A RECURSIVE APPROACH TO SOLVING PARITY GAMES IN QUASIPOLYNOMIAL TIME[*]

KAROLIINA LEHTINEN[a], PAWEŁ PARYS[b], SVEN SCHEWE[c], AND DOMINIK WOJTCZAK[c]

[a] CNRS, Aix-Marseille University and University of Toulon, LIS, Marseille
  e-mail address: lehtinen@lis-lab.fr
  URL: http://pageperso.lif.univ-mrs.fr/~karoliina.lehtinen/

[b] Institute of Informatics, University of Warsaw, Poland
  e-mail address: parys@mimuw.edu.pl
  URL: https://www.mimuw.edu.pl/~parys/

[c] University of Liverpool, UK
  e-mail address: {sven.schewe,d.wojtczak}@liverpool.ac.uk
  URL: https://www2.csc.liv.ac.uk/~sven/, https://www2.csc.liv.ac.uk/~dominik/

ABSTRACT. Zielonka's classic recursive algorithm for solving parity games is perhaps the simplest among the many existing parity game algorithms. However, its complexity is exponential, while currently the state-of-the-art algorithms have quasipolynomial complexity. Here, we present a modification of Zielonka's classic algorithm that brings its complexity down to $n^{\mathcal{O}\left(\log\left(1+\frac{d}{\log n}\right)\right)}$, for parity games of size n with d priorities, in line with previous quasipolynomial-time solutions.

1. INTRODUCTION

A parity game is an infinite two-player game in which Even and her opponent Odd build an infinite path along the edges of a graph labelled with integer priorities. Even's goal is for the highest priority seen infinitely often on this path to be even, while Odd tries to stop her.

Parity games are a central tool in automata theory, logic, and their applications to verification. In particular, the model-checking problem for the modal μ calculus [EJS01] and the synthesis problem for LTL [PR89] reduce to solving parity games. Solutions to parity games have also influenced work on ω-word automata translations [BL18, DJL19], linear optimisation [Fri11b, FHZ11] and stochastic planning [Fea10].

The complexity of solving parity games, that is, deciding which player has a winning strategy, is a long standing open problem. The problem is known to be both in UP and coUP [Jur98] but a polynomial algorithm remains, so far, out of reach. In 2017 Calude et al. published the first quasipolynomial-time algorithm [CJK+17], which was followed by several alternative algorithms [JL17, Leh18], variations and improvements thereupon [FJdK+19, Par20, DJT20]. The progress-measure approach [JL17] is based on succinct encodings of classical progress measures, while the register-game approach [Leh18] is based on a relatively involved analysis of the structure of winning strategies in parity games.

In this paper we present a quasipolynomial-time parity-game algorithm based on Zielonka's classic recursive algorithm. Zielonka's algorithm [Zie98], based on McNaughton's algorithm for solving Muller games [McN93], is perhaps the (conceptually) simplest parity game algorithm to date. Despite its exponential worst-case complexity, it is also one of the most performant algorithms in practice [vD18]. Here we show how to adapt this algorithm to make its worst-case complexity quasipolynomial. Our algorithm, its correctness proof, and complexity analysis are all remarkably simple. Its runtime complexity is roughly in line with previous quasipolynomial-time algorithms.

Our key insight is that, instead of each recursive call solving a subgame, we use a weakened induction hypothesis, which postulates that each call should return a partition that separates the *small dominions* of both players, up to a size specified by a pair of parameters. Generalising the observation that only one dominion can be larger than half the arena, we then use these parameterised calls to build an algorithm that only makes a quasipolynomial number of calls, but still finds the winning regions of each player.

The time complexity of our algorithm, which is in $n^{\mathcal{O}\left(\log\left(1+\frac{d}{\log n}\right)\right)}$ for games with n vertices and d priorities, is similar to the complexity of previous quasipolynomial-time algorithms. This also provides fixed-parameter tractability when d, the number of priorities, is treated as the parameter, as well as a polynomial bound for the common case where the number of priorities is logarithmic in the number of states. In a fine grained comparison our algorithm (similarly to the original Zielonka's algorithms, on which it is based) operates symmetrically, going through every priority, rather than just half of them (as in previous quasipolynomial-time algorithms), meaning that the $\mathcal{O}\left(1 + \log\frac{d}{\log n}\right)$ in the exponent hides a factor of 2. Thus, a very careful analysis still reveals a small gap, when compared to previous quasipolynomial-time algorithms.

We provide two versions of our algorithm. One has better worst-case complexity (outlined above), while the other turns out to be somewhat better in practice. We evaluate both versions against Zielonka's classic algorithm, which, despite its exponential worst-case behaviour, is in most cases faster than our quasipolynomial-time versions. In line with the theory, both our algorithms outperform Zielonka's algorithm on the families of games designed to exhibit its exponential worst-case behaviour.

We also briefly comment on the relationship between this recursive algorithm and universal trees, but refer the reader to Jurdziński and Morvan's work [JM20] for a more thorough analysis, which also addresses the symbolic implementation of recursive algorithms.

This is a journal version of Parys' paper [Par19], in which he turns Zielonka's exponential-time algorithm into a quasipolynomial-time one; additionally, it includes theoretical improvements suggested by Lehtinen, Schewe, and Wojtczak [LSW19], which bring the complexity of the algorithm down to roughly match the state of the art.

2. Preliminaries

A *parity game* is a two-player game between players Even and Odd played on a *game graph* defined as a tuple $\mathcal{G} = (V, V_\mathsf{E}, E, \pi)$, where (V, E) is a finite directed graph in which every vertex has at least one successor; its vertices are labelled with positive integer *priorities* by $\pi\colon V \to \{1, 2, \ldots, d\}$ (for some $d \in \mathbb{N}$), and partitioned between vertices V_E *belonging to Even* and vertices $V_\mathsf{O} = V \setminus V_\mathsf{E}$ *belonging to Odd*. As usual for directed graphs, we forbid self-loops (i.e., edges from a vertex to itself). We sometimes abbreviate Even and Odd to E and O, we use the symbol $\wp$ for a player (either Even or Odd), and we write $\bar{\wp}$ for the opponent of $\wp$.

A *play* ρ is an infinite path through the game graph. It is *winning for Even* if the highest priority occurring infinitely often on it is even; otherwise it is *winning for Odd*. We write $\rho[i]$ for the ith vertex in ρ (zero based) and $\rho[0, j]$ for its prefix of length $j + 1$.

A *strategy* from a vertex v for a player $\wp$ is a function that

- takes any prefix of a play starting in v and ending in a vertex that belongs to $\wp$, and
- returns one of successors of the latter vertex.

A strategy σ from v for Even (Odd) *agrees* with a play ρ if $\rho[0] = v$ and, whenever $\rho[i] \in V_\mathsf{E}$ (V_O, respectively), then $\rho[i + 1] = \sigma(\rho[0, i])$ (and σ agrees with a finite path, if it agrees with some infinite play extending the path). A strategy for a player is *winning* if it agrees only with plays winning for that player. Parity games are determined: from every vertex, one of the two players has a winning strategy [Mar75].

The *winning region* of a player $\wp$ is the set of vertices from which $\wp$ has a winning strategy. We are interested in the problem of computing, given a parity game G, the winning regions of each player.

Given a set of vertices $G \subseteq V$, we can consider a *subgame* of $\mathcal{G}$ *induced* by G, which is obtained simply by restricting all components of $\mathcal{G}$ to G. Notice, however, that not every set G induces a valid subgame: every vertex in G should have at least one successor in G. In the sequel, we often write "the subgame G" instead of the more precise "the subgame of $\mathcal{G}$ induced by G". For the definitions below, we assume some set $G \subseteq V$ inducing a subgame of $\mathcal{G}$.

