# A Discrete Strategy Improvement Algorithm for Solving Parity Games

**(Extended Abstract)**

Jens Vöge[1,⋆] and Marcin Jurdziński[2,⋆⋆]

[1] Lehrstuhl für Informatik VII
RWTH Aachen, D-52056 Aachen, Germany
`voege@informatik.rwth-aachen.de`

[2] BRICS[⋆⋆⋆]
Department of Computer Science, University of Århus,
Ny Munkegade Bldg. 540, DK-8000 Århus C, Denmark,
`mju@brics.dk`

**Abstract.** A discrete strategy improvement algorithm is given for constructing winning strategies in parity games, thereby providing also a new solution of the model-checking problem for the modal $\mu$-calculus. Known strategy improvement algorithms, as proposed for stochastic games by Hoffman and Karp in 1966, and for discounted payoff games and parity games by Puri in 1995, work with real numbers and require solving linear programming instances involving high precision arithmetic. In the present algorithm for parity games these difficulties are avoided by the use of discrete vertex valuations in which information about the relevance of vertices and certain distances is coded. An efficient implementation is given for a strategy improvement step. Another advantage of the present approach is that it provides a better conceptual understanding and easier analysis of strategy improvement algorithms for parity games. However, so far it is not known whether the present algorithm works in polynomial time. The long standing problem whether parity games can be solved in polynomial time remains open.

## 1 Introduction

The study of the computational complexity of solving parity games has two main motivations. One is that the problem is polynomial time equivalent to the modal $\mu$-calculus model checking [7, 18], and hence better algorithms for parity games may lead to better model checkers, which is a major objective in computer aided verification.

The other motivation is the intriguing status of the problem from the point of view of structural complexity theory. It is one of the few natural problems which

⋆ Supported by the Deutsche Forschungsgemeinschaft (DFG), project Th 352/5-3.

⋆⋆ A part of this work was done while the author stayed at Lehrstuhl für Informatik VII, RWTH Aachen, Germany.

⋆⋆⋆ Basic Research in Computer Science, Centre of Danish Science Foundation.

is in NP $\cap$ co-NP [7] (and even in UP $\cap$ co-UP [9]), and is not known to have a polynomial time algorithm, despite substantial effort of the community (see [7, 1, 17, 10] and references therein). Other notable examples of such problems include simple stochastic games [3, 4], mean payoff games [5, 21], and discounted payoff games [21]. There are polynomial time reductions of parity games to mean payoff games [15, 9], mean payoff games to discounted payoff games [21], and discounted payoff games to simple stochastic games [21]. Parity games, as the simplest of them all, seem to be the most plausible candidate for trying to find a polynomial time algorithm.

A strategy improvement algorithm has been proposed for solving stochastic games by Hoffman and Karp [8] in 1966. Puri in his PhD thesis [15] has adapted the algorithm for discounted payoff games. Puri also provided a polynomial time reduction of parity games to mean payoff games, and advocated the use of the algorithm for solving parity games, and hence for the modal $\mu$-calculus model checking.

In our opinion Puri's strategy improvement algorithm for parity games has two drawbacks.

- The algorithm uses high precision arithmetic, and needs to solve linear programming instances: both are typically costly operations. An implementation (by the first author) of Puri's algorithm, using a linear programming algorithm of Meggido [12], proved to be prohibitively slow.
- Solving parity games is a discrete, graph theoretic problem, but the crux of the algorithm is manipulation of real numbers, and its analysis is crucially based on continuous methods, such as Banach's fixed point theorem.

The first one makes the algorithm inefficient in practice, the other one obscures understanding of the algorithm.

Our discrete strategy improvement algorithm remedies both above-mentioned shortcomings of Puri's algorithm, while preserving the overall structure of the generic strategy improvement algorithm. We introduce discrete values (such as tuples of vertices, sets of vertices and natural numbers denoting lengths of paths in the game graph) which are being manipulated by the algorithm, instead of their encodings into real numbers. (One can show a precise relationship between behaviour of Puri's and our algorithms; we will treat this issue elsewhere.)

The first advantage of our approach is that instead of solving linear programming instances involving high precision arithmetic, we only need to solve instances of a certain purely discrete problem. Moreover, we develop an algorithm exploiting the structure of instances occurring in this context, i.e., relevance of vertices and certain distance information. In this way we get an efficient implementation of a strategy improvement algorithm with $O(nm)$ running time for one strategy improvement step, where $n$ is the number of vertices, and $m$ is the number of edges in the game graph.

The other advantage is more subjective: we believe that it is easier to analyze the discrete data maintained by our algorithm, rather than its subtle encodings into real numbers involving infinite geometric series [15]. The classical continuous reasoning gives a relatively clean proof of correctness of the algorithm in a more general case of discounted payoff games [15], but we think that in the case of parity games it blurs an intuitive understanding of the underlying discrete structure. However, the long standing open question whether a strategy improvement algorithm works in polynomial time [4] remains unanswered. Nevertheless, we hope that our discrete analysis of the algorithm may help either to find a proof of polynomial time termination, or to come up with a family of examples on which the algorithm requires exponential number of steps. Any of those results would mark a substantial progress in understanding the computational complexity of parity games.

