International Journal of Foundations of Computer Science


# TWO LOCAL STRATEGY ITERATION SCHEMES FOR PARITY GAME SOLVING

OLIVER FRIEDMANN

*Dept. of Computer Science*
*University of Munich, Germany*
*http://www.tcs.ifi.lmu.de/∼friedman*

MARTIN LANGE

*School of Electrical Engineering and Computer Science*
*University of Kassel, Germany*
*http://www.uni-kassel.de/∼mlange*

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

The problem of solving a parity game is at the core of many problems in model checking, satisfiability checking and program synthesis. Some of the best algorithms for solving parity game are strategy iteration algorithms. These are global in nature since they require the entire parity game to be present at the beginning. This is a distinct disadvantage because in many applications one only needs to know which winning region a particular node belongs to, and a witnessing winning strategy may cover only a fractional part of the entire game graph.

We present two local strategy iteration algorithms which explore the game graph on-the-fly whilst performing the improvement steps. We also compare them empirically with existing global strategy iteration algorithms and the currently only other local algorithm for solving parity games. It turns out that local strategy iteration can outperform these others significantly.

*Keywords*: parity games; strategy iteration; local solving; modal mu calculus

1991 Mathematics Subject Classification: 22E46, 53C35, 57S20

## 1. Introduction

Parity games are two-player games of perfect information played on labelled, directed graphs. They are at the core of various problems in computer-aided verification, namely model checking [1], satisfiability checking for branching-time logics [2] and controller synthesis [3]. They are also closely related to other games of infinite duration, in particular mean and discounted payoff as well as stochastic games [4, 1].

A variety of algorithms for solving parity games has been invented so far. Among these, strategy iteration (SI) algorithms have proved to be very successful both because of their theoretical elegance as well as their good performance in practice.

The name hints at the general method according to which these algorithms work. Starting with an initial strategy, they compute an evaluation which either witnesses that the current strategy is winning, or it offers ways to improve the current strategy. SI is therefore in fact an algorithmic scheme parametrised by an improvement rule which determines how to improve a non-winning strategy. The first *discrete* SI algorithm for parity game solving was given by Jurdziński and Vöge [5] which was inspired by and improves over Puri's SI algorithms for parity and discounted payoff games [6].

A disadvantage common to all these algorithms is their global nature: they require the entire game graph to be present at the start of the algorithm's execution. They then compute for a given player his entire winning region and a corresponding winning strategy. This is a drawback because in many applications of parity games taken from program verification one only needs to know for a specific node in the game which winning region it belongs to, together with a winning strategy witnessing this.

As an example, consider the model checking problem for branching-time temporal logics like CTL, CTL$^*$, or the modal $\mu$-calculus on finite-state systems for instance. These can easily be regarded as instances of the parity game solving problem [1, 7] in which the nodes are pairs of transition system states and (sets of) subformulas of the input formula. Solving the model checking problem means solving the parity game locally, i.e. deciding which player wins the node composed of the transition system's initial state and the input formula. A global algorithm would compute this information for every (reachable) pair of nodes and (set of) subformulas. In model checking, local solving is usually considered to be more desirable for explicitly represented state spaces [8]. Similarly, the satisfiability problem for branching-time logics reduces to the problem of solving parity games [2] and, again, in order to decide whether or not the input formula is satisfiable one needs to know whether or not player 0 wins a particular node of the resulting parity game. The winners of the other nodes are irrelevant.

It is known that the problem of solving a parity game is equivalent under linear-time reductions to the model checking problem for the modal $\mu$-calculus. There are local model checkers, in particular the one by Stevens and Stirling [9], which can therefore be rephrased as parity game solvers. The Stevens-Stirling algorithm is generally outperformed in practice by the global algorithms – when the task is to solve an entire game – in particular the SI algorithms [10]. It is therefore fair to say that local parity game solvers with reasonable efficiency in practice are desirable but do not exist yet.

Here we present two local SI algorithm schemes for solving parity games.[a] Local strategy iteration means starting the algorithm with a given initial node, expanding the game graph in an on-the-fly fashion and intertwining this expansion with evaluations needed in order to improve current strategies. It terminates as soon as

[a]Note that this is not just one scheme with two different improvement policies.

it has determined which winning region the initial node belongs to.

The rest of the paper is organized as follows. Sect. 2 recalls parity games and Sect. 3 recalls global strategy iteration. Sect. 4 introduces the foundations of local strategy iteration, and presents two concrete variants of this algorithmic class. Sect. 5 reports on empirical comparisons between the local strategy iteration variants, global strategy iteration, and the local model checking algorithm by Stevens/Stirling when rephrased as a parity game solver. We show that local strategy iteration can outperform the others significantly.

The contribution of this paper lies in the practical use of parity game solvers.[b] The presented algorithms do not improve other algorithms in asymptotic worst-case complexity. Their relevance is justified by the improvement in solving parity games in practice which they bring along.

## 2. Parity Games

A *parity game* is a tuple $G = (V, V_0, V_1, E, \Omega)$ where $(V, E)$ forms a directed graph in which each node has at least one successor. The set of nodes is partitioned into $V = V_0 \cup V_1$ with $V_0 \cap V_1 = \emptyset$, and $\Omega : V \to \mathbb{N}$ is the *priority function* that assigns to each node a natural number called the *priority* of the node. We write $|\Omega|$ for the index of the parity game, that is the number of different priorities assigned to its nodes. The graph is required to be total. Here we only consider games based on finite graphs.

We also use infix notation $vEw$ instead of $(v, w) \in E$ and define the set of all *successors* of $v$ as $vE := \{w \mid vEw\}$, as well as the set of all *predecessors* of $w$ as $Ew := \{v \mid vEw\}$.

The game is played between two players called 0 and 1 in the following way. Starting in a node $v_0 \in V$ they construct a path through the graph as follows. If the construction so far has yielded a finite sequence $v_0 \dots v_n$ and $v_n \in V_i$ then player $i$ selects a $w \in v_n E$ and the play continues with the sequence $v_0 \dots v_n w$.