A *dominion* of a player $\wp$ in a subgame G is a set of vertices $D \subseteq G$ such that from every vertex $v \in D$ the player $\wp$ has, in the subgame G, a winning strategy that agrees only with plays staying forever in D.

Given a set $S \subseteq G$, the $\wp$-*attractor* of S in the subgame G, written $\mathrm{ATTR}_\wp(S, G)$, is the set of vertices from which the player $\wp$ has a strategy in G that agrees only with plays reaching S. A set $S \subseteq G$ is $\wp$-*closed* in the subgame G if $\mathrm{ATTR}_{\bar{\wp}}(G \setminus S, G) = G \setminus S$. More concretely, this means that $\wp$ can ensure to stay inside S: every vertex of $\wp$ in S has at least one successor in S, and every vertex of $\bar{\wp}$ in S has no successor in $G \setminus S$.

Observe that the set $G \setminus \mathrm{ATTR}_\wp(S, G)$, for any $S \subseteq G$, always induces a subgame: if all successors of a vertex belong to $\mathrm{ATTR}_\wp(S, G)$, then this vertex belongs there as well. Observe also that every dominion of $\wp$ is $\wp$-closed, and that the winning region of $\wp$ is a dominion of $\wp$.

We use the following simple lemmata to prove correctness of our algorithm:

Lemma 2.1. *Let D be a dominion of a player $\wp$ in a subgame G, and let $X \subseteq G$. If D does not intersect with X, then D is also a dominion of $\wp$ in $G \setminus \mathrm{ATTR}_{\bar{\wp}}(X, G)$.*

Proof. By the definition of a dominion, from every vertex $v \in D$ player $\wp$ has a winning strategy that agrees only with plays staying in D. Observe that a play agreeing with such a strategy cannot reach a vertex in the $\bar{\wp}$-attractor of X, because from such a vertex the opponent can force to leave D and enter X. Thus, the same strategies (after restricting them appropriately) witness that D is also a dominion in the smaller subgame; in particular D does not intersect with $\text{ATTR}_{\bar{\wp}}(X, G)$. $\qquad\square$

Lemma 2.2. *Let D be a dominion of a player $\wp$ in a subgame G, and let $X \subseteq G$ be $\bar{\wp}$-closed in G. Then $D \cap X$ is a dominion of $\wp$ in X.*

Proof. Because X is $\bar{\wp}$-closed in G, no vertex of $\wp$ in $D \cap X$ has a successor in $G \setminus X$. In consequence, strategies that witness D being a dominion of $\wp$ in G, after restricting them appropriately, witness $D \cap X$ being a dominion of $\wp$ in X. $\qquad\square$

Lemma 2.3. *If all priorities in a non-empty dominion D of a player $\wp$ are at most d', and d' is not of $\wp$'s parity, then D contains a non-empty sub-dominion C without vertices of priority d'.*

Proof. Fix a strategy σ for $\wp$ from some vertex $v_0 \in D$ that is winning and agrees only with plays staying in D. To C we take all vertices v of D such that

- there exists a finite path ρ_v that ends in v and agrees with σ, and
- there is no finite path ρ' that extends ρ_v, agrees with σ, and ends in a vertex of priority d' (in particular, v itself is not of priority d').

Such a set C is nonempty; otherwise, it would be easy to construct a play that agrees with σ and visits priority d' infinitely often (start from v_0, continue to a vertex of priority d', which is possible because $v_0 \notin C$, continue to an arbitrary successor v_1, continue to a vertex of priority d', which is possible because $v_1 \notin C$, and so on). Then, for a vertex $v \in C$ we consider a strategy σ_v such that $\sigma_v(\rho) = \sigma(\rho_v \cdot \rho)$ for all plays ρ starting in v; this strategy is winning for $\wp$ (if σ_v agrees with a play ρ, then σ agrees with $\rho_v \cdot \rho$, and hence both these plays are winning) and agrees only with plays staying in C (by the definition of C). $\qquad\square$

3. A FIRST QUASIPOLYNOMIAL-TIME VERSION

In this section we present our first quasipolynomial-time algorithm solving parity games. Informally, we refer to it as the Liverpool variant, since it is (close to) an algorithm introduced by Lehtinen, Schewe, and Wojtczak [LSW19], working at University of Liverpool.

We remark that, historically, the Liverpool variant came after the Warsaw variant, presented in the next section. We present the Liverpool variant first, as the "main variant", because it has better theoretical complexity and because we think that it is more elegant.

3.1. The algorithm.
First, recall Zielonka's classic recursive algorithm. Each iteration first identifies the set N_d of vertices of the highest priority, which we can assume to be even as the odd case is symmetric, and the E-attractor of N_d. It then makes a recursive call to solve the rest of the game, which has one fewer priority. Odd's winning region in this subgame, as well as its O-attractor, is winning for Odd in the whole game, and is considered solved. The algorithm then iterates on the remaining arena, which is smaller than the original arena, until no more winning region for Odd is found, and the rest of the arena can be declared winning for Even. The depth of the recursion is bounded by the number of priorities, while

the number of iterations of the while loop is bounded by the size of the arena. This yields an exponential complexity for the algorithm of roughly $O(n^d)$.

To make this algorithm quasipolynomial-time, we weaken and parameterise the inductive guarantees of recursive calls. Instead of expecting a recursive call to solve a subgame, we ask that it provides a partition of the subgame into two regions, one which contains all of Odd's dominions of size up to a given parameter, and one which contains all of Even's dominions of size up to another parameter. These two parameters determine the precision of the recursive call; if they are both the size of the whole subgame, this corresponds to solving it entirely.

Using three recursive calls, our algorithm consecutively computes three regions, $G \setminus G_1$, $G_1 \setminus G_2$, and $G_2 \setminus G_3$, which, together, contain all Odd's dominions of size up to the first parameter and no Even's dominion of size up to the second parameter. The first and the third region are based on calls that use half the precision parameter for Odd's dominions, and only the second region is computed using the full precision. Correctness of the algorithm hinges on proving that $G \setminus G_1$ and $G_1 \setminus G_2$ together cover already over the half of any small Odd's dominion, and hence the last call handles correctly what is left.

We present $\textsc{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$, which, given a subgame G with priorities up to an even number d, returns a set of vertices containing all of Even's small dominions (of size up to p_{E}) and not intersecting any of Odd's small dominions (of size up to p_{O}). The dual, $\textsc{Solve}_{\mathsf{O}}$, is defined for odd d by swapping E with O.

Algorithm 1 $\textsc{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$

1: **if** $G = \emptyset$ or $p_{\mathsf{O}} \leq 1$ **then**
2:     **return** G
3: $G_1 = \textsc{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, \lfloor p_{\mathsf{O}}/2 \rfloor)$
4: $N_d := \{v \in G_1 \mid \pi(v) = d\}$
5: $H := G_1 \setminus \textsc{Attr}_{\mathsf{E}}(N_d, G_1)$
6: $W_{\mathsf{O}} := \textsc{Solve}_{\mathsf{O}}(H, d-1, p_{\mathsf{O}}, p_{\mathsf{E}})$
7: $G_2 := G_1 \setminus \textsc{Attr}_{\mathsf{O}}(W_{\mathsf{O}}, G_1)$
8: $G_3 := \textsc{Solve}_{\mathsf{E}}(G_2, d, p_{\mathsf{E}}, \lfloor p_{\mathsf{O}}/2 \rfloor)$
9: **return** G_3

Remark 3.1. In Lehtinen, Schewe, and Wojtczak [LSW19], Line 3 of the above algorithm was replaced by $G_1 = G \setminus \textsc{Attr}_{\mathsf{O}}(G \setminus \textsc{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, \lfloor p_{\mathsf{O}}/2 \rfloor), G)$. We can see, however, that the call to $\textsc{Attr}_{\mathsf{O}}$ in this expression adds no new vertices, meaning that such a line computes exactly the same set G_1 as our Line 3. Indeed, below, in Lemma 3.2, we show that the set returned by $\textsc{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, \lfloor p_{\mathsf{O}}/2 \rfloor)$ is E-closed. Thus, on the one hand, the presence of $\textsc{Attr}_{\mathsf{O}}$ is redundant here; on the other hand, after adding this $\textsc{Attr}_{\mathsf{O}}$ one can simplify a bit the correctness proof: Item (i) of Lemma 3.2 ceases to be needed.