So far, for all families of examples we have considered the strategy improvement algorithm needs only linear number of strategy improvement steps. Notably, a linear number of strategy improvements suffices for several families of difficult examples for which other known algorithms need exponential time.

In this extended abstract, some substantial proofs are omitted, in particular for Lemmas 3 and 5, as well as the detailed correctness proof of the algorithm of Section 4. For a more complete exposition see [19].

**Acknowledgement**

We are indebted to Wolfgang Thomas for his invaluable advice, support, and encouragement.

## 2 Preliminaries

A *parity game* is an infinite two-person game played on a finite vertex-colored graph $G = (V_0, V_1, E, c)$ where $V = V_0 \dot\cup V_1$ is the set of vertices and $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$ the set of edges with $\forall u \in V\ \exists v \in V : uEv$, and $c : V \to \{1, \dots, d\}$ is a coloring of the vertices. The two players move, in alternation, a token along the graph's edges; player 0 moves the token from vertices in $V_0$, player 1 from vertices in $V_1$. A play is an infinite vertex sequence $v_0 v_1 v_2 \dots$ arising in this way. The decision who wins refers to the coloring $c$ of the game graph: if the largest color which occurs infinitely often in $c(v_0)c(v_1)c(v_2)\dots$ is even then player 0 wins, otherwise player 1 wins. One says that player 0 (resp. player 1) has a winning strategy from $v \in V$ if starting a play in $v$, he can force a win for arbitrary choices of player 1 (resp. player 0). By the well-known determinacy of parity games, the vertex set $V$ is divided into the two sets $W_0, W_1$, called the winning sets, where $W_i$ contains the vertices from which player $i$ has a winning strategy.

Moreover, it is known (see, e.g., [6, 14]) that on $W_i$ player $i$ has a memoryless winning strategy, which prescribes the respective next move by a unique edge leaving each vertex in $V_i$.

In the following we fix the basic notations for games and strategies. Let $G = (V_0, V_1, E, c)$ be a game graph with the vertices $V = V_0 \cup V_1$ and the coloring $c : V \to \{1, \dots d\}$. Let $W_0, W_1$ be the winning sets of $G$.

A strategy for player $i$ ($i = 0, 1$) is a function $\rho : V_i \to V_{1-i}$ such that $vE\rho(v)$ for all $v \in V_i$. A strategy for player $i$ is called a winning strategy on $W \subseteq V$ if player $i$ wins every play starting in $W$ when using that strategy. A *winning strategy* for player $i$ is a winning strategy on the winning set $W_i$.

We set

$$V_+ = \{v \in V \mid c(v) \textit{ is even}\} \text{ and } V_- = \{v \in V \mid c(v) \textit{ is odd}\}$$

and we call vertices in $V_+$ positive and those in $V_-$ negative.

We introduce two orderings of $V$, the relevance ordering $<$ and the reward ordering $\prec$. The relevance order is a total order extending the pre-order given by the coloring, i.e., such that $c(u) < c(v)$ implies $u < v$. (So higher colors indicate higher relevance.)

By $\inf(\pi)$ we denote the set of vertices occurring infinitely often in the play $\pi$. Referring to $<$ over $V$, we can reformulate the winning condition for player 0 in a play $\pi$ as

$$\max\nolimits_<(\inf(\pi)) \in V_+.$$

Another ordering, the reward order $\prec$, indicates the value of a vertex as seen from player 0. The lowest reward originates from the vertex in $V_-$ of highest relevance, the highest from the vertex in $V_+$ of highest relevance. Formally, we define:

$$u \prec v \iff (u < v \wedge v \in V_+) \vee (v < u \wedge u \in V_-)$$

We extend the reward order $\prec$ to an order on sets of vertices. If $P, Q \in 2^V$ are distinct, the vertex $v$ of highest relevance in the symmetric difference $P \triangle Q$ decides whether $P \prec Q$ or $Q \prec P$: If $v \in V_+$ then the set containing $v$ is taken as the higher one, if $v \in V_-$ then the set not containing $v$ is taken as the higher one. Formally:

$$P \prec Q \iff P \neq Q \;\wedge\; \max\nolimits_<(P \triangle Q) \in Q \triangle V_-$$

Note that we use the same symbol $\prec$ for vertices and for sets of vertices.

