Parameterized Algorithms for Parity Games

Jakub Gajarský[1*], Michael Lampis[2], Kazuhisa Makino[3], Valia Mitsou[4†], and Sebastian Ordyniak[5‡]

[1]Faculty of Informatics, Masaryk University, Brno, Czech Republic

[2]LAMSADE, Université Paris Dauphine, France

[3]RIMS, Kyoto University, Japan

[4]SZTAKI, Hungarian Academy of Sciences, Budapest, Hungary

[5]Institute for Computergraphics and Algorithms, TU Wien, Vienna, Austria

Abstract

Determining the winner of a Parity Game is a major problem in computational complexity with a number of applications in verification. In a parameterized complexity setting, the problem has often been considered with parameters such as (directed versions of) treewidth or clique-width, by applying dynamic programming techniques.

In this paper we adopt a parameterized approach which is more inspired by well-known (non-parameterized) algorithms for this problem. We consider a number of natural parameterizations, such as by Directed Feedback Vertex Set, Distance to Tournament, and Modular Width. We show that, for these parameters, it is possible to obtain recursive parameterized algorithms which are simpler, faster and only require polynomial space. We complement these results with some algorithmic lower bounds which, among others, rule out a possible avenue for improving the best-known sub-exponential time algorithm for parity games.

1 Introduction

A *Parity Game* is an infinite two-player game played on a parity game arena by the two players Even and Odd. A *parity game arena* is a sinkless directed graph, where every vertex is controlled by exactly one of the two players and is addionally assigned a priority (an integer value). Initially, a token is placed on some vertex of the graph (the starting vertex) and at each step of the game the player that controls the vertex that currently contains the token has to move

[*]Supported by the research centre Institute for Theoretical Computer Science (CE-ITI), project P202/12/G061

[†]Supported by ERC Starting Grant PARAMTIGHT (No. 280152)

[‡]Supported by Employment of Newly Graduated Doctors of Science for Scientific Excellence (CZ.1.07/2.3.00/30.0009).

the token along a directed edge to any outneighbor of that vertex. This leads to an infinite path, which is won by the player that corresponds to the parity of the highest priority occuring infinitely often on that path. Given such a parity game arena and a starting vertex v, the problem is to decide which of the two players has a winning strategy on the given arena if the token is initially placed on v.

Although a parity game may, at first glance, seem like an odd kind of game to play, the problem of deciding the winner of a parity game is extremely well-studied. One of the reasons is that solving parity games captures the complexity of model-checking for the modal μ-calculus, which has a large number of applications in software verification (see e.g. [17, 14, 8]). Despite extensive efforts by the community, it is still a major open question whether there exists a polynomial-time algorithm for this problem [21, 5, 22, 13].

In addition to their wealth of practical applications, parity games are also especially interesting from a complexity-theoretic point of view because of their intriguing complexity status. Despite not being known to be in P, deciding parity games in known to be in NP$\cap$coNP [14], and in fact even in UP$\cap$coUP [20]. The former of these two inclusions follows directly from the (non-trivial) fact that optimal strategies are known to be positional (or memoryless) [23] and the fact that single-player parity games are in P. Parity games are also known to be reducible to (and therefore "easier" than) a number of other natural classes of games, including mean payoff games (which admit a pseudo-polynomial time algorithm) [29] and simple stochastic games [11]. Thus, in a sense, deciding the winner of a parity game is a problem that seems to lie only slightly out of reach of current algorithmic techniques, and could perhaps be solvable in polynomial time. However, the best currently known time upper bound is (roughly) $n^{\sqrt{n}}$ [22].

Since, despite all this effort, it is still not clear if a polynomial-time algorithm for parity games is possible, several more recent works have attempted to tackle this problem from a parameterized complexity perspective. Perhaps the most natural parameter for this problem is the maximum priority p. Although this problem is known to be in XP, i.e., solvable in polynomial-time for fixed p, (by a classical $O(n^p)$algorithm due to McNaughton [24], later improved to $O(n^{p/3})$ [7, 21, 27]), it is unclear if this parameterization could be in FPT, i.e., solvable in time $O^*(f(p))$[1] for some function f of p, though this question has been explicitly considered [3].

