Symmetric Strategy Improvement

Sven Schewe¹, Ashutosh Trivedi², and Thomas Varghese¹

¹Department of Computer Science, University of Liverpool
²Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Abstract

Symmetry is inherent in the definition of most of the two-player zero-sum games, including parity, mean-payoff, and discounted-payoff games. It is therefore quite surprising that no symmetric analysis techniques for these games exist. We develop a novel symmetric strategy improvement algorithm where, in each iteration, the strategies of both players are improved simultaneously. We show that symmetric strategy improvement defies Friedmann’s traps, which shook the belief in the potential of classic strategy improvement to be polynomial.

1 Introduction

We study turn-based graph games between two players—Player Min and Player Max—who take turns to move a token along the vertices of a coloured finite graph so as to optimise their adversarial objectives. Various classes of graph games are characterised by the objective of the players, for instance in parity games the objective is to optimise the parity of the dominating colour occurring infinitely often, while in discounted and mean-payoff games the objective is the discounted and limit-average sum of the colours.

Solving graph games is the central and most expensive step in many model checking [16, 8, 30, 6, 1], satisfiability checking [30, 16, 28, 26], and synthesis [22, 27] algorithms. More efficient algorithms for solving graph games will therefore foster the development of performant model checkers and contribute to bringing synthesis techniques to practice.

Parity games enjoy a special status among graph games and the quest for performant algorithms [16, 9, 7, 20, 32, 4, 31, 15, 19, 23, 29, 3, 21, 17, 2, 14, 24, 25, 10] for solving them has therefore been an active field of research during the last decades. Traditional forward techniques (≈ \(O(n^{\frac{4}{3}c})\) [15] for parity games with \(n\) positions and \(c\) colours), backward techniques (≈ \(O(n^c)\) [20, 9, 31]), and their combination (≈ \(O(n^{\frac{7}{3}}c)\) [24]) provide good complexity bounds. However, these bounds are sharp, and techniques with good complexity bounds [24, 15] frequently display their worst case complexity on practical examples. Strategy improvement algorithms [19, 23, 29, 3, 25, 10], on the other hand, are closely related to the Simplex algorithm for solving linear programming problems that perform well in practice.

Classic strategy improvement algorithms are built around the existence of optimal positional strategies for both players. They start with an arbitrary positional strategy for a player and iteratively compute a better positional strategy in every step until the strategy cannot be further improved. Since there are only finitely many positional strategies in a finite graph, termination is guaranteed. The crucial step in a strategy improvement algorithm is to compute a better strategy from the current strategy. Given a current strategy \(\sigma\) of a player (say, Player Max), this step is performed by first computing the globally optimal counter strategy \(\tau^c_\sigma\) of the opponent (Player Min) and then computing the value of each vertex of the game restricted to the strategies \(\sigma\) and \(\tau^c_\sigma\). For the games under discussion (parity, discounted, and mean-payoff) both of these computations are simple and tractable. This value dictates potentially locally profitable changes or switches \(\text{Prof}(\sigma)\) that Player Max can make vis-à-vis his previous strategy.
For the correctness of the strategy improvement algorithm it is required that such locally profitable changes imply a global improvement. The strategy of Player Max can then be updated according to a switching rule (akin to pivoting rule of the Simplex) in order to give an improved strategy. This has led to the following template for classic strategy improvement algorithms.

**Algorithm 1:** Classic strategy improvement algorithm

1. determine an optimal counter strategy $\tau^c_\sigma$ for $\sigma$
2. evaluate the game for $\sigma$ and $\tau^c_\sigma$ and determine the profitable changes $\text{Prof}(\sigma)$ for $\sigma$
3. update $\sigma$ by applying changes from $\text{Prof}(\sigma)$ to $\sigma$

A number of switching rules, including the ones inspired by Simplex pivoting rules, have been suggested for strategy improvement algorithms. The most widespread ones are to select changes for all game states where this is possible, choosing a combination of those with an optimal update guarantee, or to choose uniformly at random. For some classes of games, it is also possible to select an optimal combination of updates [25]. There have also been suggestions to use more advanced randomisation techniques with sub-exponential $-2^{O(\sqrt{n})}$ -- bounds [3] and snare memory [10]. Unfortunately, all of these techniques have been shown to be exponential in the size of the game [11, 12, 13].

Classic strategy improvement algorithms treat the two players involved quite differently where at each iteration one player computes a globally optimal counter strategy, while the other player performs local updates. In contrast, a symmetric strategy improvement algorithm symmetrically improves the strategies of both players at the same time, and uses the finding to guide the strategy improvement. This suggests the following naïve symmetric approach.

**Algorithm 2:** Naïve symmetric strategy improvement algorithm

1. determine $\tau' = \tau^c_\sigma$
2. update $\sigma$ to $\sigma'$
3. determine $\sigma' = \sigma^c_\tau$
4. update $\tau$ to $\tau'$

This algorithm has earlier been suggested by Condon [5] where it was shown that a repeated application of this update can lead to cycles [5]. A problem with this naïve approach is that there is no guarantee that the primed strategies are generally better than the unprimed ones. With hindsight this is maybe not very surprising, as in particular no improvement in the evaluation of running the game with $\sigma', \tau'$ can be expected over running the game with $\sigma, \tau$, as an improvement for one player is on the expense of the other. This observation led to the approach being abandoned. In this paper we propose the following more careful symmetric strategy improvement algorithm that guarantees improvements in each iteration similar to classic strategy improvement.