3.2. **Correctness.** We prove the following lemma, which guarantees that $\textsc{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$ and $\textsc{Solve}_{\mathsf{O}}(G, d, p_{\mathsf{O}}, p_{\mathsf{E}})$ partition a subgame G of maximal priority at most d into a region that contains all Odd's dominions of size up to p_{O} and a region that contains all Even's dominions of size up to p_{E}. Then $\textsc{Solve}_{\mathsf{E}}(G, d, |G|, |G|)$ (if d is even) or $\textsc{Solve}_{\mathsf{O}}(G, d, |G|, |G|)$ (if d is odd) solves G completely.

Lemma 3.2. *The procedure* $\text{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$, *where d is even and not smaller than the maximal priority in G, returns a set that*

(i) *is E-closed in G,*
(ii) *contains all Even's dominions in G of size up to p_{E}, and*
(iii) *does not intersect with any Odd's dominion in G of size up to p_{O}.*

Similarly, $\text{Solve}_{\mathsf{O}}(G, d, p_{\mathsf{O}}, p_{\mathsf{E}})$, *where d is odd and not smaller than the maximal priority in G, returns a set that*

(i) *is O-closed in G,*
(ii) *contains all Odd's dominions in G of size up to p_{O}, and*
(iii) *does not intersect with any Even's dominion in G of size up to p_{E}.*

Proof. The proof is by induction on the sum $d + p_{\mathsf{E}} + p_{\mathsf{O}}$. We present only the part concerning $\text{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$, as the other part is similar.

When $G = \emptyset$ or $p_{\mathsf{O}} \leq 1$, the whole G returned in Line 2 clearly satisfies the thesis: there exist no Odd's dominions of size 1. We proceed with $G \neq \emptyset$ and $p_{\mathsf{O}} > 1$.

We first show Item (i). By the induction hypotheses we know that G_1 is E-closed in G (which, in particular, allows us to use G_1 as a subgame in Line 5) and that G_3 is E-closed in G_2. Moreover, G_2 is E-closed in G_1, because $G_1 \setminus G_2 = \text{Attr}_{\mathsf{O}}(W_{\mathsf{O}}, G_1)$ is invariant under the Attr_{O} operation. In consequence the returned set G_3 is E-closed in G as needed. This is because every Even's vertex in G_3 has at least one successor in G_3 since G_3 is E-closed in G_2; at the same time, every Odd's vertex in $G_3 \subseteq G_2 \subseteq G_1$ has no successor in $G \setminus G_1$ (as G_1 is E-closed in G), nor in $G_1 \setminus G_2$ (as G_2 is E-closed in G_1), nor in $G_2 \setminus G_3$ (as G_3 is E-closed in G_2).

Next, we show Item (ii), saying that the set returned by $\text{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$ contains all Even's dominions of size up to p_{E}. Let D be such a dominion. According to the induction hypothesis, D is contained in G_1, and it is, moreover, an Even's dominion in $G_1 = G \setminus \text{Attr}_{\mathsf{O}}(G \setminus G_1, G)$ (cf. Lemma 2.1). Since H is O-closed in G_1, the intersection D' of D and H is an Even's dominion in H (cf. Lemma 2.2) and therefore, from the induction hypothesis, neither D' nor D (containing additionally only vertices not in H) intersects with W_{O}. In consequence, D is a dominion in G_2 (cf. Lemma 2.1) and, by the induction hypothesis, it is contained in the returned set G_3.

We proceed with showing Item (iii), saying that the set returned by $\text{Solve}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$ does not intersect with Odd's dominions of size up to p_{O}. Let D be such a dominion, let S be the union of Odd's dominions of size up to $\lfloor p_{\mathsf{O}}/2 \rfloor$ in the subgame induced by D, and let A be the O-attractor of S in this subgame.

Because D is O-closed in G (i.e., Even cannot force to leave D), A is contained in the O-attractor of S in the whole G. For the same reason, every Odd's dominion in the subgame D is an Odd's dominion in the whole G. Thus, by the induction hypothesis, S does not intersect with G_1. Since (again by the induction hypothesis) G_1 is E-closed in G, this implies that $\text{Attr}_{\mathsf{O}}(S, G)$ (hence also its subset A) does not intersect with G_1: because of $S \subseteq G \setminus G_1$ we have that $\text{Attr}_{\mathsf{O}}(S, G) \subseteq \text{Attr}_{\mathsf{O}}(G \setminus G_1, G) = G \setminus G_1$. If $A = D$, then D does not intersect with $G_3 \subseteq G_1$, and we are done.

We consider the case of $A \neq D$. Notice that D is a dominion not only in the whole G, but also in the subgame induced by D. In consequence, $D \setminus A$ is a dominion in the subgame induced by $D \setminus A$ (cf. Lemma 2.2; $D \setminus A$ is E-closed in D). It contains a nonempty Odd's dominion C without vertices of priority d (cf. Lemma 2.3). The union $C \cup A$ is an Odd's dominion in D: inside C Odd has a winning strategy, which is valid as long as the play does

not leave $D \setminus A$ entering A; in $A \setminus S$ Odd uses his strategy to reach S, and in S he uses his strategy from one of the small dominions covering S. Moreover, because D is O-closed in G, $C \cup A$ is an Odd's dominion also in the whole G. Since it is not contained in S, it is of size greater than $\lfloor p_{\mathsf{O}}/2 \rfloor$. We now show that $C \cup A$ does not intersect with G_2.

Recall that $A \cap G_1 = \emptyset$. Since G_1 is E-closed in G, the set $(C \cup A) \cap G_1$ is an Odd's dominion in G_1 (cf. Lemma 2.2), and since $(C \cup A) \cap G_1 \subseteq C$ contains no vertices of priority d, also in $H = G_1 \setminus \mathrm{ATTR}_{\mathsf{E}}(N_d, G_1)$ (cf. Lemma 2.1). By the induction hypothesis, it is contained in W_{O}, and therefore $C \cup A$ does not intersect with $G_2 \subseteq G_1 \setminus W_{\mathsf{O}}$.

Note that the proof of Item (i) above also shows that G_2 is E-closed in G. Now, since D is a dominion in G, Lemma 2.2 implies that the set $D \cap G_2$ is a dominion in G_2. Moreover, its size is at most $p_{\mathsf{O}} - (\lfloor p_{\mathsf{O}}/2 \rfloor + 1) \leq \lfloor p_{\mathsf{O}}/2 \rfloor$, because D is of size at most p_{O} and $A \cup C \subseteq D$ (not intersecting with G_2) is of size greater than $\lfloor p_{\mathsf{O}}/2 \rfloor$. So, by the induction hypothesis, $D \cap G_2$ does not intersect with the returned set G_3. The same holds for the whole D, because $G_3 \subseteq G_2$. $\qquad\square$

3.3. Analysis.

Let $R(d, \ell)$ be the maximal number of calls to $\mathrm{SOLVE}_{\mathsf{E}}$ and $\mathrm{SOLVE}_{\mathsf{O}}$ performed during an execution of $\mathrm{SOLVE}_{\mathsf{E}}(G, d, p_{\mathsf{E}}, p_{\mathsf{O}})$ if d is even or of $\mathrm{SOLVE}_{\mathsf{O}}(G, d, p_{\mathsf{O}}, p_{\mathsf{E}})$ if d is odd, where $\ell = \lfloor \log p_{\mathsf{E}} \rfloor + \lfloor \log p_{\mathsf{O}} \rfloor$ (in this paper, log denotes the binary logarithm). We assume that $p_{\mathsf{E}} \geq 1$ and $p_{\mathsf{O}} \geq 1$.

