An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games*

Sven Schewe

Universität des Saarlandes and University of Liverpool

Abstract. This paper presents a novel strategy improvement algorithm for parity and payoff games, which is guaranteed to select, in each improvement step, an optimal combination of local strategy modifications. Current strategy improvement methods stepwise improve the strategy of one player with respect to some ranking function, using an algorithm with two distinct phases: They first choose a modification to the strategy of one player from a list of *locally* profitable changes, and subsequently evaluate the modified strategy. This separation is unfortunate, because current strategy improvement algorithms have no effective means to predict the *global effect* of the individual local modifications beyond classifying them as profitable, adversarial, or stale. Furthermore, they are completely blind towards the *cross effect* of different modifications: Applying one profitable modification may render all other profitable modifications adversarial. Our new construction overcomes the traditional separation between choosing and evaluating the modification to the strategy. It thus improves over current strategy improvement algorithms by providing the *optimal improvement* in every step, selecting the best combination of local updates from a superset of all profitable and stale changes.

1 Introduction

Solving parity games is the central and most expensive step in many model checking [1–5], satisfiability checking [3, 1, 6, 7], and synthesis [8, 9] algorithms. More efficient algorithms for solving parity games will therefore foster the development of performant model checkers and contribute to bringing synthesis techniques to practice. The quest for performant algorithms [1, 10–25] for solving them has therefore been an active field of research during the last decades.

Traditional forward techniques ($\approx O(n^{\frac{1}{2}c})$ [16] for parity games with n positions and c colors), backward techniques ($\approx O(n^c)$ [12, 10, 15]), and their combination ($\approx O(n^{\frac{1}{3}c})$ [25]) provide good complexity bounds. However, these bounds are sharp, and techniques with good complexity bounds [25, 16] frequently display their worst case complexity on practical examples.

Strategy improvement algorithms [17–20], on the other hand, are fast simplex style algorithms that perform well in practice. While their complexity is wide open, they are often considered the best choice for solving large scale games.

* This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

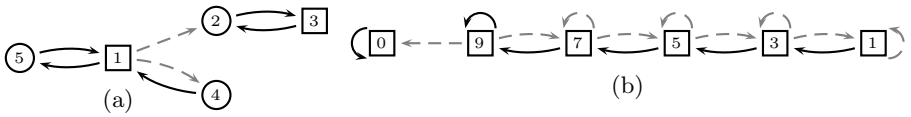

Fig. 1. The examples show situations where ignoring global effects (a) and cross effects between different updates (b) perturb the pivot rule of current strategy improvement algorithms. States of Player 0 and 1 are depicted as boxes and circles, respectively. The current strategy of Player 0 (and all options of Player 1) are depicted as full arrows, the improvement edges of Player 0 are represented by dashed arrows.

State of the Art. Strategy improvement algorithms are closely related to the simplex algorithm for solving linear programming problems. Strategy improvement algorithms assign a value to each infinite play of a parity or payoff game, and the objective of the two participating players (Player 0 and 1) is to minimize and maximize this value, respectively.

In strategy improvement algorithms for parity and payoff games [17–20], the memoryless strategies of Player 0 define the corners of a simplex. For each memoryless strategy of Player 0, her opponent has an optimal counter strategy. This pair of strategies defines a pointwise ranking function that assigns to each game position p the value (or rank) $v(p)$ of the play that starts in p.

The two distinguishing differences between strategy improvement techniques compared to the simplex algorithm are a *weak pivot rule* and the option to apply *multiple modifications* in every strategy improvement step.

Weak Pivot Rule. Different to simplex techniques for linear programming problems, current strategy improvement algorithms do not take the *global effect* of a step to an adjacent corner of the simplex into account. When estimating the improvement attached to modifying the strategy locally, they presume that changing the strategy for one game position has no influence on the rank of any other game position. In the situation depicted in Figure 1a, Player 0 can choose between two improvements of her strategy from her position colored by 1; she can either move to the position with color 2, or to the position with color 4. While the latter choice is obviously better (because Player 0 asserts the parity condition), the local analysis considers only the value of the positions for the *old* strategy [17–20]. A valuation function based on the old strategy, however, will favor the change to the position with color 2, because the dominating color in the infinity set for this position is 3 (for the old strategy), while the dominating color in the infinity set of the position colored by 4 is 5. In the whole, the local analysis alone does not provide much more information than a classification into locally profitable, adversarial, and stale modifications.

Multiple Modifications. An advantage of strategy improvement techniques over the simplex algorithm for linear programming problems is the option to consider several locally profitable modifications at the same time [18, 19]. This advantage, however, must be considered with care, because current strategy improvement algorithms are blind towards the *cross effect* between different local updates.

While any combination of profitable changes remains profitable, it frequently happens that applying one modification turns all remaining modifications ad-

Current Strategy Improvement Algs	Game-Based Strategy Improvement Algorithm
1. pick initial strategy	1. pick and evaluate initial strategy
2. evaluate current strategy	2. adjust evaluation, increasing #profitable/stale mods
3. chose from profitable modifications	3. find and evaluate optimal combination of p/s mods
4. goto 2	4. goto 2

Fig. 2. Comparison between traditional strategy improvement algorithms and the proposed optimal improvement algorithm. While current techniques first choose a particular update from profitable modifications and subsequently evaluate it, our novel technique concurrently considers all combinations of profitable and stale modifications.

versarial. In the small singleton parity game depicted in Figure 1b, Player 0 is only one step away from her optimal strategy. (It suffices to update the strategy in the position with color 9.) All local changes lead to an improvement, but after updating the strategy at the position with color 9, all remaining changes become harmful. Given this shortcoming, it is unclear whether or not simultaneous updates are a step forward for current strategy improvement algorithms.