**Algorithm 3:** Symmetric strategy improvement algorithm

1. determine $\tau^c_\sigma$
2. determine $\sigma^c_\tau$
3. determine $\text{Prof}(\sigma)$ for $\sigma$
4. determine $\text{Prof}(\tau)$ for $\tau$
5. update $\sigma$ using $\text{Prof}(\sigma) \cap \sigma^c_\tau$
6. update $\tau$ using $\text{Prof}(\tau) \cap \tau^c_\sigma$

The main difference to classic strategy improvement approaches is that we exploit the strategy of the other player to inform the search for a good improvement step. In this algorithm we select only such updates to the two strategies that agree with the optimal counter strategy to the respective other’s strategy. We believe that this will provide a gradually improving advice function that will lead to few iterations. We support this assumption by showing that this algorithm suffices to escape the traps Friedmann has laid to establish lower bounds for different types of strategy improvement algorithms [11, 12, 13].
2 Preliminaries

We focus on turn-based zero-sum games played between two players—Player Max and Player Min—over finite graphs. A game arena $A$ is a tuple $(V_{\text{Max}}, V_{\text{Min}}, E, C, \phi)$ where $(V = V_{\text{Max}} \cup V_{\text{Min}}, E)$ is a finite directed graph with the set of vertices $V$ partitioned into a set $V_{\text{Max}}$ of vertices controlled by Player Max and a set $V_{\text{Min}}$ of vertices controlled by Player Min, $E \subseteq V \times V$ is the set of edges, and $C$ is a set of colours, $\phi : V \rightarrow C$ is the colour mapping. We require that every vertex has at least one outgoing edge.

A turn-based game over $A$ is played between players by moving a token along the edges of the arena. A play of such a game starts by placing a token on some initial vertex $v_0 \in V$. The player controlling this vertex then chooses a successor vertex $v_1$ such that $(v_0, v_1) \in E$ and the token is moved to this successor vertex. In the next turn the player controlling the vertex $v_1$ chooses the successor vertex $v_2$ with $(v_1, v_2) \in E$ and the token is moved accordingly. Both players move the token over the arena in this manner and thus form a play of the game. Formally, a play of a game over $A$ is an infinite sequence of vertices $\langle v_0, v_1, \ldots \rangle \in V^\omega$ such that, for all $i \geq 0$, we have that $(v_i, v_{i+1}) \in E$. We write $\text{Plays}_A(v)$ for the set of plays over $A$ starting from vertex $v \in V$ and $\text{Plays}_A$ for the set of plays of the game. We omit the subscript when the arena is clear from the context. We extend the colour mapping $\phi : V \rightarrow C$ from vertices to plays by defining the mapping $\phi : \text{Plays} \rightarrow C^\omega$ as $\langle v_0, v_1, \ldots \rangle \mapsto (\phi(v_0), \phi(v_1), \ldots)$.

Definition 2.1 (Graph Games). A graph game $G$ is a tuple $(A, \eta, \prec)$ such that $A$ is an arena, $\eta : C^\omega \rightarrow \mathbb{D}$ is an evaluation function where $\mathbb{D}$ is the carrier set of a complete space, and $\prec$ is a preference ordering over $\mathbb{D}$.

Example 2.2. Parity, mean-payoff and discounted payoff games are graph games $(A, \eta, \prec)$ played on game arenas $A = (V_{\text{Max}}, V_{\text{Min}}, E, \mathbb{R}, \phi)$. For mean payoff games the evaluation function is $\eta : \langle c_0, c_1, \ldots \rangle \mapsto \liminf_{i \rightarrow \infty} \frac{1}{i} \sum_{j=0}^{i-1} c_j$, while for discounted payoff games with discount factor $\lambda \in [0, 1)$ it is $\eta : \langle c_0, c_1, \ldots \rangle \mapsto \sum_{i=0}^{\infty} \lambda^i c_i$ with $\prec$ as the natural order over the reals. For (max) parity games the evaluation function is $\eta : \langle c_0, c_1, \ldots \rangle \mapsto \limsup_{i \rightarrow \infty} c_i$ often used with a preference order $\prec_{\text{parity}}$ where higher even colours are preferred over smaller even colours, even colours are preferred over odd colours, and smaller odd colours are preferred over higher odd colours.

In the remainder of this paper, we will use parity games where every colour is unique, i.e., where $\phi$ is injective. All parity games can be translated into such games as discussed in [29]. For these games, we use a valuation function based on their progress measure. We define $\eta$ as $\langle c_0, c_1, \ldots \rangle \mapsto (c, C, d)$, where $c = \limsup_{i \rightarrow \infty} c_i$ is the dominant colour of the colour sequence, $d = \min \{i \in \omega \mid c_i = c \}$ is the index of the first occurrence of $c$, and $C = \{ c_i \mid i < d, c_i > c \}$ is the set of colours that occur before the first occurrence of $c$. The preference order is defined as the following: we have $(c', C', d') \prec (c, C, d)$ if

- $c' \prec_{\text{parity}} c$,
- $c = c'$, the highest colour $h$ in the symmetric difference between $C$ and $C'$ is even, and in $C$,
- $c = c'$, the highest colour $h$ in the symmetric difference between $C$ and $C'$ is odd, and in $C'$,
- $c = c'$ is even, $C = C'$, and $d < d'$, or
- $c = c'$ is odd, $C = C'$, and $d > d'$.

Definition 2.3 (Strategies). A strategy of Player Max is a function $\sigma : V^* V_{\text{Max}} \rightarrow V$ such that $\langle v, \sigma(v) \rangle \in E$ for all $v \in V^*$ and $v \in V_{\text{Max}}$. Similarly, a strategy of Player Min is a function $\tau : V^* V_{\text{Min}} \rightarrow V$ such that $\langle v, \sigma(v) \rangle \in E$ for all $v \in V^*$ and $v \in V_{\text{Min}}$. We write $\Sigma^\infty$ and $T^\infty$ for the set of strategies of Player Max and Player Min, respectively.

