Improvement in small progress measures

Improvement in Small Progress Measures

Maciej Gazda and Tim A.C. Willemse
Eindhoven University of Technology, The Netherlands

Small Progress Measures is one of the classical parity game solving algorithms. For games with n vertices, m edges and d different priorities, the original algorithm computes the winning regions and a winning strategy for one of the players in $O(dm\cdot(n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$ time. Computing a winning strategy for the other player requires a re-run of the algorithm on that player's winning region, thus increasing the runtime complexity to $O(dm\cdot(n/\lceil d/2 \rceil)^{\lceil d/2 \rceil})$ for computing the winning regions and winning strategies for both players. We modify the algorithm so that it derives the winning strategy for both players in one pass. This reduces the upper bound on strategy derivation for SPM to $O(dm\cdot(n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$. At the basis of our modification is a novel operational interpretation of the least progress measure that we provide.

1 Introduction

A parity game [3, 14, 20] is an infinite duration game played on a directed graph by two players called *even* and *odd*. Each vertex in the graph is owned by one of the players, and labelled with a natural number, called a priority. The game is played by pushing a token along the edges in the graph; the choice where to move next is made by the owner of the vertex on which the token currently resides. The winner of the thus constructed play is determined by the parity of the minimal (or maximal, depending on the convention) priority that occurs infinitely often, and the winner of a vertex is the player who has a *strategy* to force every play originating from that vertex to be winning for her. Parity games are positionally determined; that is, each vertex is won by some player [14], and each player has a positional winning strategy on her winning region. *Solving* a game essentially means deciding which player wins which vertices in the game.

Parity games play an important role in several foundational results; for instance, they allow for an elegant simplification of the hard part of Rabin's proof of the decidability of a monadic second-order theory, and a number of decision problems of importance can be reduced to deciding the winner in parity games. For instance, the model checking problem for the modal μ-calculus is equivalent, via a polynomial-time reduction, to the problem of solving parity games [4, 18]; this is of importance in computer-aided verification. Winning strategies for the players play a crucial role in supervisory control of discrete event systems, in which such strategies are instrumental in constructing a supervisor that controls a plant such that it reaches its control objectives and avoids bad situations; see *e.g.* [1] and the references therein. In model checking, winning strategies are essential in reporting witnesses and counterexamples, see [18].

A major impetus driving research in parity games is their computational status: the solution problem lies in UP and coUP and is, despite the continued research effort, not known to be in PTIME. Deterministic algorithms for solving parity games include the classical *recursive algorithm* [20] and *small progress measures* (SPM) algorithm [11], the *bigstep* algorithm [15], the deterministic subexponential algorithm [12] and a class of strategy improvement algorithms [19, 16, 5].

Strategy improvement algorithms were long perceived as likely candidates for solving parity games in polynomial time, but, save a recently introduced symmetric variant [17], they were ultimately proven to be exponential in the worst-case [6]. The deterministic subexponential algorithm, running in $n^{O(\sqrt{n})}$ where n is the number of vertices in the game, is a modification of the recursive algorithm which itself runs in $O(m \cdot n^d)$, where m is the number of edges and d is the number of different priorities in the game. Bigstep, which runs in $O(m \cdot (\kappa n/d)^{\gamma(d)})$, where κ is a small constant and $\gamma(d) \approx d/3$, is a combination of the recursive algorithm and the SPM algorithm. This latter algorithm solves a game in $O(dm \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$.

Somewhat surprisingly, our knowledge of classical algorithms such as SPM and the recursive algorithm is still far from complete. For instance, the recursive algorithm is regarded as one of the best algorithms in practice, which is corroborated by experiments [7]. However, until our recent work [8] where we showed the algorithm is well-behaved on several important classes of parity games, there was no satisfactory explanation why this would be the case. In a similar vein, in *ibid.* we provided tighter bounds on the worst-case running time, but so far, no tight bounds for this seemingly simple algorithm have been established. We expect that if improvements on the upper bound on the parity game solving problem can be made, such improvements will come from improvements in, or through a better understanding of the classical algorithms; this expectation is fuelled by the fact that these classical algorithms and the ideas behind them have been at the basis of the currently optimal algorithms.

In this paper, we focus on Jurdziński's small progress measures algorithm. Using a fixpoint computation, it computes a *progress measure*, a labelling of vertices, that witnesses the existence of winning strategies. In general, no clear, intuitive interpretation of the information contained in the progress measures has been given, and the mechanics of the algorithm remain rather technical. This is in contrast to the self-explanatory recursive algorithm, and the strategy improvement algorithm, where, thanks to the ordering of plays according to profiles, at every step, one has a clear picture of the currently known best strategy. Apart from Jurdziński's original article, some additional insight was offered by Klauck and Schewe. In [10], Klauck defines a specific parity progress measure for a solitaire game with only even simple cycles and explains that it has a particular interpretation, but his parity progress measure is not generally related to the measure computed by the SPM algorithm (not even in the setting of solitaire games). Schewe, in his paper on *bigstep* [15], analyses progress measures with restricted codomains and shows how they can be utilised to efficiently detect small dominions. Our *first* contribution is to provide a better understanding of these progress measures and the intermediate values in the fixpoint computation, see Section 4. By doing so, a better understanding of the algorithm itself is obtained.

Progress measures come in two flavours, *viz.* even-and odd-biased, and their computation time is bounded by either $O(dm \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$ or $O(dm \cdot (n/\lceil d/2 \rceil)^{\lceil d/2 \rceil})$, depending on the parity of the extreme priorities that occur in the game. From an even-biased progress measure, one immediately obtains winning regions for *both* players, but only a winning strategy for player even on its winning region and not for her opponent. Likewise for odd-biased progress measures. Obtaining the winning strategy for an opponent thus requires re-running the algorithm on the opponent's winning region. Note that the effort that needs to be taken to obtain a strategy in the same time bound as the winning region stems from a more general feature of parity games: a winning partition in itself does not allow one to efficiently compute a winning strategy (unless there is an efficient algorithm for solving parity games). This basic result, which we nevertheless were unable to find in the literature, is formalised in Section 5.