We also need a coarser version $\prec_w$ of $\prec$, using a reference vertex $w$ and taking into account only $P$- and $Q$-vertices of relevance $\geq w$:

$$P \prec_w Q \iff P \cap \{x \in V | x \geq w\} \prec Q \cap \{x \in V | x \geq w\},$$

and the corresponding equivalence relation:

$$P \sim_w Q \iff P \preceq_w Q \;\wedge\; Q \preceq_w P$$

## 3 Game Graph Valuations

In this section we present the terminology and main ideas underlying the approximative construction of winning strategies. The plays considered in the sequel are always induced by two given strategies $\sigma$ and $\tau$ for players 0 and 1 respectively. Any such pair $\sigma, \tau$ determines from any vertex a play ending in a loop $L$. The first Subsection 3.1 introduces certain values called play profiles for such plays. A play profile combines three data: the most relevant vertex of the loop $L$, the vertices occurring in the play that are more relevant than the most relevant vertex in the loop, and the number of all vertices that are visited before the most relevant vertex of the loop. The order $\prec$ above is extended to play profiles (so that $\prec$-higher play profiles are more advantageous for player 0). An arbitrary assignment of play profile values to the vertices is called a valuation.

Subsection 3.2 gives a certain condition when a valuation originates from a strategy pair $(\sigma, \tau)$ as explained. Such valuations are called locally progressive.

The algorithm will produce successively such locally progressive valuations. In each step, a valuation is constructed from a strategy $\sigma$ of player 0 and an 'optimal' response strategy $\tau$ of player 1. In Subsection 3.3 this notion of optimality is introduced. We show that a valuation which is optimal for both players 0 and 1 represents the desired solution of our problem: a pair of winning strategies for the two players.

The last Subsection 3.4 will explain the nature of an approximation step, from a given valuation $\varphi$ to a (next) 'response valuation' $\varphi'$. One should imagine that player 0 picks edges to vertices with $\prec$-maximal values given by $\varphi$, and that player 1 responds as explained above, by a locally progressive valuation $\varphi'$. The key lemma says that uniformly $\varphi'$ will majorize $\varphi$ (i.e., $\varphi(v) \preceq \varphi'(v)$ for all vertices $v \in V$). This will be the basis for the termination proof because equality $\varphi = \varphi'$ will imply that $\varphi$ is already optimal for both players.

## 3.1 Play Profiles

Let $G = (V_0, V_1, E, c)$ be a game graph. Let $\Pi$ be the set of plays that can be played on this graph. We define the function $w : \Pi \to V$ which computes the most relevant vertex that is visited infinitely often in a play $\pi$, i.e., $w(\pi) = \max_< \inf(\pi)$. Furthermore, we define a function $\alpha : \Pi \to 2^V$ which computes the vertices that are visited before the first occurrence of $w(\pi)$:

$$\alpha(\pi) = \{u \in V \mid \exists i \in \mathbb{N}_0 : \pi_i = u \wedge \forall k \in \mathbb{N}_0 : k \leq i \Longrightarrow \pi_k \neq w(\pi)\}.$$

We are interested in three characteristic values of a play $\pi$:

1. The most relevant vertex $u_\pi$ that is visited infinitely often: $u_\pi = w(\pi)$.
2. The set of vertices $P_\pi$ that are more relevant than $u_\pi$ and visited before the first visit of $u_\pi$: $P_\pi = \alpha(\pi) \cap \{v \in V \mid v > w(\pi)\}$.
3. The number of vertices visited before the first visit of $u_\pi$: $e_\pi = |\alpha(\pi)|$.

We call such a triple $(u_\pi, P_\pi, e_\pi)$ a *play profile*. It induces an equivalence relation of plays on the given game graph. By definition each profile of a play induced by a pair of strategies belongs to the following set:

$$\mathcal{D} = \{(u, P, e) \in V \times 2^V \times \{0, \dots, |V| - 1\} \mid \forall v \in P : u < v \ \wedge \ |P| \leq e\}.$$

The set $\mathcal{D}$ is divided into profiles of plays that are won by player 0, respectively 1:

$$\mathcal{D}_0 = \{(u, P, e) \in \mathcal{D} \mid u \in V_+\} \quad \text{and} \quad \mathcal{D}_1 = \{(u, P, e) \in \mathcal{D} \mid u \in V_-\}.$$

We define a linear ordering $\prec$ on $\mathcal{D}$. One play profile is greater than another with respect to this ordering if the plays it describes are 'better' for player 0, respectively smaller if they are better for player 1. Let $(u, P, e), (v, Q, f) \in \mathcal{D}$:

$$(u, P, e) \prec (v, Q, f) \iff \begin{cases} \quad u \prec v \\ \vee\ (u = v \wedge P \prec Q) \\ \vee\ (u = v \wedge P = Q \wedge v \in V_- \wedge e < f) \\ \vee\ (u = v \wedge P = Q \wedge v \in V_+ \wedge e > f) \end{cases}$$

The idea behind the last two clauses is that in case $u = v$ and $P = Q$ it is more advantageous for player 0 to have a shorter distance to the most relevant vertex $v$ if $v \in V_+$ (case $f < e$), resp. a longer distance if $v \in V_-$ (case $f > e$).