Every infinite play has a unique winner given by the *parity* of the greatest priority that occurs infinitely often in a play. The winner of the play $v_0 v_1 v_2 \dots$ is player $i$ iff $\max\{p \mid \forall j \in \mathbb{N}\ \exists k \geq j : \Omega(v_k) = p\} \equiv_2 i$ (where $i \equiv_2 j$ holds iff $|i - j|$ mod $2 = 0$). That is, player 0 tries to make an even priority occur infinitely often without any greater odd priorities occurring infinitely often, player 1 attempts the converse.

A *strategy* for player $i$ is a function $\sigma : V^* V_i \to V$, s.t. for all sequences $v_0 \dots v_n$ with $v_{j+1} \in v_j E$ for all $j = 0, \dots, n-1$, and all $v_n \in V_i$ we have: $\sigma(v_0 \dots v_n) \in v_n E$. A play $v_0 v_1 \dots$ *conforms* to a strategy $\sigma$ for player $i$ if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ then $v_{j+1} = \sigma(v_0 \dots v_j)$. Intuitively, conforming to a strategy means to always make those choices that are prescribed by the strategy. A strategy $\sigma$ for player $i$ is a *winning strategy* in node $v$ if player $i$ wins every play that begins in $v$

[b]This paper extends our previous results as presented in the proceedings of GandALF'2010 [11].

and conforms to $\sigma$.

A strategy $\sigma$ for player $i$ is called *positional* if for all $v_0 \dots v_n \in V^*V_i$ and all $w_0 \dots w_m \in V^*V_i$ we have: if $v_n = w_m$ then $\sigma(v_0 \dots v_n) = \sigma(w_0 \dots w_m)$. That is, the choice of the strategy on a finite path only depends on the last node on that path.

With $G$ we associate two sets $W_0, W_1 \subseteq V$; $W_i$ is the set of all nodes $v$ s.t. player $i$ wins the game $G$ starting in $v$. Here we restrict ourselves to positional strategies because it is well-known that a player has a (general) winning strategy iff she has a positional winning strategy for a given game. In fact, parity games enjoy positional determinacy meaning that for every node $v$ in the game either $v \in W_0$ or $v \in W_1$ [12]. Furthermore, it is not difficult to show that, whenever player $i$ has winning strategies $\sigma_v$ for all $v \in U$ for some $U \subseteq V$, then there is also a single strategy $\sigma$ that is winning for player $i$ from every node in $U$.

The problem of *(globally) solving a parity game* is to decide for every node $v \in V$ whether or not it belongs to $W_0$. The problem of *locally solving a parity game* is to decide, for a given node $v$, whether or not $v$ belongs to $W_0$. Note that it may not be necessary to visit all nodes of a game in order to answer this question. Consider the following trivial example, in which two nodes are connected to each other and have no further outgoing edges. Hence, this simple cycle is clearly won by one of the two players. A depth-first search can find this loop whilst leaving the rest of the game unexplored.

A strategy $\sigma$ for player $i$ induces a *strategy subgame* $G|_\sigma := (V, V_0, V_1, E|_\sigma, \Omega)$ where $E|_\sigma := \{(u, v) \in E \mid u \in dom(\sigma) \Rightarrow \sigma(u) = v\}$. Such a subgame $G|_\sigma$ poses the restriction on the game graph that for every node $u \in V_i$, all transitions from $u$ but $\sigma(u)$ are no longer accessible. The set of strategies for player $i$ is denoted by $\mathcal{S}_i(G)$.

## 3. Global Strategy Iteration

We recap the global strategy iteration algorithm as given by Jurdziński and Vöge [5]. The algorithm selects one of the two players to be the strategy improver, we will select player 0 for this role. For each strategy of player 0, the algorithm computes a *valuation*, which provides a measure of how good the strategy is. Player 0 strategies can be partially ordered by their valuations, i.e., some strategies are strictly better than others, but there can exist pairs of strategies that are incomparable. Strategy improvement also provides a method for improving a strategy: For every player 0 strategy that is not maximal in the partial order, we can compute a strategy that is strictly improved in the partial order.

Therefore, strategy improvement proceeds as follows. It begins with an arbitrarily chosen strategy, and computes its valuation. It then computes a new strategy that has a strictly better valuation. Since there are only a finite number of positional strategies, iterating this procedure must eventually find a strategy that is maximal in the partial order, and it can be shown that the winning sets of the two players

can be determined by looking at the properties of this optimal strategy. Therefore, the algorithm can terminate and output the sets $W_0$ and $W_1$.

Strategy valuations can be defined in different ways, but usually through *node valuations* for each node $v$. The latter are totally ordered and induced by a path starting in $v$ leading to a cycle. The valuation process then selects for each node $v$ the node valuation associated with a path $\pi$ ending in a cycle starting in $v$ conforming to the strategy $\sigma$ that yields the worst valuation.

For a given parity game $G = (V,\ V_0,\ V_1,\ E,\ \Omega)$, the *reward* of node $v$ is defined as follows: $\mathrm{rew}_G(v) := \Omega(v)$ if $\Omega(v) \equiv_2 0$ and $\mathrm{rew}_G(v) := -\Omega(v)$ otherwise. The set of *profitable nodes* for player 0 resp. 1 is defined to be $V_\oplus := \{v \in V \mid \Omega(v) \equiv_2 0\}$ resp. $V_\ominus := \{v \in V \mid \Omega(v) \equiv_2 1\}$.

The *relevance ordering* $<$ on $V$ is induced by $\Omega$: $v < u :\iff \Omega(v) < \Omega(u)$; usually, one makes the assumption that no priority is assigned twice, yielding total relevance orderings. However, since we are interested in local solving, we do not know about all occurring priorities in advance. One way of dealing with this issue is to extend the relevance ordering to an arbitrary total ordering, i.e. in the context of local solving, one could say that $v < u$ if $\Omega(v) = \Omega(u)$ and $u$ has been discovered after $v$. We assume from now on that $<$ is a total ordering. Additionally one defines the *reward ordering* $\prec$ on $V$ by $v \prec u$ iff $\mathrm{rew}_G(v) < \mathrm{rew}_G(u)$, or $\Omega(v) = \Omega(u)$ and ($v < u$ and $\Omega(v) \equiv_2 0$) or ($u < v$ and $\Omega(v) \equiv_2 1$). Note that $\prec$ is a total ordering since $<$ is a total ordering.

