Partial Solvers for Generalized Parity Games*

Véronique Bruyère[1]**, Guillermo A. Pérez[2], Jean-François Raskin[3], and Clément Tamines[1]

[1] University of Mons (UMONS), Mons, Belgium
[2] University of Antwerp (UAntwerp), Antwerp, Belgium
[3] Université libre de Bruxelles (ULB), Brussels, Belgium

Abstract. Parity games have been broadly studied in recent years for their applications to controller synthesis and verification. In practice, partial solvers for parity games that execute in polynomial time, while incomplete, can solve most games in publicly available benchmark suites. In this paper, we combine those partial solvers with the classical algorithm for parity games due to Zielonka. We also extend partial solvers to generalized parity games that are games with conjunction of parity objectives. We have implemented those algorithms and evaluated them on a large set of benchmarks proposed in the last LTL synthesis competition.

Keywords: Parity games · Generalized parity games · Partial solvers

1 Introduction

Since the early nineties, parity games have been attracting a large attention in the formal methods and theoretical computer science communities for two main reasons. First, parity games are used as intermediary steps in the solution of several relevant problems like, among others, the reactive synthesis problem from LTL specifications [18] or the emptiness problem for tree automata [9]. Second, their exact complexity is a long standing open problem: while we know that they are in $\mathsf{NP} \cap \mathsf{coNP}$ [9] (and even in $\mathsf{UP} \cap \mathsf{coUP}$ [16]), we do not yet have a polynomial time algorithm to solve them. Indeed, the best known algorithm so far has a worst-case complexity which is quasi-polynomial [3].

The classical algorithm for reactive synthesis from LTL specifications is as follows: from an LTL formula ϕ whose propositional variables are partitioned into inputs (controllable by the environment) and outputs (controlled by the system), construct a deterministic parity automaton (DPA) A_ϕ that recognizes the set of traces that are models of ϕ. This DPA can then be seen as a two player

* Work partially supported by the PDR project *Subgame perfection in graph games* (F.R.S.-FNRS), the ARC project *Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond* (Fédération Wallonie-Bruxelles), the EOS project *Verifying Learning Artificial Intelligence Systems* (F.R.S.-FNRS & FWO), the COST Action 16228 *GAMENET* (European Cooperation in Science and Technology). The full version of this article is available at https://arxiv.org/abs/1907.06913

** Corresponding author veronique.bruyere@umons.ac.be

graph game where the two players choose in turn the values of the input variables (Player 1) and of the output variables (Player 0). The winning condition in this game is the parity acceptance condition of the DPA. The two main difficulties with this approach are that the DPA may be doubly exponential in the size of ϕ and that its parity condition may require exponentially many priorities. So the underlying parity game may be hard to solve with the existing (not polynomial) algorithms. These difficulties have triggered two series of results.

First, incomplete algorithms that partially solve parity games in polynomial time have been investigated [13, 14, 1]. Although they are incomplete, experimental results show that they behave well on benchmarks generated with a random model and on examples that are forcing the worst-case behavior of the classical recursive algorithm for solving parity games due to Zielonka [19]. The latter algorithm has a worst-case complexity which is exponential in the number of priorities of the parity condition. Second, compositional approaches to generate the automata from LTL specifications have been advocated, when the LTL formula ϕ is a conjunction of smaller formulas, i.e., $\phi = \phi_1 \wedge \cdots \wedge \phi_n$. In this case, the procedure constructs a DPA A_i for each subformula ϕ_i. The underlying game is a product of the automata A_i; the winning condition, the conjunction (for Player 0) of the parity conditions of each automaton. Those games are thus generalized parity games, that are known to be co-NP-complete [6].

In this paper, we contribute to these lines of research in several ways. First, we show how to extend the partial solvers for parity games to generalized parity games. In the generalized case, we show how antichain-based data structures can be used to retain efficiency. Second, we show how to combine partial solvers for parity games and generalized parity games with the classical recursive algorithms [19, 6]. In this combination, the recursive algorithm is only executed on the portion of the game graph that was not solved by the partial solver, and this is repeated at each recursive call. Third, we provide for the first time extensive experiments that compare all those algorithms on benchmarks that are generated from LTL specifications used in the LTL synthesis competition [15]. For parity games, our experiments show behaviors that differ largely from the behaviors observed on experiments done on random graphs only. Indeed Zielonka's algorithm is faster than partial solvers on average which was not observed on random graphs in [14]. Equally interestingly, we show that there are instances of our benchmarks of generalized parity games that cannot be solved by the classical recursive algorithm or by any of the partial solvers alone, but that can be solved by algorithms that combine them. We also show that when combined with partial solvers, the performances of the classical recursive algorithms are improved on a large portion of our benchmarks for both the parity and generalized parity cases.

2 Preliminaries

Game Structures. A *game structure* is a tuple $G = (V_0, V_1, E)$ where *(i)* (V, E) is a finite directed graph, with $V = V_0 \cup V_1$ the set of vertices and $E \subseteq$

$V \times V$ the set of edges such that[4] for each $v \in V$, there exists some $(v, v') \in E$, *(ii)* (V_0, V_1) forms a partition of V where V_i is the set of vertices controlled by player i. Given $U \subseteq V$, if $G_{\restriction U} = (V_0 \cap U, V_1 \cap U, E \cap (U \times U))$ has no deadlock, then $G_{\restriction U}$ is called the *subgame structure* induced by U.

A *play* in G is an infinite sequence of vertices $\pi = v_0 v_1 \ldots \in V^\omega$ such that $(v_j, v_{j+1}) \in E$ for all $j \in \mathbb{N}$. *Histories* in G are finite sequences $h = v_0 \ldots v_j \in V^+$ defined similarly. We denote by $\mathsf{Plays}(G)$ the set of plays in G and by $\mathsf{Plays}(v_0)$ the set of plays starting in a given *initial vertex* v_0. Given a play $\pi = v_0 v_1 \ldots$, the set $\mathsf{Occ}(\pi)$ denotes the set of vertices that occur in π, and the set $\mathsf{Inf}(\pi)$ denotes the set of vertices that occur infinitely often in π.

A *strategy* σ_i for player i is a function $\sigma_i \colon V^* V_i \to V$ assigning to each history $hv \in V^* V_i$ a vertex $v' = \sigma_i(hv)$ such that $(v, v') \in E$. Given a strategy σ_i of player i, a play $\pi = v_0 v_1 \ldots$ is *consistent* with σ_i if $v_{j+1} = \sigma_i(v_0 \ldots v_j)$ for all $j \in \mathbb{N}$ such that $v_j \in V_i$. Consistency is naturally extended to histories.