For the subsequent lemmata we need a coarser ordering $\prec_w$ of play profiles:

$$(u, P, e) \prec_w (v, Q, f) \iff u \prec v \ \vee\ (u = v \ \wedge\ P \prec_w Q)$$

and a corresponding equivalence relation $\sim_w$:

$$(u, P, e) \sim_w (v, Q, f) \iff u = v \ \wedge\ P \sim_w Q$$

## 3.2 Valuations

A *valuation* is a function $\varphi : V \to \mathcal{D}$ which labels each vertex with a play profile.

We are interested in valuations where all play profiles $\varphi(v)$ $(v \in V)$ are *induced* by the same pair of strategies. An initial vertex $v$ and two strategies $\sigma$ for player 0 and $\tau$ for player 1 determine a unique play $\pi_{v,\sigma,\tau}$. For a pair of strategies $(\sigma, \tau)$ we can define the *valuation induced by* $(\sigma, \tau)$ to be the function $\varphi$ which maps $v \in V$ to the play profile of $\pi_{v,\sigma,\tau}$. This valuation $\varphi$ assigns to each vertex the play profile of the play starting in $v$ played with respect to $\sigma$ and $\tau$. To refer to the components of a play profile $\varphi(v) = (u, P, e)$ we write $\varphi_0$, $\varphi_1$ and $\varphi_2$, where $\varphi_0(v) = u$, $\varphi_1(v) = P$ and $\varphi_2(v) = e$. We call $u$ the most relevant vertex of play profile $\varphi(v)$ (or of $v$, if $\varphi$ is clear from the context).

The play profiles in a valuation induced by a strategy pair are related as follows: Let $\sigma, \tau$ be strategies and $\varphi$ their corresponding valuation. Let $x, y \in V$ be two vertices with $\sigma(x) = y$ or $\tau(x) = y$, i.e., in a play induced by $\sigma$ and $\tau$ a move proceeds from $x$ to $y$. It follows immediately that $\varphi_0(x) = \varphi_0(y)$. We can distinguish the following cases:

1. Case $x < \varphi_0(x)$: Then $\varphi_1(x) = \varphi_1(y)$ and $\varphi_2(x) = \varphi_2(y) + 1$.
2. Case $x = \varphi_0(x)$: By definition of $\varphi$ we have: $\varphi_1(x) = \emptyset$ and $\varphi_2(x) = 0$. Furthermore $\varphi_1(y) = \emptyset$, because there are no vertices on the loop through $x$ that are more relevant than $x$.
3. Case $x > \varphi_0(x)$: Then $\varphi_1(x) = \{x\} \mathbin{\dot\cup} \varphi_1(y)$ and $\varphi_2(x) = \varphi_2(y) + 1$.

These conditions define what we call the $\varphi$*-progress* relation $\lhd_\varphi$. If an edge leads from $x$ to $y$ then $x \lhd_\varphi y$ will mean that the $\varphi$-value is correctly updated when passing from $x$ to $y$ in a play. Formally we define for $x, y \in V$, assuming $\varphi(x) = (u, P, e)$, $\varphi(y) = (v, Q, f)$:

$$\begin{aligned} x \lhd_\varphi y \iff u = v \wedge \big( & (x = u \wedge P = Q = \emptyset \quad \wedge e = 0) \\ & \vee (x < u \wedge P = Q \qquad \wedge e = f + 1) \\ & \vee (x > u \wedge P = Q \mathbin{\dot\cup} \{x\} \wedge e = f + 1)\big) \end{aligned}$$

The following proposition is straightforward.

**Proposition 1.** *Let $\varphi$ be a valuation, and let $v \lhd_\varphi \sigma(v)$ and $v \lhd_\varphi \tau(v)$, for all $v \in V$. Then $(\sigma, \tau)$ induces $\varphi$.*

Note that a valuation $\varphi$ may be induced by several pairs $(\sigma, \tau)$ of strategies so that several plays starting in $v$ with play profile $\varphi(v)$ may exist. We now characterize those valuations which are induced by some strategy pair $(\sigma, \tau)$. A valuation $\varphi$ is called *locally progressive* if

$$\forall u \in V \; \exists v \in V : uEv \wedge u \lhd_\varphi v.$$

The next proposition follows immediately from definitions.

**Proposition 2.** *Let $G = (V_0, V_1, E, c)$ be a game graph. Let $\varphi$ be a valuation for $G$. Then $\varphi$ is a locally progressive valuation iff there exists a strategy $\sigma$ for player 0 and a strategy $\tau$ for player 1 such that $\varphi$ is a valuation induced by $(\sigma, \tau)$.*

We call a strategy $\rho : V_i \to V_{1-i}$ $(i \in \{0, 1\})$ *compatible* with the valuation $\varphi$ if $\forall v \in V_i : v \lhd_\varphi \rho(v)$. From Proposition 2 it follows that for a locally progressive valuation $\varphi$ at least one strategy for each player is compatible with $\varphi$.