Let $v$ be a node, $\sigma$ be a positional player 0 strategy and $\tau$ be a positional player 1 strategy. Starting in $v$, there is exactly one path $\pi_{\sigma,\tau,v}$ that conforms to $\sigma$ and $\tau$. Since $\sigma$ and $\tau$ are positional strategies, this path can be uniquely written as follows.

$$\pi_{\sigma,\tau,v} = v_1 \dots v_k (w_1 \dots w_l)^\omega$$

with $v_1 = v$, $v_i \neq w_1$ for all $1 \leq i \leq k$ and $w_1 > w_j$ for all $1 < j \leq l$.

Discrete strategy iteration relies on a more abstract description of such a play $\pi_{\sigma,\tau,v}$. In fact, we only consider the *dominating cycle node* $w_1$, the set of *more relevant nodes* – i.e. all $v_i > w_1$ – on the path to the cycle node, and the *length* $k$ of the path leading to the cycle node. More formally, the *node valuation of* $v$ *w.r.t.* $\sigma$ *and* $\tau$ is defined as follows.

$$\vartheta_{\sigma,\tau,v} := (w_1, \{v_i > w_1 \mid 1 \leq i \leq k\}, k)$$

In order to compare node valuations with each other, we introduce a total ordering on the set of node valuations. For that reason, we need to define a total ordering $\prec$ on the second component of node valuations – i.e. on subsets of $V$ – first: to determine which set of nodes is better w.r.t. $\prec$, one considers the node with the highest relevance that occurs in only one of the two sets. The set owning that node is greater than the other if and only if that node has an even priority. More formally:

$$M \prec N :\iff M \triangle N \neq \emptyset \text{ and } \max_{<}(M \triangle N) \in ((N \cap V_\oplus) \cup (M \cap V_\ominus))$$

where $M \triangle N$ denotes the symmetric difference of both sets.

Now we are able to extend the total ordering on sets of nodes to node valuations:

$$(u, M, e) \prec (v, N, f) \;:\Longleftrightarrow\; \begin{cases} (u \prec v) \text{ or } (u = v \text{ and } M \prec N) \text{ or} \\ (u = v \text{ and } M = N \text{ and } e < f \text{ and } u \in V_{\ominus}) \text{ or} \\ (u = v \text{ and } M = N \text{ and } e > f \text{ and } u \in V_{\oplus}) \end{cases}$$

The motivation behind this ordering is a lexicographic measurement of the profitability of a positional play w.r.t. player 0: the most prominent part of a positional play is the cycle in which the plays eventually stays, and here it is the reward ordering on the dominating cycle node that defines the profitability for player 0. The second important part is the loopless path that leads to the dominating cycle node. Finally, we consider the length, and the intuition behind the definition is that, assuming we have an even-priority dominating cycle node, it is better to reach the cycle quickly whereas it is better to stay as long as possible out of the cycle otherwise.

Given a player 0 strategy $\sigma$, it is player 1's goal to find a best response counterstrategy $\tau$ that minimizes the associated node valuations. A strategy $\tau$ is an *optimal counterstrategy* w.r.t. $\sigma$ iff for every other strategy $\tau'$ and for every node $v$ we have: $\vartheta_{\sigma,\tau,v} \preceq \vartheta_{\sigma,\tau',v}$.

Is is well-known that an optimal counterstrategy always exists and that a witnessing optimal counterstrategy is efficiently computable.

**Lemma 1.** [5] *Let $G$ be a parity game and $\sigma$ be a player 0 strategy. An optimal counterstrategy for player 1 w.r.t. $\sigma$ exists and can be computed in time $\mathcal{O}(|V| \cdot |E|)$.*

For each strategy $\sigma$, a fixed but arbitrary optimal counterstrategy will be denoted by $\tau_\sigma$ from now on. The associated *game valuation* $\Xi_\sigma$ is a map that assigns to each node the node valuation w.r.t. $\sigma$ and $\tau_\sigma$:

$$\Xi_\sigma : v \mapsto \vartheta_{\sigma,\tau_\sigma,v}$$

Game valuations are used to measure the performance of a strategy of player 0: for a fixed strategy $\sigma$ of player 0 and a node $v$, the associated valuation states which is the worst cycle that can be reached from $v$ conforming to $\sigma$ as well as the worst loopless path leading to that cycle (also conforming to $\sigma$).

We also write $v \prec_\sigma u$ to compare the $\Xi_\sigma$-valuations of two nodes, i.e. to abbreviate $\Xi_\sigma(v) \prec \Xi_\sigma(u)$.

A run of the strategy iteration algorithm can be expressed by a sequence of *improving* game valuations; a partial ordering on game valuations is quite naturally defined as follows:

$$\Xi \lhd \Xi' \;:\Longleftrightarrow\; (\Xi(v) \preceq \Xi'(v) \text{ for all } v \in V) \text{ and } (\Xi \neq \Xi')$$

A valuation $\Xi_\sigma$ can be used to create a new strategy of player 0. The strategy iteration algorithm is only allowed to select new strategy decisions for player 0

occurring in the *improvement arena* $\mathcal{A}_{G,\sigma} := (V,\ V_0,\ V_1,\ E',\ \Omega)$ where

$$vE'u :\iff$$
$$vEu \text{ and } (v \in V_1 \text{ or } (v \in V_0 \text{ and } \sigma(v) \preceq_\sigma u))$$

Thus all edges performing worse than the current strategy are removed from the game. A strategy $\sigma$ is *improvable* iff there is a node $v \in V_0$, a node $u \in V$ with $vEu$ s.t. $\sigma(v) \prec_\sigma u$.