Contribution. We introduce a strategy improvement algorithm that is based on a reduction to simple update *games*, which can be solved in a single sweep. It provides substantial advantages over current strategy improvement algorithms:

1. *The reduction is more natural.* It reduces solving parity (or mean payoff) games to solving a series of simplified games, where the options of Player 0 are restricted, but not to the extreme of a singleton game. It thus preserves and exploits the game character of the problem to improve a strategy.
2. *The improvements are greater.* The game-based approach allows us to take the global and cross effects of different local modifications to the strategy into account. We thus overcome the blind spot of current strategy improvement algorithms and can make full use of simultaneous modifications.
3. *The game-based analysis is cheaper.* Reductions to graph-based algorithms need to exclude stale cycles. Both, for parity and payoff games, the codomain of the pointwise ranking function needs to be increased by a factor linear in the size n of the game, which raises the estimation for the amount of iterations by a factor of n and slows down the arithmetic operations.

From Graph-Based to Game-Based Updates. The suggested optimal strategy improvement algorithm reduces solving parity games to solving a series of simpler two player games. Turning to a game-based (rather than to a graph-based) approach allows for considering *all combinations* of profitable and stale modifications in every update step, taking all global and cross effects into account.

This advancement is achieved by a novel technique that resolves the separation between choosing and evaluating the modifications to a strategy. Where current strategy improvement algorithms first update the strategy and then evaluate the resulting singleton game, our game-based approach exploits a natural preorder for the evaluation of game positions that allows for simultaneously constructing optimal strategies for both players, such that every game position is considered only once. Following this preorder, the evaluation of each individual position can be reduced to a cheap local analysis.

The intuition for the preorder is that, in most cases, the game-based approach allows for fixing an optimal decision for a game position *after* all of its successors have been reevaluated. If all positions do have unevaluated successors, we can immediately determine the optimal choice for some position of Player 1.

The Ranking Function. We change the rules of parity games by allowing one player, say Player 0, to terminate the game in her positions. This is related to the finite unraveling of mean payoff games [13] and the controlled single source shortest path problem from the reduction of Björklund and Vorobyov [20].

The objective of Player 0 remains to assert an infinite path with even maximal priority in the infinity set. However, we add the natural secondary objective for the case that she has not yet found (or there is no) such strategy. If Player 0 cannot assert such a path, she eventually stops the unraveling of the game, optimizing the finite occurrences of the different priorities, but *disregarding* the number of positions with priority 0. (Disregarding this number leads to a coarser ranking function, and thus to an improved estimation of the number of improvement steps. It also leads to greater improvements by increasing the number of profitable or stale modifications.) Second, if the highest occurring priority is odd, there is no need to keep track of the number of occurrences of this priority. It suffices to store the information that this maximal number occurs on a finite path, resulting again in a coarser ranking function.

For parity games with n positions, m edges, and c colors, the coarser ranking function leads to an improved estimation of the number of updates from the currently best bound $O\big(n\,(\frac{n+c}{c})^{c+1}\big)$ [20] for the number of arithmetic operations needed by strategy improvement algorithms to $O\big(n\,(\frac{n+c}{c})^{c-1}\big)$ for parity games with an even number of colors, and to $O\big(n\,(\frac{n+c}{c})^{c}\big)$ if the numbers of colors is odd, reducing the bound by a factor quadratic and linear in the number of states, respectively. The bound is reduced further by decreasing the discounted cost of arithmetic operations from $O(c)$ to $O(1)$ when the number of iterations is high.

2 Parity Games

A game is composed of a finite arena and an evaluation function. We will first discuss arenas, and then turn to the evaluation functions for parity games.

Arena. A finite arena is a triple $\mathcal{A} = (V_0, V_1, E)$, where V_0 and V_1 are disjoint finite sets of positions, called the positions of Player 0 and 1, and $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$ is a set of edges; that is, $(V_0 + V_1, E)$ is a bipartite directed graph. For infinite games, the arena is also required not to contain sinks; that is, every position $p \in V = V_0 \cup V_1$ has at least one outgoing edge $(p, p') \in E$.

Plays. Intuitively, a game is played by placing a pebble on the arena. If the pebble is on a position $p_0 \in V_0$, Player 0 chooses an edge $e = (p_0, p_1) \in E$ from p_0 to a position $p_1 \in V_1$ and moves the pebble to p_1. Symmetrically, Player 1 chooses a successor if the pebble is on a position $p_1 \in V_1$. This way, they successively construct an infinite *play* $\pi = p_0 p_1 p_2 p_3 \ldots \in V^\omega$.

Strategies. For a finite arena $\mathcal{A} = (V_0, V_1, E)$, a (memoryless) *strategy* for Player 0 is a function $f : V_0 \to V_1$ that maps every position $p_0 \in V_0$ of Player 0 to a position $v_1 \in V_1$ such that there is an edge $(p_0, p_1) \in E$ from p_0 to p_1. A play is called *f-conform* if every decision of Player 0 in the play is in accordance with f. For a strategy f of Player 0, we denote with $\mathcal{A}_f = (V_0, V_1, E_f)$ the arena obtained from $\mathcal{A}$ by deleting the transitions from positions of Player 0 that are not in accordance with f. The analogous definitions are made for Player 1.

Parity Games. A *parity game* is a game $\mathcal{P} = (V_0, V_1, E, \alpha)$ with arena $\mathcal{A} = (V_0, V_1, E)$ and a surjective coloring function $\alpha : G \cup R \to \mathcal{C} \subset \mathbb{N}$ that maps each position of $\mathcal{P}$ to a natural number. $\mathcal{C}$ denotes the finite set of colors. For technical reasons we assume that the minimal color of a parity game is $0 = min\{\mathcal{C}\}$.