Objectives. An *objective for player i* is a set of plays $\Omega \subseteq \mathsf{Plays}(G)$. A *game* (G, Ω) is composed of a game structure G and an objective Ω for *player* 0. A play π is *winning* for player 0 if $\pi \in \Omega$, and losing otherwise. The games that we study are *zero-sum*: player 1 has the opposite objective $\overline{\Omega} = V^\omega \setminus \Omega$, meaning that a play π is winning for player 0 if and only if it is losing for player 1. Given a game (G, Ω) and an initial vertex v_0, a strategy σ_0 for player 0 is *winning from* v_0 if all the plays $\pi \in \mathsf{Plays}(v_0)$ consistent with σ_0 belong to Ω. We say that v_0 is *winning* for player 0 and that player 0 is winning from v_0. We denote by $\mathsf{Win}(G, 0, \Omega)$ the set of such winning vertices v_0. Similarly we denote by $\mathsf{Win}(G, 1, \overline{\Omega})$ the set of vertices from which player 1 can ensure his objective $\overline{\Omega}$.

A game (G, Ω) is *determined* if each vertex of G belongs to either $\mathsf{Win}(G, 0, \Omega)$ or $\mathsf{Win}(G, 1, \overline{\Omega})$. Martin's theorem [17] states that all games with Borel objectives are determined. The problem of *solving a game* (G, Ω) means to decide, given an initial vertex v_0, whether player 0 is winning from v_0 for Ω (or dually whether player 1 is winning from v_0 for $\overline{\Omega}$ when the game is determined). The sets $\mathsf{Win}(G, 0, \Omega)$ and $\mathsf{Win}(G, 1, \overline{\Omega})$ are also called the *solutions* of the game.

Parity and Generalized Parity Objectives. Let G be a game structure and $d \in \mathbb{N}$ be an integer. Let $\alpha \colon V \to [d]$, with $[d] = \{0, 1, \ldots, d\}$, be a *priority* function that associates a priority with each vertex. The *parity* objective $\Omega = \mathsf{EvenParity}(\alpha)$ asks that the maximum priority seen infinitely often along a play is even, i.e., $\mathsf{EvenParity}(\alpha) = \{\pi \in \mathsf{Plays}(G) \mid \max_{v \in \mathsf{Inf}(\pi)} \alpha(v) \text{ is even}\}$. Games $(G, \mathsf{EvenParity}(\alpha))$ are called *parity games*. In those games, player 1 has the opposite objective $\overline{\Omega}$ equal to $\{\pi \in \mathsf{Plays}(G) \mid \max_{v \in \mathsf{Inf}(\pi)} \alpha(v) \text{ is odd}\}$. We denote $\overline{\Omega}$ by $\mathsf{OddParity}(\alpha)$. In the sequel, an *$i$-priority* means an even priority if $i = 0$ and an odd priority if $i = 1$. For convenience, we also use notation $i\mathsf{Parity}(\alpha)$ such that $0\mathsf{Parity}(\alpha) = \mathsf{EvenParity}(\alpha)$ and $1\mathsf{Parity}(\alpha) = \mathsf{OddParity}(\alpha)$.

The *generalized parity* objective $\Omega = \mathsf{ConjEvenParity}(\alpha_1, \ldots, \alpha_k)$ is the conjunction of $k \geq 1$ parity objectives, that is, $\Omega = \bigcap_{\ell=1}^{k} \mathsf{EvenParity}(\alpha_\ell)$, where each

[4] This condition guarantees that there is no deadlock.

$\alpha_\ell \colon V \to [d_\ell]$ is a priority function. The opposite objective $\overline{\Omega}$ for player 1 is equal to $\mathsf{DisjOddParity} = \bigcup_{\ell=1}^{k} \mathsf{OddParity}(\alpha_\ell)$. Games $(G, \mathsf{ConjEvenParity}(\alpha_1, \ldots, \alpha_k))$ are called *generalized parity games*.

Parity games and generalized parity games are determined because their objectives are ω-regular and thus Borel. Solving parity games is in $\mathsf{UP} \cap \mathsf{co\text{-}UP}$ [16] and solving generalized parity games is $\mathsf{co\text{-}NP}$-complete [6].

Partial Solvers. We study *partial solvers* for parity games and generalized parity games. A partial solver returns two partial sets of winning vertices $Z_0 \subseteq \mathsf{Win}(G, 0, \Omega)$ and $Z_1 \subseteq \mathsf{Win}(G, 1, \overline{\Omega})$ such that $G_{\restriction U}$ is a subgame structure with $U = V \setminus (Z_0 \cup Z_1)$. In the next sections, we present the polynomial time partial solvers proposed in [13, 14] for parity games and show how to extend them to generalized parity games.

Other ω-Regular Objectives. We recall some other useful ω-regular objectives. Let G be a game structure and $U, U_1, \ldots, U_k \subseteq V$ be subsets: the *reachability objective* $\mathsf{Reach}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Occ}(\pi) \cap U \neq \emptyset\}$ asks to visit U at least once; the *safety objective* $\mathsf{Safe}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Occ}(\pi) \cap U = \emptyset\}$ asks to avoid visiting U; the *Büchi objective* $\mathsf{Büchi}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Inf}(\pi) \cap U \neq \emptyset\}$ asks to visit infinitely often a vertex of U; the *co-Büchi objective* $\mathsf{CoBüchi}(U) = \{\pi \in \mathsf{Plays}(G) \mid \mathsf{Inf}(\pi) \cap U = \emptyset\}$ asks to avoid visiting infinitely often U; the *generalized Büchi objective* $\mathsf{GenBüchi}(U_1, \ldots, U_k)$ is equal to the intersection $\bigcap_{\ell=1}^{k} \mathsf{Büchi}(U_\ell)$. The next theorem summarizes the time complexities for solving those games as implemented in our prototype tool.[5]

Theorem 1. *For solving games (G, Ω), we have the following time complexities.*

- *Reachability, safety objectives: $O(|E|)$ [12].*
- *Büchi, co-Büchi, Büchi $\cap$ safety objectives: $O(|V| \cdot |E|)$ [12].*
- *Generalized Büchi, generalized Büchi $\cap$ safety objectives: $O(k \cdot |V| \cdot |E|)$.*[6]

Attractors. Let G be a game structure. The *controllable predecessors* for player i of a set $U \subseteq V$, denoted by $\mathsf{Cpre}_i(G, U)$, is the set of vertices from which player i can ensure to visit U in *one step*. Formally, $\mathsf{Cpre}_i(G, U)$ is equal to

$$\{v \in V_i \mid \exists (v, v') \in E, v' \in U\} \cup \{v \in V_{1-i} \mid \forall (v, v') \in E, v' \in U\}. \tag{1}$$

The *attractor* $\mathsf{Attr}_i(G, U)$ *for player* i is the set of vertices from which he can ensure to visit U in *any* number of steps (including zero steps). It is constructed as follows: $\mathsf{Attr}_i(G, U) = \bigcup_{j \geq 0} X_j$ such that $X_0 = U$, and for all $j \in \mathbb{N}$, $X_{j+1} = X_j \cup \mathsf{Cpre}_i(G, X_j)$. It is thus the winning set $\mathsf{Win}(G, i, \mathsf{Reach}(U))$. The *positive attractor* $\mathsf{PAttr}_i(G, U)$ is the set of vertices from which player i can ensure to visit U in any *positive* number of steps, that is, $\mathsf{PAttr}_i(G, U) = \bigcup_{j \geq 0} X_j$ with:

$$X_0 = \mathsf{Cpre}_i(G, U), \;\; X_{j+1} = X_j \cup \mathsf{Cpre}_i(G, X_j \cup U) \text{ for all } j \in \mathbb{N}. \tag{2}$$

[5] A better algorithm in $O(|V|^2)$ for Büchi objectives is proposed in [5], and in $O(k \cdot |V|^2)$ for generalized Büchi objectives in [4].