A direction which has been much more fruitful is to add a second parameter to the problem, usually some structural graph parameter of the input graph. Here, several non-trivial algorithmic results are known. For digraphs whose underlying graph has treewidth k the problem is solvable in time (roughly) $p^{O(k)}$ [25], while a similar running time is achievable if k is the (directed) clique-width of the input digraph [26]. For several directed variations of the notion of treewidth, such as entanglement, DAG-width and kelly-width, algorithms

[1]As usual in parameterized complexity the $O^*(f(k))$-notation, for some function f of the parameter k, means that there is an algorithm running in time $O(f(k)n^{O(1)})$, where n is the input size of the problem.

running in $n^{O(k)}$ time are known [2, 1, 19]. More recently, an algorithm running in $f(p, k)n^{O(1)}$ time for these measures was given in [6], for some computable function f.

Thanks to the above results we now know that parity games are FPT for the most standard structural graph parameters, if parameterized by *both* k and the maximum priority p. It is however, worthy of note that, to the best of our knowledge, this problem is not known to be FPT when the number of priorities is not part of the parameter for any non-trivial graph width. Furthermore, all the above algorithms, which use somewhat complicated dynamic programming techniques, also require space exponential in k. Can this be improved?

Our contribution: This paper provides a number of algorithms for various natural structural parameterizations of parity games. More specifically, we show the following results:

- There exists an algorithm running in time $O^*\big(4^k p^{\log k}\big)$ when k is any of the following parameters: the size of the graph's directed feedback vertex set, the number of vertices controlled by one of the two players, or the number of vertices whose deletion makes the graph a tournament.
- There exists an algorithm running in time $O^*\Big(k^{O(\sqrt{k})}\Big)$ when k is the modular width of the input graph.

Conceptually, the main contribution of this paper is in repurposing the ideas of the classical algorithm of McNaughton [24] (and its sub-exponential time variation by Jurdzinski, Paterson and Zwick [22]) in the parameterized complexity setting. By avoiding the dynamic programming paradigm, we are able to give a number of very simple to implement, recursive algorithms. Notably, all the algorithms of this paper run in polynomial space.

Our first set of algorithmic results introduces the notion of a *dominion hitter*. Informally, a dominion is a confined area of the graph where one player has a winning strategy. The algorithmic importance of dominions was recognized in [22], where the algorithm begins by exhaustively looking for a small dominion in the graph, before running the simple recursive algorithm of [24]. We give a parameterized counterpart of this idea which can be applied whenever we can find a small set of vertices that intersects all dominions. We then also provide three natural example parameterizations where this is the case, showing that this may be an idea of further interest.

We then consider graphs of modular width k. Modular width is a graph parameter that has recently attracted attention in the parameterized complexity community as a more restricted alternative to clique-width [15, 16]. We show that, in this more restricted case, the $O^*\big(p^k\big)$ complexity of the algorithm for clique-width can be improved to a running time that is FPT parameterized only by k, with a *sub-exponential* parameter dependence. The core algorithmic idea again combines the recursive algorithm of McNaughton, with a judicious search for dominions.

We complement these algorithms with a couple of hardness results. First, as mentioned, one of the key steps in the sub-exponential algorithm of [22] (and in most of our algorithms) consists of exhaustively searching the graph for a dominion. In [22] searching for a dominion of size k is done by checking all sets of k vertices. We give a reduction from Multi-Colored Clique showing that this is likely optimal, thus ruling out one possible avenue for improving the algorithm of [22]. Furthermore, in order to demonstrate that the parameterizations we consider are non-trivial, we give a reduction showing that Rabin games (a more general class of infinite games) remain hard for many of the cases of this paper.

2 Definitions and Preliminaries

2.1 Parity Games, Dominions and Attractors

In a parity game the input is a sinkless directed graph $G(V, E)$, where V is partitioned into two sets V_0, V_1, a priority function $\mathrm{Pr} : V \to \mathbb{N}$, and a starting vertex $v \in V$. Throughout the paper we use n to denote $|V|$ and p to denote the maximum priority given to any vertex of G $\max\{\mathrm{Pr}(u) : u \in V\}$. We also use $N^+(u)$ (respectively $N^-(u)$) to denote the set of out-neighbors (in-neighbors) of the vertex u.