Definition 2.4 (Valuation). For a strategy pair $\langle \sigma, \tau \rangle \in \Sigma^\infty \times T^\infty$ and an initial vertex $v \in V$ we denote the unique play starting from the vertex $v$ by $\pi(v, \sigma, \tau)$ and we write $\text{val}_{G}(v, \sigma, \tau)$ for the value of the vertex $v$ under the strategy pair $\langle \sigma, \tau \rangle$ defined as

$$\text{val}_{G}(v, \sigma, \tau) \triangleq \eta(\phi(\pi(v, \sigma, \tau))).$$
Thus, a positional strategy can be viewed as a function \( \pi, \pi \) that only depends on the last state, i.e. for all \( v \), \( \pi(v) = \pi(v') \). We write \( (\pi, \pi) \) if it only depends on the last state. We also extend the valuation for vertices to a valuation for the whole game by defining \( \text{val} \), \( \text{val} \). We define the concept of the value of a strategy \( \sigma \) as
\[
\text{val}_G(v, \sigma) \overset{\text{def}}{=} \inf_{\tau \in T} \text{val}_G(v, \sigma, \tau) \quad \text{and} \quad \text{val}_G(v, \tau) \overset{\text{def}}{=} \sup_{\sigma \in \Sigma} \text{val}_G(v, \sigma, \tau).
\]

We say that a strategy \( \sigma \in \Sigma^\infty \) is memoryless or positional if it only depends on the last state, i.e. for all \( \pi, \pi' \in \Sigma^\infty \) and \( v \in V_{\text{Max}} \) we have that \( \sigma(\pi v) = \sigma(\pi' v) \). Thus, a positional strategy can be viewed as a function \( \sigma : V_{\text{Max}} \to V \) such that for all \( v \in V_{\text{Max}} \) we have that \( (v, \sigma(v)) \in E \). The concept of positional strategies of Player Min is defined in an analogous manner. We write \( \Sigma \) and \( T \) for the set of positional strategies of Players Max and Min, respectively. We say that a game is positionally determined if:

- \( \text{val}_G(v, \sigma) = \min_{\tau \in T} \text{val}_G(v, \sigma, \tau) \) holds for all \( \sigma \in \Sigma \),
- \( \text{val}_G(v, \tau) = \max_{\sigma \in \Sigma} \text{val}_G(v, \sigma, \tau) \) holds for all \( \tau \in T \),
- **Existence of value**: for all \( v \in V \), \( \text{val}_G(v) = \max_{\sigma \in \Sigma} \text{val}_G(v, \sigma) = \min_{\tau \in T} \text{val}_G(v, \tau) \) holds, and we use \( \text{val}_G(v) \) to denote this value, and
- **Existence of positional optimal strategies**: there is a pair \( \tau_{\text{min}}, \sigma_{\text{max}} \) of strategies such that, for all \( v \in V \), \( \text{val}_G(v) = \text{val}_G(v, \sigma_{\text{max}}) = \text{val}_G(v, \tau_{\text{min}}) \) holds. Observe that for all \( \sigma \in \Sigma \) and \( \tau \in T \) we have that \( \text{val}_G(\sigma_{\text{max}}) \subseteq \text{val}_G(\sigma) \) and \( \text{val}_G(\tau_{\text{min}}) \subseteq \text{val}_G(\tau) \).

Observe that (first and second item above) that classes of games with positional strategies guarantee an optimal positional counter strategy for Player Min to all strategies in \( \sigma \in \Sigma \) of Player Max. We denote these strategies by \( \tau_{\text{pos}}^\sigma \). Similarly, we denote the optimal positional counter strategy for Player Max to a strategy \( \tau \in T \) by \( \sigma_{\text{pos}}^\tau \) of Player Min. While this counter strategy is not necessarily unique, we use the **convention** in all proofs that \( \tau_{\text{pos}}^\sigma \) is always the same counter strategy for \( \sigma \in \Sigma \), and \( \sigma_{\text{pos}}^\tau \) is always the same counter strategy for \( \tau \in T \).

**Example 2.6.** Consider the parity game arena shown in Figure 1. We use circles for the vertices of Player Max and squares for Player Min. We label each vertex with its colour. Notice that a positional strategy can be depicted just by specifying an outgoing edge for all the vertices of a player. The positional strategies \( \sigma \) of Player Max is depicted in blue and the positional strategy \( \tau \) of Player Min is depicted in red. In the example, \( \text{val}(1, \sigma, \tau) = (1, \emptyset, 0), \text{val}(4, \sigma, \tau) = (3, \{4\}, 1), \text{val}(3, \sigma, \tau) = (3, \emptyset, 0), \) and \( \text{val}(0, \sigma, \tau) = (0, \emptyset, 0) \).

**2.1 Classic Strategy Improvement Algorithm**

As discussed in the introduction, classic strategy improvement algorithms work well for classes of games that are positionally determined. Moreover, the evaluation function should be such that one can easily identify the set \( \text{Prof}(\sigma) \) of profitable updates and reach an optimum exactly where there are no profitable updates. We formalise these prerequisites for a class of games to be good for strategy improvement algorithm in this section.
**Definition 2.7** (Profitable Updates). For a strategy $\sigma \in \Sigma$, an edge $(v, v') \in E$ with $v \in V_{\text{Max}}$ is a profitable update if $v' \in \Sigma$ with $\sigma' : v \mapsto v'$ and $\sigma' : v' \mapsto \sigma(v'')$ for all $v'' \neq v$ has a strictly greater valuation than $\sigma$, $\text{val}(\sigma') \supseteq \text{val}(\sigma)$. We write $\text{Prof}(\sigma)$ for the set of profitable updates.

**Example 2.8.** In our example from Figure 1, $\tau = \tau_2^c$ is the optimal counter strategy to $\sigma$, such that $\text{val}(\tau) = \text{val}(\sigma, \tau)$. $\text{Prof}(\tau) = \{(3, 4), (3, 0)\}$, because both the successor to the left and the successor to the right have a better valuation, $(3, \{4\}, 1)$ and $(0, \emptyset, 0)$, respectively, than the successor on the selected self-loop, $(3, \emptyset, 0)$.

