current vertex and the measure of its successor:

- If the priority is even, it preserves or decreases the measure.
- If the priority is odd, it increments the corresponding component of the measure or sets it to $\top$ (infinity) if it exceeds the bound.

Complexity Analysis

The time complexity of the small progress measures algorithm is $O(d \cdot m \cdot (n/d)^{d/2})$, where:

- n is the number of vertices
- m is the number of edges
- d is the number of different priorities

This is better than the recursive algorithm for games with a small number of priorities, but still exponential in the worst case.

Advantages

The small progress measures algorithm has several advantages:

1. It is amenable to efficient implementation using data structures like priority queues.

2. It can be parallelized more easily than the recursive algorithm.

3. It provides a different perspective on parity games, which has led to further algorithmic developments.

The algorithm has been the basis for several improvements and variations, including the succinct progress measures algorithm that achieves quasi-polynomial time complexity.

4.3 Strategy improvement algorithms

Strategy improvement algorithms represent a different approach to solving parity games, based on the principle of iteratively improving a strategy until an optimal one is found. This approach was pioneered by Jürgen Vöge and Marcin Jurdziński.

Algorithm Concept

The strategy improvement algorithm starts with an arbitrary strategy for Player 0 and iteratively improves it until a winning strategy is found or it is determined that no winning strategy exists.

Key Components

1. **Strategy Evaluation**: Given a strategy σ for Player 0, compute a valuation function that assigns a value to each vertex, representing how good the strategy is from that vertex.

2. **Strategy Improvement**: Identify vertices where the strategy can be improved (i.e., where a different move would lead to a better valuation), and update the strategy accordingly.

3. **Termination**: The algorithm terminates when no further improvements are possible, at which point the current strategy is optimal.

Algorithm Description

Valuation Function

The valuation function val_σ assigns to each vertex a value that represents the quality of the play from that vertex when Player 0 follows strategy σ and Player 1 plays optimally against it. The valuation is typically a tuple that encodes:

- Whether Player 0 wins from the vertex.

- The highest priority seen infinitely often in the play.

Algorithm 3 Strategy Improvement Algorithm

```
 1: function SolveWithStrategyImprovement($G = (V, V_0, V_1, E, \Omega)$)
 2:     Initialize a strategy $\sigma$ for Player 0
 3:     improved $\leftarrow$ true
 4:     while improved do
 5:         Compute the valuation $\mathrm{val}_\sigma$ for the current strategy $\sigma$
 6:         improved $\leftarrow$ false
 7:         for each $v \in V_0$ do
 8:             for each $(v, w) \in E$ do
 9:                 if $\mathrm{val}_\sigma(w) > \mathrm{val}_\sigma(\sigma(v))$ then
10:                     $\sigma(v) \leftarrow w$                    ▷ Improve the strategy
11:                     improved $\leftarrow$ true
12:                 end if
13:             end for
14:         end for
15:     end while
16:     Compute winning regions $(W_0, W_1)$ based on the final strategy $\sigma$ return $(W_0, W_1)$
17: end function
```

- Additional information about the structure of the play.

Complexity Analysis

The strategy improvement algorithm has the following complexity characteristics:

- In the worst case, it may require exponentially many iterations, as there can be exponentially many different strategies to consider.
- Each iteration involves computing the valuation function, which takes polynomial time.
- In practice, the algorithm often converges quickly, making it efficient for many instances.

Advantages and Variations

Strategy improvement algorithms have several advantages:

1. They provide not just the winning regions but also explicit winning strategies.
2. They can be adapted to handle more general games, such as mean-payoff games and discounted payoff games.

3. They are amenable to various heuristics that can improve practical performance.

Several variations and improvements of the basic strategy improvement algorithm have been proposed:

- Discrete strategy improvement, which uses a simpler valuation function.
- Randomized strategy improvement, which makes random choices in the improvement step to avoid certain worst-case scenarios.
- Strategy improvement with different switching policies, which determine how to select among multiple possible improvements.

Strategy improvement algorithms represent a powerful approach to solving parity games and continue to be an active area of research.

4.4 Quasi-polynomial algorithms

The development of quasi-polynomial algorithms for parity games represents a major breakthrough in the field. These algorithms significantly improved upon the previous best algorithms, which had exponential or sub-exponential time complexity.

The Calude et al. Algorithm

In 2017, Calude, Jain, Khoussainov, Li, and Stephan presented the first quasi-polynomial time algorithm for solving parity games. The algorithm has a time complexity of approximately $n^{O(\log d)}$, where n is the number of vertices and d is the number of different priorities.

Key Insights

The algorithm is based on several key insights:

1. **Separating Automata**: The algorithm uses automata that can separate plays that are winning for Player 0 from those winning for Player 1.
2. **Succinct Witnesses**: The algorithm shows that winning strategies can be represented by succinct witnesses of quasi-polynomial size.
3. **Universal Trees**: The concept of universal trees provides a framework for organizing the strategy information in a compact way.