An induction on $\ell + d$ shows that $R(d, \ell) \leq 2^{\ell+1} \cdot \binom{d+\ell}{\ell} - 1$. Indeed, if $d = 0$ (implying $G = \emptyset$) or $\ell = 0$ (implying $p_{\mathsf{E}} = p_{\mathsf{O}} = 1$), then the execution finishes immediately, so $R(d, \ell) = 1 \leq 2^{\ell+1} \cdot \binom{d+\ell}{\ell} - 1$. For positive d and ℓ we have

$$R(d, \ell) \leq 1 + 2 \cdot R(d, \ell - 1) + R(d - 1, \ell),$$

where 1 counts the main call itself, $2 \cdot R(d, \ell - 1)$ counts calls performed while executing Lines 3 and 8, and $R(d - 1, \ell)$ counts calls performed while executing Line 6. Then, applying the induction hypothesis, we obtain that

$$R(d, \ell) \leq 1 + 2 \cdot \left(2^{\ell-1+1} \cdot \binom{d+\ell-1}{\ell-1} - 1 \right) + 2^{\ell+1} \cdot \binom{d-1+\ell}{\ell} - 1$$

$$= 2^{\ell+1} \cdot \binom{d+\ell}{\ell} - 2 \leq 2^{\ell+1} \cdot \binom{d+\ell}{\ell} - 1.$$

We now apply the inequality $\binom{k}{\ell} \leq \left(\frac{ek}{\ell} \right)^{\ell}$ for $k = d + \ell$, obtaining

$$R(d, \ell) \leq 2^{\ell+1} \cdot \left(\frac{e \cdot (d+\ell)}{\ell} \right)^{\ell} = 2 \cdot 2^{\ell \left(1 + \log e + \log \left(1 + \frac{d}{\ell} \right) \right)}.$$

Recall that while solving a parity game with n vertices, we start with $p_{\mathsf{E}} = p_{\mathsf{O}} = n$, that is, with $\ell = 2 \cdot \lfloor \log n \rfloor$. One can check that the above function is increasing, meaning that we can replace ℓ by $2 \cdot \log n$. We obtain that

$$R(d, 2 \cdot \lfloor \log n \rfloor) \leq 2 \cdot 2^{2 \cdot (\log n) \cdot \left(1 + \log e + \log \left(1 + \frac{d}{2 \cdot \log n} \right) \right)} = 2 \cdot n^{2 \cdot \left(1 + \log e + \log \left(1 + \frac{d}{2 \cdot \log n} \right) \right)}.$$

Observe that the cost of each call, excluding the cost of further calls performed recursively, is dominated by the attractor construction in Lines 5 and 7, which is linear in the number of edges in the game. We can also estimate $\log e \leq 1.45$.

Regarding the space complexity, we notice that the recursion has depth $\mathcal{O}(\log n)$, and at each level of recursion we only need to store some sets of vertices of size $\mathcal{O}(n)$.

The complexity is summarised in the following theorem:

Theorem 3.3. *Algorithm 1 computes winning regions in a given parity game. For a parity game with n vertices, m edges, and maximal priority d, its memory usage is in $\mathcal{O}(m+n\cdot\log n)$, and its running time is in*

$$\mathcal{O}\left(m \cdot n^{4.9+2\cdot\log\left(1+\frac{d}{2\cdot\log n}\right)}\right) = n^{\mathcal{O}\left(\log\left(1+\frac{d}{\log n}\right)\right)}.$$

The complexity result can be reformulated as follows:

- When $d = o(\log n)$, the component $\frac{d}{2\cdot\log n}$ converges to 0, meaning that the complexity becomes $\mathcal{O}\left(m \cdot n^{4.9}\right)$ (for n large enough we have that $\log\left(1 + \frac{d}{2\cdot\log n}\right) \leq 1.45 - \log e$).
- This implies that our algorithm is an fpt-algorithm when d is treated as a parameter.
- When $d = \mathcal{O}(\log n)$, the component $\frac{d}{2\cdot\log n}$ is in $\mathcal{O}(1)$, so the complexity of the algorithm is polynomial (with an exponent depending on the precise relation between d and $\log n$).
- When $d = \omega(\log n)$, the component $\frac{d}{2\cdot\log n}$ dominates the component 1, implying that $\log\left(1 + \frac{d}{2\cdot\log n}\right) = \log\frac{d}{\log n} - 1 + o(1)$; in consequence, the complexity can be written as $\mathcal{O}\left(m \cdot n^{2.9+2\cdot\log\frac{d}{\log n}}\right)$ or $n^{\mathcal{O}\left(\log\frac{d}{\log n}\right)}$.

4. A variation

We now present Algorithm 2, the original algorithm of Parys [Par19], as a variation, which we refer to as the Warsaw version.

In Algorithm 1 there is one "central" call to $\textsc{Solve}_\mathsf{O}$ for a subgame with less priorities, with unchanged precision parameters. This call is surrounded by two calls to $\textsc{Solve}_\mathsf{E}$, where we do not decrease the number of priorities, but we divide the precision parameter p_O by 2. While executing each of these calls, we perform one call to $\textsc{Solve}_\mathsf{O}$ for a subgame with less priorities, with p_O divided by 2, which is now surrounded by two calls to $\textsc{Solve}_\mathsf{E}$ for p_O divided by 4. Thus, while looking at calls to $\textsc{Solve}_\mathsf{O}$ for subgames with less priorities, we see that there is one central call with unchanged p_O; it is surrounded by two call with p_O divided by 2; each of them is surrounded by two call with p_O divided by 4, surrounded, in turn, by calls with p_O divided by 8, and so on (this ends when the result of the division becomes smaller than 2).

In the variation, which we now present, this structure of recursive calls to $\textsc{Solve}_\mathsf{O}$ is modified: we again perform one call with unchanged p_O, but this time all the surrouning recursive calls use p_O divided by 2 (not by powers of 2).

4.1. **The algorithm.** As previously, the procedure $\textsc{Solve}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$, given a subgame G with priorities up to an even number d, returns a set of vertices containing all of Even's small dominions (of size up to p_E) and not intersecting any of Odd's small dominions (of size up to p_O). The dual, $\textsc{Solve}_\mathsf{O}$, is defined for odd d by swapping E with O.

Algorithm 2 $\text{SOLVE}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$

1: **if** $G = \emptyset$ or $p_\mathsf{E} \leq 1$ **then**
2:     **return** $\emptyset$
3: **repeat**
4:     $N_d := \{v \in G \mid \pi(v) = d\}$
5:     $H := G \setminus \text{ATTR}_\mathsf{E}(N_d, G)$
6:     $W_\mathsf{O} := \text{SOLVE}_\mathsf{O}(H, d - 1, \lfloor p_\mathsf{O}/2 \rfloor, p_\mathsf{E})$
7:     $G := G \setminus \text{ATTR}_\mathsf{O}(W_\mathsf{O}, G)$
8: **until** $W_\mathsf{O} = \emptyset$
9: $W_\mathsf{O} := \text{SOLVE}_\mathsf{O}(H, d - 1, p_\mathsf{O}, p_\mathsf{E})$
10: $G := G \setminus \text{ATTR}_\mathsf{O}(W_\mathsf{O}, G)$
11: **while** $W_\mathsf{O} \neq \emptyset$ **do**
12:     $N_d := \{v \in G \mid \pi(v) = d\}$
13:     $H := G \setminus \text{ATTR}_\mathsf{E}(N_d, G)$
14:     $W_\mathsf{O} := \text{SOLVE}_\mathsf{O}(H, d - 1, \lfloor p_\mathsf{O}/2 \rfloor, p_\mathsf{E})$
15:     $G := G \setminus \text{ATTR}_\mathsf{O}(W_\mathsf{O}, G)$
16: **end while**
17: **return** G

The variation is closer to the classic recursive algorithms, using only calls to games with less priorities. As a consequence, the number of calls is not capped (beyond the trivial cap of up to n calls for games with n vertices), but all calls refer to sub-games with reduced maximal priority, while all but one asks for lesser guarantees.