In this game there are two players, player 0 (the even player) and player 1 (the odd player), controlling the vertices of V_0 and V_1 respectively. A token is initially placed on the starting vertex v and, in each turn, the player who controls the vertex that currently contains the token selects one of its out-neighbors and moves the token there (an out-neighbor always exists, since the graph has no sinks). The play then goes on to the next round and continues infinitely. Player 0 is the winner of a play if and only if the vertex that has maximum priority out of all the vertices that appear infinitely often in the play has even priority; otherwise, player 1 is the winner. A player has a winning strategy from a starting vertex v, if she has a way to construct a winning play when the game starts from v no matter how the opponent plays. For more information on parity games see for example [17].

We use $W_i(G)$, for $i \in \{0, 1\}$ to denote the winning region of player i in the game graph G, that is, the set of vertices from which player i has a winning strategy (and we will simply write W_i if G is clear from the context). It is a well-known (but non-trivial) fact that each player has a memory-less *positional* strategy, i.e a strategy where decisions depend on the current position of the token and not on the history, that allows her to win all the vertices of her winning region [4, 14, 23]. We will make use of the following basic fact:

Fact 1. *For all $i \in \{0, 1\}$ and all vertices $v \in V_i$ we have $v \in W_i$ if and only if $N^+(v) \cap W_i \neq \emptyset$.*

Let us now formally define two notions that will be crucial throughout this paper: dominions and attractors.

Definition 1. A set of vertices $D \subseteq V$ is an i-dominion, for some $i \in \{0, 1\}$ if the following hold:

1. There are no arcs from $D \cap V_{1-i}$ to $V \setminus D$ and every vertex of $D \cap V_i$ has an out-neighbor in D.

2. Player i has a winning strategy for all starting vertices in the parity game on the induced subgraph $G[D]$.

Informally, an i-dominion is a region of the graph where player i can force the token to remain, and by doing so she manages to win the game. It is not hard to see that $W_i(G)$ is always an i-dominion, but smaller i-dominions could also exist.

Definition 2. An i-attractor of a set of vertices S, for some $i \in \{0, 1\}$, denoted $\mathrm{attr}_i(S)$, is the smallest superset of S that satisfies the following:

1. There are no arcs from $V_i \setminus \mathrm{attr}_i(S)$ to $\mathrm{attr}_i(S)$.

2. Every vertex of $V_{1-i} \setminus \mathrm{attr}_i(S)$ has an out-neighbor outside of $\mathrm{attr}_i(S)$.

Intuitively, an i-attractor of a set S is a region inside which player i can force the token to eventually enter S (this should obviously include S itself).

It is not hard to see that an i-attractor can be calculated in polynomial time by iteratively adding vertices to S, as dictated by the above specifications. One of the reasons that we are interested in attractors is that they allow us to simplify the game once we identify some part of one player's winning region, thanks to the following fact, which is e.g. shown in [22, Lemma 4.5].

Fact 2. *If, for some $i \in \{0, 1\}$ and $S \subseteq V$, we have $S \subseteq W_i$, then $W_i(G) = \mathrm{attr}_i(S) \cup W_i(G \setminus \mathrm{attr}_i(S))$.*

We will often make use of the fact that an attractor for one player cannot expand into the region of a dominion controlled by the opponent.

Lemma 1. *For $i \in \{0, 1\}$ let D be an i-dominion and $S \subseteq V$ be a set of vertices such that $S \cap D = \emptyset$. Then $\mathrm{attr}_{1-i}(S) \cap D = \emptyset$.*

Proof. Consider the iterative process that adds vertices to S to build $\mathrm{attr}_{1-i}(S)$. No vertex of D can be the first to be added to the attractor in this process: vertices of $D \cap V_i$ have an out-neighbor in D and vertices of $D \cap V_{1-i}$ have all their out-neighbors in D. $\square$

Finally, let us point out that a dominion that is completely controlled by a single player can be found (and by Fact 2 removed from the graph) in polynomial time. We will therefore always assume that in all the graphs we consider, no player can win without at some point passing the token to her opponent.