Algorithm 4 Quasi-Polynomial Algorithm (Calude et al.)

```
1: function SolveQuasiPolynomial($G = (V, V_0, V_1, E, \Omega)$)
2:     Construct a separation game $G'$ based on $G$
3:     Solve $G'$ using a modified version of the attractor computation
4:     Map the solution of $G'$ back to a solution of $G$ return Winning regions ($W_0, W_1$)
5: end function
```

Algorithm Description

At a high level, the algorithm works as follows:

The separation game G' is constructed in such a way that winning strategies in G' correspond to winning strategies in the original game G, but with the advantage that G' has a structure that allows for more efficient solution methods.

Subsequent Quasi-Polynomial Algorithms

Following the initial breakthrough, several other quasi-polynomial algorithms have been developed:

1. **Succinct Progress Measures (Jurdziński and Lazić, 2017)**: This algorithm uses a more compact representation of progress measures, achieving quasi-polynomial time complexity with a different approach.
2. **Register Games (Lehtinen, 2018)**: This approach reduces parity games to equivalent parity games with a logarithmic number of priorities, leading to a quasi-polynomial algorithm.
3. **Universal Attractor Decomposition (Parys, 2019)**: This algorithm simplifies the original approach and improves its practical performance.

Practical Considerations

While quasi-polynomial algorithms represent a significant theoretical advancement, their practical performance can vary:

- They often have large constant factors and complex data structures, which can make them less efficient than simpler algorithms for small or structured instances.
- Implementations of these algorithms continue to be refined to improve their practical performance.
- Hybrid approaches that combine ideas from different algorithms can be effective in practice.

The development of quasi-polynomial algorithms has narrowed the gap between the upper and lower bounds for the complexity of parity games, but the question of whether a polynomial-time algorithm exists remains open.

5 Own approach

5.1 Decomposition-based hybrid algorithm

In this section, I propose a novel approach to solving parity games that combines elements from existing algorithms with a focus on exploiting the structural properties of the game graph. The key insight is that real-world parity games often have structure that can be exploited for more efficient solutions.

Key Insights

The proposed approach is based on the following insights:

1. **SCC Decomposition**: Parity games can be decomposed into strongly connected components (SCCs), which can be solved independently and then combined.

2. **Priority Compression**: Many parity games have priorities that can be compressed without changing the winning regions, reducing the effective number of priorities.

3. **Algorithm Selection**: Different algorithms perform better on different types of game structures, suggesting a hybrid approach.

4. **Partial Solving**: Often, large portions of a parity game can be solved quickly using simple rules, leaving a smaller residual game.

Algorithm Description

The proposed hybrid algorithm consists of the following steps:

Components of the Algorithm

1. **SCC Decomposition**: The game graph is decomposed into strongly connected components, which can be done in linear time using Tarjan's algorithm.

2. **Priority Compression**: The priorities in each SCC are compressed to a contiguous range, potentially reducing the effective number of priorities.

Algorithm 5 Decomposition-Based Hybrid Algorithm

```
function SolveHybrid($G = (V, V_0, V_1, E, \Omega)$)
    Decompose $G$ into strongly connected components (SCCs)
    Sort SCCs in topological order
    for each SCC $S$ in topological order do
        if IsTrivialSCC($S$) then
            SolveTrivialGame($S$)
        else
            $S' \leftarrow$ CompressPriorities($S$)
            $(Solved, Remaining) \leftarrow$ AttractorDecomposition($S'$)
            if not Empty($Remaining$) then
                if IsSmall($Remaining$) then
                    SolveWithRecursiveAlgorithm($Remaining$)
                else if HasLowTreewidth($Remaining$) then
                    SolveWithTreewidthAlgorithm($Remaining$)
                else
                    SolveWithQuasiPolynomialAlgorithm($Remaining$)
                end if
            end if
            CombineSolutions($Solved$, $Remaining$)
        end if
    end for
    PropagateSolutions(SCCs) return WinningRegions
end function
```

3. **Attractor Decomposition**: A partial solving technique that identifies vertices that can be immediately classified as winning for one player.
4. **Algorithm Selection**: Based on the properties of the remaining game (size, treewidth, etc.), an appropriate algorithm is selected:
    - Recursive algorithm for small games
    - Specialized algorithms for games with low treewidth
    - Quasi-polynomial algorithm for general cases
5. **Solution Propagation**: The solutions for individual SCCs are combined and propagated through the topological order to obtain the solution for the entire game.