For a strategy $\sigma$ and a functional (right-unique) subsets $P \subseteq \text{Prof}(\sigma)$ we define the strategy $\sigma^P$ with $\sigma^P : v \mapsto v'$ if $(v, v') \in P$ and $\sigma^P : v \mapsto \sigma(v)$ if there is no $v' \in V$ with $(v, v') \in P$. For a class of graph games, profitable updates are *combinable* if, for all strategies $\sigma$ and all functional (right-unique) subsets $P \subseteq \text{Prof}(\sigma)$ we have that $\text{val}(\sigma^P) \supseteq \text{val}(\sigma)$. Moreover, we say that a class of graph games is *maximum identifying* if $\text{Prof}(\sigma) = \emptyset \iff \text{val}(\sigma) = \text{val}_G$. Algorithm 4 provides a generic template for strategy improvement algorithms.

**Algorithm 4:** Classic strategy improvement algorithm

1. Let $\sigma_0$ be an arbitrary positional strategy. Set $i := 0$.
2. If $\text{Prof}(\sigma_i) = \emptyset$ return $\sigma_i$.
3. $\sigma_{i+1} := \sigma_i^P$ for some functional subset $P \subseteq \text{Prof}(\sigma)$. Set $i := i + 1$. Go to 2.

We say that a class of games is *good for max strategy improvement* if they are positionally determined and have combinable and maximum identifying improvements.

**Theorem 2.9.** If a class of games is good for max strategy improvement then Algorithm 4 terminates with an optimal strategy $\sigma$ ($\text{val}_G(\sigma) = \text{val}_G$) for Player Max.

As a remark, we can drop the combinability requirement while maintaining correctness when we restrict the updates to a single position, that is, when we require $P$ to be singleton for every update. We call such strategy improvement algorithms *slow*, and a class of games *good for slow max strategy improvement* if it is maximum identifying and positionally determined.

**Theorem 2.10.** If a class of games is positionally determined games with maximum identifying improvement then all slow strategy improvement algorithms terminate with an optimal strategy $\sigma$ ($\text{val}_G(\sigma) = \text{val}_G$) for Player Max.

The proof for both theorems is the same.

**Proof.** The strategy improvement algorithm will produce a sequence $\sigma_0, \sigma_1, \sigma_2 \ldots$ of positional strategies with increasing quality $\text{val}_G(\sigma_0) \supseteq \text{val}_G(\sigma_1) \supseteq \text{val}_G(\sigma_2) \supseteq \ldots$. As the set of positional strategies is finite, this chain must be finite. As the game is maximum identifying, the stopping condition provides optimality.

Various concepts and results extend naturally for analogous claims about Player Min. We call a class of game *good for strategy improvement* if it is good for max strategy improvement and good for min strategy improvement. Parity games, mean payoff games, and discounted payoff games are all good for strategy improvement (for both players). Moreover, the calculation of $\text{Prof}(\sigma)$ is cheap in all of these instances, which makes them well suited for strategy improvement techniques.

### 3 Symmetric Strategy Improvement Algorithm

We first extend the termination argument for classic strategy improvement techniques (Theorems 2.9 and 2.10) to symmetric strategy improvement given as Algorithm 5.
Algorithm 5: Symmetric strategy improvement algorithm

1. Let $\sigma_0$ and $\tau_0$ be arbitrary positional strategies. Set $i := 0$.
2. Determine $\sigma^c_i$ and $\tau^c_i$.
3. $\sigma_{i+1} := \sigma_i^P$ for $P \subseteq \text{Prof}(\sigma) \cap \sigma^c_i$.
4. $\tau_{i+1} := \tau_i^P$ for $P \subseteq \text{Prof}(\tau) \cap \tau^c_i$.
5. If $\sigma_{i+1} = \sigma_i$ and $\tau_{i+1} = \tau_i$ return $(\sigma_i, \tau_i)$.
6. Set $i := i + 1$. Go to 2.

3.1 Correctness

Lemma 3.1. The symmetric strategy improvement algorithm terminates for all classes of games that are good for strategy improvement.

Proof. We first observe that the algorithm yields a sequence $\sigma_0, \sigma_1, \sigma_2, \ldots$ of Player Max strategies for $G$ with improving values $\text{val}_G(\sigma_0) \subseteq \text{val}_G(\sigma_1) \subseteq \text{val}_G(\sigma_2) \subseteq \ldots$, where equality, $\text{val}_G(\sigma_i) \equiv \text{val}_G(\sigma_{i+1})$, implies $\sigma_i = \sigma_{i+1}$. Similarly, for the sequence $\tau_0, \tau_1, \tau_2, \ldots$ of Player Min strategies for $G$, the values $\text{val}_G(\tau_0) \supseteq \text{val}_G(\tau_1) \supseteq \text{val}_G(\tau_2) \supseteq \ldots$, improve (for Player Min), such that equality, $\text{val}_G(\tau_i) \equiv \text{val}_G(\tau_{i+1})$, implies $\tau_i = \tau_{i+1}$. As the number of values that can be taken is finite, eventually both values stabilise and the algorithm terminates.

What remains to be shown is that the symmetric strategy improvement algorithm cannot terminate with an incorrect result. In order to show this, we first prove the weaker claim that it is optimal in $G(\sigma, \tau, \sigma^c, \tau^c) = (V_{\text{max}}, V_{\text{min}}, E', \text{val})$ such that $E' = \{(v, \sigma(v)) \mid v \in V_{\text{max}}\} \cup \{(v, \tau(v)) \mid v \in V_{\text{min}}\}$ is the subgame of $G$ whose edges are those defined by the four positional strategies, when it terminates with the strategy pair $\sigma, \tau$.

Lemma 3.2. When the symmetric strategy improvement algorithm terminates with the strategy pair $\sigma, \tau$ on games that are good for strategy improvement, then $\sigma$ and $\tau$ are the optimal strategies for Players Max and Min, respectively, in $G(\sigma, \tau, \sigma^c, \tau^c)$.