Fact 3. *If there exists a dominion D such that $D \subseteq V_i$ for some $i \in \{0, 1\}$, then such a dominion can be found in polynomial time.*

2.2 Attractor-based Algorithms

We give here a short descriptive sketch of the attractor-based algorithms of [24, 22]. One of the main approaches for solving parity games is the attractor-based approach often referred to as McNaughton's algorithm [24, 28]. We sketch here the algorithm, which has served as the starting point of many subsequent improvements, as well as most of the algorithms of this paper. The first step in McNaughton's algorithm is to locate the set of vertices S that have maximum priority p in the input graph. Let $i := p \bmod 2$. One then calculates the set $A := \mathrm{attr}_i(S)$. We now recursively call the algorithm on the graph $G \setminus A$, that is, the graph obtained by deleting the vertices of A. This allows us to calculate the winning regions of $G \setminus A$.

There are now two cases. If player i wins from every vertex of this new graph, then she has a strategy to win from every vertex of G. To see this, observe that when the token enters A player i can follow a strategy that leads the token to S, therefore, if the token enters A infinitely often in a play, the maximum priority will be p, which is winning for i. If the token does not enter A infinitely often then play is eventually restricted to $G \setminus A$ where player i wins everywhere. So, in this case we are done. If, on the other hand, player $1-i$ wins some vertices of $G \setminus A$ then $W_{1-i}(G \setminus A)$, which is a $(1-i)$-dominion in $G \setminus A$, is also a $(1-i)$-dominion in G. Therefore, we can use Fact 2 to simplify the graph. We then recursively solve the remaining game.

Theorem 1. *[24] McNaughton's algorithm solves parity games in time $O^*(n^p)$.*

Proof. Let $T(n,p)$ denote the worst-case running time of the algorithm. Then $T(n,p) \leq T(n-1,p-1) + T(n-1,p) + n^{O(1)}$, since in the first recursive call the maximum priority has been decreased and in the second call (if it is made) we have deleted at least one vertex. Solving this recurrence gives the claimed bound. □

Perhaps one of the most notable improvements building upon McNaughton's algorithm is the deterministic sub-exponential algorithm of Jurdzinski, Paterson and Zwick [22] (before their work the only sub-exponential algorithms known for this problem were randomized LP-based algorithms [5]). Let us also sketch this approach, which we extend in this paper. The idea is to precede the running of McNaughton's algorithm by a pre-processing step, which looks for a "small" dominion in the input graph. The pre-processing is done by brute force, trying out all sets of size at most c, where c is a parameter to be optimized later (in section 5 we argue that this brute force step cannot be improved). If such a dominion is found, it can be removed from the graph using Fact 2. Otherwise, we know that if McNaughton's algorithm makes a second recursive call, a dominion of size at least c will have been removed from the graph. Judiciously selecting c to balance the two steps gives a sub-exponential running time.

Theorem 2. *[22] The algorithm of Jurdzinski, Paterson and Zwick solves parity games in time $n^{O(\sqrt{n})}$.*

Proof. We select c to be $\sqrt{n}$. Then, if $T(n)$ is the worst-case running time on n vertices, we have $T(n) \leq \binom{n}{c}c^c + T(n-1) + T(n-c)$. $\square$

3 Strong Dominion Hitters

In this section we introduce the notion of a *dominion hitter* and show several (parameterized) algorithmic applications. The ideas presented here are heavily inspired by the sub-exponential algorithm of [22], which looks for small dominions in the input graph. Here we will be interested in the intersection of the located dominion with a special set that intersects all dominions. We give two straight-forward applications of this idea (for parameters directed feedback vertex set and number of vertices of one player) and a slightly more involved one (for parameter distance from tournament). Before moving forward, let us give some useful definitions.

Definition 3. A set $S \subseteq V$ is a *dominion hitter* if for every (non self-controlled) dominion D of G we have $S \cap D \neq \emptyset$. A set $S \subseteq V$ is a *strong dominion hitter* if, for every $V' \subseteq V$ for which $G[V']$ is a game, $S \cap V'$ is a dominion hitter in $G[V']$.