5.2 Analysis and examples

Theoretical Analysis

The complexity of the hybrid algorithm depends on the structure of the input game:

- **Best Case**: If the game decomposes into small SCCs or has a simple structure, the algorithm can run in near-linear time.
- **Worst Case**: In the worst case, the algorithm falls back to the quasi-polynomial algorithm, resulting in a time complexity of $n^{O(\log d)}$.
- **Average Case**: For many practical instances, the algorithm is expected to perform significantly better than the worst-case bound due to the exploitation of structure.

Example: Solving a Structured Parity Game

Consider a parity game with the following structure:

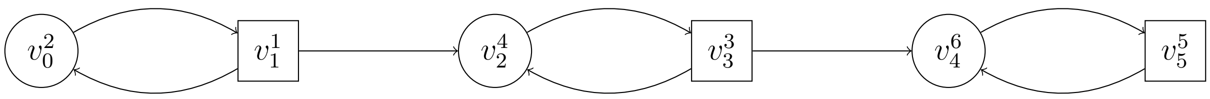

Figure 2: A parity game with three strongly connected components.

Applying the hybrid algorithm to this game:

1. **SCC Decomposition**: The game is decomposed into three SCCs: $\{v_0, v_1\}$, $\{v_2, v_3\}$, and $\{v_4, v_5\}$.

2. **Solving SCCs in Topological Order**:

    - SCC 3 $\{v_4, v_5\}$: The highest priority is 6 (even), so Player 0 wins both vertices.
    - SCC 2 $\{v_2, v_3\}$: Since Player 0 wins all successors (SCC 3), the highest priority is 4 (even), so Player 0 wins both vertices.
    - SCC 1 $\{v_0, v_1\}$: Since Player 0 wins all successors (SCC 2), the highest priority is 2 (even), so Player 0 wins both vertices.

3. **Result**: Player 0 wins all vertices in the game.

This example illustrates how the decomposition approach can efficiently solve structured parity games by breaking them down into smaller, manageable components.

Advantages of the Hybrid Approach

The proposed hybrid algorithm offers several advantages:

1. **Adaptability**: It adapts to the structure of the input game, using the most appropriate algorithm for each component.

2. **Efficiency**: By exploiting structure and using partial solving techniques, it can solve many instances more efficiently than general-purpose algorithms.

3. **Scalability**: The approach scales well with the size of the game, as the decomposition allows for parallel processing of independent components.

4. **Robustness**: Even in the worst case, the algorithm falls back to the best known general algorithm, ensuring competitive performance.

The hybrid approach represents a practical solution to parity games that combines theoretical insights with engineering considerations, aiming to provide efficient solutions for real-world instances.

6 References

1. Calude, C. S., Jain, S., Khoussainov, B., Li, W., & Stephan, F. (2017). Deciding parity games in quasipolynomial time. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (pp. 252-263).

2. Jurdziński, M. (2000). Small progress measures for solving parity games. In Annual Symposium on Theoretical Aspects of Computer Science (pp. 290-301).

3. Zielonka, W. (1998). Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science, 200(1-2), 135-183.

4. Jurdziński, M., & Lazić, R. (2017). Succinct progress measures for solving parity games. In 2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (pp. 1-9).

5. Lehtinen, K. (2018). A modal μ perspective on solving parity games in quasi-polynomial time. In Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (pp. 639-648).

6. Vöge, J., & Jurdziński, M. (2000). A discrete strategy improvement algorithm for solving parity games. In International Conference on Computer Aided Verification (pp. 202-215).

7. Fearnley, J., Jain, S., Schewe, S., Stephan, F., & Wojtczak, D. (2017). An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software (pp. 112-121).

8. Parys, P. (2019). Parity games: Zielonka's algorithm in quasi-polynomial time. In International Symposium on Mathematical Foundations of Computer Science (pp. 10:1-10:13).

9. Emerson, E. A., & Jutla, C. S. (1991). Tree automata, mu-calculus and determinacy. In Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (pp. 368-377).

10. Friedmann, O., & Lange, M. (2009). Solving parity games in practice. In International Symposium on Automated Technology for Verification and Analysis (pp. 182-196).

11. Benerecetti, M., Dell'Erba, D., & Mogavero, F. (2016). Solving parity games via priority promotion. In International Conference on Computer Aided Verification (pp. 270-290).

12. Schewe, S. (2008). An optimal strategy improvement algorithm for solving parity and payoff games. In International Workshop on Computer Science Logic (pp. 369-384).

13. Grädel, E., Thomas, W., & Wilke, T. (Eds.). (2002). Automata, logics, and infinite games: A guide to current research (Vol. 2500). Springer Science & Business Media.

14. Kupferman, O., & Vardi, M. Y. (1998). Weak alternating automata and tree automata emptiness. In Proceedings of the 30th Annual ACM Symposium on Theory of Computing (pp. 224-233).

15. Piterman, N. (2007). From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science, 3(3).