An *improvement rule* now selects a strategy for player 0 in a given improvement arena. More formally: an improvement rule is a map $\mathcal{I}_G : \mathcal{S}_0(G) \to \mathcal{S}_0(G)$ fulfilling the following two conditions for every strategy $\sigma$.

(1) For every node $v \in V_0$ it holds that $(v, \mathcal{I}_G(\sigma)(v))$ is an edge in $\mathcal{A}_{G,\sigma}$.
(2) If $\sigma$ is improvable then there is a node $v \in V_0$ s.t. $\sigma(v) \prec_\sigma \mathcal{I}_G(\sigma)(v)$.

**Theorem 2 ([5])** *Let $G$ be a parity game, $\sigma$ be an improvable strategy and $\mathcal{I}_G$ be an improvement rule. Then $\Xi_\sigma \lhd \Xi_{\mathcal{I}_G(\sigma)}$.*

If a strategy is not improvable, the strategy iteration procedure comes to an end and the winning sets for both players as well as associated winning strategies can easily be derived from the given valuation.

**Theorem 3 ([5])** *Let $G$ be a parity game and $\sigma$ be a non-improvable strategy. Then the following holds:*

*(1)* $W_0 = \{v \mid \Xi_\sigma(v) = (w, \_, \_) \text{ and } w \in V_\oplus\}$
*(2)* $W_1 = \{v \mid \Xi_\sigma(v) = (w, \_, \_) \text{ and } w \in V_\ominus\}$
*(3) $\sigma$ is a winning strategy for player 0 on $W_0$*
*(4)* $\tau := \tau_\sigma$ *is a winning strategy for player 1 on $W_1$*
*(5) $\sigma$ is $\trianglelefteq$-optimal*

The strategy iteration starts with an initial strategy $\iota_G$ and runs for a given improvement rule $\mathcal{I}_G$ as outlined in the pseudo-code of Algorithm 1.

**Algorithm 1** Strategy Iteration

```
σ ← ι_G
while σ is improvable do
    σ ← I_G(σ)
end while
return W_0, W_1, σ, τ as in Theorem 3
```

We call player 1 in this setting the *responder*, as it always chooses the best-response counterstrategy against player 0's current strategy, and we call player 0 the *iterator*, since we iterate over a sequence of player 0 strategies to find the equilibrium. Note that by symmetry, both players can change their roles, i.e. strategy

iteration with a player 0 responder and a player 1 iterator finds the same equilibrium.

We note that there are many improvement policies, the most prominent one being the *locally optimizing rule* $\mathcal{I}_G^{\mathtt{loc}}$ due to Jurdziński and Vöge [5].[c] It selects a most profitable strategy decision in every point with respect to the current valuation. More formally, it holds for every strategy $\sigma$, every player 0 node $v$ and every $w \in vE$ that $w \preceq_\sigma \mathcal{I}_G^{\mathtt{loc}}(\sigma)(v)$.

**Lemma 4 ([5])** *The locally optimizing rule can be computed in polynomial time.*

This rule is generally considered to be the most natural choice, particularly because it directly corresponds to the canonical versions of strategy iteration in related parts of game theory like discounted payoff games or simple stochastic games.

## 4. Local Strategy Iteration

There is no strict definition of an algorithm being local. This is due to the variety of algorithms in various application domains. In general, a local algorithm constructs or explores the input on-the-fly whilst executing its computation. In the context of strategy iteration for parity game solving we require a local algorithm to combine on-the-fly expansion of the game graph with the re-use of precomputations. This means that the algorithm should expand the game in an on-the-fly fashion for as long as the winner of a particular input node cannot be determined within the already partially expanded game.

### 4.1. *Game Expansion*

Let $G = (V, V_0, V_1, E, \Omega)$ be a parity game, and $U \subseteq V$. Let $G|_U = (V \cap U, V_0 \cap U, V_1 \cap U, E \cap (U \times U), \Omega|_U)$ denote the *$U$-induced subgame*. Note that $G|_U$ is not necessarily a total graph. Given a node $v \in U$, we define the *$U$-exterior of $v$* as the set of successors that is not contained in $U$, i.e. $D_G(U, v) := vE \cap (V \setminus U)$. The *exterior of $G|_U$* is defined to be the set of all $U$-exteriors, i.e. $D_G(U) := \bigcup_{v \in U} D_G(U, v)$. We say that an $U$-induced subgame $G|_U$ is *expandable* iff $D_G(U) \neq \emptyset$.

We call an $U$-induced subgame a *partially expanded game* $G|_U$ iff $G|_U$ still is a total graph.

Given a partially expanded game $G|_U$, further expansion is based on two selections. First, a non-empty subset of $S \subseteq D_G(U)$ is selected that contains all exterior nodes that are to be expanded. Second, since expanding needs to ensure that we end up with a total subgame again, we need to make a subsequent expansion selection for every node $v \in S$, s.t. at least one successor of every such node is contained in the subgame. Obviously, including further nodes can again lead to further expansion.

[c]Note that in this context, local means only considering neighbouring nodes. The algorithm scheme is still global in the sense that it does not work in an on-the-fly fashion.

Expanding therefore depends on two functions. The *primary expansion function* $\mathcal{E}_1 : 2^V \to 2^V$ selects, given an expandable partially expanded game, a non-empty subset of the exterior, i.e. $\emptyset \subsetneq \mathcal{E}_1(U) \subseteq D_G(U)$ for expandable $U$. The *secondary expansion function* $\mathcal{E}_2 : 2^V \times V \to 2^V$ selects, given an $U$-induced subgame and a node $v \in U$ with $D_G(U, v) \neq \emptyset$, a subset of the $U$-exterior of $v$, i.e. $\mathcal{E}_2(U, v) \subseteq D_G(U, v)$. If $D_G(U, v) = vE$, we additionally require that $\mathcal{E}_2(U, v) \neq \emptyset$.