Let us explain the intuition behind the definition of strong dominion hitters. First, by Fact 3 we can remove self-controlled dominions in polynomial time from G and all of its subgames. Now, if G contains a strong dominion hitter S with $|S| = k$ then McNaughton's algorithm will run in time $O^*\big(n^k\big)$, since the second recursive call will always delete at least one vertex of S (in fact the running time can also be upper-bounded by $O^*\big(p^k\big)$, since the first recursive call decreases p).

Our strategy in this section will be to improve upon McNaughton's algorithm significantly by following a strategy similar to that of [22]. Suppose we are given a strong dominion hitter S. First, we look for a dominion that has a small intersection with S. As we will argue, this can be done by solving parity games (recursively) on a much simpler graph where a large part of S has been deleted. If such a dominion is found, it can be removed from the graph. If that fails, then every dominion must have a large intersection with S, therefore we know that McNaughton's algorithm's second recursive call will be on a graph where much of S has been deleted. Proper balancing allows us to obtain a running time of $O^*\big(p^{\log k}4^k\big)$ (Lemma 2). Let us now describe exactly how dominion hitters can be exploited.

Lemma 2. *There is an algorithm which, given a parity game instance $G(V,E)$ and a strong dominion hitter S with $|S| = k$ decides the problem in $O^*\big(p^{\log k}4^k\big)$ time.*

Proof. The algorithm proceeds as follows: first, for every $S' \subset S$ with $|S'| = s$, where s is a parameter to be fixed later, we will try to find a dominion D such that $D \cap S \subseteq S'$. For $i \in \{0,1\}$ we will describe how to find such an i-dominion, if it exists.

Let $A := \mathrm{attr}_{1-i}(S \setminus S')$. We solve parity games on the graph $G \setminus A$ by recursively calling this algorithm. Now, we claim that $W_i(G \setminus A)$ (if non-empty) is an i-dominion in G. To see this, observe that, by definition, the game restricted to $W_i(G \setminus A)$ is a win for i, and vertices of $V_{1-i} \cap W_i(G \setminus A)$ have no outgoing edges to A, by the definition of attractors, or $W_{1-i}(G \setminus A)$, by Fact 1. On the other hand, we claim that if an i-dominion D exists in G such that $D \cap S \subseteq S'$ then $D \subseteq W_i(G \setminus A)$. This follows from Lemma 1. Thus, we can conclude whether an i-dominion with the requested intersection with S exists (and find it if it does) by solving parity games on a graph where at least $|S \setminus S'| \geq k - s$ vertices of S have been deleted. If a dominion is found, we can remove it from the graph using Fact 2.

If no dominion (of either player) is found in the first phase, since S is a dominion hitter we know that any dominion includes at least s vertices of S. We now essentially follow McNaughton's approach: remove an attractor of the maximum priority vertices, solve the remaining graph recursively and, if necessary, remove the discovered winning region and solve the remainder recursively.

Let us analyze the running time of the above procedure. Suppose that $T(n,p,k)$ is the worst-case running time for a graph with n vertices, maximum priority p and a strong dominion hitter of size k. The first phase requires time at most $2\binom{k}{s}[T(n,p,s) + O(n^a)]$ (removing a dominion takes polynomial time $O(n^a)$). If a dominion D is found, we solve recursively an instance where we have removed an attractor of D in time $T(n,p,k-1) + O(n^a)$. If no dominion is found, we first make a recursive call that takes time $T(n,p-1,k) + O(n^a)$ since the maximum priority is decreased and then a recursive call that takes time at most $T(n,p,k-s) + O(n^a)$. Removing self-controlled dominions takes polynomial time $O(n^b)$. So, overall we have

$$\begin{aligned} T(n,p,k) \leq 2\binom{k}{s}[T(n,p,s) + O(n^a)] + \max\{&T(n,p,k-1), \\ &T(n,p-1,k) + T(n,p,k-s)\} + O(n^a + n^b) \end{aligned}$$

Observe that the algorithm we have described generalizes the algorithm of [22]: in the worst case $k = n$ (the dominion hitter is all the vertices) and we can select $s = \sqrt{n}$ to obtain essentially the same running time. However, here we want to leverage the fact that we may have a small dominion hitter. We will set $s = k/2$ and prove inductively that $T(n,p,k) \leq 4^k p^{\log k} O(n^c)$, where $c > \max\{a,b\}$ is a constant.