4.2. **Correctness.** We prove the following lemma, which guarantees that $\text{SOLVE}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$ and $\text{SOLVE}_\mathsf{O}(G, d, p_\mathsf{O}, p_\mathsf{E})$, for a subgame G of maximal priority at most d, returns two regions that contain all Even's dominions of size up to p_E and all Odd's dominions of size up to p_O, respectively. Then $\text{SOLVE}_\mathsf{E}(G, d, |G|, |G|)$ (if d is even) or $\text{SOLVE}_\mathsf{O}(G, d, |G|, |G|)$ (if d is odd) solves G completely.

We remark that, although the correctness proofs of Algorithms 1 and 2 are quite similar, there is also a lot of differences in details. For readability reasons, we have choosen to leave the two proofs independent.

Lemma 4.1. *The procedure* $\text{SOLVE}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$, *where d is even and not smaller than the maximal priority in G, returns a set that*

(i) *contains all Even's dominions in G of size up to p_E, and*
(ii) *does not intersect with any Odd's dominion in G of size up to p_O.*

Similarly, $\text{SOLVE}_\mathsf{O}(G, d, p_\mathsf{O}, p_\mathsf{E})$, *where d is odd and not smaller than the maximal priority in G, returns a set that*

(i) *contains all Odd's dominions in G of size up to p_O, and*
(ii) *does not intersect with any Even's dominion in G of size up to p_E.*

Proof. The proof is by induction on the sum $d + p_\mathsf{E} + p_\mathsf{O}$. We present only the part concerning $\text{SOLVE}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$, as the other part is similar.

When $G = \emptyset$ or $p_\mathsf{E} \leq 1$, the empty set returned in Line 2 clearly satisfies the thesis: there exist no Even's dominions of size 1. We proceed with $G \neq \emptyset$ and $p_\mathsf{E} > 1$.

Before starting, we remark that the loops terminate, as G shrinks in size in all iterations but the last one of each loop.

We first show Item (i), saying that the set returned by $\textsc{Solve}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$ contains all Even's dominions of size up to p_E. Let D be such a dominion.

For each repetition of the body of the **repeat** (Lines 4–7) and **while** (Lines 12–15) loops, as well as for Lines 9–10, we have the following inductive argument. Initially, D is an Even's dominion in G. Since H is O-closed in G, the intersection D' of D and H is an Even's dominion in H (cf. Lemma 2.2) and therefore, from the induction hypothesis, neither D' nor D (containing additionally only vertices not in H) intersects with the set W_O (computed in Line 6/9/14). In consequence, D is a dominion in the newly computed set G in Line 7/10/15 (cf. Lemma 2.1).

From this argument it follows that D is contained in the set G when it is returned in Line 17.

We proceed with showing Item (ii), saying that the set returned by $\textsc{Solve}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$ does not intersect with Odd's dominions of size up to p_O. Let D be such a dominion.

For each repetition of the body of the **repeat** (Lines 4–7) and **while** (Lines 12–15) loops, as well as for Lines 9–10, we have the following inductive argument. Initially, $D \cap G$ is an Odd's dominion in G. Due to Lemma 2.2, $D \cap G$ is then again an odd dominion in G after execution of Line 7/10/15 (independent of how the respective set W_O is calculated).

We now assume for contradiction that the intersection of D and G is non-empty in Line 17, and therefore that this intersection is non-empty throughout the whole execution of the procedure.

To establish the contradiction, we first observe that, in the last iteration of the **repeat** loop, W_O is empty. By Lemma 2.3, the intersection of D and G, being non-empty, contains a nonempty Odd's dominion C in G without vertices of priority d and, by Lemma 2.1, C is an Odd's dominion in the subgame $H = G \setminus \textsc{Attr}_\mathsf{E}(N_d, G)$ computed in Line 5. As W_O is empty, C (as well as its superset H) must be of size greater than $\lfloor p_\mathsf{O}/2 \rfloor$, as C would otherwise be contained in the empty set W_O by the induction hypothesis.

As the set H from Line 5 is re-used in Line 9, it therefore contains the same Odd's dominion C. By the induction hypothesis, C is also contained in the region W_O computed in Line 9. The intersection of D and the remaining game G produced in Line 10 is of size at most $p_\mathsf{O} - (\lfloor p_\mathsf{O}/2 \rfloor + 1) \leq \lfloor p_\mathsf{O}/2 \rfloor$, because all vertices in C are removed from G at that point.

Then, the same argument as for the **repeat** loop ensures that when the **while** loop finishes, the intersection of D and G contains an Odd's dominion C' of size greater than $\lfloor p_\mathsf{O}/2 \rfloor$. This leads to a contradiction with the upper bound on the size of $D \cap G$ established in the previous paragraph. $\qquad\square$

4.3. **Analysis.** As in the previous section, let $R(d, \ell)$ be the maximal number of calls to $\textsc{Solve}_\mathsf{E}$ and $\textsc{Solve}_\mathsf{O}$ performed during an execution of $\textsc{Solve}_\mathsf{E}(G, d, p_\mathsf{E}, p_\mathsf{O})$ if d is even or of $\textsc{Solve}_\mathsf{O}(G, d, p_\mathsf{O}, p_\mathsf{E})$ if d is odd, where $\ell = \lfloor \log p_\mathsf{E} \rfloor + \lfloor \log p_\mathsf{O} \rfloor$. We assume that $p_\mathsf{E} \geq 1$ and $p_\mathsf{O} \geq 1$.

An induction on $\ell + d$ shows that $R(d, \ell) \leq 2 \cdot n^\ell \cdot \binom{d+\ell}{\ell} - 1$, where $n \geq 1$ is the number of vertices in the considered game $\mathcal{G}$. Indeed, if $d = 0$ (implying $G = \emptyset$) or $\ell = 0$ (implying $p_\mathsf{E} = p_\mathsf{O} = 1$), then the execution finishes immediately, so $R(d, \ell) = 1 \leq 2 \cdot n^\ell \cdot \binom{d+\ell}{\ell} - 1$.

For positive d and ℓ we have

$$R(d,\ell) \leq 1 + n \cdot R(d-1,\ell-1) + R(d-1,\ell),$$

where 1 counts the main call itself, $n \cdot R(d-1,\ell-1)$ counts calls performed while executing Lines 6 and 14, and $R(d-1,\ell)$ counts calls performed while executing Line 9. The calls with reduced parameters (Lines 6 and 14) are always done with games of decreasing size, and there is no game of size 1; thus, there is at most n of these calls (for games of size $n, n-1, \ldots, 2$ and then 0). Then, applying the induction hypothesis to the above formula, we obtain that

$$R(d,\ell) \leq 1 + n \cdot \left(2 \cdot n^{\ell-1} \cdot \left(\binom{d-1+\ell-1}{\ell-1} - 1 \right) \right) + 2 \cdot n^\ell \cdot \binom{d-1+\ell}{\ell} - 1$$

$$\leq 2 \cdot n^\ell \cdot \left(\binom{d-1+\ell}{\ell-1} + \binom{d-1+\ell}{\ell} \right) - 1 = 2 \cdot n^\ell \cdot \binom{d+\ell}{\ell} - 1.$$