An essential consequence of this is that the algorithm solves a parity game in $O(dm \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$, as one can always compute one of the two types of progress measures in the shorter time bound. Contrary to what is stated in [11], the same reasoning does not apply to computing the winning strategy for a fixed player; this still requires $O(dm \cdot (n/\lceil d/2 \rceil)^{\lceil d/2 \rceil})$ in the worst case, as also observed by Schewe in [15]. An intriguing open problem is whether it is at all possible to derive the winning strategies for both players while computing one type of measure only, as this would lower the exponent in the time bound to $\lfloor d/2 \rfloor$. Our *second* contribution is to give an affirmative answer to the above problem. We modify the generic SPM by picking a partial strategy when a vertex won by player $\Box$ is discovered, and subsequently adjust the lifting policy so that it prioritises the area which contains an $\Box$-dominion. Both steps are inspired by the interpretation of the progress measures that we discuss in Section 4. Like the original algorithm, our solution, which we present in Section 5, still works in polynomial space. Formal proofs of all results can be found in our technical report [9].

2 Parity Games

A parity game is an infinite duration game, played by players *odd*, denoted by $\Box$ and *even*, denoted by $\Diamond$, on a directed, finite graph. The game is formally defined as follows.

Definition 1 A parity game is a tuple $(V, E, \mathscr{P}, (V_\Diamond, V_\Box))$, where

- V is a finite set of vertices, partitioned in a set $V_\Diamond$ of vertices owned by player $\Diamond$, and a set of vertices $V_\Box$ owned by player $\Box$,
- $E \subseteq V \times V$ is a total edge relation, *i.e.* all vertices have at least one outgoing edge,
- $\mathscr{P}$:$V \to \mathbb{N}$ is a priority function that assigns priorities to vertices.

Henceforth, we assume that $\bigcirc$ denotes an arbitrary player; that is $\bigcirc \in \{\Box, \Diamond\}$. We write $\overline{\bigcirc}$ for $\bigcirc$'s opponent: $\overline{\Diamond} = \Box$ and $\overline{\Box} = \Diamond$. Parity games are depicted as graphs; diamond-shaped nodes represent vertices owned by player $\Diamond$, box-shaped nodes represent vertices owned by player $\Box$ and the priority assigned to a vertex is written inside the node, see the game depicted in Figure 1.

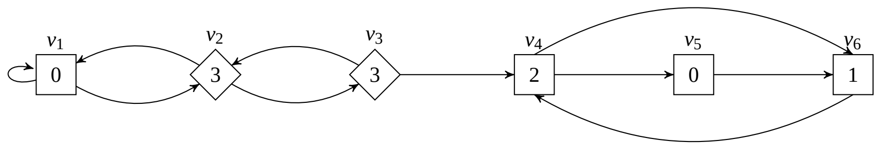

Figure 1: A simple parity game with 4 different priorities, in which 4 vertices are owned by player odd and 2 vertices are owned by player even.

Throughout this section, assume that $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$ is an arbitrary parity game. We write $v \to w$ iff $(v, w) \in E$, and we write $v^\bullet$ to denote the set of successors of v, *i.e.* $\{w \in V \mid v \to w\}$. For a set of vertices $W \subseteq V$, we will denote the minimal priority occurring in W with $\mathsf{min}_{\mathscr{P}}(W)$; by V_i we denote the set of vertices with priority i; likewise for $V_{\leq k}$. For a set $A \subseteq V$, we define $G \cap A$ as the structure $(A, E \cap (A \times A), \mathscr{P}|_A, (V_\Diamond \cap A, V_\Box \cap A))$; the structure $G \setminus A$ is defined as $G \cap (V \setminus A)$. The structures $G \cap A$ and $G \setminus A$ are again parity games if their edge relations are total.

Plays and Strategies A sequence of vertices $v_1, \ldots, v_n$ is a *path* if $v_m \to v_{m+1}$ for all $1 \leq m < n$. Infinite paths are defined in a similar manner. We write π_i to denote the i^{th} vertex in a path π.

A game starts by placing a token on some vertex $v \in V$. Players $\Diamond$ and $\Box$ move the token indefinitely according to a single simple rule: if the token is on some vertex that belongs to player $\bigcirc$, that player moves the token to an adjacent vertex. An infinite sequence of vertices created this way is called a *play*.

The *parity* of the *least* priority that occurs infinitely often on a play defines the *winner* of the play: player $\Diamond$ wins if, and only if this priority is even.

A *strategy* for player $\bigcirc$ is a partial function $\sigma{:}V^+ \to V$ satisfying that whenever it is defined for a finite path $u_1 \cdots u_n \in V^+$, both $u_n \in V_\bigcirc$ and $\sigma(u_1 \cdots u_n) \in u_n^\bullet$. We denote the set of strategies of player $\bigcirc$ by $\mathbb{S}^*_\bigcirc$. An infinite play $u_1\ u_2\ u_3 \cdots$ is *consistent* with a strategy σ if for all prefixes $u_1 \cdots u_n$ of the play for which $\sigma(u_1\ \cdots u_n)$ is defined, $u_{n+1} = \sigma(u_1 \cdots u_n)$. The set of all plays, consistent with some strategy σ, and starting in v is denoted $\Pi(\sigma, v)$. Some strategy σ is winning for player $\bigcirc$ from vertex v iff all plays consistent with σ are won by player $\bigcirc$. Vertex v is won by player $\bigcirc$ whenever she has a winning strategy for all plays starting in vertex v. Parity games are *determined* [3], meaning that every vertex is won by one of the players; they are even *positionally determined*, meaning that if $\bigcirc$ wins a vertex then she has a winning *positional strategy*: a strategy that determines where to move the token next based solely on the vertex on which the token currently resides. Such strategies can be represented by a function $\sigma{:}V_\bigcirc \to V$, and the set of all such strategies for player $\bigcirc$ is denoted $\mathbb{S}_\bigcirc$. *Solving* a parity game G essentially means computing the partition $(\mathsf{Win}_\Diamond(G), \mathsf{Win}_\Box(G))$ of V into vertices won by player $\Diamond$ and player $\Box$, respectively.

Example 1 In the parity game depicted in Figure 1, vertices v_1, v_2 and v_3 are won by player $\Diamond$ while vertices v_4, v_5 and v_6 are won by player $\Box$. A winning positional strategy for player $\Diamond$ is to play from v_2 to v_1 and from v_3 to v_2. A winning strategy for $\Box$ is to move from v_4 to v_6 and from v_5 to v_6.

Attractors and Dominions An $\bigcirc$-*attractor* into a set $U \subseteq V$ contains all those vertices from which player $\bigcirc$ can force any play to U; it is formally defined as follows.