For $p, k > 4$, and sufficiently large n:

(a)

$$\begin{aligned}
2\binom{k}{\frac{k}{2}}[T(n,p,k/2)+O(n^a)]+T(n,p,k-1)+O(n^a+n^b) &\leq \\
2^k 4^{\frac{k}{2}} p^{\log\frac{k}{2}} O(n^c) + 4^{k-1} p^{\log k} O(n^c) + 2^k O(n^a+n^b) &\leq \\
\left(\frac{1}{p}+\frac{1}{4}+\frac{2^k O(n^a+n^b)}{4^k p^{\log k} O(n^c)}\right) 4^k p^{\log k} O(n^c) &\leq \\
4^k p^{\log k} O(n^c).
\end{aligned}$$

(b)

$$\begin{aligned}
2\binom{k}{\frac{k}{2}}[T(n,p,k/2)+O(n^a)]+T(n,p-1,k)+T(n,p,k/2)+O(n^a+n^b) &\leq \\
2^k T(n,p,k/2)+T(n,p-1,k)+2^k O(n^a+n^b) &\leq \\
2^k 4^{\frac{k}{2}} p^{\log\frac{k}{2}} O(n^c) + 4^k (p-1)^{\log k} O(n^c) + 2^k O(n^a+n^b) &\leq \\
\left(\frac{1}{p}+\left(\frac{p-1}{p}\right)^{\log k}+\frac{2^k O(n^a+n^b)}{4^k p^{\log k} O(n^c)}\right) 4^k p^{\log k} O(n^c) &\leq \\
4^k p^{\log k} O(n^c).
\end{aligned}$$

□

3.1 Direct applications

Let us first consider two direct applications of the idea of strong dominion hitters. One is the parameterization of the problem where the parameter is the directed feedback vertex set S of the graph, i.e. a set of vertices whose removal makes the graph a DAG. Since every dominion should contain a directed cycle, S is clearly a strong dominion hitter. Application of Lemma 2 is straightforward.

Theorem 3. *Given an instance of parity games $G(V,E)$ and a directed feedback vertex set $S, |S| \leq k$ of G, there exists an algorithm which decides the problem in time $O^*(p^{\log k} 4^k)$.*

Another example is the parameterization of the problem where the parameter k is equal to $|V_1|$, the number of vertices controlled by one of the players. Because of Fact 3, V_1 is a strong dominion hitter. Once again Lemma 2 can be applied directly.

Theorem 4. *There exists an algorithm which given an instance of parity games $G(V,E)$ such that $|V_1| = k$ decides the problem in time $O^*(4^k p^{\log k})$.*

3.2 More involved case

Now we will see how we can apply the idea of strong dominion hitters to a less straightforward case, where the graph is almost a tournament (a tournament is a directed graph having at least one arc between every pair of vertices).

Once again, due to Fact 3, G is free of i-controlled i-dominions (we call these dominions *happy*). In fact, this implies something even stronger: that no happy dominion exists in any subgame of G. Then all i-dominions in G shall have an *unhappy* vertex, i.e. a vertex controlled by player $(1-i)$. The next fact shall prove useful.

Fact 4. *Let $v_0 \in V_0 \cap W_1$ and $v_1 \in V_1 \cap W_0$ be unhappy vertices. Then v_0 and v_1 are not neighbors.*

Proof. $(v_0, v_1) \notin E$, otherwise the even player would have had a way to escape from v_0 to W_0. With similar reasoning $(v_1, v_0) \notin E$. □

We first need to establish that parity games on tournaments can be solved in polynomial time, using the above observation. In fact, we can show that in tournaments, after we remove happy dominions, one player wins all the vertices. The fact that parity games are polynomially decidable on tournaments was already shown in [12], but a proof is included here for the sake of completeness.