Proof. For $G(\sigma, \tau, \sigma^c, \tau^c)$, both update steps are not restricted: the changes Player Max can potentially select his updates from are the edges defined by $\sigma^c$ at the vertices $v \in V_{\text{max}}$ where $\sigma$ and $\sigma^c$ differ ($\sigma(v) \neq \sigma^c(v)$). Consequently, $\text{Prof}(\sigma) = \text{Prof}(\sigma) \cap \sigma^c$. Thus, $\sigma = \sigma'$ holds if, and only if, $\sigma$ is the result of an update step when using classic strategy improvement in $G(\sigma, \tau, \sigma^c, \tau^c)$ when starting in $\sigma$. As game is maximum identifying, $\sigma$ is the optimal Player Max strategy for $G(\sigma, \tau, \sigma^c, \tau^c)$. Likewise, the Player Min can potentially select every updates from $\tau^c$, at vertices $v \in V_{\text{min}}$ and we first get $\text{Prof}(\tau) = \text{Prof}(\tau) \cap \tau^c$ with the same argument. As the game is minimum identifying, $\tau$ is the optimal Player Min strategy for $G(\sigma, \tau, \sigma^c, \tau^c)$.

We are now in a position to expand the optimality in the subgame $G(\sigma, \tau, \sigma^c, \tau^c)$ from Lemma 3.2 to global optimality the valuation of these strategies for $G$.

Lemma 3.3. When the symmetric strategy improvement algorithm terminates with the strategy pair $\sigma, \tau$ on a game $G$ that is good for strategy improvement, then $\sigma$ is an optimal Player Max strategy and $\tau$ an optimal Player Min strategy.

Proof. Let $\sigma, \tau$ be the strategies returned by the symmetric strategy improvement algorithm for a game $G$, and let $L = G(\sigma, \tau, \sigma^c, \tau^c)$ denote the local game from Lemma 3.2 defined by them. Lemma 3.2 has established optimality in $L$. Observing that the optimal responses in $G$ to $\sigma$ and $\tau$, $\sigma^c$ and $\tau^c$, respectively, are available in $L$, we first see that they are also optimal in $L$. Thus, we have

- $\text{val}_L(\sigma) \equiv \text{val}_L(\sigma, \tau^c) \equiv \text{val}_G(\sigma, \tau^c)$ and
\begin{itemize}
\item \(\text{val}_L(\tau) \equiv \text{val}_L(\sigma_\tau^+, \tau) \equiv \text{val}_G(\sigma_\tau^+, \tau).\)
\end{itemize}

Optimality in \(L\) then provides \(\text{val}_L(\sigma) = \text{val}_L(\tau).\) Putting these three equations together, we get
\(\text{val}_G(\sigma, \tau_\sigma^+) \equiv \text{val}_G(\sigma_\tau^+, \tau).\)

Taking into account that \(\tau_\sigma^+\) and \(\sigma_\tau^+\) are the optimal responses to \(\sigma\) and \(\tau\), respectively, in \(G\), we expand this to \(\text{val}_G \subseteq \text{val}_G(\sigma, \tau_\sigma^+) \equiv \text{val}_G(\sigma_\tau^+, \tau) \equiv \text{val}_G(\sigma, \tau) \subseteq \text{val}_G\) and get \(\text{val}_G \equiv \text{val}_G(\sigma) \equiv \text{val}_G(\tau) \equiv \text{val}_G(\sigma, \tau).\)

The Lemmas in this subsection yield the following results.

**Theorem 3.4.** The symmetric strategy improvement algorithm is correct for games that are good for strategy improvement.

**Theorem 3.5.** The slow symmetric strategy improvement algorithm is correct for positionally determined games that are maximum and minimum identifying.

We implemented our symmetric strategy improvement algorithm based on the progress measures introduced by Vöge and Jurdziński [29]. The first step is to determine the valuation for the optimal counter strategies to and the valuations for \(\sigma\) and \(\tau\).

**Example 3.6.** In our running example from Figure[1] we have discussed in the previous section that \(\tau\) is the optimal counter strategy \(\tau_\sigma^+\) and that \(\text{Prof}(\sigma) = \{(3,4), (3,0)\}\). In the optimal counter strategy \(\sigma_\tau^+\) to \(\tau\), Player Max moves from 3 to 4, and we get \(\text{val}(1, \tau) = (1, \emptyset, 0), \text{val}(4, \tau) = (4, \emptyset, 0), \text{val}(3, \tau) = (4, \emptyset, 1), \text{val}(0, \tau) = (0, \emptyset, 0)\). Consequently, \(\text{Prof}(\tau) = \{(4,1)\}\). For the update of \(\sigma\), we select the intersection of \(\text{Prof}(\sigma)\) and \(\sigma_\tau^+\). In our example, this is the edge from 3 to 4 (depicted in green). To update \(\tau\), we select the intersection of \(\text{Prof}(\tau)\) and \(\tau_\sigma^+\). In our example, this intersection is empty, as the current strategy \(\tau\) agrees with \(\tau_\sigma^+\).

### 3.2 A minor improvement on stopping criteria

In this subsection, we look at a minor albeit natural improvement over Algorithm[5] shown in Algorithm[6]. There we used termination on both sides as a condition to terminate the algorithm. We could alternatively check if either player has reached an optimum. Once this is the case, we can return the optimal strategy and an optimal counter strategy to it.

**Algorithm 6:** Symmetric strategy improvement algorithm (Improved Stopping criteria)

1. Let \(\sigma_0\) and \(\tau_0\) be arbitrary positional strategies. set \(i := 0\).
2. Determine \(\sigma_\tau^+\) and \(\tau_\sigma^+\).
3. if \(\text{Prof}(\sigma_i) = \emptyset\) return \((\sigma_i, \tau_\sigma^+)\);
4. if \(\text{Prof}(\tau_i) = \emptyset\) return \((\sigma_\tau^+, \tau_i)\);
5. \(\sigma_{i+1} := \sigma_i^P\) for \(P \subseteq \text{Prof}(\sigma) \cap \sigma_\tau^+\).
6. \(\tau_{i+1} := \tau_i^P\) for \(P \subseteq \text{Prof}(\tau) \cap \tau_\sigma^+\).
7. set \(i := i + 1.\) go to 2.