Each play is evaluated by the highest color that occurs infinitely often. Player 0 wins a play $\pi = p_0 p_1 p_2 p_3 \dots$ if the highest color occurring infinitely often in the sequence $\alpha(\pi) = \alpha(p_0)\alpha(p_1)\alpha(p_2)\alpha(p_3)\dots$ is even, while Player 1 wins if the highest color occurring infinitely often in $\alpha(\pi)$ is odd.

A strategy f of Player 0 or 1 is called *p-winning* if all f-conform plays starting in p are winning for Player 0 or 1, respectively. A position of $\mathcal{P}$ is p-winning for Player 0 or 1 if Player 0 or 1, respectively, has a p-winning strategy. We call the p-winning positions for Player 0 or 1 the *winning region* of Player 0 or 1, respectively. Parity games are memoryless determined:

Theorem 1. *[11] For every parity game $\mathcal{P}$, the game positions are partitioned into a winning region W_0 of Player 0 and a winning region W_1 of Player 1. Moreover, Player 0 and 1 have memoryless strategies that are p-winning for every position p in their respective winning region.* □

3 Escape Games

Escape games are total reward games that are tailored for the game-based improvement method. They generalize parity games by allowing Player 0 to terminate every play immediately on each of her positions. Technically this is done by extending the arena with a fresh *escape position* $\perp$, which forms a sink of the extended arena, and can be reached from every position of Player 0. Every play of an *escape game* either eventually reaches the escape position and then terminates, or it is an infinite play in the non-extended arena.

Extended Arena. In an escape game, the finite arena $\mathcal{A} = (V_0, V_1, E)$ is extended to the directed graph $\mathcal{A}' = (V_0, V_1', E')$, which extends the arena $\mathcal{A}$ by a fresh position $\perp$ of Player 1 ($V_1' = V_1 \uplus \{\perp\}$) that is reachable from every position of Player 0 ($E' = E \cup V_0 \times \{\perp\}$). The escape position is a sink in $\mathcal{A}'$.

Finite Plays. Since the escape position $\perp$ is a sink, every play terminates when reaching $\perp$. The set of plays is therefore extended by the finite plays $\pi = p_0 p_1 p_2 p_3 \dots p_n \perp \in V^* \cdot \{\perp\}$.

Escape Games. An *escape game* is a game $\mathcal{E} = (V_0, V_1, E, \alpha)$, where $\mathcal{A} = (V_0, V_1, E)$ is a finite arena, and $\alpha : V \to \mathcal{C} \subset \mathbb{N}$ is a coloring function. An escape game is played on the extended arena $\mathcal{A}' = (V_0, V_1', E')$.

An infinite play $\pi = p_0 p_1 p_2 \ldots \in V^\omega$ of an escape game is evaluated to ∞ if the highest color occurring infinitely often is even, and to $-\infty$ otherwise. A finite play $\pi = p_0 p_1 p_2 \ldots p_n \perp$ is evaluated by a function $\rho(\pi) : \mathcal{C}_0 \to \mathbb{Z}$ (where $\mathcal{C}_0 = \mathcal{C} \setminus \{0\}$ is the codomain of the coloring function without 0) that maps an element c' of $\mathcal{C}_0$ to the number of positions p_i in π with $i > 0$ that are colored by $c' = \alpha(p_i)$. (Disregarding the color of the first position is technically convenient.)

The potential values of a path are ordered by the obvious alphabetic order $>$ that sets $\rho > \rho'$ if (1) the highest color c' with $\rho(c') \neq \rho'(c')$ is even and $\rho(c') > \rho'(c')$, or (2) if the highest color c' with $\rho(c') \neq \rho'(c')$ is odd and $\rho'(c') > \rho(c')$. Additionally, we define $\infty > \rho > -\infty$. The objective of Player 0 is to maximize this value, while it is the objective of Player 1 to minimize it.

We introduce an operator $\oplus$ for the evaluation of finite paths. For $\mathcal{R} = (\mathcal{C}_0 \to \mathbb{Z}) \cup \{\infty\}$, $\oplus : \mathcal{R} \times \mathcal{C} \to \mathcal{R}$ maps a function ρ and a color c' to the function ρ' that deviates from ρ only by assigning the respective successor $\rho'(c') = \rho(c') + 1$ to c' (and leaves $\rho'(d) = \rho(d)$ for $d \neq c'$). We fix $\infty \oplus c' = \infty$ and $\rho \oplus 0 = \rho$.

Estimations. We introduce *estimations* $v : V' \to \mathcal{R}$ for an escape game $\mathcal{E} = (V_0, V_1', E, \alpha)$ as witnesses for the existence of a memoryless strategy f of Player 0, which guarantees that every f-conform play π starting in some position p is evaluated to $\rho(\pi) \geq v(p)$. Formally, an estimation v has to satisfy the following side conditions:

- $v(\perp) = \mathbf{0}$ ($\mathbf{0}$ denotes the constant function that maps all colors in $\mathcal{C}_0$ to 0),
- for every $p_1 \in V_1$ and every edge $e = (p_1, p_0) \in E$, $v(p_1) \leq v(p_0) \oplus \alpha(p_0)$ holds true,
- for every position $p_0 \in V_0$ there is an edge $e = (p_0, p_1) \in E'$ such that $v(p_0) \leq v(p_1) \oplus \alpha(p_1)$ holds true, and
- Player 0 has a strategy f_∞ that maps every position $p_0 \in V_0$ with $v(p_0) = \infty$ to a position $v_1 = f_\infty(p_0)$ with $v(p_1) = \infty$, and which guarantees that every f_∞-conform play π starting in g is evaluated to $\rho(\pi) = \infty$.