[6] This result is obtained with a classical reduction to games with Büchi objectives [2].

Given $U \subseteq V$, we say that U is an *i-trap* if for all $v \in U \cap V_i$ and all $(v, v') \in E$, we have $v' \in U$ (player i cannot leave U), and for all $v \in U \cap V_{1-i}$, there exists $(v, v') \in E$ such that $v' \in U$ (player $1-i$ can ensure to stay in U). Therefore $G_{\restriction U}$ is a subgame structure. When $V \setminus U$ is an i-trap, we also use the notation $G \setminus U$ (instead of $G_{\restriction V \setminus U}$) for the subgame structure induced by $V \setminus U$. It is well-known that the set $V \setminus \mathsf{Attr}_i(G, U)$ is an i-trap [12].

3 Zielonka's Algorithm with Partial Solvers

The classical algorithm used to solve parity games is the algorithm proposed by Zielonka in [19]. Despite its relatively bad theoretical $O(|V|^d)$ time complexity, it is known to outperform other algorithms in practice [11, 7]. This algorithm solves parity games $(G, \mathsf{EvenParity}(\alpha))$ by working in a divide-and-conquer manner, combining solutions of subgames to obtain the solution of the whole game. It returns two sets $\{W_0, W_1\}$ such that $W_i = \mathsf{Win}(G, i, i\mathsf{Parity}(\alpha))$ is the winning set for player i. See Algorithm 1 in which no call to a partial solver is performed (therefore line 4 is to be replaced by $\{Z_0, Z_1\} = \{\emptyset, \emptyset\}$).

Let us explain how Zielonka's algorithm can be combined with a partial solver for parity games (see Algorithm 1). When V is not empty, we first execute the partial solver. If it solves the game completely, we are done. Otherwise, let $\overline{G}$ be the subgame of G that was not solved. We then execute the Zielonka instructions on $\overline{G}$ and return the union of the partial solutions obtained by the partial solver with the solutions obtained for $\overline{G}$. Proposition 2 below guarantees the soundness of this approach under the hypothesis that if some player wants to escape from $\overline{G}$, then he necessarily goes to the partial solution of the other player.

Algorithm 1 Ziel&PSolver(G, α)

```
1  if V = ∅ then return {W_0, W_1} = {∅, ∅}
2  else
3      {Z_0, Z_1} = PSolver(G, α)
4      Ḡ = G \ (Z_0 ∪ Z_1);   V̄ = V \ (Z_0 ∪ Z_1)
5      if V̄ = ∅ then return {Z_0, Z_1}
6      else
7          p = max{α(v) | v ∈ V̄};   i = p mod 2
8          U = {v ∈ V̄ | α(v) = p};   X = Attr_i(Ḡ, U)
9          {W'_i, W'_{1-i}} = Ziel&PSolver(Ḡ \ X, α)
10         if W'_{1-i} = ∅ then
11             W_i = Z_i ∪ W'_i ∪ X;  W_{1-i} = Z_{1-i}
12         else
13             X = Attr_{1-i}(Ḡ, W'_{1-i})
14             {W''_i, W''_{1-i}} = Ziel&PSolver(Ḡ \ X, α)
15             W_i = Z_i ∪ W''_i;   W_{1-i} = Z_{1-i} ∪ W''_{1-i} ∪ X
16         return {W_i, W_{1-i}}
```

Proposition 2. *Suppose that the partial solver used in Algorithm 1 computes partial solutions Z_0, Z_1 such that for all $(v, v') \in E$ and $i \in \{0,1\}$, if $v \in \overline{V} \cap V_i$ and $v' \notin \overline{V}$, then $v' \in Z_{1-i}$. Then Algorithm Ziel&PSolver correctly computes the sets $\mathsf{Win}(G, i, i\mathsf{Parity}(\alpha))$, for $i \in \{0,1\}$.*

An extension of Zielonka's algorithm to generalized parity games[7] is introduced in [6]. This algorithm, that we call GenZielonka, has $O(|E|\cdot|V|^{2D})\binom{D}{d_1,\ldots,d_k}$ time complexity where $D = \Sigma_{\ell=1}^{k} d_\ell$. While more complex, it has the same behavior with respect to the recursive call: an attractor X is computed as part of the solution of one player and a recursive call is executed on the subgame $G \setminus X$. Therefore, this algorithm can be combined with a partial solver for generalized parity games, as long as the latter satisfies the assumptions of Proposition 2.

In the next sections, we present three partial solvers for parity games and their extension to generalized parity games. They all satisfy the assumptions of Proposition 2 since the partial solutions that they compute are composed of one or several attractors.

4 Algorithms BüchiSolver and GenBüchiSolver

In this section, we present simple partial solvers for parity games and generalized parity games. More elaborate and powerful partial solvers are presented in the next two sections. These first partial solvers are based on Propositions 3 and 4 that are direct consequences of the definition of parity games and generalized parity games. The first proposition states that for parity games, if player i can ensure to visit infinitely often an i-priority without visiting infinitely often a greater $(1-i)$-priority, then he is winning for $i\mathsf{Parity}(\alpha)$.

Proposition 3. *Given a parity game $(G, \mathsf{EvenParity}(\alpha))$ and an i-priority $p \in [d]$, let $U = \{v \in V \mid \alpha(v) = p\}$ and $U' = \{v \in V \mid \alpha(v)$ is a $(1-i)$-priority and $\alpha(v) > p\}$. If $v_0 \in \mathsf{Win}(G, i, \mathsf{Büchi}(U) \cap \mathsf{CoBüchi}(U'))$, then $v_0 \in \mathsf{Win}(G, i, i\mathsf{Parity}(\alpha))$.*

The second proposition states that for generalized parity games, *(i)* if player 0 can ensure to visit infinitely often a 0-priority p_ℓ without visiting infinitely often a 1-priority greater than p_ℓ on *all dimensions* ℓ, then he is winning in the generalized parity game, and *(ii)* if player 1 can ensure to visit infinitely often a 1-priority p_ℓ without visiting infinitely often a 0-priority greater than p_ℓ on *some dimension* ℓ, then he is also winning.