The correctness of this stopping condition is provided by Theorems[2.9] and [2.10], and checking this stopping condition is usually cheap: it suffices to check if \(\text{Prof}(\sigma)\) or \(\text{Prof}(\tau)\) is empty. This provides us with a small optimisation, as we can stop as soon as one of the strategies involved is optimal. However this small optimisation can only provide a small advantage.

**Theorem 3.7.** The difference in the number of iterations of Algorithm[5] and Algorithm[6] is at most linear in the number of states of \(G\).
Proof. Let $\sigma$ be an optimal strategy for $G$. When starting with a strategy pair $\sigma, \tau_0$ for some strategy $\tau_0$ of Player Min, we first construct the optimal counter strategies $\tau_c^\sigma$ and $\sigma^{\tau_0}$. As $\sigma$ is optimal and $G$ maximum identifying, $\text{Prof}(\sigma) = \emptyset$, and strategy improvement will not change it. In particular, our algorithm will always provide $\sigma' = \sigma$, irrespective of the optimal counter strategy $\sigma_c^\tau$ to a strategy $\tau_i$ of Player Min. This also implies that $\tau_c^\sigma$ will not change. It is now easy to see that, unless $\tau_i' = \tau_i$, $\tau_{i+1}$ differs from $\tau_i$ in at least one decision, and it differs by adhering to $\tau_c^\sigma$ at the positions where it differs ($\forall v \in V_{\text{min}}. \tau_i(v) \neq \tau_{i+1}(v) \Rightarrow \tau_{i+1}(v) = \tau_c^\sigma(v)$). Such an update can happen at most once for each Player Min position. The argument for starting with an optimal strategy $\tau$ of Player Min is similar.

4 Friedmann’s Traps

In a seminal work on the complexity of strategy improvement [11], Friedmann uses a class of parity games called 1-sink parity games. These games contain a sink node with the weakest odd parity in a max-parity game. This sink node is reachable from every other node in the game and such a game is won by Player Min eventually. Figure 2 shows a lower bound game from [11].

In order to obtain an exponential lower bound for the classic strategy improvement algorithm with the locally optimising policy, these sink games implement a binary counter realised by a gadget called a cycle gate which consists of two components. With $n$ cycle gates, we have a representation of the $n$ bits for an $n$ bit counter. The first component of a cycle gate is called a simple cycle. In Figure 2 the three smaller boxes shown in yellow are the simple cycles of the game. These simple cycles encode the bits of the counter. The second component of the cycle gate gadget is called a deceleration lane. This structure serves to ensure that any profitable updates to strategies are postponed by cycling through seemingly more profitable improvements, in the order $r, s, a_1, a_2, \ldots$, before eventually turning to $e_i$. This structure is shown as a shaded blue rectangle in Figure 2.

A simple cycle consists of exactly one Player Max controlled node $d$ with a weak odd colour $k$ and
one Player Min controlled node $e$ with the even colour $k + 1$. The Player Max node is also connected to some set of external nodes in the game and the Player Min node is connected to an output node with a high even colour on a path to the sink node. Given a strategy $\sigma$, we say that a simple cycle is closed if we have an edge $\sigma(d) = e$. Otherwise, we say that the simple cycle is open. Opening and closing cycles correspond to unsetting and setting bits. We then say a cycle gate is open or closed when its corresponding simple cycle is open or closed respectively.

In these lower bound games, the simple cycles are connected to the deceleration lane in such a way that lower valued cycles have less edges entering the deceleration lane ensuring that lower open cycles close before higher open cycles. This allows the lesser significant bits to be set and reset before the higher significant bits.

The deceleration lane hides sensible improvements, thus making the players take more iterations before taking the best improvement. It is then shown in [11] that incrementing a bit state always requires more than one strategy iteration in 4 different phases. This gadget thus counts an exponential number of improvement steps taken by the strategy improvement algorithm to flip $n$ bits. For a detailed exposition of the gadget and the exponential lower bound construction, we refer the reader to [11].

4.1 Escaping the traps with symmetric strategy improvement

We discuss the effect of symmetric strategy improvement on Friedmann’s traps, with a focus on the simple cycles. Simple cycles are the central component of the cycle gates and the heart of the lower bound proof. As described above, an $n$-bit counter is represented by $n$ cycle gates, each cycle gate embedding a smaller simple cycle. These simple cycles are reused exponentially often to represent $n$ bits. Both players have the choice to open or close the simple cycles.

The optimal strategy of both players in the simple cycles of Figure 2 is to turn right. (For Player Max, one could say that he wants to leave the cycle, and for Player Min, one could say that she wants to stay in it.) When the players agree to stay in the cycle, Player Max wins the parity game. In fact these are the only places where Player Max can win positionally in this parity game. When running the symmetric strategy improvement algorithm for Player Max, the optimal counter strategy by Player Min is to move to the right in simple cycles where Player Max is moving to the right, and to move left in all other simple cycles.

As mentioned before, Friedmann [11] showed that, when looking at an abstraction of the Player Max strategy that only distinguishes the decisions of turning right or not turning right in the simple cycles, then they essentially behave like a binary counter that, with some delay (caused by the deceleration lane) will ‘count up’. More precisely, one step after the $i^{th}$ bit has been activated, all lower bits are reset.

We now discuss how symmetric strategy improvement can beat this mechanism by taking the view of both players into account. For this, we consider a starting configuration, where Player Min moves to the right in the $j$ most significant simple cycle positions, where $j$ can be 0. Note that, when Player Min moves right in all of these positions, she has found her optimal strategy and we can invoke Theorem 3.7 to show that the algorithm terminates in a linear number of steps—or simply stop when using the alternative stopping condition.