We then study the parameterization where the parameter is the vertex-deletion distance of the graph from being a tournament. In this case, the graph is basically a tournament plus a set S of at most k additional vertices which might be missing arcs (this case is more general than the case of a tournament missing a set of k edges).

The algorithm is similar to Lemma 2. If S happens to be a dominion hitter the procedure reduces S by at least $\frac{k}{2}$ either during the preprocessing or during the second recursive call of McNaughton's algorithm. If S is not a dominion hitter, then the dominion found after removing $\mathrm{attr}_i(p)$ might not intersect S. In this case however, we argue that the $(1-i)$-attractor of the winning region $W_{1-i}(G \setminus \mathrm{attr}_i(p))$ of the smaller graph should absorb all vertices of $V_{1-i} \setminus S$, leaving an instance where player $(1-i)$ has vertices only in S (up to k vertices). We can then use Theorem 4. The algorithmic results of this section are thus summarized in the following two theorems.

Theorem 5. *Parity games on tournaments can be solved in polynomial time.*

Proof. Given a tournament T, we use Fact 3 to obtain a tournament $G(V, E)$ with no happy dominions.

We will first prove that in G all vertices are won by one player, that is either W_0 or W_1 of G is empty. Suppose that W_0, W_1 are both non-empty. Then, by Fact 4 there would be two unhappy vertices which would not be connected. This is a contradiction because G is a tournament.

What we need to figure out is which of the two players wins. This can be done once again by running McNaughton's algorithm, though we show that in the case of tournaments we don't need the second recursive call. Indeed, suppose we have removed $\mathrm{attr}_i(p)$ of the maximum priority p and solved the remaining game $G'(V', E')$. This too is a tournament, so one player should win the whole graph. If player i wins everything in G', then this is clearly the case for G. If player $1-i$ wins everything in G', then V' is an $(1-i)$-dominion of G and by the observation that all vertices are won by one player this makes player $(1-i)$ the winner in the rest of G.

All that remains is to find a strategy for player $(1-i)$ in $\mathrm{attr}_i(p) = V \setminus V'$. We know that player $(1-i)$ should be winning the whole graph, so any vertex belonging in $V_i \cap V'$ should not be able to escape to $\mathrm{attr}_i(p)$ (we know that there exists at least one such vertex, call it u). That means that all vertices in $\mathrm{attr}_i(p) \cap V_{1-i}$ should have at least one outgoing edge towards V' (say to u). So the strategy for player $1-i$ is to send the token to u. $\square$

Theorem 6. *Given a graph $G(V, E)$ on n vertices and a set $S \subseteq V$ with $|S| \leq k$ such that $G \setminus S$ is a tournament, parity games can be solved in time $O^*\big(p^{\log k} 4^k\big)$.*

Proof. Again, we first pre-process the graph to obtain graph $G(V, E)$ with no happy dominions. G should also have a set S of at most k vertices the removal of which leaves a tournament.

We proceed in a similar way as in Lemma 2. First, for $i \in \{0, 1\}$, for all $S' \subset S$ of size $|S'| \leq \frac{k}{2}$ we try to find a dominion $D \cap S' \neq \emptyset$. If such a dominion exists we remove it from the graph using Fact 2 and solve a smaller instance. Otherwise we know that every dominion either uses at least $\frac{k}{2}$ vertices from S or it has an empty intersection with S.

In the latter case, we run McNaughton's algorithm: remove $A = \mathrm{attr}_i(p)$ and potentially find a winning region $W_{1-i}(G \setminus A)$ with at least one unhappy vertex $v_i \in W_{1-i}(G \setminus A) \cap V_i$.

- If $W_{1-i}(G \setminus A) \cap S \neq \emptyset$, this case should be similar to Lemma 2.
- If $W_{1-i}(G \setminus A) \cap S = \emptyset$, observe that all vertices in $A \setminus S$ should have v_i as an out-neighbor, otherwise v_i would belong in A. Furthermore, by Fact 4, all unhappy vertices of $W_i(G \setminus A)$ should belong to S. Thus, when we compute $A' = \mathrm{attr}_{1-i}(W_{1-i}(G \setminus A))$, all vertices of player $(1-i)$ but those in S will be absorbed, so $G \setminus A'$ should be a graph where player $(1-i)$ controls possibly only vertices from S. We shall then proceed similarly to Theorem 4.