Proposition 4. *Let $(G, \mathsf{ConjEvenParity}(\alpha_1, \ldots, \alpha_k))$ be a generalized parity game.*

- *Let $p = (p_1, \ldots, p_k)$, with $p_\ell \in [d_\ell]$, be a vector of 0-priorities. For all ℓ, let $U_\ell = \{v \in V \mid \alpha_\ell(v) = p_\ell\}$. Let $U' = \{v \in V \mid \exists \ell,\ \alpha_\ell(v)$ is a 1-priority and $\alpha_\ell(v) > p_\ell\}$. Then for all $v_0 \in V$,*

$$v_0 \in \mathsf{Win}(G, 0, \mathsf{GenBüchi}(U_1, \ldots, U_k) \cap \mathsf{CoBüchi}(U')) \implies v_0 \in \mathsf{Win}(G, 0, \mathsf{ConjEvenParity}(\alpha_1, \ldots, \alpha_k)).$$

[7] This algorithm is referred to as "the classical algorithm" in [6].

– *Let* $\ell \in \{1, \ldots, k\}$ *and* $p_\ell \in [d_\ell]$ *be a* 1*-priority. Let* $U = \{v \in V \mid \alpha_\ell(v) = p_\ell\}$ *and* $U' = \{v \in V \mid \alpha_\ell(v)$ *is a* 0*-priority and* $\alpha_\ell(v) > p_\ell\}$*. Then for all* $v_0 \in V$*,*

$$\begin{aligned} & v_0 \in \mathsf{Win}(G, 1, \mathsf{Büchi}(U) \cap \mathsf{CoBüchi}(U')) \\ \implies & v_0 \in \mathsf{Win}(G, 1, \mathsf{DisjOddParity}(\alpha_1, \ldots, \alpha_k)). \end{aligned}$$

Partial Solver for Parity Games. A partial solver for parity games is easily derived from Proposition 3. The polynomial time algorithm BüchiSolver (see Algorithm 2) computes winning vertices for each player by applying this proposition as follows. Let us denote by W_p the computed winning set for priority p (line 5) and let us suppose for simplicity that the loop in line 1 treats the priorities from the highest one d to the lowest one 0. This algorithm computes W_d, $W_{d-1}, \ldots$, until finding $W_p \neq \emptyset$. At this stage, it was able to find some winning vertices. It then repeats the process on $G \setminus W_p$ to find other winning vertices. At the end of the execution, it returns the computed partial solutions $\{Z_0, Z_1\}$.

Algorithm 2 BüchiSolver(G, α)

```
1 for each p ∈ [d] do
2     i = p mod 2
3     U = {v ∈ V | α(v) = p}
4     U' = {v ∈ V | α(v) is a (1 − i)-priority > p}
5     W = Win(G, i, Büchi(U) ∩ CoBüchi(U'))
6     if W ≠ ∅ then
7         {Z_i, Z_{1−i}} = BüchiSolver(G \ W, α)
8         return {Z_i ∪ W, Z_{1−i}}
9 return {∅, ∅}
```

Notice that we could replace line 5 by $W' = \mathsf{Win}(G, i, \mathsf{Büchi}(U) \cap \mathsf{Safe}(U'))$ and $W = \mathsf{Attr}_i(G, W')$ since $\mathsf{Attr}_i(G, W') \subseteq \mathsf{Win}(G, i, \mathsf{Büchi}(U) \cap \mathsf{CoBüchi}(U'))$ (as the parity objective is closed under attractor; and computing the attractor is necessary to get a subgame for the recursive call). This variant is investigated in [13, 14] under the name of Algorithm psolB.

Partial Solver for Generalized Parity Games. Similarly, a partial solver for generalized parity games can be derived from Proposition 4. This algorithm is called GenBüchiSolver (see Algorithm 3). Instead of considering all $p \in [d]$ as done in line 1 of Algorithm 2, we here have a loop on all elements, stored in a list L (see line 1), that are either a 1-*priority* p for some given *dimension* ℓ (case of player 1), or a *vector* $(p_1, \ldots, p_k)$ with 0-*priorities* (case of player 0). As in Algorithm BüchiSolver, objective $\mathsf{CoBüchi}(U')$ can be replaced by $\mathsf{Safe}(U')$ in lines 5 and 12 (with the addition of an attractor computation). This modification yields an algorithm in $O((\Pi_{\ell=1}^{k} \frac{d_\ell}{2}) \cdot k \cdot |V|^2 \cdot |E|)$ time by Theorem 1.

Algorithm 3 GenBüchiSolver$(G, \alpha_1, \dots, \alpha_k, L)$

for each *element* $\in L$ **do**
  if *element* $= (p, \ell)$ **then**
    $U = \{v \in V \mid \alpha_\ell(v) = p\}$
    $U' = \{v \in V \mid \alpha_\ell(v)$ is a 0-priority and $\alpha_\ell(v) > p\}$
    $W = \mathsf{Win}(G, 1, \mathsf{Büchi}(U) \cap \mathsf{CoBüchi}(U'))$
    if $W \neq \emptyset$ **then**
      $\{Z_0, Z_1\} =$ GenBüchiSolver$(G \setminus W, \alpha_1, \dots, \alpha_k, L)$
      return $\{Z_0, Z_1 \cup W\}$
  else {We know that *element* $= (p_1, \dots, p_k)$}
    for each ℓ **do** $U_\ell = \{v \in V \mid \alpha_\ell(v) = p_\ell\}$
    $U' = \{v \in V \mid \exists \ell,\ \alpha_\ell(v)$ is a 1-priority and $\alpha_\ell(v) > p_\ell\}$
    $W = \mathsf{Win}(G, 0, \mathsf{GenBüchi}(U_1, \dots, U_k) \cap \mathsf{CoBüchi}(U'))$
    if $W \neq \emptyset$ **then**
      $\{Z_0, Z_1\} =$ GenBüchiSolver$(G \setminus W, \alpha_1, \dots, \alpha_k, L)$
      return $\{Z_0 \cup W, Z_1\}$
return $\{\emptyset, \emptyset\}$

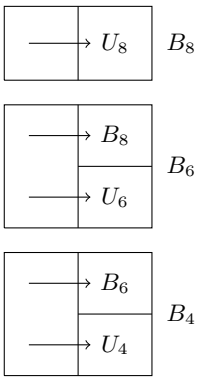

Fig. 1. Layered structure of attractors with $U_8 \subseteq U_6 \subseteq U_4 = U$ and $B_8 \subseteq B_6 \subseteq B_4$.

5 Algorithms GoodEpSolver and GenGoodEpSolver

In this section, we present a second polynomial time partial solver for parity games as proposed in [14]. We then explain how to extend it for partially solving generalized parity games. We finally explain how the latter solver can be transformed into a more efficient (in practice) antichain-based algorithm.

Partial Solver for Parity Games. We consider the *extended* game structure $G \times M$ with $M = [d]$ such that $m \in M$ records the *maximum visited priority*. More precisely, the set of vertices of $G \times M$ is equal to $V \times M$ (where $V_i \times M$ are the vertices controlled by player i), and the set E_M of its edges is composed of all pairs $((v, m), (v', m'))$ such that $(v, v') \in E$ and $m' = \max\{m, \alpha(v)\}$. Clearly, with this construction, in G, player i can ensure to visit v from v_0 such that the maximum visited priority (v excluded) is an i-priority if and only if in $G \times M$, he can ensure to visit (v, m) from $(v, \alpha(v_0))$ for some i-priority m.