The first observation is that changing the decision to moving left will not lead to an improvement, as it produces a winning cycle of a quality (leading even colour) higher than the quality of any cycle available for Player Max under the current strategy of Player Min. Let us now consider the less significant position $j + 1$. First, we observe that moving to the right is a superior strategy. This can easily be seen: moving to the left produces a cycle with a dominating even colour and thus turns out to be winning for Player Max. Moving to the right in position $j + 1$ and (by our assumption) all more significant positions removes this cycle and implies that the leading colour from this position is 1. This is clearly better for Player Min. If Player Min uses a strategy where $j + 1$ is the most significant position where she decides to move to the left, we have the following case distinctions for Player Max’s strategy in this simple cycle:

1. Player Max moves to the right in this simple cycle. Then moving to the right is also the optimal counter strategy for Player Min, and her strategy will be updated accordingly.
2. Player Max does not move right in this simple cycle with her current strategy \( \sigma \). Moving right in this simple cycle is among \( \text{Prof}(\sigma) \), as one even colour is added to the set in the quality measure in the local comparison. It is also the choice for the optimal counter strategy \( \sigma'_c \) to the current strategy \( \tau \) of Player Min, as this is the only way for Player Max to produce a valuation with the dominating even colour of this simple cycle, while to valuation with a higher even colour is possible.

Taking these two cases into consideration, Player Min will move to the right in the \( j \) most significant positions after \( 2^j \) improvement steps. When Player Max has found his optimal strategy, we can invoke Theorem 5.7 to show termination in linear steps for the algorithm.

There are similar arguments for all kinds of traps that Friedmann has developed for strategy improvement algorithms. We have not formalised these arguments on other instances, but provided the number of iterations needed by our symmetric strategy improvement algorithm for all of them in the next section.

Note that the way in which Friedmann traps asymmetric strategy improvement has proven to be quite resistant to the improvement policy (snare [10], random facet [19, 3], globally optimal [25], etc.). From the perspective of the traps, the different policies try to aim at a minor point in the mechanism of the traps, and this minor point is adjusted. The central mechanism, however, is not affected. All of these examples have some variant of simple cycles at the heart of the counter and a deceleration lane to orchestrate the timely counting.

Symmetric strategy improvement aims at the mechanism of the traps themselves. It seems that examples that trap symmetric strategy improvement algorithms need to do more than just trapping both players (which could be done by copying the trap with inverse roles), they need to trap them simultaneously. It is not likely to find a proof that such traps do not exist, as this would imply a proof that symmetric strategy improvement solves parity (or, depending on the proof, mean or discounted payoff) games in polynomial time. But it seems that such traps would need a different structure. A further difference to asymmetric strategy improvement is that the deceleration lane ceases to work.

Taking into account that finding traps for asymmetric strategy improvement took decades and was very insightful, this looks like an interesting challenge for future research.

5 Experimental Results

We have implemented the symmetric strategy improvement algorithm for parity games and compared it with the standard strategy improvement algorithm with the popular locally optimising and other switching rules. To generate various examples we used the tools steadygame and stratimprgen that comes as a part of the parity game solver collection PGSolver [18]. We have compared the performance of our algorithm on parity games with 100 positions (see appendix) and found that the locally optimising policy outperforms other switching rules. We therefore compare our symmetric strategy improvement algorithm with the locally optimising strategy improvement below.

Since every iteration of both algorithms is rather similar—one iteration of our symmetric strategy improvement algorithm essentially runs two copies of an iteration of a classical strategy improvement algorithm—and can be performed in polynomial time, the key data to compare these algorithms is the number of iterations taken by both algorithms.

Symmetric strategy improvement will often rule out improvements at individual positions: it disregards profitable changes of Player Max and Min if they do not comply with \( \sigma'_c \) and \( \tau'_c \), respectively. It is well known that considering fewer updates can lead to a significant increase in the number of updates on random examples and benchmarks. An algorithm based on the random-facet method [19, 3], e.g., needs around a hundred iterations on the random examples with 100 positions we have drawn, simply because it updates only a single position at a time. The same holds for a random-edge policy where only a single position is updated. The figures for these two methods are given in the appendix.

It is therefore good news that symmetric strategy improvement does not display a similar weakness.
Figure 3: These plots compare the performance of the symmetric strategy improvement algorithm (data points in cyan circles) with standard strategy improvement using the locally optimising policy rule (data points in orange squares). The plot on the left side is for random examples generated using the `steadygame 1000 2 4 3 5 6` command, while the plot on the right is for Friedmann’s trap from the previous section generated by the command `stratimprgen -pg switchallsubexp i`.

It even uses less updates when compared to classic strategy improvement with the popular locally optimising and locally random policy rules. Note also that having less updates can lead to a faster evaluation of the update, because unchanged parts do not need to be re-evaluated [3].

As shown in Figure 3, the symmetric strategy improvement algorithm not only performs better (on average) in comparison with the traditional strategy improvement algorithm with the locally optimising policy rule, but also avoids Friedmann’s traps for the strategy improvement algorithm. The following table shows the performance of symmetric strategy improvement algorithm for Friedmann’s traps for other common switching rules. It is clear that our algorithm is not exponential for these classes of examples.

| Switch Rule            | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  |
|------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Cunningham             | 2   | 6   | 9   | 12  | 15  | 18  | 21  | 24  | 27  | 30  |
| CunninghamSubexp       | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
| FearnleySubexp         | 4   | 7   | 11  | 13  | 17  | 21  | 25  | 29  | 33  | 37  |
| FriedmannSubexp        | 4   | 9   | 13  | 15  | 19  | 23  | 27  | 31  | 35  | 39  |
| RandomEdgeExpTest      | 1   | 2   | 2   | 2   | 2   | 2   | 2   | 2   | 2   | 2   |
| RandomFacetSubexp      | 1   | 2   | 7   | 9   | 11  | 13  | 15  | 17  | 19  | 21  |
| SwitchAllBestExp       | 4   | 5   | 8   | 11  | 12  | 13  | 15  | 17  | 18  | 19  |
| SwitchAllBestSubExp    | 5   | 7   | 9   | 11  | 13  | 15  | 17  | 19  | 21  | 23  |
| SwitchAllSubExp        | 3   | 5   | 7   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| SwitchAllExp           | 3   | 4   | 6   | 8   | 10  | 11  | 12  | 14  | 16  | 18  |
| ZadehExp               | -   | 6   | 10  | 14  | 18  | 21  | 25  | 28  | 32  | 35  |
| ZadehSubexp            | 5   | 9   | 13  | 16  | 20  | 23  | 27  | 30  | 34  | 37  |