Definition 2 The $\bigcirc$-*attractor* into a set $U \subseteq V$, denoted $\bigcirc$-*Attr*(U), is the least set $A \subseteq V$ satisfying:

1. $U \subseteq A$
2. (a) if $w \in V_\bigcirc$ and $w^\bullet \cap A \neq \emptyset$, then $w \in A$
   (b) if $w \in V_{\bar{\bigcirc}}$ and $w^\bullet \subseteq A$, then $w \in A$

Observe that the complement of an attractor set of any subset of a parity game induces a parity game, *i.e.* $G \setminus \bigcirc$-*Attr*(U) for any U and $\bigcirc$ is a well-defined parity game. Whenever we refer to an *attractor strategy* associated with $\bigcirc$-*Attr*(U), we mean the positional strategy that player $\bigcirc$ can employ to force play to U; such a strategy can be computed in $\mathcal{O}(|V| + |E|)$ using a straightforward fixpoint iteration.

A set of vertices U is an $\bigcirc$-dominion whenever there is a strategy σ for player $\bigcirc$ such that every play starting in U and conforming to σ is winning for $\bigcirc$ and stays within U. A game is a *paradise* for player $\bigcirc$ if the entire game is an $\bigcirc$-dominion.

Example 2 Reconsider the parity game of Figure 1. We have $\Diamond$-*Attr*$(v_2) = \{v_2, v_3\}$ and $\Box$-*Attr*$(v_4) = \{v_4, v_5, v_6\}$. The winning region $\{v_1, v_2, v_3\}$ is an $\Diamond$-dominion, but the subset $\{v_2, v_3\}$ is not; the set $\{v_4, v_6\}$ is an $\Box$-dominion.

3 Jurdziński's Small Progress Measures Algorithm

The SPM algorithm works by computing a *measure* associated with each vertex. This measure is such that it decreases along the play with each "bad" odd priority encountered, and only increases upon reaching a beneficial even priority. In what follows, we recapitulate the essentials for defining and studying the SPM algorithm; our presentation is—as in the original paper by Jurdziński—from the perspective of player $\Diamond$.

Let $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$ be a parity game. Let d be a positive number and let $\alpha \in \mathbb{N}^d$ be a *d-tuple* of natural numbers. We number its components from 0 to $d-1$, *i.e.* $\alpha = (\alpha_0, \alpha_1, \dots, \alpha_{d-1})$, and let $<$ on such d-tuples be given by the lexicographic ordering. We define a derived ordering $<_i$ on k-tuples and l-tuples of natural numbers as follows:

$$(\alpha_0, \alpha_1, \dots, \alpha_k) <_i (\beta_0, \beta_1, \dots, \beta_l) \text{ iff } (\alpha_0, \alpha_1, \dots, \alpha_i) < (\beta_0, \beta_1, \dots, \beta_i)$$

where, if $i > k$ or $i > l$, the tuples are suffixed with 0s. Analogously, we write $\alpha =_i \beta$ to denote that α and β are identical up-to and including position i. The derived ordering provides enough information to reason about the interesting bits of plays: when encountering a priority i in a play, we are only interested in how often we can or must visit vertices of a more significant (*i.e.* smaller) priority than i, whereas we no longer care about the less significant priorities that we shall encounter; a more precise interpretation of the information that will be encoded will be given in Section 4.

Now, assume from hereon that $d-1$ is the largest priority occurring in G; *i.e.*, d is one larger than the largest priority in G. For $i \in \mathbb{N}$, let $n_i = |V_i|$. Define $\mathbb{M}^\Diamond \subseteq \mathbb{N}^d \cup \{\top\}$, as the largest set containing $\top$ ($\top \notin \mathbb{N}^d$) and only those d-tuples with 0 on *even* positions and natural numbers $\leq n_i$ on *odd* positions i.

The lexicographical ordering $<$ and the family of orderings $<_i$ on d-tuples is extended to an ordering on $\mathbb{M}^\Diamond$ by setting $\alpha < \top$ and $\top = \top$. Let $\rho{:}V \to \mathbb{M}^\Diamond$ and suppose $w \in v^\bullet$. Then $\mathsf{Prog}(\rho, v, w)$ is the least $m \in \mathbb{M}^\Diamond$, such that

- $m \geq_{\mathscr{P}(v)} \rho(w)$ if $\mathscr{P}(v)$ is even,
- $m >_{\mathscr{P}(v)} \rho(w)$, or $m = \rho(w) = \top$ if $\mathscr{P}(v)$ is odd.

Definition 3 Function ρ is a *game parity progress measure* if for all $v \in V$:

- if $v \in V_\Diamond$, then for some $w \in v^\bullet$, $\rho(v) \geq_{\mathscr{P}(v)} \mathsf{Prog}(\rho, v, w)$
- if $v \in V_\Box$, then for all $w \in v^\bullet$, $\rho(v) \geq_{\mathscr{P}(v)} \mathsf{Prog}(\rho, v, w)$

The following proposition is due to Jurdziński [11]; it shows that the least game parity progress measure characterises the winning regions of a parity game.

Proposition 1 If ρ is the *least* game parity progress measure for G, then for all $v \in V$: $\rho(v) \neq \top$ iff $v \in \mathsf{Win}_\Diamond(G)$.

The least game parity progress measure can be described as the least fixpoint of a monotone transformer on the complete lattice we define next. Let $\rho, \rho'{:}V \to \mathbb{M}^\Diamond$ and define $\rho \sqsubseteq \rho'$ as $\rho(v) \leq \rho'(v)$ for all $v \in V$. We write $\rho \sqsubset \rho'$ if $\rho \sqsubseteq \rho'$ and $\rho \neq \rho'$. Then the set of all functions $V \to \mathbb{M}^\Diamond$ with $\sqsubseteq$ is a complete lattice. The monotone transformer defined on this set is given as follows:

$$\mathsf{Lift}(\rho, v) = \begin{cases} \rho[v \mapsto \max\{\rho(v), \min\{\mathsf{Prog}(\rho, v, w) \mid v \to w\}\}] & \text{if } v \in V_\Diamond \\ \rho[v \mapsto \max\{\rho(v), \max\{\mathsf{Prog}(\rho, v, w) \mid v \to w\}\}] & \text{if } v \in V_\Box \end{cases}$$

As a consequence of Tarski's fixpoint theorem, we know the least fixpoint of the above monotone transformer exists and can be computed using a standard fixpoint iteration scheme. This leads to the original SPM algorithm, see Algorithm 1. Upon termination of the iteration within the SPM algorithm, the computed game parity progress measure ρ is used to compute the sets $\mathsf{Win}_\Diamond(G)$ and $\mathsf{Win}_\Box(G)$ of vertices won by player $\Diamond$ and $\Box$, respectively.

Theorem 1 Algorithm 1 solves a parity game in $O(dm \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$, see [11].