## 3.3 Optimal Valuations

A valuation $\varphi$ is called *optimal* for player 0 if the $\varphi$-progress relation $x \lhd_\varphi y$ only applies to edges $xEy$ where the value $\varphi(y)$ is $\preceq$-maximal among the $E$-successors of $x$ (i.e., among the values $\varphi(z)$ with $xEz$). However, a weaker requirement is made if $x$ is the most relevant vertex associated to $x$ via $\varphi$ (i.e., $\varphi_0(x) = x$). In this case we discard the distance component $\varphi_2$; formally, we replace $\preceq$ by $\preceq_x$. (Recall that $(u, P, e) \prec_x (v, Q, f)$ holds if $u \prec v$ holds, or $u = v$ and $P \prec_x Q$, i.e., $e$ and $f$ are not taken into account.) Formally, $\varphi$ is called optimal for player 0 if for all $x \in V_0$ and $y \in V_1$ with an edge $xEy$:

$$x \lhd_\varphi y \iff \forall z \in V_1 : xEz \Rightarrow \varphi(z) \preceq \varphi(y) \;\vee\; \big(\varphi_0(x) = x \;\wedge\; \varphi(z) \preceq_x \varphi(y)\big)$$

In the last case, the vertex $y$ succeeds $x$ in the loop of a play given by a strategy pair $(\sigma, \tau)$ which is compatible with $\varphi$; of course, there may be several such $y$ with $x \lhd_\varphi y$. Similarly $\varphi$ is called optimal for player 1 if for all $x \in V_1$ and $y \in V_0$ with an edge $xEy$:

$$x \lhd_\varphi y \iff \forall z \in V_0 : xEz \Rightarrow \varphi(y) \preceq \varphi(z) \;\vee\; \big(\varphi_0(x) = x \;\wedge\; \varphi(y) \preceq_x \varphi(z)\big)$$

A valuation that is optimal for both players is called optimal valuation.

It is useful to note the following fact: If $\varphi$ is optimal for player 0, then (since $\preceq_x$ is coarser than $\preceq$) $x \lhd_\varphi y$ implies $\varphi(z) \preceq_x \varphi(y)$ for $xEz$. Similarly if $\varphi$ is optimal for player 1 then $x \lhd_\varphi y$ implies $\varphi(y) \preceq_x \varphi(z)$ for $xEz$.

The optimal valuations are closely related to the desired solution of a game, namely a pair of winning strategies. Consider a valuation $\varphi$ which is optimal for player 0, and let $W_1$ be the set of vertices $u$ whose $\varphi$-value $\varphi(u)$ is in $\mathcal{D}_1$ (i.e., the most relevant vertex $\varphi_0(u)$ associated to $u$ is in $V_-$, signalling a win of player 1). Any strategy for player 1 compatible with $\varphi$ turns out to be a winning strategy for him on $W_1$, whatever strategy player 0 chooses *independently* of $\varphi$. Applying this symmetrically for both players we shall obtain a pair of winning strategies.

**Lemma 3.** *Let $G = (V_0, V_1, E, c)$ be a game graph. Let $i \in \{0,1\}$ and $\varphi$ be a locally progressive valuation of $G$ which is optimal for player $i$. Let $W_{1-i} = \{v \in V \mid \varphi(v) \in \mathcal{D}_{1-i}\}$. Then the strategies for player $1-i$ compatible with $\varphi$ are winning strategies on $W_{1-i}$ (against an arbitrary strategy of player $i$).*

Applying this lemma symmetricly for both players leads to the following.

**Theorem 4.** *Let $G = (V_0, V_1, E, c)$ be a game graph. Let $\varphi$ be an optimal locally progressive valuation of $G$. Then all strategies compatible with $\varphi$ are winning strategies.*

### 3.4 Improving Valuations

Given two locally progressive valuations $\varphi$ and $\varphi'$, we call $\varphi'$ *improved for player* 0 *with respect to* $\varphi$ if

$$\forall x \in V_0 \; \exists y \in V : xEy \;\wedge\; x \lhd_{\varphi'} y \;\wedge\; \forall z \in V_1 : xEz \Rightarrow \varphi(z) \preceq \varphi(y).$$

A locally progressive valuation $\varphi'$, which is improved for player 0 with respect to $\varphi$, can be constructed from a given locally progressive valuation $\varphi$ by extracting a strategy $\sigma : V_0 \to V_1$ for player 0 that chooses maximal $E$-successors with respect to $\varphi$ and constructing a locally progressive valuation $\varphi'$ such that $\sigma$ is compatible with $\varphi'$.

**Lemma 5.** *Let $G = (V_0, V_1, E, c)$ be a game graph. Let $\varphi$ be a locally progressive valuation optimal for player* 1 *and $\varphi'$ a locally progressive valuation that is improved for player* 0 *with respect to $\varphi$. Then for all $v \in V$, we have $\varphi(v) \preceq \varphi'(v)$.*

## 4 The Algorithm

In this section we give an algorithm for constructing an optimal locally progressive valuation for a given parity game graph. This will lead to winning strategies for the game. The algorithm is split into three functions: *main()*, *valuation()*, and *subvaluation()*.