A trivial estimation is simple to construct: we denote with v_0 the estimation that maps the escape position to $v_0(\perp) = \mathbf{0}$, every position $p_0 \in V_0$ to $v_0(p_0) = \mathbf{0}$, and every position $p_1 \in V_1$ of Player 1 to $v_0(p_1) = min\{\mathbf{0} \oplus \alpha(p_0) \mid (p_1, p_0) \in E\}$[1].

Lemma 1. *For every estimation v of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ there is a memoryless strategy f for Player 0 such that every f-conform play π starting in any position p satisfies $\rho(\pi) \geq v(p)$.*

Proof. We fix an arbitrary strategy f for Player 0 that agrees with f_∞ on every position $p \in V_0$ with infinite estimation ($v(p) = \infty \Rightarrow f(p) = f_\infty(p)$), and

[1] Having a simple-to-construct initial estimation is the reason for the restriction to bipartite games. Constructing an initial estimation for general games is not hard, and the improvement algorithm proposed in Section 4 extends to non-bipartite games.

chooses some successor that satisfies $v(p) \leq v\big(f(p)\big) \oplus \alpha\big(f(p)\big)$ otherwise. Every cycle reachable in an f-conform play has nonnegative weight (that is, weight $\mathbf{0} \oplus \alpha(p_0) \oplus \ldots \oplus \alpha(p_n)$ of every cycle $p_0 \ldots p_n p_0$ is $\geq \mathbf{0}$) by construction of f; every infinite f-conform play π is therefore evaluated to $\rho(\pi) = \infty \geq v(p)$.

By induction over the length of finite f-conform plays π that start in some position p, we can show that $\rho(\pi) \geq v(p)$. □

We call an estimation v' an *improvement* of an estimation v if $v'(p) \geq v(p)$ holds for all positions $p \in V$, and we call an improvement *strict* if $v' \neq v$.

For every estimation v, we define the *improvement arena* $\mathcal{A}_v = (V_0, V_1', E_v)$ that contains an edge $e = (p, p')$ if it satisfies $v(p) \leq v(p') \oplus \alpha(p')$ (i.e., $E_v = \{(p, p') \in E' \mid v(p) \leq v(p') \oplus \alpha(p')\}$), and the 0-*arena* $\mathcal{A}_v^0 = (V_0, V_1', E_v^0)$ which contains an edge $e = (p, p') \in E_v$ of the improvement arena if it satisfies (1) $v(p) = v(p') \oplus \alpha(p')$, and, if e originates from a position $p \in V_0$ of Player 0, if additionally (2) no edge $e' = (p, p')$ with $v(p) < v(p') \oplus \alpha(p')$ originates from p ($E_v^0 = \{(p, p') \in E_v \mid v(p) = v(p') \oplus \alpha(p')$ and $p \in V_0 \Rightarrow \forall (p, p') \in E_v.\, v(p) = v(p') \oplus \alpha(p')\}$).

Attractors. The 0-*attractor* $A \subseteq V$ of a set $F \subseteq V$ of game positions is the set of those game positions from which Player 0 has a strategy to force the pebble into a position in F. The 0-attractor A of a set F can be defined as the least fixed point of sets that contain F, and that contain a position $p \in V_0$ of Player 0 if they contain some successor of p, and a position $p \in V_1$ of Player 1 if p has some successor, and all successors of p are contained in A. The 1-attractor is defined accordingly. Constructing this least fixed point is obviously linear in the number of positions and edges in the arena, and we can fix a memoryless strategy (the attractor strategy) for the respective player to reach F in finitely many steps during this construction.

Lemma 2. *For a given arena $\mathcal{A} = (V_0, V_1, E)$ with n positions and m edges that may contain sinks, and for a set $F \subseteq V$ of game positions, we can compute the respective attractor A of F and a memoryless strategy for Player 0 or 1, respectively, on $A \setminus F$ to reach F in finitely many steps in time $O(m + n)$.* □

We call an estimation *improvable* if the 1-attractor of the escape position in the 0-arena $\mathcal{A}_v^0$ does not cover all positions that are not estimated to ∞.

Theorem 2. *For every non-improvable estimation v of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ Player 1 has a memoryless strategy f' such that every f'-conform play π starting in any position p satisfies $\rho(\pi) \leq v(p)$.*

Proof. We fix a strategy f' for Player 1 that agrees on all positions $V_1 \setminus v^{-1}(\infty)$ with some 1-attractor strategy of the escape position $\perp$ in the 0-arena $\mathcal{A}_v^0$.

For plays starting in some position p that is evaluated to ∞, $\rho(\pi) \leq v(p) = \infty$ holds trivially. For plays starting in some position p that is not evaluated to ∞, we can show by induction over the length of f'-conform plays starting in p that no f'-conform play can reach a position p' that is evaluated to $v(p') = \infty$. By

construction of f', every reachable cycle in an f'-conform play that does not reach a position in $v^{-1}(\infty)$ has negative weight (that is, a weight $< \mathbf{0}$), and every infinite f'-conform play which starts in a position p that is not evaluated to ∞ thus satisfies $-\infty = \rho(\pi) < v(p)$.

For every finite f'-conform play π starting in some position p, we can show by induction over the length of π that $\rho(\pi) \leq v(p)$ holds true. □

The non-improvable estimation of an escape game can be used to derive the winning regions ($v^{-1}(\infty)$ for Player 0) and the winning strategy for Player 1 on his winning region in the underlaying parity game. f_∞ defines the winning strategy of Player 0 on her winning region.

4 Solving Escape Games

In this section we introduce a game-based strategy improvement algorithm for the fast improvement of estimations of escape games. Every estimation (for example, the trivial estimation v_0) can be used as a starting point for the algorithm.