Let $G$ be a parity game, $\mathcal{E}_1$ be a primary expansion function and $\mathcal{E}_2$ be a secondary expansion function. Given a subset $U \subseteq V$ and a subset $A \subseteq D_G(U)$, we define the *secondary expansion operator* $\textsc{Expand}_{\mathcal{E}_2}(U, A) : 2^V \to 2^V$ by $\textsc{Expand}_{\mathcal{E}_2}(U, A) = U$ if $A = \emptyset$, and otherwise

$$\textsc{Expand}_{\mathcal{E}_2}(U, A) = \textsc{Expand}_{\mathcal{E}_2}(U \cup A, \bigcup_{v \in A} \mathcal{E}_2(U \cup A, v))$$

The intuition behind this operator is that, given an expanded region $U$ and a *recently expanded exterior subset* $A$, we apply the secondary expansion function to every node in $A$, in order to obtain the secondary expanded exterior $A'$. Then, we recursively operate on the expanded region $U \cup A$ and consider $A'$ to be the next recently expanded exterior subset. It is not hard to see that this recursion always terminates.

Given a partially expanded game $G|_U$, we define the *primary expansion operator* $\textsc{Expand}_{\mathcal{E}_1,\mathcal{E}_2} : 2^V \to 2^V$ by $\textsc{Expand}_{\mathcal{E}_1,\mathcal{E}_2}(U) = U$ if $D_G(U) = \emptyset$, and otherwise

$$\textsc{Expand}_{\mathcal{E}_1,\mathcal{E}_2}(U) = \textsc{Expand}_{\mathcal{E}_2}(U, \mathcal{E}_1(U))$$

In other words, the primary expansion operator applies the primary expansion function to obtain an expanded exterior subset, and then applies the secondary expansion operator to it.

### 4.2. *Strategy Iteration*

Let $G$ be a parity game. A tuple $L = (G|_U, \sigma, \tau, \Xi)$ is called *local strategy iteration instance – instance* for short – if $G|_U$ is a partially expanded game, $\sigma$ is a player 0 strategy in $G|_U$, $\tau$ is a player 1 strategy in $G|_U$ s.t. $\tau$ is an optimal counterstrategy against $\sigma$ in $G|_U$ and $\Xi = \Xi_{G|_U,\sigma}$ is an associated game valuation.

Let $G$ be a parity game, $\mathcal{E}_1$, resp. $\mathcal{E}_2$, be primary resp. secondary expansion operators and $\mathcal{I}$ be an improvement rule. We define some operations on instances and leave the parameterization by $G$, $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{I}$ implicit.

The *expansion operation* takes instances with expandable partially expanded games, expands them and recomputes the valuation as well as an optimal counterstrategy. Let $L = (G|_U, \sigma, \tau)$ be an instance s.t. $G|_U$ is expandable. The expansion operation $\textsc{Exp}(L)$ results in a new instance $\textsc{Exp}(L) = (G|_{U'}, \sigma', \tau')$ s.t. $U' = \textsc{Expand}(U)$, $\sigma'|_U = \sigma$, $\sigma'$ is defined on all nodes $U' \cap V_0$ and $\tau'$ is an optimal counterstrategy against $\sigma'$ on $G|_{U'}$. By Lemma 1 we have the following:

**Lemma 5.** *Let $L = (G|_U, \sigma, \tau)$ be an instance. Computing the expansion operation $\textsc{Exp}(L)$ takes time $\mathcal{O}(|U'| \cdot |E \cap (U' \times U')|)$ where $U' = \textsc{Expand}(U)$.*

The *improvement operation* takes instances with 0-improvable strategies $\sigma$, applies the improvement rule to them and recomputes the valuation along with an optimal counterstrategy. Let $L = (G|_U, \sigma, \tau)$ be an instance s.t. $\sigma$ is 0-improvable. The improvement operation $\textsc{Imp}(L)$ results in a new instance $\textsc{Imp}(L) = (G|_U, \sigma', \tau')$ s.t. $\sigma' = \mathcal{I}_{G|_U}(\sigma)$, and $\tau'$ is an optimal counterstrategy against $\sigma'$ on $G|_{U'}$. Again by Lemma 1 we have the following:

**Lemma 6.** *Let $L = (G|_U, \sigma, \tau)$ be an instance. Computing the improvement operation $\textsc{Imp}(L)$ takes time $\mathcal{O}(|U| \cdot |E \cap (U \times U)|)$.*

The next step is to get definitive answers about the winning player of a given node in an instance in the context of the original game. Knowing that player 0 wins node $v$ in an instance, for example, can only be transferred to the original game iff player 1 could not have made better choices that are not available in the partially expanded game.

Let $E^*$ denote the reflexive-transitive closure of the edge set $E$. The *escape set* for player $i$ and node $v \in U$ is the exterior subset that can be reached when conforming to the instance optimal strategy of the other player, i.e. $\mathtt{E}^0_L(v) := vE^*_\tau \cap D_G(U)$ and $\mathtt{E}^1_L(v) := vE^*_\sigma \cap D_G(U)$, where $E_\varrho = \{(v, w) \mid v \in \mathtt{dom}(\varrho) \Rightarrow \varrho(v) = w\}$. The *definitive winning sets* of an instance $L$ are two subsets $W_0(L), W_1(L) \subseteq U$ s.t.

$$W_0(L) = \{v \in U \mid \mathtt{E}^1_L(v) = \emptyset \text{ and } \Xi(v)_1 \text{ even}\}$$
$$W_1(L) = \{v \in U \mid \mathtt{E}^0_L(v) = \emptyset \text{ and } \Xi(v)_1 \text{ odd}\}$$

It is easy to see that definitive winning sets indeed correspond to winning sets in the original game.