We now apply the inequality $\binom{k}{\ell} \leq \left(\frac{ek}{\ell} \right)^\ell$ for $k = d + \ell$, obtaining

$$R(d,\ell) \leq 2 \cdot n^\ell \cdot \left(\frac{e \cdot (d+\ell)}{\ell} \right)^\ell = 2 \cdot 2^{\ell \left(\log n + \log e + \log \left(1 + \frac{d}{\ell} \right) \right)}.$$

Recall that while solving a parity game with n vertices, we start with $p_{\mathsf{E}} = p_{\mathsf{O}} = n$, that is, with $\ell = 2 \cdot \lfloor \log n \rfloor$. One can check that the above function is increasing, meaning that we can replace ℓ by $2 \cdot \log n$. We obtain that

$$R(d, 2 \cdot \lfloor \log n \rfloor) \leq 2 \cdot 2^{2 \cdot (\log n) \cdot \left(\log n + \log e + \log \left(1 + \frac{d}{2 \cdot \log n} \right) \right)} = 2 \cdot n^{2 \cdot \left(\log n + \log e + \log \left(1 + \frac{d}{2 \cdot \log n} \right) \right)}.$$

This time, to each call to the procedure we associate the cost of the two attractor constructions, preceding and following this call. In consequence, the cost allocated to each call is again linear in the number of edges. Estimating $\log e \leq 1.45$, we obtain the following theorem:

Theorem 4.2. *Algorithm 2 computes winning regions in a given parity game. For a parity game with n vertices, m edges, and maximal priority d, its memory usage is in $\mathcal{O}(m+n\cdot\log n)$, and its running time is in*

$$\mathcal{O}\left(m \cdot n^{2.9+2\cdot\log n+2\cdot\log\left(1+\frac{d}{2\cdot\log n}\right)} \right) = n^{\mathcal{O}\left(\log n+\log\left(1+\frac{d}{\log n}\right)\right)} = n^{\mathcal{O}(\log n)}.$$

The last equality above holds under the assumption $d \leq n$. This is a reasonable assumption: clearly there are at most d different priorities, and they can be renumbered so that only consecutive numbers, from 1 to some upper bound, are used.

The main difference between the complexity here and in Section 3 is that here we have $\log n$ in the exponent. This means that the algorithm does not speed up in the common case where the number of priorities is significantly smaller than the number of vertices.

5. Optimisations

In this section we present some optimisations that can be applied to Algorithms 1 and 2. They do not improve the theoretical complexity, but they may (and actually they do) speed up the algorithm in practice, on some inputs.

(1) After the **repeat** loop in Algorithm 2 we can check whether $|H| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$, and if this is the case, we can return G immediately, without executing Lines 9-16. Indeed, the induction hypothesis guarantees that W_{O}, calculated as $\textsc{Solve}_{\mathsf{O}}(H, d-1, \lfloor p_{\mathsf{O}}/2 \rfloor, p_{\mathsf{E}})$, contains all Odd's dominions in H of size up to $\lfloor p_{\mathsf{O}}/2 \rfloor$. When $|H| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$, then these are actually all Odd's dominions in H, and it makes no sense to call $\textsc{Solve}_{\mathsf{O}}$ again with precision p_{O}.

(2) Likewise, in Algorithm 1, if $|G| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$, then the first call to $\textsc{Solve}_{\mathsf{E}}$ handles already all Odd's dominions in G; we can return G_1 immediately after Line 3.

(3) To strengthen the effect of testing whether $|G| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$ or $|H| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$ in Algorithms 1 and 2, respectively, the initial call can be made with $p_{\mathsf{E}} = p_{\mathsf{O}} = 2^{\lfloor \log |G| \rfloor + 1} - 1$ instead of $p_{\mathsf{E}} = p_{\mathsf{O}} = |G|$. The former number is slightly greater, but the depth of the recursion remains unchanged.

(4) In Algorithm 2, if during the recursive evaluation of the last call to $\textsc{Solve}_{\mathsf{O}}$ in the **repeat** loop, the condition "$p_{\mathsf{O}} \leq 1$" (Line 1 of $\textsc{Solve}_{\mathsf{E}}$) was never true, then, again, we can return G immediately after this loop, without executing Lines 9-16. Indeed, in such a case, the call $\textsc{Solve}_{\mathsf{O}}(H, d-1, \lfloor p_{\mathsf{O}}/2 \rfloor, p_{\mathsf{E}})$ is equivalent to a call with any greater value (e.g., $|H|$) in place of $\lfloor p_{\mathsf{O}}/2 \rfloor$, so the the returned set contains all Odd's dominions in H, and there is no need to call $\textsc{Solve}_{\mathsf{O}}$ with precision p_{O}.

   The check can be implemented by returning and using a "dirty" flag that marks whether the "$p_{\mathsf{O}} \leq 1$" condition has been true in any of the sub-routines.

(5) As in implementations of standard Zielonka's algorithm, the conditions in the **repeat** and **while** loops in Algorithm 2 can be changed from "$W_{\mathsf{O}} = \emptyset$" to W_{O} being invariant under O-attractor (i.e., to "$\textsc{Attr}_{\mathsf{O}}(W_{\mathsf{O}}, G) = W_{\mathsf{O}}$" for the value of G before it is updated in Lines 7, 10, or 15). This requires to re-calculate H after the **repeat** loop, before Line 9. While this is clearly compatible with the optimisations above, it complicates the correctness argument, namely the part where we follow what the algorithm does to a small (size up to p_{O}) Odd's dominion D.

   For the basic version of the algorithm, at every step we were considering a nonempty Odd's dominion C in the subgame $D \cap G$, not containing vertices of priority d. Now, instead of choosing such a dominion C arbitrarily, we should take the greatest one (it exists, because the union of two dominions is a dominion). This maximality implies that if C is strictly smaller than $D \cap G$, then $\textsc{Attr}_{\mathsf{O}}(C, D \cap G)$ contains a vertex of the maximal priority d. Indeed, note that $\textsc{Attr}_{\mathsf{O}}(C, D \cap G)$ is itself an Odd's dominion in $D \cap G$; thus it either equals C or contains a vertex of priority d. In the latter case we are done. In the former case, there is a nonempty dominion C' in $(D \cap G) \setminus C$, not containing vertices of priority d; the union $C' \cup C$ is an Odd's dominion in $D \cap G$, contradicting the maximality of C.

   Having this, we observe as previously that if $|C| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$, then C is included in W_{O} and thus removed from G. Moreover, if C does not cover the whole $D \cap G$, then $\textsc{Attr}_{\mathsf{O}}(C, D \cap G) \subseteq \textsc{Attr}_{\mathsf{O}}(W_{\mathsf{O}}, G)$ contains a vertex of priority d; this vertex is not included in $G \setminus N_d \supseteq H \supseteq W_{\mathsf{O}}$, so the current loop continues. If $|C| > \lfloor p_{\mathsf{O}}/2 \rfloor$, then the **repeat** loop may finish without removing the whole C from G, and the remaining part of C is removed from G by the full precision call in Lines 9–10 (since clearly $|C| \leq p_{\mathsf{O}}$). After that, in the **while** loop, we always have $|C| \leq |D \cap G| \leq \lfloor p_{\mathsf{O}}/2 \rfloor$, which, by the above, guarantees removal of all remaining vertices of D.

(6) With a similar argument, the third call of Algorithm 1 (Line 8) can be skipped if the O-attractor of W_{O} in G_1 equals W_{O}.