6 Discussion

We have introduced symmetric approaches to strategy improvement, where the players take inspiration from the respective other’s strategy when improving theirs. This creates a rather moderate overhead, where each step is at most twice as expensive as a normal improvement step. For this moderate price, we have shown that we can break the traps Friedmann has introduced to establish exponential bounds for the different update policies in classic strategy improvement [11,12,13].
In hindsight, attacking a symmetric problem with a symmetric approach seems so natural, that it is quite surprising that it has not been attempted immediately. There are, however, good reasons for this, but one should also consent that the claim is not entirely true: the concurrent update to the respective optimal counter strategy has been considered quite early [11][12][13], but was dismissed, because it can lead to cycles [5].

The first reason is therefore that it was folklore that symmetric strategy improvement does not work. The second reason is that the argument for the techniques that we have developed in this paper would have been restricted to beauty until some of the appeal of classic strategy improvement was caught in Friedmann’s traps. Friedmann himself, however, remained optimistic:

We think that the strategy iteration still is a promising candidate for a polynomial time algorithm, however it may be necessary to alter more of it than just the improvement policy.

This is precisely, what the introduction of symmetry and co-improvement tries to do.

References

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[2] Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. Dag-width and parity games. In *Proc. of STACS*, pages 524–436. Springer-Verlag, 2006.

[3] Henrik Björklund and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Appl. Math.*, 155(2):210–229, 2007.

[4] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *TCS*, 178(1–2):237–255, 1997.

[5] Anne Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[6] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *Proc. of LICS*, pages 279–290. IEEE Computer Society Press, June 2001.

[7] E. Allen Emerson and Charanjit S. Jutla. Tree automata, \(\mu\)-calculus and determinacy. In *Proc. of FOCS*, pages 368–377. IEEE Computer Society Press, October 1991.

[8] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of \(\mu\)-calculus. In *Proc. of CAV*, pages 385–396, 1993.

[9] E. Allen Emerson and C. Lei. Efficient model checking in fragments of the propositional \(\mu\)-calculus. In *Proc. of LICS*, pages 267–278. IEEE Computer Society Press, 1986.

[10] John Fearnley. Non-oblivious strategy improvement. In *Proc. of LPAR*, Lecture Notes in Computer Science, pages 212–230, 2010.

[11] Oliver Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *LMCS*, 7(3), 2011.

[12] Oliver Friedmann. A subexponential lower bound for zadeh’s pivoting rule for solving linear programs and games. In *Proc. of IPCO*, LNCS, pages 192–206, 2011.
[13] Oliver Friedmann. A superpolynomial lower bound for strategy iteration based on snare memo-
ration. *Discrete Applied Mathematics*, 161(10-11):1317–1337, 2013.

[14] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving
parity games. In *Proc. of SODA*, pages 117–123. ACM/SIAM, 2006.

[15] Marcin Jurdziński. Small progress measures for solving parity games. In *Proc. of STACS*, pages
290–301. Springer-Verlag, 2000.

[16] Dexter Kozen. Results on the propositional $\mu$-calculus. *TCS*, 27:333–354, 1983.

[17] M. Lange. Solving parity games by a reduction to SAT. In *Proc. of Int. Workshop on Games in
Design and Verification*, 2005.

[18] M. Lange and O. Friedmann. The PGSolver collection of parity game solvers. Technical report,
Institut für Informatik Ludwig-Maximilians-Universität, Germany, February 2010.

[19] Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem.
*Inf. Comput.*, 117(1):151–155, 1995.

[20] Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–
184, 1993.

[21] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *Proc. of CAV*,
pages 80–92. Springer-Verlag, 2003.

[22] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata.
In *Proc. of LICS*, pages 255–264. IEEE Computer Society, 2006.

[23] Anuj Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, Computer Science
Department, University of California, Berkeley, 1995.

[24] Sven Schewe. Solving parity games in big steps. In *Proc. of FSTTCS*, pages 449–460. Springer-
Verlag, 2007.

[25] Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In
*Proc. of CSL 2008*, pages 368–383. Springer-Verlag, 2008.

[26] Sven Schewe and Bernd Finkbeiner. The alternating-time $\mu$-calculus and automata over concurrent
game structures. In *Proc. of CSL*, pages 591–605. Springer-Verlag, 2006.

[27] Sven Schewe and Bernd Finkbeiner. Synthesis of asynchronous systems. In *Proc. of LOPSTR*,
pages 127–142. Springer-Verlag, 2006.

[28] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of ICALP*, pages
628–641. Springer-Verlag, 1998.

[29] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity
games. In *Proceedings of the CAV*, pages 202–215. Springer-Verlag, July 2000.

[30] Thomas Wilke. Alternating tree automata, parity games, and modal $\mu$-calculus. *Bull. Soc. Math.
Belg.*, 8(2), May 2001.

[31] Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on
infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

[32] Uri Zwick and Mike S. Paterson. The complexity of mean payoff games on graphs. *Theoretical
Computer Science*, 158(1–2):343–359, 1996.
A Symmetric Strategy Improvement algorithm versus classic strategy improvement algorithm with various switching rules

Figure 4: These plots compare the performance of the symmetric strategy improvement algorithm (data points in cyan circles) with standard strategy improvement using the locally optimising policy rule (data points in orange squares), random-edge switching rule (data points in red triangles), random-facet rule (data points in blue triangles), and switch-half rule (data point in green triangles). These plots are for random examples generated using the steadygame 100 2 4 3 5 6 command from PGSolver. The results from randomized switching rules (random-edge, random-facet, and switch-half) presented here are taken as average number of iterations over four executions of the corresponding algorithms.