**Lemma 7.** *Let $G$ be a parity game and $L$ be an instance. Then $W_0(L) \subseteq W_0(G)$, $W_1(L) \subseteq W_1(G)$, $\sigma$ is a winning strategy on $W_0(L)$ in $G$ and $\tau$ is a winning strategy on $W_1(L)$ in $G$.*

**Proof.** Let $\sigma$ be defined arbitrarily on nodes outside of $L$. Let $v \in W_0(L)$ and let $\varrho$ be an arbitrary player 1 strategy in $G$. The play $\pi_{\sigma,\varrho,v}$ does not leave $L$, by definition of $W_0(L)$. Since $\tau$ is an optimal counterstrategy against $\sigma$ in $L$, it follows that $\vartheta_{\sigma,\tau,v} \preceq \vartheta_{\sigma,\varrho,v}$. Hence, $v \in W_0(G)$. Similarly for $v \in W_1(G)$. □

The membership test $v \in W_i(L)$ may seem to be expensive. However, by keeping an additional data structure that remembers for every player $i$ and every node $v$ in the partially expanded graph whether $v$ cannot reach a node $u \in U \cap V_{1-i}$ with $D_G(U, u) \neq \emptyset$ by following player $i$'s strategy, a dynamic update procedure can be applied that maintains this data structure automatically when changing the strategies $\sigma$ and $\tau$ with the same asymptotic complexity as updating the associated valuation.

Now let $G$ be a parity game and $v^*$ be a designated node. The local strategy iteration algorithm works as outlined in the pseudo-code of Algorithm 2.

**Algorithm 2** Local Strategy Iteration

```
1: U ← Expand({v*})
2: σ ← arbitrary 0-strategy on G|_U
3: τ ← optimal 1-strategy against σ on G|_U
4: L ← (U, σ, τ)
5: while v* ∉ W_0(L) ∪ W_1(L) do
6:     if σ is 0-improvable w.r.t L then
7:         L ← Imp(L)
8:     else
9:         L ← Exp(L)
10:    end if
11: end while
12: return σ, τ, W_0(L), W_1(L)
```

**Lemma 8.** *The Local Strategy Iteration terminates, determines whether* $v^* \in W_0(G)$ *or* $v^* \in W_1(G)$ *and returns a witnessing winning strategy for the respective player. Particularly, every improvement step can be performed in time* $\mathcal{O}(|V| \cdot |E|)$.

We specify two variants of local strategy iteration in the following by specifying expansion functions. We do not commit to a particular improvement rule as it is usually done for global strategy iteration, since it is an open question whether there is a particularly clever one.

### 4.3. *Asymmetric Algorithm*

Global strategy iteration is formulated w.r.t. one player $i$: starting with a strategy for player $i$, it iteratively improves it until it becomes a winning strategy on some part of the game graph. It is based on the fact that a counterstrategy of the opposing player is always globally optimal. This does not hold in general for local strategy iteration. However, if we always include *every* successor of player 1 controlled nodes, the property still holds.

The primary expansion function considers the parity of $\Xi(v^*)_1$ in order to determine where to expand. If $\Xi(v^*)_1$ is even but $v^* \notin W_0(L)$, it must be the case that the escape of player 1 $\mathtt{E}^1_L(v^*)$ is not empty. Hence, it selects one of the elements of player 1's escape set. Otherwise, if $\Xi(v^*)_0$ is odd but $v^* \notin W_1(L)$, it follows that the escape $\mathtt{E}^0_L(v^*)$ of player 0 is not empty, hence we select one of those.

Formally, the primary expansion function is $\mathcal{E}_1(U) = w$ for some $w$ with $w \in \mathtt{E}^{1-i}_L(v^*)$ and $\Xi_1(v^*) \equiv_2 i$.

The secondary expansion function includes, given a node $v$, every successor of $v$ if it is controlled by player 1, one single successor if it is controlled by player 0 with no successors already in $L$ and no successor otherwise.

Formally, the secondary expansion function is $\mathcal{E}_2(U, w) = wE \setminus U$ if $w \in V_1$, $\mathcal{E}_2(U, w) = u$ for some $u \in wE$ if $w \in V_0$ and $wE \cap U = \emptyset$, and $\mathcal{E}_2(U, w) = \emptyset$

otherwise.

The asymmetric expansion functions yield an upper bound on the number of iterations that almost matches the trivial upper bound for global strategy iteration, $|V|^{|V_0|}$, which is the number of strategies. Since we include every player 1 successor here for every contained player 1 node, it is the case that valuations are never decreasing. Hence, we get the same upper bound for the number of improving steps plus the number of potential expansion steps.

**Lemma 9.** *The asymmetric local strategy iteration requires at most* $|V| + |V|^{|V_0|}$ *iterations.*

The essential idea behind the asymmetric algorithm is to bound the number of necessary iterations as closely as possible to the number of iterations required by global strategy iteration without compromising the goal of extending the game only when necessary.

However, this algorithm is counterproductive if player 1 wins $v^*$: if local strategy iteration is done for one player only, and the opponent is the winner of the designated input node, then the algorithm would have to expand the entire reachable part of the game graph and it would therefore be at most as good as global strategy iteration.

The *asymmetric algorithm* therefore executes strategy iteration *simultaneously* for both players. The first process that yields a definitive answer for $v^*$ yields a termination of the whole algorithm. Also, the proposed optimization that removes definitive winning sets from the known game can be shared between the two processes to speed up the overall procedure.

### 4.4. *Symmetric Algorithm*

The symmetric algorithm is based on a single process that does not distinguish between the two players. Instead, it expands again a single node of the escape sets and only includes further nodes if that is necessary to keep a total graph.

Formally, the primary expansion function is $\mathcal{E}_1(U) = w$ for some $w$ with $w \in \mathtt{E}_L^{1-i}(v^*)$ and $\Xi_1(v^*) \equiv_2 i$ again. The secondary expansion function is here $\mathcal{E}_2(U, w) = u$ for some $u \in wE$ if $wE \cap U = \emptyset$, and $\mathcal{E}_2(U, w) = \emptyset$ otherwise.

The symmetric expansion functions now yield an upper bound on the number of iterations that is much higher than in the symmetric case: now, it may happen that the valuation degrades when expanding the game. Hence, we get a much worse bound on the number of iterations.

**Lemma 10.** *The symmetric local strategy iteration requires at most* $|V| \cdot |V|^{|V_0|}$ *iterations.*

On the other hand, it is well-known that strategy iteration requires only a few iterations in practice. Moreover, symmetric local strategy iteration expands more moderately than the asymmetric one, and hence, could be more efficient in situations in which the computation of the transition relation is very expensive.