6. Evaluation

Our evaluation focuses on the performance of the presented algorithms on random parity games on the one hand and on so-called worst-case families on the other hand. In a nutshell, the results are without great surprises. On random games, which usually don't present much of a challenge for Zielonka's algorithm, its quasipolynomial-time variations are in general orders of magnitude slower. The exception is games that can be solved in very few iterations by Zielonka, such as random high-degree games, which seem to be solvable by the Warsaw quasipolynomial-time algorithm in just as few iteration. Furthermore, the Liverpool quasipolynomial-time algorithm, which has the best worst-case complexity, reliably requires more iterations than the Warsaw version, which has a higher worst-case complexity. On the families designed to trigger worst-case complexity in Zielonka's algorithm [Fri11b, BDM20, Gaz16], the story is different. By game-size 54, both quasipolynomial-time algorithms require fewer iterations than the standard recursive algorithm, and the difference in performance from then on grows at an exponential rate. On these families of worst-case games, as the theory predicts, the algorithm with better worst-case behaviour indeed requires fewer iterations.

6.1. **Implementation.** We have implemented the two algorithms presented in the paper, using all the optimisations from Section 5. For comparison, we have also implemented standard Zielonka's algorithm, where we apply Optimisation 5 from Section 5 (i.e., we use "$\text{ATTR}_{\mathsf{O}}(W_{\mathsf{O}}, G) = W_{\mathsf{O}}$" instead of "$W_{\mathsf{O}} = \emptyset$" for the condition of a loop). Our implementation involves the Oink framework [vD18] to read files with games and to confirm correctness of our solutions, but apart from that the implementation is independent.

To compare performance of the three algorithms, we use the number of iterations. Because the most costly operation in the procedure is the computation of an attractor, as a single iteration we count the fragment of code where an attractor for one player is computed, a subprocedure is called, and an attractor for the other player is computed (the first computation of an attractor is not present in Lines 9–10 of Algorithm 2, but anyway we count these two lines as an iteration). Note, however, that iterations are exactly the places where a subprocedure is called, so the number of iterations is actually equal to the number of recursive calls, minus one.

With the number of iterations as a convenient running time measure, our evaluation becomes independent from low-level optimisations or from a choice of hardware.

The implementations, together with detailed results of the evaluation, are available at https://github.com/pparys/qpt-parity.

6.2. **Benchmarks.**

Constructed parity games. Friedmann [Fri11a] was the first to show an exponential lower bound to the running time of Zielonka's recursive algorithm. Since then, several families designed to trigger worst-case behaviour in recursive algorithms have been proposed: some are more robust [BDM20], in the sense that they also provide lower bounds for many variations and optimisations of the standard algorithm; others provide stronger lower bounds [Gaz16]. Since here we do not study optimisation strategies, we choose to present the performance of our algorithm on Gazda's family of games, which provide the strongest lower bound we are aware of to date, namely $\Omega(2^{\frac{n}{3}})$ [Gaz16]. From our observations, this is representative

Figure 1: Number of iterations needed by each algorithm on Gazda's family of games

of the algorithms' performance on any of the mentioned families. We have run the three algorithms on games of size up to 231, above which point all the three algorithms time out at 15 minutes.

Random parity games. We use the PG-solver tools [FL14] to generate random parity games. We study the algorithms both on games with high vertex degree and with low vertex degree. Overall, we have run our algorithms on 320 random games of sizes ranging from 100 to 2000; the number of priorities always corresponds to the number of vertices. For random games, the game size is a poor predictor of how many iterations are needed—for instance, in our sample, Zielonka's algorithm solves more random games with high degree of size 2000 than of size 200 in two iterations—we therefore don't report the sizes of the sample games in our graphs.

6.3. **Results.** We compare the implementations of the three algorithms, with all the optimisations from Section 5 included. We use the number of iterations required by each algorithm as a proxy for their performance.

Constructed parity games. As shown in Figure 1, on parity games of Gazda's family, the performance of both quasipolynomial-time versions overtake the standard version of Zielonka's algorithm around size 50. From there on, the performance gap grows exponentially. The Liverpool version of the algorithm, which enjoys better worst-case complexity, outperforms the Warsaw version consistently. Note that the logarithmic scale hides the growth of this performance gap. The number of iterations required by both quasipolynomial-time algorithms seems to jump abruptly at parity games of size 30, 63 and 126 (these sizes are
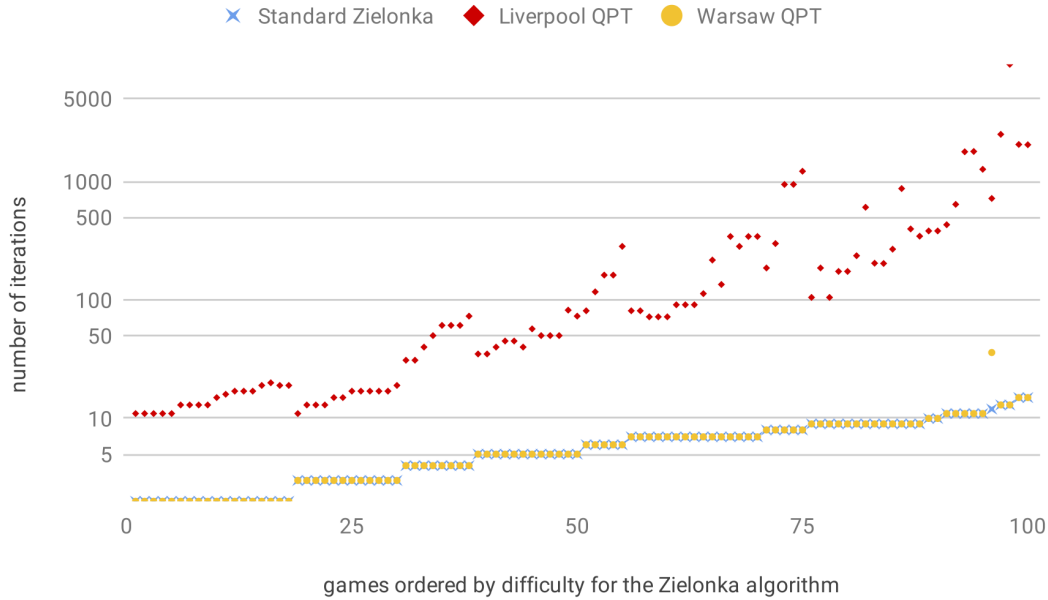
Figure 2: Number of iterations needed by each algorithm on randomly generated games of high degree with between 100 and 2000 vertices

around the powers of 2, where the search depth of the algorithms change); this behaviour is persistent throughout the different worst-case families.

Random parity games. The size of randomly generated games does not generally predict the performance of the algorithms. On games of high degree, the algorithms agree on the relative difficulty of games; see Figure 2. Both Zielonka's algorithm and the Warsaw quasipolynomial-time version consistently solve these games in under 15 iterations, while the Liverpool version needs one to two orders of magnitude more iterations. The reason why the Warsaw version behaves like Zielonka's algorithm in this case is that differences only start if the recursion depth of the algorithms exceeds the logarithm of the game size; here the Warsaw version never runs out of bound, its call structure is the same as Zielonka's. On random games of low degree, see Figure 3, the two quasipolynomial-time versions only broadly tend to agree with Zielonka's algorithm on the relative difficulty of games. Again, the Warsaw quasipolynomial-time version seems to match Zielonka's algorithm on the easiest games, namely those that Zielonka's algorithm solves in under 15 iterations. However, in general both the Warsaw and Liverpool version require several orders of magnitude more iterations and the Warsaw version consistently outperforms the Liverpool version throughout.