Algorithm 1 The original Small Progress Measures Algorithm

```
1: function SPM($G$)
2:     Input $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$
3:     Output Winning partition $(Win_\Diamond(G), Win_\Box(G))$
4:     $\rho \leftarrow \lambda v \in V.\ (0, \ldots, 0)$
5:     while $\rho \sqsubset \mathsf{Lift}(\rho, v)$ for some $v \in V$ do
6:         $\rho \leftarrow \mathsf{Lift}(\rho, v)$ for some $v \in V$ such that $\rho \sqsubset \mathsf{Lift}(\rho, v)$
7:     end while
8:     return $(\{v \in V \mid \rho(v) \neq \top\}, \{v \in V \mid \rho(v) = \top\})$
9: end function
```

The runtime complexity of SPM is obtained by considering the more optimal runtime of solving a game G, or G's 'dual'; the latter is obtained by shifting all priorities by one and swapping ownership of all vertices (alternatively, a 'dual' algorithm can be used, computing with a domain $\mathbb{M}^\Box$). The runtime complexity for computing winning strategies for both players using the SPM algorithm is worse than the runtime complexity of solving the game. A winning strategy $\sigma_\Diamond : V_\Diamond \to V$ for player $\Diamond$ can be extracted from ρ by setting $\sigma_\Diamond(v) = w$ for $v \in V_\Diamond \cap \mathsf{Win}_\Diamond(G)$ and $w \in v^\bullet$ such that $\rho(w) \leq \rho(w')$ for all $w' \in v^\bullet$. A winning strategy for player $\Box$ cannot be extracted from ρ *a posteriori*, so, as also observed in [15], the runtime complexity of computing a winning strategy cannot be improved by considering the dual of a game (contrary to what is stated in [11]).

Theorem 2 [See also [15]] Algorithm 1 can compute winning strategies for both players in $O(dm \cdot (n/\lceil d/2 \rceil)^{\lceil d/2 \rceil})$.

As an illustration of the above observations, consider the family of games depicted in Figure 2. The more optimal runtime of $O(dm \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$ will be achieved by solving the games themselves (using $\mathbb{M}^\Diamond$) and not their dual. As all games in the family are $\Box$-paradises, we cannot extract a winning strategy for player $\Box$ from the computed progress measure and the only option we have is to solve the dual games with the less favourable runtime of $O(dm \cdot (n/\lceil d/2 \rceil)^{\lceil d/2 \rceil})$. In fact, all instances of the family of games depicted in Figure 2 are solved exponentially faster than their dual, underlining the potential practical implications of re-running the algorithm.

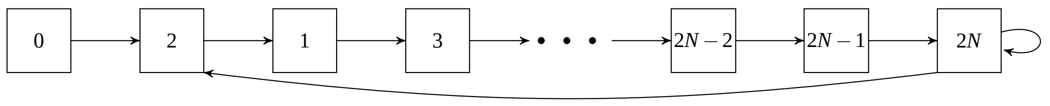

Figure 2: A parity game won by player $\Box$. Solving the game using $\mathbb{M}^\Diamond$, the first $\top$ value is reached after the first full pass of the cycle containing priority 1 ($O(N^2)$ using a fair lifting strategy), and it will then propagate through the game. Solving the dual game, or using $\mathbb{M}^\Box$ takes exponential time to lift the node with priority $2N$.

To facilitate the analysis of SPM, we will use the following terms and notions. The term *measure* refers to the intermediate values of ρ in the lifting process as well. Given a tuple $m \in \mathbb{M}^\Diamond$, we say that the position i in m is *saturated*, if $(m)_i = |V_i|$.

4 An operational interpretation of progress measures

While, from a technical perspective, SPM is a relatively simple algorithm in the sense that it is concise and its individual steps are elementary operations, it lacks a clear and appealing explanation of the devices used. It is therefore difficult to understand, and possibly enhance. Apart from the formal definition of progress measures, little explanation of what is hidden behind the values in tuples is offered. Notable exceptions are [13], which explains that for $\Box$-solitaire games with only even simple cycles, one can use the maximal degrees of 'odd stretches' (a concept we make precise below) in order to define a parity progress measure, and Schewe's bigstep paper [15], where it is shown that dominions of a bounded size can be detected using measures with a restricted codomain. Klauck's construction for a specific parity progress measure breaks down for arbitrary parity games and the constructed parity progress measure is not related to the measure that is computed by the SPM algorithm, nor to any of the intermediate measures. In general, the high-level intuition is that the larger progress measure values indicate more capabilities of player $\Box$, and a value at a given position in the tuple is somehow related to the number of priorities that $\Box$ can enforce to visit.

In what follows, we present a precise and operational interpretation of measures. Our interpretation is similar in spirit to the one used in [13], but applicable to all parity games, and uses a concept known from the realm of strategy improvement algorithms – a value (or profile) of a play. Here, values are defined in terms of maximal odd-dominated stretches occurring in a prefix of a play. With this notion at hand, we can consider an optimal valuation of vertices, being the best lower bound on play values that player $\Diamond$ can enforce, or the worst upper bound that $\Box$ can achieve, *i.e.* it is an *equilibrium*. The optimal valuations range over the same codomain as progress measures, and the main result of this section states that the least game parity progress measure is equal to the optimal valuation.

Let $\mathbb{M}^{\Diamond}_{ext}$ denote all tuples in $\mathbb{N}^d \cup \{\top\}$ such that for all $m \in \mathbb{M}^{\Diamond}_{ext}$ and even positions $i \leq d$, $(m)_i = 0$; *i.e.* compared to $\mathbb{M}^{\Diamond}$, we drop the requirement that values on odd positions i are bounded by $|V_i|$. Elements in $\mathbb{M}^{\Diamond}_{ext}$ are ordered in the same fashion as those in $\mathbb{M}^{\Diamond}$. Given a play π, a *stretch* is a subsequence $\pi_s\pi_{s+1}\dots\pi_{s+l}$ of π. For a priority k, a *k-dominated stretch* is a stretch in which the minimal priority among all vertices in the stretch is k. The *degree* of a k-dominated stretch is the number of vertices with priority k occurring in the stretch.

Definition 4 Let us denote all plays in the parity game by Π. An $\Diamond-$*value* (or simply value) of a play is a function $\theta_\Diamond : \Pi \longrightarrow \mathbb{M}^{\Diamond}_{ext}$ defined as follows:

- if π is winning for $\Box$, then $\theta_\Diamond(\pi) = \top$
- if π is winning for $\Diamond$, then $\theta_\Diamond(\pi) = m$, where $m \neq \top$, and for every odd position i, $(m)_i$ is the *degree* of the maximal i-dominated stretch that is a prefix of π

Observe that a play value is well-defined, as an infinite i-dominated stretch for an odd i implies that a game is won by $\Box$, and its value is $\top$ in such case.