Given a player i, we then compute the following fixpoint $F^{(i)} = \bigcap_{j \geq 0} F_j$. Initially $F_0 = V$ and for $j \geq 1$, F_j is computed from F_{j-1} as follows:

$$T_j = \{(v, m) \in V \times M \mid v \in F_{j-1} \text{ and } m \text{ is an } i\text{-priority}\} \tag{3}$$

$$A_j = \mathsf{PAttr}_i(G \times M, T_j) \tag{4}$$

$$F_j = \{v \in V \mid (v, \alpha(v)) \in A_j)\} \cap F_{j-1}. \tag{5}$$

Intuitively, if $v_0 \in F_j$, then player i has a strategy to ensure to visit some vertex $v \in F_{j-1}$ such that the maximum visited priority along the consistent history hv from v_0 to v (priority of v excluded) is some i-priority. We say that h is a *good episode*. Notice that h is non empty since each A_j is a positive attractor.

Algorithm 4 GoodEpSolver(G, α)

```
for each i ∈ {0,1} do
  W = GoodEp_i(G, α)
  if W ≠ ∅ then
    X = Attr_i(G, W)
    {Z_i, Z_{1−i}} = GoodEpSolver(G\X, α)
    return {Z_i ∪ X, Z_{1−i}}
return ∅, ∅
```

Algorithm 5 LaySolver(G, α)

```
for each P_{≥q} with q ∈ [d] do
  i = parity of the priorities in P_{≥q}
  W = LayEp_i(G, α, P_{≥q})
  if W ≠ ∅ then
    X = Attr_i(G, W)
    {Z_i, Z_{1−i}} = LaySolver(G\X, α)
    return {Z_i ∪ X, Z_{1−i}}
return ∅, ∅
```

We denote the fixpoint $F^{(i)}$ by $\mathsf{GoodEp}_i(G, \alpha)$. From a vertex v_0 in this fixpoint, player i can ensure a succession of good episodes in which the maximum visited priority is an i-priority. Thus he is winning from v_0 for $i\mathsf{Parity}(\alpha)$ as formalized in the next proposition (notice that if v_0 belongs to the attractor for player i of $\mathsf{GoodEp}_i(G, \alpha)$, then v_0 still belongs to $\mathsf{Win}(G, i, i\mathsf{Parity}(\alpha))$). From Proposition 5, we derive a polynomial time algorithm called GoodEpSolver (see Algorithm 4). This algorithm is called psolC in [14].

Proposition 5 ([14]). *Let $(G, \mathsf{EvenParity}(\alpha))$ be a parity game. For all $v_0 \in V$, if $v_0 \in \mathsf{Attr}_i(G, F^{(i)})$, then $v_0 \in \mathsf{Win}(G, i, i\mathsf{Parity}(\alpha))$.*

Partial Solver for Generalized Parity Games. The same approach can be applied to generalized parity games with some adaptations that depend on the player. For player 1, instead of applying Proposition 3, we now apply the GoodEpSolver approach by computing $W = \mathsf{GoodEp}_1(G, \alpha_\ell)$ for each dimension ℓ.

For player 0, we have to treat vectors of 0-priorities. We thus consider the extended game structure $G \times M_1 \times \ldots \times M_k$ such that for all ℓ, M_ℓ is equal to $[d_\ell]$, and where $m_\ell \in M_\ell$ records the maximum visited priority in *dimension* ℓ according to function α_ℓ. Hence, the edges of this game structure have the form $((v, m_1, \ldots, m_k), (v', m'_1, \ldots, m'_k))$ such that $(v, v') \in E$ and $m'_\ell = \max\{m_\ell, \alpha_\ell(v)\}$ for all ℓ. A good episode is now a history h such that for all ℓ, the maximum priority visited along h in dimension ℓ is a 0-priority. And the related set $\mathsf{GoodEp}_0(G, \alpha_1, \ldots, \alpha_\ell)$ equal to $\bigcap_{j \geq 0} F_j$ is computed with Equations (3-5) modified as expected. The resulting algorithm, called GenGoodEpSolver, has a time complexity in $O((\Pi_{\ell=1}^{k} d_\ell) \cdot |V|^2 \cdot |E|)$ by Theorem 1.

Partial Solver with Antichains. Algorithm GenGoodEpSolver has exponential time complexity due to the use of the game structure $G \times M_1 \times \ldots \times M_k$, and in particular to the computation of $\mathsf{GoodEp}_0(G, \alpha_1, \ldots, \alpha_k)$. We here show that the vertices of this extended game can be partially ordered in a way to obtain an antichain-based algorithm for $\mathsf{GoodEp}_0(G, \alpha_1, \ldots, \alpha_k)$. The antichains allow compact representation and efficient manipulation of partially ordered sets [8].

Let us first recall the basic notions about antichains. Consider a *partially ordered set* $(S, \preceq)$ where S is a finite set and $\preceq \subseteq S \times S$ is a partial order on S.

Given $s, s' \in S$, we write $s \sqcap s'$ their *greatest lower bound* if it exists. A *lower semilattice* is a partially ordered set such that this greater lower bound always exists for all $s, s' \in S$. Given two subsets $R, R' \subseteq S$, we denote by $R \sqcap R'$ the set $\{s \sqcap s' \mid s \in R, s' \in R'\}$. An *antichain* $A \subseteq S$ is a set composed of pairwise incomparable elements with respect to $\preceq$. Given a subset $R \subseteq S$, we denote $\lceil R \rceil$ the set of its *maximal* elements (which is thus an antichain). We say that R is *closed* if whenever $s \in R$ and $s' \preceq s$, then $s' \in R$. If A is an antichain, we denote by $\downarrow A$ the closed set that it *represents*, that is, $A = \lceil \downarrow A \rceil$. Hence when R is closed, we have $R = \downarrow \lceil R \rceil$. The benefit of antichains is that they provide a *compact representation* of closed sets. Moreover some operations on those closed sets can be done at the level of their antichains:

Proposition 6 ([8]). *Let $(S, \preceq)$ be a lower semilattice, and $R, R' \subseteq S$ be two closed sets represented by their antichains $A = \lceil R \rceil, A' = \lceil R' \rceil$. Then*

- *for all $s \in S$, $s \in R$ if and only if there exists $s' \in A$ such that $s \preceq s'$,*
- *$R \cup R'$, $R \cap R'$ are closed, and $\lceil R \cup R' \rceil = \lceil A \cup A' \rceil$, $\lceil R \cap R' \rceil = \lceil A \sqcap A' \rceil$.*

For *simplicity*, we focus on the extended structure $G \times M$ associated to a parity game and begin to explain an antichain-based algorithm for the computation of the set $\mathsf{GoodEp}_0(G, \alpha)$ (this algorithm is inspired from [10]). We explain later what are the required adaptations to compute $\mathsf{GoodEp}_0(G, \alpha_1, \ldots, \alpha_k)$ in generalized parity games. We equip $V \times M$ with the following partial order:

Definition 7. *We define the strict partial order $\prec$ on $V \times M$ such that $(v', m') \prec (v, m)$ if and only if $v = v'$ and (i) either m, m' are even and $m' > m$, (ii) or m, m' are odd and $m' < m$, (iii) or m is odd and m' is even. We define $(v', m') \preceq (v, m)$ if either $(v', m') = (v, m)$ or $(v', m') \prec (v, m)$.*

For instance, if $[d] = [4]$, then $(v, 4) \prec (v, 2) \prec (v, 0) \prec (v, 1) \prec (v, 3)$. With this definition, two elements $(v, m), (v', m')$ are incomparable as soon as $v \neq v'$. It follows that in Proposition 6, if $R \subseteq \{v\} \times M$ and $R' \subseteq \{v'\} \times M$ with $v \neq v'$, then the union $A \cup A'$ of their antichains is already an antichain.