In the function *main()* a sequence of strategies for player 0 is generated and for each of these strategies a locally progressive valuation is computed by calling *valuation()*. This valuation is constructed such that the strategy of player 0 is compatible with it and that it is optimal for player 1. The first strategy for player 0 is chosen randomly. Subsequent strategies for player 0 are chosen such that they are optimal with respect to the previous valuation. The main loop terminates if the strategy chosen for player 0 is the same as in the previous iteration. Finally a strategy for player 1 is extracted from the last computed valuation.

```
main(G):
 1. for each $v \in V_0$                                   {Select initial strategy for player 0.}
 2.       select $\sigma(v) \in V_1$ with $v\, E_G\, \sigma(v)$
 3. repeat
 4.       $G_\sigma = (V_0, V_1, E', c)$, where $\forall u, v \in V$:
                $uE'v \iff (\sigma(u) = v \wedge u \in V_0) \vee (uE_Gv \wedge u \in V_1)$
 5.       $\varphi = valuation(G_\sigma)$
 6.       $\sigma' = \sigma$                                               {Store $\sigma$ under name $\sigma'$}
 7.       for each $v \in V_0$                      {Optimize $\sigma$ locally according to $\varphi$}
 8.             if $\varphi(\sigma(v)) \prec \max_\prec\{d \in D | \exists v' \in V : v\, E_G\, v' \wedge d = \varphi(v')\}$ then
 9.                   select $\sigma(v) \in V_1$
                            with $\varphi(\sigma(v)) = \max_\prec\{d \in D | \exists v' \in V : v\, E_G\, v' \wedge d = \varphi(v')\}$
10. until $\sigma = \sigma'$
11. for each $v \in V_1$
12.       select $\tau(v) \in V_0$
                with $\varphi(\tau(v)) = \min_\prec\{d \in D | \exists v' \in V : v\, E_G\, v' \wedge d = \varphi(v')\}$
13. $W_0 = \{v \in V \mid \varphi_0(v) \in V_+\}$
14. $W_1 = \{v \in V \mid \varphi_0(v) \in V_-\}$
15. return $W_0, W_1, \sigma, \tau$
```

*Fig. 1:* This function computes the winning sets and a winning strategy for each player from a given game graph.

In the functions *valuation()* and *subvaluation()*, we use the functions *reach*$(G, u)$, *minimal_distances*$(G, u)$, *maximal_distances*$(G, u)$. The functions work on the graph $G$ and perform a backward search on the graph starting in vertex $u$.

- The function *reach*$(G, u)$ produces the set of all vertices from which the vertex $u$ can be reached in $G$ (done by a backward depth first search).
- The function *minimal_distances*$(G, u)$ computes a vector $\delta : V_G \to \{0, \dots, |V_G| - 1\}$ where $\delta(v)$ is the length of the shortest path from $v$ to $u$ (done by a backward breadth first search starting from $u$).
- The function *maximal_distances*$(G, u)$ yields a vector $\delta : V_G \to \{0, \dots, |V_G| - 1\}$ where $\delta(v)$ is the length of the longest path from $v$ to $u$ that does not contain $u$ as an intermediate vertex. This is done by a backward search

starting from $u$ where a new vertex is visited if all its successors are visited before. This algorithm only works if every cycle in the graph contains $u$.

The *valuation(H)*-function produces a locally progressive valuation for the graph $H$ that is optimal for player 1. It does this by splitting the graph into a set of subgraphs for which *subvaluation()* can compute a locally progressive valuation of this kind.

The function searches a loop and then the set of vertices $R$ from which this loop can be reached. Computing the locally progressive valuation for the subgraph induced by $R$ is done by *subvaluation()*. The rest of the graph (i.e., the subgraph induced by $V_H \setminus R$) is recursively treated in the same way.

```
valuation(H):
 1. for each $v \in V$ do
 2.     $\varphi(v) = \perp$.
 3. for each $w \in V$ (ascending order with respect to $\prec$) do
 4.     if $\varphi(w) = \perp$ then
 5.         $L = reach(H|_{\{v \in V | v \leq w\}}, w)$
 6.         if $E_H \cap \{w\} \times L \neq \emptyset$ then
 7.             $R = reach(H, w)$
 8.             $\varphi|_R = subvaluation(H|_R, w)$
 9.             $E_H = E_H \setminus (R \times (V \setminus R))$
10. return $\varphi$
```

*Fig. 2:* This function computes for graph $H$ a locally progressive valuation that is optimal for player 1.