Example 3 Suppose that the sequence of priorities corresponding to a certain play π is $453453213(47)^*$. Then $\theta_\Diamond(\pi) = (0,1,0,2,0,0,0,0)$.

The theorem below links the progress measure values to players' capabilities to enforce beneficial plays or avoid harmful ones, where the benefit from a play is measured by a specially devised play value, as it is done in strategy improvement algorithms. This offers a more operational view on progress measure values, which, combined with a more fine-grained analysis of the mechanics of SPM allows us to extract winning strategies for both players in the next section.

Theorem 3 If $\overline{\rho}$ is the least progress measure of a parity game G, then, for all v:

1. there is a strategy $\sigma_\Box \in \mathbb{S}^*_\Box$ such that for every $\pi \in \Pi(\sigma_\Box, v)$, $\theta_\Diamond(\pi) \geq \overline{\rho}(v)$
2. there is a strategy $\sigma_\Diamond \in \mathbb{S}_\Diamond$ such that for every $\pi \in \Pi(\sigma_\Diamond, v)$, $\theta_\Diamond(\pi) \leq \overline{\rho}(v)$

A notable difference between strategy improvement algorithms and SPM is that SPM does not work with explicit strategies, and the intermediate measure values do not represent any proper valuation induced by strategies – only the final least progress measure does. Still, these intermediate values give an underapproximation of the capabilities of player $\Box$ in terms of odd-dominated stretches that she can enforce.

Note that a consequence of Theorem 3 is that the least (resp. greatest) play values that player $\Box$ (resp. $\Diamond$) can enforce are equal, and coincide with the least game parity progress measure $\overline{\rho}$ computed by SPM. Observe also that player $\Diamond$ can always achieve the strategy guaranteeing the optimal even-biased play value with a memoryless strategy, whereas player $\Box$ may require to that end a strategy that depends on a play's history.

5 Strategy construction for player $\Box$

Computing winning strategies is typically part of a practical solution to a complex verification or a controller synthesis problem. In such use cases, these strategies are employed to construct witnesses and counterexamples for the verification problems, or for constructing control strategies for the controller [1]. As we explained in Section 3, the SPM algorithm can easily be extended to construct a winning strategy for player $\Diamond$. The problem of deriving a winning strategy for player $\Box$ in SPM (other than by running the algorithm on the 'dual' game, or by using a 'dual' domain $\mathbb{M}^\Box$) has, however, never been addressed. Note that the problem of computing strategies is at least as hard as solving a game. Indeed, even if we are equipped with a valid winning partition $(\mathsf{Win}_\Diamond(G), \mathsf{Win}_\Box(G))$ for a game G, then deriving the strategies witnessing these partitions involves the same computational overhead as the one required to compute $(\mathsf{Win}_\Diamond(G), \mathsf{Win}_\Box(G))$ in the first place.

Proposition 2 The problem of finding the winning partition $(\mathsf{Win}_\Diamond(G), \mathsf{Win}_\Box(G))$ of a given game G can be reduced in polynomial time to the problem of computing a winning strategy for player $\bigcirc$ in a given $\bigcirc$-dominion.

Deriving a strategy for both players by using the SPM to compute $\mathbb{M}^\Diamond$ measures and $\mathbb{M}^\Box$ measures consecutively, or even simultaneously, affects, as we already discussed in Section 3, SPM's runtime. Being able to compute $\Box$ strategies without resorting to the aforementioned methods would also allow us to potentially significantly improve efficiency on such instances as given by Figure 2. It may be clear, though, that extracting a winning strategy from the small progress measures algorithm for vertices with measure $\top$ will require modifying the algorithm (storing additional data, augmenting the lifting process). For instance, simply following the vertex that caused the update to top, fails, as the example below shows.

Example 4 Reconsider the game depicted in Figure 1. Recall that vertices v_4, v_5 and v_6 are won by player $\Box$, and in all possible lifting schemes, the first vertex whose measure becomes $\top$ is v_6. After that, a possible sequence of liftings is that first $\rho(v_5)$ is set to $\top$ (due to v_6), followed by $\rho(v_4) = \top$ (due to v_5). If we set the strategy based on the vertex that caused the given vertex to be lifted to top, we obtain $\sigma(v_4) = v_5$, which is not winning for $\Box$.

The key problem is that for vertices won by player $\Box$, from some point onward, the lifting process discards significant information. This is best seen in case of lifting to $\top$ – a partial characterisation of reachable odd priorities contained in a tuple (see also our previous section) is ultimately replaced with a mere indication that player □ can win.

5.1 A Bounded □-Dominion

Consider a game G on which a standard SPM algorithm with an arbitrary lifting policy has been applied. Suppose that at some point a vertex v is the first one to be lifted to $\top$, and after lifting of v the algorithm is suspended, resulting in some temporary measure ρ. Let k be the priority of v.

We will start with two straightforward observations. The first one is that k must be an odd number; this is because a vertex with an even priority obtains, after lifting, a ρ-value equal to the ρ-value of one of its successors, and therefore it cannot be the first vertex lifted to $\top$. Another immediate conclusion is that at least one (or all, if $v \in V_\diamond$) successor(s) of v has (have) a ρ-value saturated up to the k-th position, *i.e.* it is of the form $m = (0, |V_1|, 0, |V_3|, \dots, 0, |V_k|, *\,*\,*)$; were it not the case, then a non-top value m' such that $m' >_k m$ would exist, which would be inconsistent with the definition of Prog.

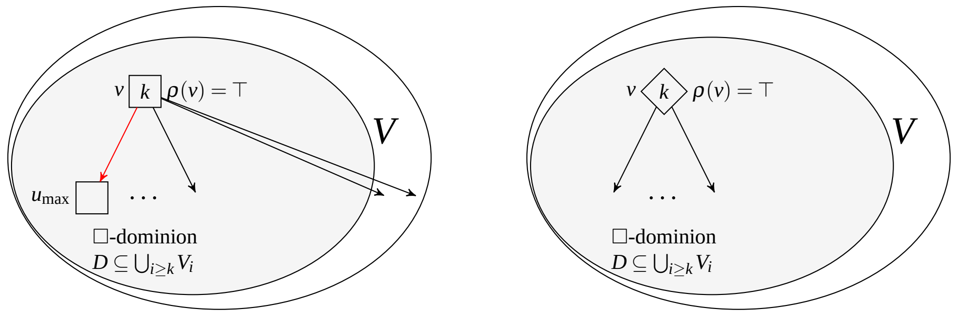

Figure 3: Snapshot of the SPM algorithm after the first vertex v is lifted to top.