## 5. Experimental Evaluation

We report on the performance of both local SI algorithm in practice. We compare it in particular to two other algorithms for parity games: global SI by Jurdziński and Vöge [5] and Stevens and Stirling's local $\mu$-calculus model checker [9]. Local SI is supposed to remedy one of global SI's distinct disadvantages, hence this comparison. Note that both global SI algorithms [5, 13] perform quite well in practice although Zielonka's recursive algorithm [14] is usually even a bit better [10]. However, it is the case that, whenever local SI is faster than global SI, it is also faster than the recursive algorithm. We also remark that the recursive algorithm is inherently global in nature, and we know of no idea how to create a local variant of this one. Furthermore, the parity game solving algorithm derived from the local model checking algorithm is of course a local algorithm already. It is therefore also a natural contender for local SI. Previous tests have shown that it can be much slower than the global algorithms [10]. This comparison therefore shows that this is not due to the locality but to the solving technique.

Both local SI have been implemented in PGSolver,[d] a publicly available tool for solving parity games which is written in OCaml and which contains implementations of the two competing algorithms as well as many others. Furthermore, PGSolver uses a specialized routine for global solving called *generic solver*. It does not feed the entire game graph to the chosen solving algorithm but performs optimizing preprocessing steps like decomposition into SCCs and removal of trivial winning regions etc. These greatly speed up global solving [10]. The runtime results of the global algorithm reported here are obtained with all optimizing preprocessing steps. On the other hand, these steps do not make sense in the context of local solving because the removal of trivial winning regions for instance is in fact a global operation. Hence, we do in fact compare local SI as it is, against global SI embedded in this optimizing framework. Local SI is run in these experiments with the standard all-max improvement rule on the respective improvement sets.

All tests have been carried out on a 64-bit machine with four quad-core Opteron™ CPUs and 128GB RAM space. As benchmarks we use some handcrafted verification problems and random games. We report on the times needed to solve the games (globally vs. locally) as well as the number of nodes in the games that have been visited. This is an important parameter indicating the memory consumption of the algorithms on these benchmarks. The times for global SI are composed of the time it takes to generate the parity game (which is not part of the solving, strictly speaking, but needs to be considered too in comparison to the local algorithms since they generate the game while they are solving it) and the time it takes to perform the strategy iteration. We only present instances of non-negligible running times. On the other hand, the solving of larger instances not presented in the figures anymore has experienced time-outs after one hour, marked †.

[d] `http://www.tcs.ifi.lmu.de/pgsolver`

| | | | global SI | | | local MC | | asymmetric SI | | symmetric SI | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n$ | \|Game\| | $t_{\text{generate}}$ | $t_{\text{iterate}}$ | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ |
| FIFO elevator | 3 | 603 | 0.02s | 0.00s | 0.02s | 603 | 0.12s | 603 | 0.28s | 603 | 0.31s |
| | 4 | 3,120 | 0.08s | 0.02s | 0.10s | 3,120 | 2.32s | 3,120 | 8.66s | 3,120 | 8.57s |
| | 5 | 19,263 | 0.66s | 0.25s | 0.91s | 19,263 | 206.32s | 19,263 | 414.55s | 19,263 | 443.73s |
| | 6 | 138,308 | 5.64s | 2.68s | 8.32s | – | † | – | † | – | † |
| | 7 | 1,130,884 | 57.57s | 35.94s | 93.51s | – | † | – | † | – | † |
| LIFO elevator | 5 | 20,126 | 0.80s | 0.34s | 1.14s | 1,237 | 0.02s | 404 | 0.03s | 658 | 0.06s |
| | 6 | 142,720 | 7.30s | 4.13s | 11.43s | 3,195 | 0.07s | 581 | 0.06s | 809 | 0.10s |
| | 7 | 1,154,799 | 83.45s | 66.91s | 150.36s | 22,503 | 0.66s | 806 | 0.09s | 974 | 0.17s |
| | 8 | 10,505,651 | 2342.61s | 3596.42s | 5939.03s | 60,247 | 2.11s | 1,085 | 0.15s | 1,153 | 0.29s |

(a) $\mu$-Calculus Model Checking

| | | | global SI | | | local MC | | asymmetric SI | | symmetric SI | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n$ | \|Game\| | $t_{\text{generate}}$ | $t_{\text{iterate}}$ | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ |
| Philo-sophers | 9 | 1,084,808 | 29.48s | 28.19s | 57.67s | 33,599 | 0.95s | 3,439 | 0.75s | 8,489 | 1.89s |
| | 10 | 3,615,173 | 177.93s | 329.53s | 507.46s | 90,101 | 3.05s | 4,374 | 1.21s | 11,357 | 3.57s |
| | 11 | 11,927,966 | 1427.76s | 2794.55s | 4222.31s | 201,566 | 8.16s | 5,450 | 2.04s | 14,817 | 5.56s |
| | 12 | ? | † | – | – | 854,922 | 70.38s | 6,673 | 4.32s | 18,925 | 12.78s |
| | 13 | ? | † | – | – | 2,763,751 | 525.45s | 8,052 | 6.82s | 23,737 | 18.73s |
| | 14 | ? | † | – | – | – | † | 9,595 | 10.00s | 29,309 | 29.44s |

(b) LTL Model Checking

| | | global SI | local MC | | asymmetric SI | | symmetric SI | |
|---|---|---|---|---|---|---|---|---|
| | \|Game\| | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ | visited | $t_{\text{solve}}$ |
| Random games | 1,000 | 0.03s | 37.93 | 0.00s | 93.02 | 0.01s | 78.43 | 0.01s |
| | 2,000 | 0.07s | 103.99 | 0.01s | 211.05 | 0.02s | 199.84 | 0.02s |
| | 5,000 | 0.24s | 159.14 | 0.11s | 255.78 | 0.03s | 234.43 | 0.03s |
| | 10,000 | 0.33s | 592.42 | 0.95s | 430.36 | 0.08s | 332.56 | 0.04s |
| | 20,000 | 0.77s | 6794.45 | 8.73s | 453.05 | 0.10s | 446.92 | 0.08s |
| | 50,000 | 2.98s | – | † | 669.90 | 0.26s | 532.61 | 0.21s |
| | 100,000 | 6.53s | – | † | 1001.80 | 0.48s | 743.98 | 0.32s |
| | 200,000 | 17.92s | – | † | 1154.40 | 1.15s | 942.29 | 0.78s |
| | 500,000 | 68.33s | – | † | 3443.40 | 5.82s | 2016.93 | 3.83s |

(c) Random Games

Fig. 1. Runtime results.