The algorithm *valuation(H)* is given in Fig. 2. It works as follows:

1. $\varphi$ is set 'undefined' for all $v$.
2. In ascending reward order those vertices $w$ are found which belong to a loop $L$ consisting of solely of $\leq$-smaller vertices. Then, for a fixed $w$, the set $R_w$ of all vertices is determined from which this loop (and hence $w$) is reachable (excluding vertices which have been used in sets $R_{w'}$ for previously considered $w'$). The valuation is updated on the set $R_w$.
   The new valuation is determined as follows. By backward depth-first search from $w$ the vertices $v$ in $R_w$ are scanned, and edges leading outside $R_w$ deleted, in order to prohibit entrance to $R_w$ by a later search (from a different $w'$).
   In *subvaluation(H)* the role of vertices $v$ in $R_w$ w.r.t. $\prec_w$ is analyzed. We shall have $\varphi_0(v) = w$ for them. One proceeds in decreasing relevance order:
   - If $u$ happens to be $w$, a backward breadth first search is done and only the distance decreasing edges are kept.
   - Otherwise one distinguishes whether $u$ is positive or negative. If $u$ is positive, one computes the set $\overline{U}$ of vertices from where $w$ can be reached

```
subvaluation($K, w$):
 1. for each $v \in V_K$ do
 2.     $\varphi_0(v) = w$
 3.     $\varphi_1(v) = \emptyset$
 4. for each $u \in \{v \in V_K \mid v > w\}$ (descending order with respect to $<$) do
 5.     if $u \in V_+$ then
 6.         $\overline{U} = reach(K|_{V_K \setminus \{u\}}, w)$
 7.         for each $v \in V_K \setminus \overline{U}$ do
 8.             $\varphi_1(v) = \varphi_1(v) \cup \{u\}$
 9.         $E_K = E_K \setminus ((\overline{U} \cup \{u\}) \times (V \setminus \overline{U}))$
10.     else
11.         $U = reach(K|_{V_K \setminus \{w\}}, u)$
12.         for each $v \in U$ do
13.             $\varphi_1(v) = \varphi_1(v) \cup \{u\}$
14.         $E_K = E_K \setminus ((U \setminus \{u\}) \times (V \setminus U))$
15. if $w \in V_+$ then
16.     $\varphi_2 = maximal\_distances(K, w)$
17. else
18.     $\varphi_2 = minimal\_distances(K, w)$
19. return $\varphi$
```

*Fig. 3:* This function computes a locally progressive valuation for a subgraph $K$ with most relevant vertex $w$.

while avoiding $u$, and for those $v$ from which a visit to $u$ is unavoidable, add $u$ to $\varphi_1(v)$, and delete edges from $v$ to vertices in $\overline{U}$.

- If $u$ is negative then let $U$ be the set of vertices $v$, such that $u$ can be visited on a path from $v$ to $w$. We add $u$ to $\varphi_1(v)$ for all $v \in U$, and we remove edges leading from $U \setminus \{u\}$ to $V \setminus U$.

The function *subvaluation(H)* is given in Fig. 3. It computes the paths to $w$ that have minimal reward with respect to $\prec_w$ and stores for each vertex the resulting path in $\varphi$-value. Edges that belong to paths with costs that are not minimal are removed successively.

## 5 Time complexity

In the analysis of the running time of a strategy improvement algorithm there are two parameters of major interest:

1. the time needed to perform a single strategy improvement step,
2. the number of strategy improvement steps needed.

We argue that our discrete strategy improvement algorithm for parity games achieves a satisfactory bound on the former parameter. Let $n$ be the number of vertices, and let $m$ be the number of edges in the game graph.

**Proposition 6.** *A single strategy improvement step, i.e., lines 4.-9. in Figure 1, is carried out in time $O(nm)$.*

A satisfactory analysis of the latter parameter is missing in our work. Despite the long history of strategy improvement algorithms for stochastic and payoff games [8, 4, 15] very little is known about the number of strategy improvement steps needed. The best upper bounds are exponential [4, 15] but to our best knowledge no examples are known which require more than linear number of improvement steps. We believe that our purely discrete description of strategy improvement gives new insights into the behaviour of the algorithm in the special case of parity games. The two long standing questions: whether there is a polynomial time algorithm for solving parity games [7], and, more concretely, whether a strategy improvement algorithm for parity games terminates in polynomial time [4, 15], remain open. Below we discuss some disjoint observations we have come up with so far, and some questions which we believe are worth pursuing.

We say that a vertex is *switchable* if the condition in line 8. of the algorithm in Figure 1 is satisfied; we say that a vertex is *switched* in line 9. of Figure 1. Note that switching an arbitrary non-empty subset of the set of switchable vertices in every strategy improvement step gives a correct strategy improvement algorithm for parity games. Therefore, one can view our algorithm as a generic algorithm which can be instantiated to a fully deterministic algorithm by providing a *policy* for choosing the set of vertices to switch in every strategy improvement step. Melekopoglou and Condon [13] exhibit families of examples on which several natural policies switching only one switchable vertex in every strategy improvement step require an exponential number of strategy improvement steps. It is open whether there are families of examples of parity games on which there are policies requiring super-polynomial number of strategy improvement steps. On the other hand, for every parity game and every initial strategy, there *exists* a policy requiring only linear number of strategy improvement steps.