Optimal Improvement. The estimations we construct intuitively refer to strategies of Player 0 for the extended arena. (Although estimations are a more general concept; not all estimations refer to a strategy.) The edges of the improvement arena $\mathcal{A}_v = (V_0, V_1, E_v)$ of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ and an estimation v refer to all promising strategy updates, that is, all strategy modifications that *locally* lead to a—not necessarily strict—improvement (profitable and stale modifications). We call an improvement v' of v *optimal* if it is the non-improvable estimation for the restricted escape game $\mathcal{E}_v = (V_0, V_1, E_v, \alpha)$. v' is optimal in the sense that it dominates all other estimations $\widehat{v}$ that refer to strategies of Player 0 for $\mathcal{E}_v$, that is, to strategies that contain only improvement edges. Finding this optimal improvement v' thus relates to solving an *update game* $\mathcal{E}_v$, which deviates from the full escape game $\mathcal{E}$ only by restricting the choices of Player 0 to her improvement edges.

Basic Update Step. Instead of computing the optimal improvement v' of an estimation v directly, we compute the optimal update $u = v' - v$. (The operator $+ : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ maps a pair ρ, ρ' of functions to the function ρ'' that satisfies $\rho''(c') = \rho(c') + \rho'(c')$ for all $c' \in \mathcal{C}_0$. $- : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ is defined accordingly.)

For a given escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ with estimation v, we define the *improvement potential* of an edge $e = (p, p') \in E_v$ in the improvement arena $\mathcal{A}_v$ as the value $P(e) = v(p') \oplus \alpha(p') - v(p) \geq \mathbf{0}$ by which the estimation would locally be improved when the respective player chose to turn to p' (disregarding the positive global effect that this improvement may have). To determine the optimal update, we construct the improvement arena, and evaluate the optimal update of the escape position to $u(\bot) = 0$. We then evaluate the improvement of the remaining positions successively by applying the following evaluation rule:

1. <u>if</u> there is a position $p \in V_1$ of Player 1 that has only evaluated successors, we evaluate the improvement of p to $u(p) = min\{u(p') + P\big((p, p')\big) \mid (p, p') \in E\}$,

2. else if there is a position $p \in V_1$ of Player 1 that has an evaluated successor p' with $u(p') = P\big((p,p')\big) = 0$, we evaluate the improvement of p to $u(p) = 0$,
3. else if there is a position $p \in V_0$ of Player 0 that has only evaluated successors, we evaluate its improvement to $u(p) = max\{u(p') + P\big((p,p')\big) \mid (p,p') \in E_v\}$[2],
4. else we choose a position $p \in V_1$ of Player 1 with minimal intermediate improvement $u'(p) = min\{u(p') + P\big((p,p')\big) \mid p'$ is evaluated and $(p,p') \in E\}$ and evaluate the improvement of p to $u(p) = u'(p)$. (Note that $min\{\emptyset\} = \infty$.)

Correctness. The basic intuition for the optimal improvement algorithm is to re-estimate the value of a position only *after* all its successors have been re-estimated. In this situation, it is easy to determine the optimal decision for the respective player. In a situation where all unevaluated positions do have a successor, we exploit that every cycle in $\mathcal{A}_v$ has non-negative weight (weight $\geq \mathbf{0}$), and every infinite play in $\mathcal{A}_v$ is evaluated to ∞. An optimal strategy of Player 1 will thus turn, for some position of Player 1, to an evaluated successor. It is safe to chose a transition such that the minimality criterion on the potential improvement u' is satisfied, because, independent of the choice of Player 1, no better potential improvement can arise at any later time during this update step. Following these evaluation rules therefore provides an optimal improvement.

Theorem 3. *For every estimation v of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$, the algorithm computes the optimal improvement $v' = v + u$. If v is improvable, then the optimal improvement $v' \neq v$ is strictly better than v.*

Proof. During the reevaluation, we can fix optimal strategies f and f' for Player 0 and 1, respectively, by fixing $f(p)$ or $f'(p)$, respectively, to be some successor of p that satisfies the respective maximality or minimality requirement. (In Rule 2, we implicitly apply the same minimality requirement as in Rule 4.)

Every infinite f-conform play is evaluated to ∞, and for every finite f-conform play π that starts in some position p, we can show by induction over the length of π that $\rho(\pi) \geq v'(p)$ holds true.

No f'-conform play $\pi = p_0 p_1 p_2 \ldots$ in $\mathcal{A}_v$ (that is, under the restriction that Player 0 can chose only transitions in E_v), which does not start in a position p_0 that is evaluated to ∞, can contain a cycle, because p_{i+1} has been evaluated prior to p_i by construction. Thus, every such f'-conform play in $\mathcal{A}_v$ is finite. For every finite f'-conform play π in $\mathcal{A}_v$ that starts in some position p, we can show by induction over the length of π that $\rho(\pi) \leq v'(p)$ holds true.

It remains to show that the algorithm guarantees progress for improvable estimations. If at least one improvement edge e that originates from a position of Player 0 has a positive improvement potential $P(e) > \mathbf{0}$, the claim holds trivially. Let us consider the case that the improvement potential is $P(e) = \mathbf{0}$ for every improvement edge e that originates from a position of Player 0. According to the update rules, the algorithm will successively assign $u'(p) = \mathbf{0}$ to all positions in the 1-attractor of $\perp$ in the 0-arena $\mathcal{A}_v^0$. If the attractor covers all positions of $\mathcal{E}$, v is non-improvable by Theorem 2. Otherwise, $u'(p) > \mathbf{0}$ holds by definition

[2] We could also choose $u(p) = max\{u(p') + P\big((p,p')\big) \mid p'$ is evaluated and $(p,p') \in E\}$.

for every remaining position $p \in V_1$ of Player 1 that is not in the 1-attractor of the escape position $\perp$. This implies $u > \mathbf{0}$ and thus $v' = v + u > v$. □

Complexity. In spite of the wide variety of strategies that are considered simultaneously, the update complexity is surprisingly low. The optimal improvement algorithm generalizes Dijkstra's single source shortest path algorithm to two player games. The critical part of the algorithm is to keep track of the intermediate update u', and the complexity of the algorithm depends on the used data structure. The default choice is to use binary trees, resulting in an update complexity of $O(m \log n)$. However, using advanced data structures like 2-3 heaps (cf. [26]) reduces this complexity slightly to $O(m + n \log n)$.