There are two more complex properties, which we can utilise to modify the SPM algorithm and compute the winning strategy for player □ (see Figure 3).

1. Vertex v belongs to an □-dominion D within G such that the minimal priority in D is k.
2. If $v \in V_\Box$, then picking the successor u_{max} of v with the maximal current ρ-value among $v^\bullet$ is a part of a (positional) winning strategy for □ that stays within such a dominion D as described above.

The intuition concerning the above facts is as follows. Vertices with a measure value m saturated up to but possibly excluding a certain position i have a neat interpretation of the measure value at position i:

Player □ can force the following outcome of a play:

1. *priority i appears m_i times without any lower priority in between*
2. *the play will reach a $\top$-labelled vertex via priorities not more significant than i*
3. *the play enters a cycle with an odd dominating priority larger (less significant) than i.*

Therefore, in the situation as described above, □ can force a play starting at v to first go in one step to the successor u_{max} of v with a measure of the form $(0, |V_1|, 0, |V_3|, \dots, 0, |V_k|, *\,*\,*)$, and then to play further and either force a less significant odd-dominated cycle (cases 2 and 3, since v is the only $\top$-labelled vertex), or to visit vertices with priority k $|V_k|$ times without any lower priority in between. But in the latter case, since v has priority k, we have in fact $|V_k| + 1$ vertices with priority k not "cancelled" by a lower priority. Hence player □ has forced an odd-dominated cycle with the lowest (most significant) priority k. Note that this does not imply we can simply construct a winning strategy for $\Box$ by always picking a successor with the maximal measure to further vertices that can be visited along the play; such a method may lead to an erroneous strategy, as illustrated by Figure 4.

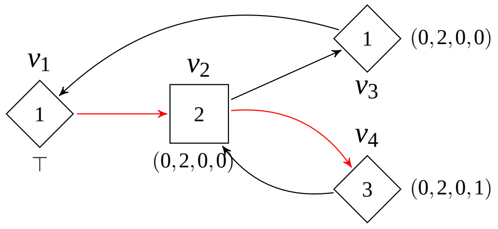

Figure 4: A game, won entirely by player $\Box$, and demonstrating that a strategy defined by a greedy choice of vertex with the maximal tuple does not work. After lifting the vertices in order v_1, v_3, v_2, v_4, v_1, we obtain the measure values as above. Player $\Box$ would then choose v_3, which leads to a losing play, whereas the choice of the other successor (v_4) yields a winning play for $\Box$.

Propagating a top value only to vertices with less significant priorities is, however, safe. This can be achieved efficiently by a slightly modified attractor that works within a given context of vertices W, which we call a *guarded* attractor.

Definition 5 Let k be some priority and U, W some sets for which $U \subseteq W \cap V_{\geq k}$. Then $\Box$-$Attr_W^{\geq k}(U)$ is the least set A satisfying:

1. $U \subseteq A \subseteq W \cap V_{\geq k}$
2. (a) if $u \in V_\Box$ and $u^\bullet \cap A \neq \emptyset$, then $u \in A$
   (b) if $u \in V_\Diamond$ and $u^\bullet \cap W \subseteq A$, then $u \in A$

If $W = V$, we drop the subscript W from the guarded attractor.

The theorem below forms the basis of our algorithm; it describes the relevant information about an $\Box$-dominion that can be extracted once the first vertex in the game is lifted to top.

Theorem 4 Let G be a parity game on which an arbitrary lifting sequence is applied, such that at some point a vertex v with $\mathscr{P}(v) = k$ is the first vertex whose measure value becomes top. Let ρ be the temporary measure at that point. The following holds:

- if $v \in V_\Box$, then for every successor u of v with a maximal measure among $v^\bullet$ there is an $\Box$-dominion D_u containing $\Box$-$Attr^{\geq k}(\{v\})$ such that for all $w \in D_u$, $\mathscr{P}(w) \geq k$. Moreover, $\Box$ has a winning strategy that is closed on D_u, and which is defined on v as $\sigma(v) = u$, and on $\Box$-$Attr^{\geq k}(\{v\}) \setminus \{v\}$ as the strategy attracting towards v,
- if $v \in V_\Diamond$, then there is an $\Box$-dominion D containing $\Box$-$Attr^{\geq k}(\{v\})$ such that for all $w \in D$, $\mathscr{P}(w) \geq k$. Moreover, $\Box$ has a winning strategy σ that is closed on D, and defined on $\Box$-$Attr^{\geq k}(\{v\}) \setminus \{v\}$ as the strategy attracting towards v. Note that in this case $v^\bullet \subseteq D$.

5.2 The Extended SPM Algorithm

Theorem 4 captures the core idea behind our algorithm. It provides us with the means to locally resolve (*i.e.* define a local strategy for) at least one vertex in $\mathsf{Win}_\Box(G)$, once a top value is found while lifting. Moreover, it indicates in which part of the game the remainder of the $\Box$-dominion resides, implying that

Algorithm 2 Modified SPM with strategy derivation for player Odd

```
 1: function SOLVE($G$)
 2:     Input $G = (V, E, \mathscr{P}, (V_\Diamond, V_\Box))$
 3:     Output Winning partition and strategies $((Win_\Diamond(G), \sigma'), (Win_\Box(G), \sigma))$
 4:     initialise $\sigma$ to an empty assignment
 5:     $\rho \leftarrow \lambda w \in V.\ (0, \ldots, 0)$
 6:     SPM-WITHIN($V$)
 7:     compute strategy $\sigma'$ for player Even by picking min. successor w.r.t. $\rho$
 8:     return $((\{v \in V \mid \rho(v) \neq \top\}, \sigma'), (\{v \in V \mid \rho(v) = \top, \sigma))$
 9:
10:     procedure SPM-WITHIN($W$)
11:         while $(W \neq \emptyset)$ do
12:             while $\rho \sqsubset \mathsf{Lift}(\rho, w)$ for some $w \in W$ and for all $w \in W{:}\rho(w) \neq \top$ do
13:                 $\rho \leftarrow \mathsf{Lift}(\rho, w)$ for $w \in W$ such that $\rho \sqsubset \mathsf{Lift}(\rho, w)$
14:             end while
15:             if for all $w \in W$: $\rho(w) \neq \top$ break end if
16:             $v \leftarrow$ the (unique) vertex in $W$ such that $\rho(v) = \top$
17:             $k \leftarrow \mathscr{P}(v)$
18:             $\sigma(v) \leftarrow u$ where $u \in v^\bullet \cap W$ for which $\rho(u') \leq_k \rho(u)$ for all $u' \in v^\bullet \cap W$
19:             $\mathsf{RES} \leftarrow \Box\text{-}Attr_W^{\geq k}(\{v\})$
20:             for all $w \in \mathsf{RES} \setminus \{v\}$ do
21:                 $\rho(w) \leftarrow \top$
22:                 if $w \in V_\Box$ then assign $\sigma(w)$ the strategy attracting to $v$ end if
23:             end for
24:             $\mathsf{DOM} \leftarrow \mathsf{RES}$
25:             $\mathsf{IRR} \leftarrow \Diamond\text{-}Attr_W(\{w \in W \mid \mathscr{P}(w) < k\})$
26:             $\mathsf{REM} \leftarrow W \setminus (\mathsf{RES} \cup \mathsf{IRR})$
27:             SPM-WITHIN($\mathsf{REM}$)
28:             $\mathsf{DOM} \leftarrow \mathsf{DOM} \cup \{w \in \mathsf{REM} \mid \rho(w) = \top\}$
29:             $A \leftarrow \Box\text{-}Attr_W(\mathsf{DOM})$
30:             for all $w \in A \setminus \mathsf{DOM}$ do
31:                 $\rho(w) \leftarrow \top$
32:                 if $w \in V_\Box$ then assign $\sigma(w)$ to be the strategy attracting to $\mathsf{DOM}$
33:                 end if
34:             end for
35:             $W \leftarrow W \setminus A$
36:         end while
37:     end procedure
38: end function
```