The computation of $\mathsf{GoodEp}_0(G, \alpha)$ is based on Equations (3-5). Equation (4) involves the computation of positive attractors over $G \times M$ thanks to Equations (1-2). Let us show that $\mathsf{GoodEp}_0(G, \alpha)$ is a closed set and that so are all the intermediate sets used to compute it. We already know from Proposition 6 that the family of closed sets is stable under union and intersection. So we now focus on the operation $\mathsf{Cpre}_0(G \times M, U)$. We introduce the following functions *up* and *down*. Given $(v, m) \in V \times M$, let $up(m, \alpha(v)) = \max\{m, \alpha(v)\}$. Recall that such an update from m to $m' = up(m, \alpha(v))$ is used in the edges $((v, m), (v', m'))$ of $G \times M$. Function *down* is the inverse reasoning of function *up*: given $(v', m') \in V \times M$ and $(v, v') \in E$, the value $down(m', \alpha(v))$ yields the maximal value m such that $(v', up(m, \alpha(v))) \preceq (v', m')$.

Definition 8. *Given $(v', m') \in V \times M$ and $p = \alpha(v)$, we define $m = down(m', p)$ as follows: (i)* **Case** *p* **even***: if $p < m'$ then $m = m'$, else $m = \max\{p - 1, 0\}$; (ii)* **Case** *p* **odd***: if $p \leq m'$ then $m = m'$, else $m = p + 1$ except if $p = d$ in which case $down(m', p)$ is not defined.*

The next proposition states that if a set $U \subseteq V \times M$ is closed, then the set $\mathsf{Cpre}_0(G \times M, U)$ is also closed. It also indicates how to design an antichain-based algorithm for computing $\mathsf{Cpre}_0(G \times M, U)$.

Proposition 9. *If $U \subseteq V \times M$ is a closed set, then $\mathsf{Cpre}_0(G \times M, U)$ is closed. Let $A = \lceil U \rceil$ be the antichain representing U. Then $\lceil \mathsf{Cpre}_0(G \times M, U) \rceil = \lceil B_0 \cup B_1 \rceil$ where B_0, B_1 are the following antichains:*

$$B_0 = \bigcup_{v \in V_0} \Big\lceil \{(v, \mathit{down}(m', \alpha(v))) \mid (v, v') \in E, (v', m') \in A\} \Big\rceil,$$
$$B_1 = \bigcup_{v \in V_1} \Big\lceil \prod_{(v,v') \in E} \lceil \{(v, \mathit{down}(m', \alpha(v))) \mid (v', m') \in A\} \rceil \Big\rceil.$$

Let us come back to the computation of $\mathsf{GoodEp}_0(G, \alpha)$, which depends on Equations (3-5) and Equations (1-2). As each T_j of Equation (3) is a closed set represented by the antichain $\lceil T_j \rceil = \{(v, 0) \mid v \in F_j\}$, by Propositions 6 and 9, we get an antichain-based algorithm for computing $\mathsf{GoodEp}_0(G, \alpha)$ as announced.

This antichain approach can be extended to generalized parity games and their extended game $G \times M_1 \times \ldots \times M_k$. The approach is similar and works *dimension by dimension* as we did before for parity games. First, we define a partial order on $V \times M_1 \times \ldots \times M_k$ such that the partial order of Definition 7 is used on each dimension. More precisely, $(v', m'_1, \ldots, m'_k) \prec (v, m_1, \ldots, m_k)$ if and only if $v = v'$ and for all ℓ, both m_ℓ and m'_ℓ respect Definition 7. Second, functions up_ℓ and down_ℓ, with $\ell \in \{1, \ldots, k\}$, are defined exactly as previous functions up and down (see Definition 8), for each dimension ℓ and with respect to priority function α_ℓ. Third, we adapt (as expected) Proposition 9 for the computation of $\mathsf{Cpre}_0(G \times M_1 \times \ldots \times M_k, U)$ for a closed set $U \subseteq V \times M_1 \times \ldots \times M_k$. Finally, we obtain an antichain-based algorithm for $\mathsf{GoodEp}_0(G, \alpha_1, \ldots, \alpha_k)$ as each set T_j in (3) is a closed set represented by the antichain $\lceil T_j \rceil = \{(v, 0, \ldots, 0) \mid v \in F_j\}$.

6 Algorithms LaySolver and GenLaySolver

In [13], the authors study another polynomial time partial solver for parity games, called psolQ, that has similarities with the GoodEpSolver approach of Section 5. It also generalizes the BüchiSolver approach of Section 4. It is a more complex partial solver that we present on an example. We then explain how to modify it for generalized parity games.

Partial Solver for Parity Games. The new partial solver works on the initial game structure G, focuses on a *subset $P_{\geq q}$ of i-priorities* and computes a set similar to $\mathsf{GoodEp}_i(G, \alpha)$ such that the positive attractor $\mathsf{PAttr}_i(G \times M, T_j)$ of Equation (4) is replaced by a *layered attractor* $\mathsf{LayAttr}_i(G, \alpha, P_{\geq q}, U)$ (one layer per priority $p \in P_{\geq q}$).

Let us explain on an example. First, we denote by $\mathsf{PSafeAttr}_i(G, U, U')$ the *positive safe attractor* composed of vertices from which player i can ensure to visit U in any positive number of steps while *not visiting* U'. Now, take the example of a parity game with $d = 9$ and fix a 0-priority $q = 4$. Let $P_{\geq q}$

be the set of all 0-priorities $p \geq q$, that is, $P_{\geq q} = \{4, 6, 8\}$. Given some set $U \subseteq \{v \in V \mid \alpha(v) \in P_{\geq q}\}$, we consider U_8 (resp. U_6, U_4) being the set of vertices of U with priority 8 (resp. priorities in $\{6, 8\}$, in $\{4, 6, 8\}$). Notice that $U_8 \subseteq U_6 \subseteq U_4 = U$. We also consider U'_8 (resp. U'_6, U'_4) the set of vertices with priority 9 (resp. priorities in $\{7, 9\}$, in $\{5, 7, 9\}$).