**Proposition 7.** *For every parity game and initial strategy, there is a policy for which the strategy improvement algorithm switches every vertex at most once, and therefore it terminates after at most $n$ strategy improvement steps.*

This contrasts with an algorithm for solving parity games based on progress measures [10], for which there are families of examples on which every policy requires an exponential number of steps.

Examples of Melekopoglou and Condon [13] are Markov decision processes, i.e., one-player simple stochastic games [3]. It is an open question whether the strategy improvement algorithm using the standard policy, i.e., switching all switchable vertices in every strategy improvement step, works in polynomial time for one-player simple stochastic games [13]. In contrast, our discrete strategy improvement algorithm terminates in polynomial time for one-player parity games.

**Proposition 8.** *The discrete strategy improvement algorithm terminates after $O(n^3)$ strategy improvement steps for one-player parity games.*

Most algorithms for solving parity games studied in literature have $O\big((n/d)^d\big)$ or $O\big((n/d)^{d/2}\big)$ worst-case running time bounds (see [10] and references therein), where $d$ is the number of different priorities assigned to vertices. The best upper bound we can give at the moment for the number of strategy improvement steps needed by our discrete strategy improvement algorithm is the trivial one, i.e., the number of different strategies for player 0, which can be $2^{\Omega(n)}$.

**Proposition 9.** *The discrete strategy improvement algorithm terminates after* $\prod_{v \in V_0} \text{out-deg}(v)$ *many strategy improvement steps.*

There is, however, a variation of the strategy improvement algorithm for parity games, for which the number of strategy improvement steps is bounded by $O\big((n/d)^d\big)$.

**Proposition 10.** *There is a strategy improvement algorithm for parity games which terminates after* $O\big((n/d)^d\big)$ *improvement steps, and a single strategy improvement step can be performed in* $n^{O(1)}$ *time.*

Note that in every strategy improvement step the current valuation strictly improves in at least one vertex. We say that a strategy improvement step is *substantial* if in the current valuation the first component of a profile of some vertex strictly improves. Observe that there can be at most $O(n^2)$ substantial strategy improvement steps. It follows that in search for superpolynomial examples one has to manufacture gadgets allowing long sequences of non-substantial strategy improvement steps.

We have collected a little experimental evidence that in practice most improvement steps are non-substantial. There are few interesting scalable families of hard examples of parity games known in literature. Using an implementation of our discrete strategy improvement algorithm due to the first author [16] we have run some experiments on families of examples taken from [2] and from [10], and on a family of examples mentioned in [10] which make Zielonka's version [20] of the McNaughton's algorithm [11] work in exponential time. For all these families only linear number of strategy improvement steps were needed and, interestingly, the number of non-substantial strategy improvement steps was in all cases constant, i.e., not dependent of the size of the game graph.

## References

1. A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1–2):237–255, May 1997.
2. Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite games with Streett and Rabin chain winning conditions. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 207–224, Passau, Germany, 27–29 March 1996. Springer.

3. Anne Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.
4. Anne Condon. On algorithms for simple stochastic games. In Jin-Yi Cai, editor, *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.
5. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.
6. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (Extended abstract). In *Proceedings of 32nd Annual Symposium on Foundations of Computer Science*, pages 368–377. IEEE Computer Society Press, 1991.
7. E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of $\mu$-calculus. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV'93*, volume 697 of *LNCS*, pages 385–396, Elounda, Greece, June/July 1993. Springer-Verlag.
8. A. Hoffman and R. Karp. On nonterminating stochastic games. *Management Science*, 12:359–370, 1966.
9. Marcin Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, November 1998.
10. Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, February 2000. Springer.
11. Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.
12. Nimrod Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12:347–353, 1983.
13. Mary Melekopoglou and Anne Condon. On the complexity of the policy improvement algorithm for stochastic games. *ORSA (Op. Res. Soc. of America) Journal of Computing*, 6(2):188–192, 1994.
14. A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.
15. Anuj Puri. *Theory of Hybrid Systems and Discrete Event Systems.* PhD thesis, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, December 1995. Memorandum No. UCB/ERL M95/113.
16. Dominik Schmitz and Jens Vöge. Implementation of a strategy improvement algorithm for finite-state parity games. manuscipt, unpublished, April 2000.
17. Helmut Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, September 1996.
18. Colin Stirling. Local model checking games (Extended abstract). In Insup Lee and Scott A. Smolka, editors, *CONCUR'95: Concurrency Theory, 6th International Conference*, volume 962 of *LNCS*, pages 1–11. Springer-Verlag, 1995.
19. Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. Aachener Informatik-Berichte 2000-2, RWTH Aachen, Fachgruppe Informatik, 52056 Aachen, Germany, February 2000.
20. Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.
21. Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.