Theorem 4. *For an escape game with n positions and m edges, the optimal improvement can be computed using $O(m + \delta \log \delta)$ arithmetic operations, where $\delta \leq n$ is the number of positions of Player 1 for which the improvement is strict.*

Proof. Let us consider a run of our algorithm that, when applying rule 3, gives preference to updates of positions with improvement 0. Keeping track of these updates is cheap, and giving them preference guarantees that all positions with 0-update are removed before the remainder of the graph is treated.

Let us partition the operations occurring after these 0-updates into

1. operations needed for keeping track of the number of unevaluated successors for positions of Player 1 and for finding the direction with maximal improvement for positions of Player 0, and
2. all remaining operations.

Obviously, (1) contains only $O(m)$ operations, while the restriction to (2) coincides with a run of Dijkstra's algorithm on a subgraph of the improvement arena. (On the subgraph defined by the strategy f of Player 0 referred to in Theorem 3.) Dijkstra's algorithm can be implemented to run in $O(m + \delta \log \delta)$ arithmetic operations [26]. □

Theorem 5. *The algorithm can be implemented to solve a parity game with n positions, m edges, and c colors in time $O\big(m \left(\frac{n+c}{c}\right)^{c'}\big)$, where $c' = c - 1$ if c is even, and $c' = c$ if c is odd.*

Proof. If both players follow the strategies f and f' from the proof of Theorem 3 starting in a position p_0 that is not evaluated to $\infty \neq v'(p_0)$, they reach the escape position $\perp$ on a finite acyclic path $p_0 p_1 \dots p_i \perp$. By induction over the length of this path we can show that $v(p_0) = \mathbf{0} \oplus \alpha(p_i) \oplus \dots \oplus \alpha(p_0)$. Note that, for odd highest color $c-1$ (and thus for even c), only p_i may be colored by $c-1$. Thus, the number of updates is, for each position, in $O\big(\left(\frac{n+c}{c}\right)^{c'-1}\big)$.

Let us, for the estimation of the running time, assume that only one small update occurs in every step. 'Only one' leads to a small δ (removing the $\delta \log \delta$ part from the estimation), while 'small update' can be used to reduce the discounted cost for the arithmetic operations on $\mathcal{R}$ to $O(1)$: Before computing the

improvement potential P, the update u, and the intermediate update u', we first compute an *abstraction* $a : \mathcal{R} \to \mathbb{Z}$ of these values that maps a function $\rho \in \mathcal{R}$ to 0 if $\rho = \mathbf{0}$, and to $\pm$ the highest integer h with $\rho(h) \neq 0$ otherwise (+ if and only if $r > \mathbf{0}$). Computing the concrete value is then linear in the absolute value of the abstraction (rather than in c). For every edge $e = (p, p')$, updating the improvement potential $a \circ P(e)$ to its new value requires $O(max\{a \circ u(p), a \circ u(p')\})$ steps (using the old u). All other operations on abstract values are in $O(1)$.

To compute u', we proceed in two steps. In a first step, we maintain a 2-3 heap that stores only the abstraction of u', and that contains all positions where u' is above a threshold t that is initialized to $t = 0$. For positions with abstract value t, we keep a 2-3 heap with concrete values for u'. Every time we use rule 4 and find an empty concrete 2-3 heap, we increase t to the minimal value of the abstract 2-3 heap, remove all positions with this abstract value from the abstract heap, and add them (with concrete value) to the concrete heap. The required concrete arithmetic operations are linear in the value of the abstraction $a \circ u(r)$ of the concrete update (rather than in c). In the worst case scenario, 'small updates' implies that the discounted cost of the operations is in $O(1)$. □

Extended Update Step. The basic update step can be improved to an extended update step by three simple and cheap additional computation steps:

1. Recursively remove all positions from $\mathcal{E}$ that have no predecessors, and push them on a solve-me-later stack.
2. Adapt the valuation function v to v' such that the values of positions of Player 1 are left unchanged ($v'(p) = v(p) \ \forall p \in V_1$), and the values of all positions of Player 0 are maximally decreased ($v'(p) = max\{v'(p') \ominus \alpha(p) \mid (p', p) \in E\} \forall p \in V_0$). (This step again exploits that the game is bipartite.)
3. Apply a basic update step.
4. Remove the 0-attractor of all positions that are evaluated to ∞ from $\mathcal{E}$.

Step 1 simplifies the game—positions without predecessors have no impact on the value of other game positions, and their evaluation can safely be postponed until after the remainder of the game has been evaluated—and strengthens the second step. In Step 2, we exploit the fact that the basic update step benefits from a high number of improvement edges that originate from positions of Player 0. This number is increased by changing the estimation v such that the estimations of positions of Player 1 remain unchanged, while the estimation of positions of Player 0 is decreased. The last step is again used to simplify the game.

An interesting side effect of Step 4 is that our game-based improvement algorithm behaves like standard fixed point algorithms [12, 10, 15] on Büchi and CoBüchi games (parity games with only two colors, w.l.o.g. 0 and 1). Like in these standard algorithms, we iteratively compute the set of states from which Player 0 can stay forever in positions with color 0, and then remove their 0-attractor from the game. The game-based approach described in this section can therefore also be viewed as an alternative generalization of the well accepted algorithms for Büchi and CoBüchi games to general parity games, which preserves different properties than McNaughton's generalization [12, 10, 15].