We compute the following sequence of positive safe attractors (see Figure 1): Initially $B_{10} = \emptyset$ and for all $p \in P_{\geq q}$, B_p is computed from B_{p+2} as follows:

$$B_p = B_{p+2} \cup \mathsf{PSafeAttr}_0(G, U_p \cup B_{p+2}, U'_p \setminus B_{p+2}). \tag{6}$$

The last computed set B_4 is the layered attractor $\mathsf{LayAttr}_0(G, \alpha, P_{\geq q}, U)$. Notice that $B_8 \subseteq B_6 \subseteq B_4$. Let us give some intuition. From a vertex in $B_4 \setminus B_6$ (lowest layer 4), player 0 can ensure to visit $U_4 \cup B_6$ without visiting $U'_4 \setminus B_6$. In case of a visit to U_4, this is a good episode for himself (in the sense of Section 5) since the maximum visited priority is a 0-priority ≥ 4. In case of a visit to some $v \in B_6 \setminus B_8$, player 0 can now ensure to visit $U_6 \cup B_8$ without visiting $U'_6 \setminus B_8$. In case of a visit to U_6, this is again a good episode for him, otherwise it is a visit to B_8 in the highest layer from which player 0 can ensure to visit U_8 without visiting U'_8 (since B_{10} is empty). Thus from all vertices of B_4, player 0 can ensure a good episode for himself.

The new partial solver, called LaySolver (see Algorithm 5), is the same as Algorithm GoodEpSolver of Section 5 except that *(i)* in Equation (4), the layered attractor $\mathsf{LayAttr}_i(G, \alpha, P_{\geq q}, U)$ in the game G replaces $\mathsf{PAttr}_i(G \times M, T)$ in the extended game $G \times M$, and *(ii)* the subset $P_{\geq q} = \{q, q+2, q+4, \ldots\}$ of i-priorities replaces the set of all i-priorities.

Partial Solver for Generalized Parity Games. We explain how to adapt the LaySolver approach to generalized parity games, only for player 0. Indeed for player 1 we can apply the previous LaySolver approach on each α_ℓ separately.

For player 0, take the example of a generalized parity game $d_\ell = 9$ for all ℓ. We fix $q = (4, \ldots, 4)$ and $P_{\geq q} = \{(4, \ldots, 4), (6, \ldots, 6), (8, \ldots, 8)\}$. Take a vector $p \in P_{\geq q}$, (say with $p_\ell = 6, \forall \ell$) and a subset $U \subseteq V$. In a first step, let us focus on how player 0 can ensure to visit U such that along the history, for all ℓ, a 0-priority $\geq p_\ell$ is visited and no 1-priority $> p_\ell$ is visited (in a way to extend Equation (6) temporarily without set B_{p+2}). Such a generalized reachability can be reduced to reachability by working with an extended game G_p such that in vertex (v, N), the memory $N \subseteq \{1, \ldots, k\}$ records the dimensions ℓ for which a vertex with 0-priority $\geq p_\ell$ is already visited. We then work with the positive safe attractor $\mathsf{PSafeAttr}_0(G_p, T_p, T'_p)$ such that $T_p = \{(v, N) \mid v \in U, N = \{1, \ldots, k\}\}$ and $T'_p = \{(v, N) \mid \exists \ell,\ \alpha_\ell(v)$ is a 1-priority $> p_\ell\}$. In a second step, let us show how to manage the set B_{p+2} in Equation (6). For parity games we explained how player 0 has to adapt his attractor strategy when he shifts from layer p to some higher layer $p' > p$ due to the visit to some $v \in B_{p+2}$. Here when player 0 visits some vertex (v, N) in layer p for which he has to shift to layer p', he stops applying his current strategy, and begins applying his strategy for layer p' from the vertex $(v, N_{p'}(v))$ belonging to layer p' with

initial memory $N_{p'}(v) = \{\ell \mid \alpha_\ell(v) \text{ is a 0-priority} \geq p_\ell\}$. All these modifications lead to the layered attractor $\mathsf{LayAttr}_0(G, \alpha_1, \ldots, \alpha_k, P_{\geq q}, F_j)$ used in place of $\mathsf{LayAttr}_0(G, \alpha, P_{\geq q}, F_j)$. In this way, we derive an algorithm called GenLaySolver with $O((\max_{\ell=1}^{k} \frac{d_\ell}{2})^2 \cdot |V|^2 \cdot |E| \cdot 2^k)$ time complexity.

7 Empirical Evaluation

For parity games, the polynomial time partial solvers BüchiSolver[8], GoodEpSolver, and LaySolver are theoretically compared in [13, 14]. It is proved that the partial solutions computed by Algorithm BüchiSolver are included in those computed by Algorithm LaySolver themselves included in those computed by Algorithm GoodEpSolver. Examples of parity games are also given that distinguish the three partial solvers (strict inclusion of partial solutions), as well as an example that is not completely solved by the most powerful of these partial solvers. This behavior also holds for the partial solvers proposed here for generalized parity games. Moreover, their time complexity is exponential in the number k of priority functions while the classical algorithm for generalized parity games is exponential in both k and all d_ℓ [6].

For both parity games and generalized parity games, we implemented in Python 2.7 the three partial solvers (with the antichain approach for Algorithm GenGoodEpSolver), Algorithm Zielonka (resp. GenZielonka) and its combination Ziel&PSolver (resp. GenZiel&PSolver) with each partial solver, and we executed all these algorithms on a large set of benchmarks. Our benchmarks were generated from TLSF specifications used for the Reactive Synthesis Competition (SYNTCOMP [15]) using the compositional translation explained in the introduction.[9] The source code for our prototype tool along with all the information about our benchmarks is made publicly available at https://github.com/Skar0/generalizedparity. Our experiments have been carried out on a server with Mac OS X 10.13.4 (build 17E199). As hardware, the server had as CPU one 6-Core Intel Xeon; as processor speed, 3.33 GHz; as L2 Cache (per Core), 256 KB; as L3 Cache, 12 MB; as memory, 32 GB; and as processor interconnect speed, 6.4 GT/s.

Experiments. We considered 240 benchmarks for parity games. Those games have a mean size $|V|$ around 46K with a maximal size of 3157K, and a mean number d of priorities of 4.1 with a maximal number $d = 15$. The statistics about the behaviors of the different algorithms are summarized in Table 1 and divided into two parts: the first part concerns all the 240 benchmarks and the second part the 20 most difficult benchmarks for Zielonka's algorithm. Column 1 indicates the name of the solver, Columns 2 and 6 count the number of benchmarks completely solved (for the partial solvers, the second number is the number of incomplete solutions), Columns 3 and 7 count the number of timeouts (fixed

[8] The variant with safety objectives.