one can temporarily restrict the lifting to that area until the dominion is fully resolved. We will give a description of our solution (Algorithm 2), and informally argue the correctness of our approach. For a (intricate and rather involved) formal proof, we refer to [9].

The algorithm proceeds as follows. First, a standard SPM is run until the first vertex reaches top [l. 12–14 in Alg. 2]. Whenever v is the first vertex lifted to top, then the issue of the winning strategy for

v can be resolved immediately [l. 18], as well as for vertices in the guarded attractor of v (if there are any). We will denote this set of 'resolved' vertices with RES. Moreover, we can restrict our search for the remainder of the $\Box$-dominion D only to vertices with priorities not more significant than k, in fact only those from which player $\Diamond$ cannot attract a play to visit a priority more significant than k. Hence we can remove from the current computation context the set $\mathsf{IRR} = \Diamond\text{-}Attr(\{w \in W \mid \mathscr{P}(w) < k\})$, vertices that may be considered at the moment irrelevant [l. 25–26].

After discarding the resolved and currently irrelevant vertices, the algorithm proceeds in the remaining set of vertices that constitutes a proper subgame (i.e. without dead ends) induced by the set REM. After the subroutine returns [l. 27], all vertices labelled with top are won by player $\Box$ in the subgame $G \cap \mathsf{REM}$. In other words, those vertices are won by $\Box$ provided that the play does not leave REM. Since the only way for player $\Diamond$ to escape from REM is to visit RES, where every vertex is won by player $\Box$, the top-labelled vertices from REM are in fact won by $\Box$ in the context of the larger game $G \cap W$. Therefore the set DOM computed in line 28 constitutes an $\Box$-dominion within the game $G \cap W$, in which we have moreover fully defined a winning strategy σ for player $\Box$. Finally, every vertex from $V \setminus \mathsf{DOM}$ that can be attracted by player $\Box$ to the dominion DOM is certainly won by $\Box$, and for those vertices we assign the standard strategy attracting to DOM. The $\Box$-dominion A is then removed, and the computation continues in the remaining subgame.

The algorithm may at first sight appear to deviate much from the standard SPM algorithm. However, the additional overlay, apart from defining the strategy, is no more than a special lifting policy that temporarily restricts the lifting to parts where an $\Box$ dominion resides.

Every attractor computation takes $O(n+m)$ time, and whenever it occurs, at least one new vertex is 'resolved'. Hence the total extra time introduced by the attractor computations is bounded by $O(n(n+m))$. As with the standard SPM, the lifting operations dominate the running time, and their total number for every vertex is bounded by the size of $\mathbb{M}^{\Diamond}$.

Theorem 5 For a game G with n vertices, m edges, and d priorities, SOLVE solves G and computes winning strategies for player $\Diamond$ and $\Box$ in worst-case $O(dm \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor})$.

6 Illustrating the new algorithm

We illustrate the various aspects of Algorithm 2 on the game G depicted in Figure 5, with two (overlapping) subgames G_1 and G_2. Note that the entire game is an $\Box$-paradise: every vertex eventually is assigned measure $\top$ by Algorithm 2 (and Algorithm 1, for that matter). Suppose we use a lifting strategy prioritising v_2, v_3, v_7 and v_8; then vertex v_3's measure is the first to reach $\top$, and the successor with maximal measure is v_7. Therefore, $\Box$'s strategy is to move from v_3 to v_7. The set RES, computed next consists of vertices v_3 and v_2; the strategy for v_2 is set to v_3 and its measure is set to $\top$. The $\Diamond$-attractor into those vertices with priorities ≥ 3, *i.e.*, vertices v_1 and v_4, is exactly those vertices, so, next, the algorithm zooms in on solving the subgame G_1.

Suppose that within the latter subgame, we prioritise the lifting of vertex v_7 and v_8; then vertex v_7's measure is set to $\top$ first, and v_7's successor with the largest measure is v_8; therefore $\Box$'s strategy is to move from v_7 to v_8. At this point in the algorithm, RES is assigned the set of vertices v_7 and v_8, and the measure of v_8 is set to $\top$. Note that in this case, in this subgame, the winning strategy for $\Box$ on v_7 is to remain within the set RES. Since all remaining vertices have more signficant priorities than v_7, or are forced by $\Diamond$ to move there, the next recursion is run on an empty subgame and immediately returns without changing the measures. Upon return, the $\Box$-attractor to all $\Box$-won vertices (within the

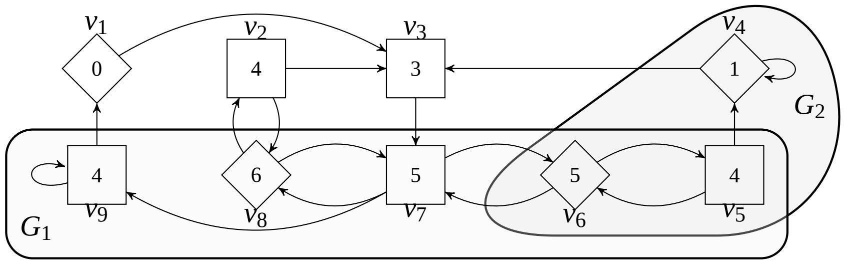

Figure 5: An example game G with two (overlapping) subgames G_1 and G_2.

subgame G_1, so these are only the vertices v_7 and v_8) is computed, and the algorithm continues solving the remaining subgame (*i.e.* the game restricted to vertices v_5, v_6 and v_9), concluding that no vertex in this entire game will be assigned measure $\top$.