5 Benchmarks and Results

To evaluate the applicability of the game-based strategy improvement algorithm, a prototype of the algorithm was implemented and evaluated on different benchmarks, including random games with and without structure as well as other benchmarks for parity games. This section provides an overview on the results.

A first estimation of the performance of our algorithm on random games showed that the expected number of update games depends mainly on the number of colors and the outdegree, but it seems to be *constant* in the number of positions. This low expected number of updates has been confirmed by the following benchmarks. This restricts the potential competitors: The randomized subexponential algorithms of Ludwig [17], and Björklund and Vorobyov [20] change the strategy in exactly one position in every update step. It is therefore almost sure that the required number of update steps is at least linear in the size of the game. Ludwig's algorithm also has a much higher update complexity.

For the first benchmark, we therefore focused on the algorithm of Vöge and Jurdziński [19], and a (not subexponential) variant of Björklund and Vorobyov's algorithm [20] that chooses, in every step, a locally profitable modification uniformly at random for every position, for which a profitable modification exists.

The following table compares the expected number of iterations of our algorithm (*game*) with the variant of Björklund and Vorobyov (*rand*) and Vöge and Jurdziński's algorithm (VJ) for random games with 3 colors and outdegree 6.

positions	30	100	300	1000	3000	10000	30000	100000	300000
game	1.1	1.4	1.7	1.7	1.9	2.0	2.0	2.0	2.0
rand	2.5	2.9	3.1	3.0	3.0	3.1	3.2	3.7	4.0
VJ	5.3	12.2	26.1	66.1	182.0	573.1	1665.3[3]	—	—

The algorithm of Vöge and Jurdziński was not considered in the following benchmarks, because it took several days even for small random game with only 30000 positions and outdegree 6. This is partly due to the fact that the observed number of iterations grows linearly in the size of the game, and partly due to the much higher update complexity of $O(m\,n)$.

Different to the algorithm of Vöge and Jurdziński, the performance of the variant of Björklund and Vorobyov's algorithm (*rand*) is, on random games, close to the performance of our game-based strategy improvement algorithm. The cost of the individual updates for *rand* is slightly higher, because 0 cycles need to be excluded in their approach, which results in higher numbers (by a factor linear in the size of the game). Together with the smaller number of iterations, the running time of our algorithm improves over theirs by a factor of approximately 2 on the considered random games.

The difference between the two algorithms becomes apparent once structure is added to the game. Figure 5 compares the behavior of *game* and *rand* on different benchmarks. Benchmark 1 adds very little structure (favoring edges to

[3] For 30000 positions, each sample took approximately four days on a 2.6 GHz Dual Core AMD Opteron machine (compared to 1.5 seconds for the game-based strategy improvement algorithm); the experiment was therefore terminated after ten samples.

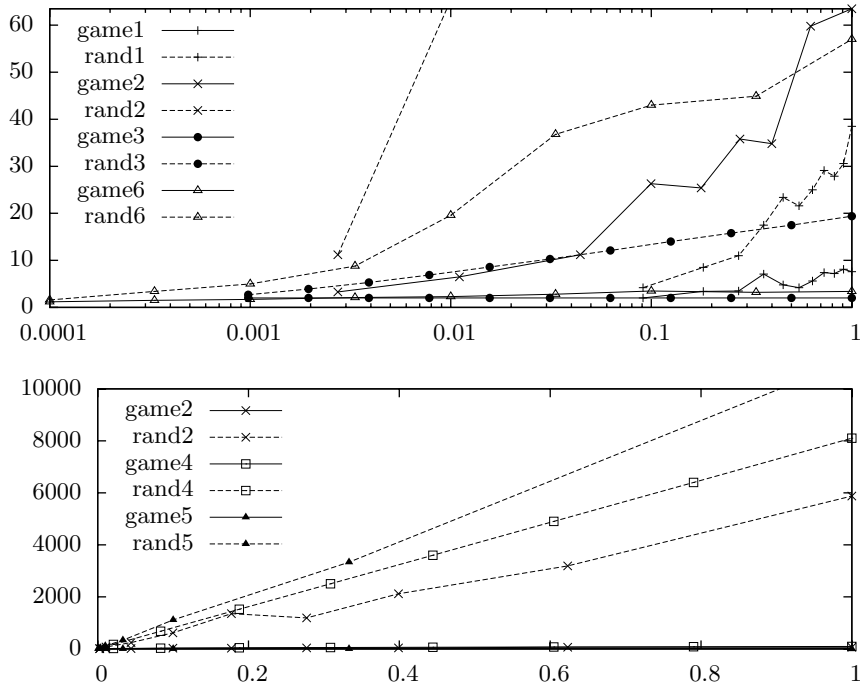

Fig. 3. The tables compare the performance of the variant of *game* (solid lines) and *rand* (dashed lines) on various benchmarks, measured in the number of iterations. The maximal size of all benchmarks is normalized to 1 (x-axis) for better readability.
For all benchmarks, the number of iterations (y-axis) needed by *game* is significant below the number needed by *rand*. The difference is particularly apparent in examples with much structure (Benchmarks 2 through 5).

'close' vertices in the randomized construction of the samples), while Benchmark 2 adds much structure (random chains of sparsely linked subgames). Benchmark 3 is a bipartite version of the games used in [16] to estimate the worst case complexity of Jurdziński's algorithm. Benchmark 4 refers to medium hard Büchi games, and Benchmark 5 is a test for the sensitivity of strategy improvement algorithms to 'traps' that lure them to a wrong direction (as in the example of Figure 1b). Finally, Benchmark 6 refers to the analysis of mean payoff games with small payoffs[4]. The results indicate that the more structure is added to the game, the greater becomes the advantage of *game* over *rand*. The detailed results are omitted due to space restrictions; they can be found in [27].