**$\mu$-Calculus Model Checking.** As a first benchmark we use the problem of verifying an elevator system. It serves $n$ floors and maintains a list of requests which may contain each floor at most once. A detailed description can be found in [10].

The task is to verify that all paths satisfy the following fairness property: if the top floor is infinitely often requested then it is infinitely often served. This property can be formalized in the modal $\mu$-calculus with alternation depth 2, and the resulting benchmarking parity games are obtained as $\mu$-calculus model checking games.

We consider two different implementations of the system, differing in the way that the request list is stored: in FIFO or in LIFO style. Note that the FIFO implementation satisfies this fairness property because any requested floor will eventually be served whereas the LIFO implementation does not satisfy it. This is because new requests will be served before older ones and therefore requests can starve.

With the FIFO system, all local algorithms need to explore the whole game in order to detect that the universally path quantified formula is satisfied or, equivalently, to find a winning strategy for player 0 which comprises the entire game graph. It is not too surprising that global SI with all its optimizations outperforms all of them in this case. But note that on the LIFO system, local algorithms explore only a tiny part of the whole game and hence outperform the global algorithm. Moreover, both local SI algorithms are much faster than the local model checker. See Fig. 1(a) for all the runtime results.

**LTL Model Checking.** We take the well-known dining philosophers problem as a system to be verified. It consists of $n$ forks, each of which can be in three different states (on the table or held by the left or right philosopher). Transitions are being done by lifting up or placing down one fork. Philosopher $i$ eats as soon as he holds two forks. We check the LTL formula $\bigvee_{i=1}^{n} \mathtt{FG}\mathit{eats}_i$ on the system which asks for a scheduling of the fork lifting that enables one of them to eventually eat all the time. Note that LTL model checking can also be viewed as parity game solving [7].

Here the local algorithms only need to find a particular scheduling out of all possible ones which enables one of the philosophers to eventually eat all the time. Both the local SI and the local model checker outperform the global algorithm, again just because they only explore a small part of the whole game graph. Furthermore, both local SI are remarkably more efficient than the local model checker in these test cases, i.e. it explores only a fractional part of the entire game in comparison and therefore needs much less time. See Fig. 1(b) for all the runtime results.

**Random Games.** Finally, we consider games generated by a random model distributed with PGSolver. It creates small random games from a uniform distribution and connects them randomly in order to obtain a non-trivial SCC structure and therefore to prevent the creation of atypically special games [10]. The initial node is the – arbitrarily chosen – first node in the games representation. The results in Fig. 1(c) show that local SI algorithms are again much better than both global SI and the local model checker on these games. Note that the number of visited nodes given there – as well as the running times – is the average over 100 runs and therefore not necessarily an integer value. Also, these random games cannot be created locally. Thus, the times given for all algorithms are the pure solving times

without generation.

## 6. Conclusions and Further Work

We have presented two on-the-fly strategy iteration algorithms for solving parity games and shown that they can outperform existing global and even other local methods by several orders of magnitude. To be precise, the local SI algorithms presented here beat global algorithms in cases where a winning strategy does not stretch over the entire game graph, even when the global algorithms are embedded into the optimizing framework of the generic engine in PGSolver.

It remains to see whether or not some of the optimizations that help global solving significantly [10] can be used for tuning local SI. Another natural question arising with this work is how to localize other global solvers, in particular Zielonka's recursive one [14] and Jurdziński's small progress measures algorithm [15].

**Acknowledgment.** We would like to thank the anonymous referees for their thorough reports containing many comments that helped to improve the presentation of this paper.

## References

[1] C. Stirling. Local model checking games. In *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[2] O. Friedmann and M. Lange. Tableaux with automata. In *Proc. Workshop on Tableaux vs. Automata as Logical Decision Procedures, AutoTab'09*, 2009.

[3] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 303(1):7–34, 2003.

[4] M. Jurdziński. Deciding the winner in parity games is in $UP \cap$co-$UP$. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[5] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proc. 12th Int. Conf. on Computer Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.

[6] A. Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, University of California, Berkeley, 1995.

[7] M. Lange and C. Stirling. Model checking games for branching time logics. *Journal of Logic and Computation*, 12(4):623–639, 2002.

[8] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal $\mu$-calculus. In *Proc. 2nd Int. Workshop on Tools and Alg. for Constr. and Analysis of Systems, TACAS'96*, volume 1055 of *LNCS*, pages 107–126. Springer, 1996.

[9] P. Stevens and C. Stirling. Practical model-checking using games. In *Proc. 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[10] O. Friedmann and M. Lange. Solving parity games in practice. In *Proc. 7th Int. Symp. on Automated Technology for Verification and Analysis, ATVA'09*, volume 5799 of *LNCS*, pages 182–196, 2009.

[11] O. Friedmann and M. Lange. Local strategy improvement for parity game solving. In *First International Symposium on Games, Automata, Logics and Formal Verification, GandALF'10*, volume 25 of *EPTCS*, pages 118–131, 2010.

[12] E. A. Emerson and C. S. Jutla. Tree automata, $\mu$-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377. IEEE, 1991.

[13] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. 17th Ann. Conf. on Computer Science Logic, CSL'08*, volume 5213 of *LNCS*, pages 369–384. Springer, 2008.

[14] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1–2):135–183, 1998.

[15] M. Jurdziński. Small progress measures for solving parity games. In *Proc. 17th Ann. Symp. on Theoretical Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.