At this point, the algorithm returns to the global game again and computes the $\Box$-attractor to the vertices won by player $\Box$ at that stage (*i.e.* vertices v_2, v_3, v_7 and v_8), adding vertices v_1 and v_9, setting their measure to $\top$ and setting $\Box$'s strategy for v_9 to move to v_1.

As a final step, the algorithm next homes in on the subgame G_2, again within the larger game. The only vertex assigned measure $\top$ in the above subgame is vertex v_4; at this point RES is assigned all vertices in G_2, the measure of v_5 and v_6 is set to $\top$ and the $\Box$ strategy for vertex v_5 is set to v_4. This effectively solves the entire game.

7 Conclusions and Future Work

In this paper, we studied the classical Small Progress Measures algorithm for solving parity games. The two key contributions of our work are as follows:

1. We have proposed a more operational interpretation of progress measures by characterising the types of plays that players can enforce.

2. We have provided a modification of the SPM algorithm that allows to compute the winning strategies for both players in one pass, thus improving the worst-case running time of strategy derivation.

The second enhancement has been made possible due to a thorough study of the contents of progress measures, and their underapproximations in the intermediate stages of the algorithm (building on the proposed operational interpretation).

As for the future work, we would like to perform an analysis of SPM's behaviour on special classes of games, along the same lines as we have done in case of the recursive algorithm [8]. More specifically, we would like to identify the games for which SPM runs in polynomial time, and study enhancements that allow to solve more types of games efficiently. It would also be interesting to see whether the ideas behind our modification of the SPM algorithm carry over to the algorithm for small energy progress measures [2] for mean payoff games.

References

[1] A. Arnold, A. Vincent & I. Walukiewicz (2003): *Games for synthesis of controllers with partial observation*. *TCS* 303(1), pp. 7–34, doi:10.1016/S0304-3975(02)00442-5.

[2] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini & J.-F. Raskin (2011): *Faster algorithms for mean-payoff games*. *Formal Methods in System Design* 38(2), pp. 97–118, doi:10.1007/s10703-010-0105-x.

[3] E.A. Emerson & C.S. Jutla (1991): *Tree automata, Mu-Calculus and determinacy*. In: *FOCS'91*, IEEE Computer Society, Washington, DC, USA, pp. 368–377, doi:10.1109/SFCS.1991.185392.

[4] E.A. Emerson, C.S. Jutla & A.P. Sistla (1993): *On Model-Checking for Fragments of μ-Calculus*. In: *CAV*, *Lecture Notes in Computer Science* 697, Springer, pp. 385–396, doi:10.1007/3-540-56922-7_32.

[5] J. Fearnley (2010): *Non-oblivious Strategy Improvement*. In: *LPAR-16*, *Lecture Notes in Computer Science* 6355, Springer, pp. 212–230, doi:10.1007/978-3-642-17511-4_13.

[6] O. Friedmann (2011): *Recursive algorithm for parity games requires exponential time*. *RAIRO – Theor. Inf. and Applic.* 45(4), pp. 449–457, doi:10.1051/ita/2011124.

[7] O. Friedmann & M. Lange (2009): *Solving Parity Games in Practice*. In: *ATVA*, *Lecture Notes in Computer Science* 5799, Springer, pp. 182–196, doi:10.1007/978-3-642-04761-9_15.

[8] M. Gazda & T.A.C. Willemse (2013): *Zielonka's Recursive Algorithm: dull, weak and solitaire games and tighter bounds*. In: *GandALF*, *EPTCS* 119, pp. 7–20, doi:10.4204/EPTCS.119.4.

[9] M. Gazda & T.A.C. Willemse (2014): *Strategy Derivation for Small Progress Measures*. `http://arxiv.org/abs/1407.2149`.

[10] E. Grädel, W. Thomas & T. Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research*. *Lecture Notes in Computer Science* 2500, Springer.

[11] M. Jurdziński (2000): *Small Progress Measures for Solving Parity Games*. In: *STACS'00*, *Lecture Notes in Computer Science* 1770, Springer, pp. 290–301, doi:10.1007/3-540-46541-3_24.

[12] M. Jurdziński, M. Paterson & U. Zwick (2006): *A Deterministic Subexponential Algorithm for Solving Parity Games*. In: *SODA'06*, ACM/SIAM, pp. 117–123, doi:10.1145/1109557.1109571.

[13] H. Klauck (2001): *Algorithms for Parity Games*. In: *Automata, Logics, and Infinite Games: A Guide to Current Research*, chapter 7, *Lecture Notes in Computer Science* 2500, Springer, pp. 107–129, doi:10.1007/3-540-36387-4_7.

[14] R. McNaughton (1993): *Infinite games played on finite graphs*. *APAL* 65(2), pp. 149–184, doi:10.1016/0168-0072(93)90036-D.

[15] S. Schewe (2007): *Solving Parity Games in Big Steps*. In: *FSTTCS'07*, *Lecture Notes in Computer Science* 4855, Springer, pp. 449–460, doi:10.1007/978-3-540-77050-3_37.

[16] S. Schewe (2008): *An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games*. In: *CSL*, *Lecture Notes in Computer Science* 5213, Springer, pp. 369–384, doi:10.1007/978-3-540-87531-4_27.

[17] S. Schewe, A. Trivedi & T. Varghese (2015): *Symmetric Strategy Improvement*. In: *ICALP*, *Lecture Notes in Computer Science* 9135, Springer, pp. 388–400, doi:10.1007/978-3-662-47666-6_31.

[18] P. Stevens & C. Stirling (1998): *Practical Model Checking Using Games*. In: *TACAS'98*, *Lecture Notes in Computer Science* 1384, Springer, pp. 85–101, doi:10.1007/BFb0054166.

[19] J. Vöge & M. Jurdziński (2000): *A Discrete Strategy Improvement Algorithm for Solving Parity Games*. In: *CAV*, *Lecture Notes in Computer Science* 1855, Springer, pp. 202–215, doi:10.1007/10722167_18.

[20] W. Zielonka (1998): *Infinite games on finitely coloured graphs with applications to automata on infinite trees*. *TCS* 200(1-2), pp. 135 – 183, doi:10.1016/S0304-3975(98)00009-7.