7. Conclusion

We have presented two quasipolynomial-time algorithms solving parity games, working recursively. The main advantage of our recursive approach over previous quasipolynomial-time algorithms solving parity games lies in its simplicity: we perform a small modification of the straightforward Zielonka's algorithm.
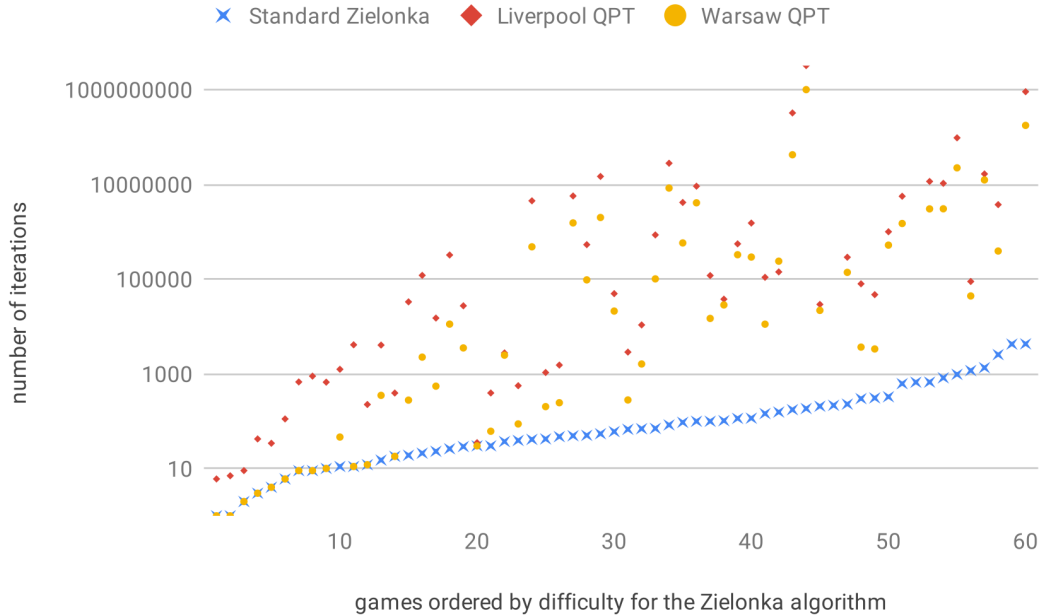
Figure 3: Number of iterations needed by each algorithm on randomly generated games of low degree with between 100 and 500 vertices

Our original hope was that, in practice, the quasipolynomial-time algorithms can be as fast as standard Zielonka's algorithm on typical inputs, while being faster (quasipolynomial instead of exponential) on worst-case inputs. This turned out to be only partially true: although our algorithms are not dramatically slower, a significant slowdown is visible (except for the Warsaw variant, in case of a really small number of iterations).

Czerwiński et al. [CDF+19] argue that previous quasipolynomial-time algorithms solving parity games [CJK+17, JL17, Leh18, FJdK+19] are instances of a so-called separator approach. Then, they prove that every algorithm accomplishing this approach has to follow a structure of a universal tree (a tree into which every tree of a given size can be embedded), and they show a quasipolynomial lower bound for the size of such a tree—hence also for the running time of the algorithm. Our algorithms, on the one hand, are of a different style; they cannot be seen as instances of the separator approach, hence the lower bound does not apply to them. On the other hand, however, one can see that the trees of recursive calls in our two algorithms follow two particular constructions of universal trees. Furthermore, Jurdziński and Morvan [JM20] generalise our algorithms to a generic algorithm parameterised by a universal tree (any universal tree gives a correct algorithm, while our two variants are obtained by using particular universal trees). Arnold, Niwiński, and Parys [ANP21] show a similar generalisation, from a slightly different perspective of fixed-point evaluation; they also prove that universal trees are persistently present in their approach, resulting in a quasipolynomial lower bound for the number of recursive calls.

Finally, let us mention a recent result of Jurdziński, Morvan, Ohlmann, and Thejaswini [JMOT20]: they design an algorithm solving parity games that is symmetric (like our recursive algorithms), but simultaneously works in time proportional to the size of one

universal tree, not to the size of a product of two such trees (thus the logarithm in the exponent is not multiplicated by 2, unlike for our Liverpool variant).

REFERENCES

[ANP21]   André Arnold, Damian Niwiński, and Paweł Parys. A quasi-polynomial black-box algorithm for fixed point evaluation. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CSL.2021.9`.

[BDM20]   Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Robust worst cases for parity games algorithms. *Inf. Comput.*, 272:104501, 2020. `doi:10.1016/j.ic.2019.104501`.

[BL18]   Udi Boker and Karoliina Lehtinen. On the way to alternating weak automata. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 21:1–21:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.FSTTCS.2018.21`.

[CDF+19]   Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Paweł Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2333–2349. SIAM, 2019. `doi:10.1137/1.9781611975482.142`.

[CJK+17]   Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. `doi:10.1145/3055399.3055409`.

[DJL19]   Laure Daviaud, Marcin Jurdziński, and Karoliina Lehtinen. Alternating weak automata from universal trees. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.18`.

[DJT20]   Laure Daviaud, Marcin Jurdziński, and K. Thejaswini. The Strahler number of a parity game. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 123:1–123:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.123`.

[EJS01]   E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001. `doi:10.1016/S0304-3975(00)00034-7`.

[Fea10]   John Fearnley. Exponential lower bounds for policy iteration. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2010. `doi:10.1007/978-3-642-14162-1_46`.

[FHZ11]   Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 283–292. ACM, 2011. `doi:10.1145/1993636.1993675`.

[FJdK+19]   John Fearnley, Sanjay Jain, Bart de Keijzer, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space. *Int. J. Softw. Tools Technol. Transf.*, 21(3):325–349, 2019. `doi:10.1007/s10009-019-00509-3`.

[FL14]   Oliver Friedmann and Martin Lange. The PGSolver collection of parity game solvers. `https://www.win.tue.nl/~timw/downloads/amc2014/pgsolver.pdf`, 2014. Version 3.

[Fri11a]   Oliver Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO Theor. Informatics Appl.*, 45(4):449–457, 2011. `doi:10.1051/ita/2011124`.

[Fri11b]   Oliver Friedmann. A subexponential lower bound for Zadeh's pivoting rule for solving linear programs and games. In Oktay Günlük and Gerhard J. Woeginger, editors, *Integer Programming and Combinatoral Optimization - 15th International Conference, IPCO 2011, New York, NY, USA, June 15-17, 2011. Proceedings*, volume 6655 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2011. `doi:10.1007/978-3-642-20807-2_16`.

[Gaz16]   Maciej Gazda. *Fixpoint logic, games, and relations of consequence*. PhD thesis, Eindhoven University of Technology, March 2016.

[JL17]   Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9. IEEE Computer Society, 2017. `doi:10.1109/LICS.2017.8005092`.

[JM20]   Marcin Jurdziński and Rémi Morvan. A universal attractor decomposition algorithm for parity games. *CoRR*, abs/2001.04333, 2020. `arXiv:2001.04333`.

[JMOT20]   Marcin Jurdziński, Rémi Morvan, Pierre Ohlmann, and K. S. Thejaswini. A symmetric attractor-decomposition lifting algorithm for parity games. *CoRR*, abs/2010.08288, 2020. `arXiv:2010.08288`.

[Jur98]   Marcin Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998. `doi:10.1016/S0020-0190(98)00150-1`.

[Leh18]   Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 639–648. ACM, 2018. `doi:10.1145/3209108.3209115`.

[LSW19]   Karoliina Lehtinen, Sven Schewe, and Dominik Wojtczak. Improving the complexity of Parys' recursive algorithm. *CoRR*, abs/1904.11810, 2019. `arXiv:1904.11810`.

[Mar75]   Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. URL: `http://www.jstor.org/stable/1971035`.

[McN93]   Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993. `doi:10.1016/0168-0072(93)90036-D`.

[Par19]   Paweł Parys. Parity games: Zielonka's algorithm in quasi-polynomial time. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 10:1–10:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.MFCS.2019.10`.

[Par20]   Paweł Parys. Parity games: Another view on Lehtinen's algorithm. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 32:1–32:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CSL.2020.32`.

[PR89]   Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. `doi:10.1145/75277.75293`.

[vD18]   Tom van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, volume 10805 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2018. `doi:10.1007/978-3-319-89960-2_16`.

[Zie98]   Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. `doi:10.1016/S0304-3975(98)00009-7`.