6 Discussion

The applicability of strategy improvement algorithms crucially depends on the quality of the individual improvement step, or, likewise, on the expected amount

[4] To extend our game-based approach to finding the 0-mean partition of mean payoff games, it suffices to replace the codomain $\mathcal{R}$ of the ranking function by $\mathbb{Z} \cup \{\infty\}$.

of improvement steps. Current strategy improvement techniques suffer from deficiencies in estimating the effect of strategy modifications: They cannot predict their global effect, let alone the cross effect between different modifications. The introduced game-based strategy improvement algorithm overcomes this deficiency by selecting an optimal combination of these modifications in every update step. While still greedy in nature, it allows us to make full use of the advantages attached to concurrent strategy modifications for the first time.

From a practical point of view, the amount of improvement steps used by simplex style algorithms tends to be linear in the amount of constraints that define the simplex [28]. For strategy improvement algorithms, these constraints are defined by the edges that originate from positions of Player 0. In the benchmarks, approximately 30% (at the end) to 50% (at the beginning) of these edges are improvement edges, which leads to an *exponential* number of concurrently considered improved strategies in every update game, and to a linear number of applied updates.

While the update complexity of the algorithms is low ($O(m + n \log n)$ arithmetic operations), finding a non-trivial bound on the number of updates remains an intriguing future challenge. The algorithm inherits the $\Omega(n)$ bound on the required number of updates from Büchi games, while the size of the codomain implies an $O((1 + \frac{n}{c})^c)$ upper bound, and either of these bounds may be sharp.

Understanding the complexity of the game-based strategy improvement algorithm would either lead to a proof that parity and/or mean payoff games can be solved in polynomial time, or would greatly help to understand the hardness of the problems. Hardness proofs for game-base strategy improvement, however, will not be simple. It took a quarter of a century to find a family of examples, for which the improvement complexity of the simplex algorithm is exponential [29]. These classical examples from linear programming, however, do not extend to game-based improvement methods. The Klee Minty polytope [29], for example, requires only a single update step from the origin (and at most linearly many steps from any arbitrary corner of the polytope) if we can consider all combinations of profitable and stale base changes in every improvement step.

References

1. Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. **27** (1983) 333–354
2. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Proc. CAV. (1993) 385–396
3. Wilke, T.: Alternating tree automata, parity games, and modal μ-calculus. Bull. Soc. Math. Belg. **8**(2) (2001)
4. de Alfaro, L., Henzinger, T.A., Majumdar, R.: From verification to control: Dynamic programs for omega-regular objectives. In: Proc. LICS, IEEE Computer Society Press (2001) 279–290
5. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (2002) 672–713
6. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Proc. ICALP, Springer-Verlag (1998) 628–641

7. Schewe, S., Finkbeiner, B.: The alternating-time μ-calculus and automata over concurrent game structures. In: Proc. CSL, Springer-Verlag (2006) 591–605
8. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. LICS, IEEE Computer Society (2006) 255–264
9. Schewe, S., Finkbeiner, B.: Synthesis of asynchronous systems. In: Proc. LOPSTR, Springer-Verlag (2006) 127–142
10. Emerson, E.A., Lei, C.: Efcient model checking in fragments of the propositional μ-calculus. In: Proc. LICS, IEEE Computer Society Press (1986) 267–278
11. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proc. FOCS, IEEE Computer Society Press (1991) 368–377
12. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Logic **65**(2) (1993) 149–184
13. Zwick, U., Paterson, M.S.: The complexity of mean payoff games on graphs. Theoretical Computer Science **158**(1–2) (1996) 343–359
14. Browne, A., Clarke, E.M., Jha, S., Long, D.E., Marrero, W.: An improved algorithm for the evaluation of fixpoint expressions. Theoretical Computer Science **178**(1–2) (1997) 237–255
15. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**(1-2) (1998) 135–183
16. Jurdziński, M.: Small progress measures for solving parity games. In: Proc. STACS, Springer-Verlag (2000) 290–301
17. Ludwig, W.: A subexponential randomized algorithm for the simple stochastic game problem. Inf. Comput. **117**(1) (1995) 151–155
18. Puri, A.: Theory of hybrid systems and discrete event systems. PhD thesis, Computer Science Department, University of California, Berkeley (1995)
19. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Proc. CAV, Springer-Verlag (2000) 202–215
20. Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Appl. Math. **155**(2) (2007) 210–229
21. Obdržálek, J.: Fast mu-calculus model checking when tree-width is bounded. In: Proc. CAV, Springer-Verlag (2003) 80–92
22. Lange, M.: Solving parity games by a reduction to SAT. In: Proc. Int. Workshop on Games in Design and Verification. (2005)
23. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: Dag-width and parity games. In: Proc. STACS, Springer-Verlag (2006) 524–436
24. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proc. SODA, ACM/SIAM (2006) 117–123
25. Schewe, S.: Solving parity games in big steps. In: Proc. FSTTCS, Springer-Verlag (2007) 449–460
26. Takaoka, T.: Theory of 2-3 heaps. In: Proc. COCOON, Springer-Verlag (1999) 41–50
27. Schewe, S.: Synthesis of Distributed Systems. PhD thesis, Saarland University, Saarbrücken, Germany (2008)
28. Smale, S.: On the average number of steps of the simplex method of linear programming. Mathematical Programming **27**(3) (1983) 241–262
29. Klee, F., Minty, G.J.: How good is the simplex algorithm? Inequalities III (1972) 159–175