[9] The tool we implemented to realize this translation can be fetched from https://github.com/gaperez64/tlsf2gpg

Solver	Solved	T.O.	Fastest	Mean	Solved	T.O.	Fastest	Mean
Zielonka	240 (100%)	0	150 (63 %)	272 ms	20 (100%)	0	11 (55%)	451 ms
Ziel&BüchiSolver	240 (100%)	0	89 (37 %)	480 ms	20 (100%)	0	8 (40%)	7746 ms
Ziel&GoodEpSolver	233 (97%)	7	0 (0%)	1272 ms	13 (65%)	7	0 (0%)	20025 ms
Ziel&LaySolver	238 (99%)	2	1 (0%)	587 ms	18 (99%)	2	1 (5%)	9079 ms
BüchiSolver	203 (84%) - 37	0	-	-	15 (75%) - 5	0	-	-
GoodEpSolver	233 (97%) - 0	7	-	-	13 (65%) - 0	7	-	-
LaySolver	232 (97%) - 6	2	-	-	18 (90%) - 0	2	-	-

Table 1. Statistics on the one dimensional benchmarks.

Solver	Solved	T.O.	Fastest	Mean
GenZielonka	128 (84%)	24	33 (25%)	66 ms
GenZiel&GenBüchiSolver	130 (86%)	22	72 (55%)	56 ms
GenZiel&GenGoodEpSolver	112 (74%)	40	24 (18%)	644 ms
GenZiel&GenLaySolver	110 (72%)	42	3 (2%)	1133 ms
GenBüchiSolver	110 (72%) - 20	22	-	-
GenGoodEpSolver	112 (74%) - 0	40	-	-
GenLaySolver	104 (68%) - 6	42	-	-

Table 2. Statistics on the multi-dimensional benchmarks.

at 60000 ms), and Columns 4 and 8 count how many times the solver was the fastest (excluding examples with timeout). For the 233 (resp. 13) benchmarks without timeout for Zielonka's algorithm and all its combinations with a partial solver, Column 5 (resp. 9) indicates the mean execution time in milliseconds.

We considered 152 benchmarks for generalized parity games. Those games have a mean size $|V|$ around 207K with a maximal size of 7009K. The mean number of priority functions is equal to 4.53 with a maximum number of 17. The statistics about the behaviors of the different algorithms are summarized in Table 2. The columns have the same meaning as before and the last column concerns the 87 benchmarks without timeout for all Algorithms GenZielonka and GenZiel&PSolver.

Observations. Our experiments show that for parity games, Zielonka's algorithm is faster than partial solvers on average which was not observed on random graphs in [14]. For generalized parity games, they show that 4 benchmarks can be solved only by the combination of GenZielonka with a partial solver. Our experiments also show that the combination with a partial solver improves the performances of Zielonka's algorithm (resp. GenZielonka): for 90 cases over 240 (38%) (resp. for 99 cases over 132 (75%)). For generalized parity games, they suggest that it is interesting to launch in parallel all three Algorithms GenZiel&PSolver, as none appears to dominate the other ones. Nevertheless, the combination of GenZielonka with GenBüchiSolver is a good compromise.

References

1. Ah-Fat, P., Huth, M.: Partial solvers for parity games: Effective polynomial-time composition. In: GandALF Proceedings. EPTCS, vol. 226, pp. 1–15 (2016). https://doi.org/10.4204/EPTCS.226.1
2. Bloem, R., Chatterjee, K., Greimel, K., Henzinger, T.A., Jobstmann, B.: Robustness in the presence of liveness. In: CAV Proceedings. Lecture Notes in Computer Science, vol. 6174, pp. 410–424. Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_36
3. Calude, C.S., Jain, S., Khoussainov, B., Li, W., Stephan, F.: Deciding parity games in quasipolynomial time. In: STOC Proceedings. pp. 252–263. ACM (2017). https://doi.org/10.1145/3055399.3055409
4. Chatterjee, K., Dvorák, W., Henzinger, M., Loitzenbauer, V.: Conditionally optimal algorithms for generalized Büchi games. In: MFCS Proceedings. LIPIcs, vol. 58, pp. 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.25
5. Chatterjee, K., Henzinger, M.: Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. J. ACM **61**(3), 15:1–15:40 (2014). https://doi.org/10.1145/2597631
6. Chatterjee, K., Henzinger, T.A., Piterman, N.: Generalized parity games. In: Seidl, H. (ed.) FOSSACS Proceedings. Lecture Notes in Computer Science, vol. 4423, pp. 153–167. Springer (2007). https://doi.org/10.1007/978-3-540-71389-0_12
7. van Dijk, T.: Oink: An implementation and evaluation of modern parity game solvers. In: TACAS Proceedings, Part I. Lecture Notes in Computer Science, vol. 10805, pp. 291–308. Springer (2018). https://doi.org/10.1007/978-3-319-89960-2_16
8. Doyen, L., Raskin, J.: Antichain algorithms for finite automata. In: TACAS Proceedings. Lecture Notes in Computer Science, vol. 6015, pp. 2–22. Springer (2010). https://doi.org/10.1007/978-3-642-12002-2_2
9. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy (extended abstract). In: FOCS Proceedings. pp. 368–377. IEEE Computer Society (1991). https://doi.org/10.1109/SFCS.1991.185392
10. Filiot, E., Jin, N., Raskin, J.: Exploiting structure in LTL synthesis. STTT **15**(5-6), 541–561 (2013). https://doi.org/10.1007/s10009-012-0222-5
11. Friedmann, O., Lange, M.: Solving parity games in practice. In: ATVA Proceedings. Lecture Notes in Computer Science, vol. 5799, pp. 182–196. Springer (2009). https://doi.org/10.1007/978-3-642-04761-9_15
12. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001], Lecture Notes in Computer Science, vol. 2500. Springer (2002). https://doi.org/10.1007/3-540-36387-4
13. Huth, M., Kuo, J.H., Piterman, N.: Fatal attractors in parity games. In: FOSSACS Proceedings. Lecture Notes in Computer Science, vol. 7794, pp. 34–49. Springer (2013). https://doi.org/10.1007/978-3-642-37075-5_3
14. Huth, M., Kuo, J.H., Piterman, N.: Static analysis of parity games: Alternating reachability under parity. In: Semantics, Logics, and Calculi - Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their 60th Birthdays. Lecture Notes in Computer Science, vol. 9560, pp. 159–177. Springer (2016). https://doi.org/10.1007/978-3-319-27810-0_8

15. Jacobs, S., Bloem, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, P.J., Michaud, T., Sakr, M., Sickert, S., Tentrup, L., Walker, A.: The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results. CoRR **abs/1904.07736** (2019), http://arxiv.org/abs/1904.07736
16. Jurdzinski, M.: Deciding the winner in parity games is in UP ∩ co-UP. Inf. Process. Lett. **68**(3), 119–124 (1998). https://doi.org/10.1016/S0020-0190(98)00150-1
17. Martin, D.A.: Borel determinacy. Annals of Mathematics **102**, 363–371 (1975)
18. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL Proceedings. pp. 179–190. ACM Press (1989). https://doi.org/10.1145/75277.75293
19. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**(1-2), 135–183 (1998). https://doi.org/10.1016/S0304-3975(98)00009-7