The complexity of this algorithm is similar to that of Lemma 2. If we call $A(n, p, k)$ the worst-case running time for a graph with n vertices, maximum priority p and a set of k vertices missing arcs and $O(n^a)$ the complexity of removing a dominion, the first phase again requires time at most $2\binom{k}{\frac{k}{2}}[A(n, p, k/2) + O(n^a)]$. If we find a dominion D intersecting S, we solve recursively in time $A(n, p, k-1) + O(n^a)$ an instance where at least one element of S is removed. If no dominion is found, then the first recursive call takes time $A(n, p-1, k) + O(n^a)$. After that, either S is reduced by $\frac{k}{2}$ and the second recursive call takes time $A(n, p, k/2) + O(n^a)$, or one of the players is left with no more than k vertices, which by Theorem 4 shall take time $4^k p^{\log k} O(n^c)$ to solve. So, overall we have:

$$\begin{aligned}A(n,p,k) \leq 2\binom{k}{\frac{k}{2}}[A(n,p,k/2)+O(n^a)] + \max\{A(n,p,k-1),\\ A(n,p-1,k) + \max\{A(n,p,k/2), 4^k p^{\log k} O(n^c)\}\} + O(n^a)\end{aligned}$$

This recursive function can be proven inductively to be $\leq 4^k p^{\log k+1} O(n^d)$ for sufficiently large values of n, p, k, d. Most computations are similar to Lemma 2 and perhaps what remains is to prove the following part of the inductive step:

For $p, k > 4$, $d \geq c$ and sufficiently large n:

$$\begin{aligned}2\binom{k}{\frac{k}{2}}[A(n,p,k/2)+O(n^a)] + A(n,p-1,k) + 4^k p^{\log k} O(n^c) + O(n^a) &\leq\\ 2^k 4^{\frac{k}{2}} p^{\log \frac{k}{2}+1} O(n^d) + 4^k (p-1)^{\log k+1} O(n^d) + 4^k p^{\log k} O(n^c) + 2^k O(n^a) &\leq\\ 4^k p^{\log k} O(n^d) + \left(\frac{p-1}{p}\right)^{\log k+1} 4^k (p-1)^{\log k+1} O(n^d) + 2^k O(n^a) &\leq\\ \left(\frac{1}{p} + \left(\frac{p-1}{p}\right)^{\log k+1} + \frac{2^k O(n^a)}{4^k p^{\log k+1} O(n^c)}\right) 4^k p^{\log k+1} O(n^d) &\leq\\ 4^k p^{\log k+1} O(n^d).&\end{aligned}$$

This concludes the proof. $\square$

4 Modular Width and Graphs with k Modules

The main result of this section is an FPT algorithm which solves parity games on graphs with small modular width. In fact, the algorithm we present is able to handle the more general case of strongly connected graphs whose vertex set can be partitioned into $k > 1$ modules. For such graphs, we are able to obtain a *sub-exponential FPT* algorithm, even for unbounded p (that is, parameterized just by k), and from this we can then obtain the same result for modular width.

Recall that a set of vertices M of a directed graph $G(V, E)$ is a *module* if for any two vertices $u, v \in M$ we have $N^+(u) \setminus M = N^+(v) \setminus M$ and $N^-(u) \setminus M = N^-(v) \setminus M$, that is, all vertices of M have the same in- and out-neighbors outside of M. Let us begin this section by presenting an algorithm that decides parity games on G in time $O^*(4^k)$. We then give an improved version with sub-exponential running time.

4.1 Exponential FPT algorithm

Suppose that we are given a strongly connected graph $G(V, E)$, along with a partition of V into $k > 1$ non-empty modules $M_1, \ldots M_k$. We will call these the graph's *basic* modules.

On a high level, the strategy we will follow is again a variation of McNaughton's algorithm. This algorithm makes two recursive calls, each on a graph obtained from G by removing an attractor of some set. If the removed set contains all of $M_j \cap V_i$ for some $j \in \{1, \ldots, k\}$, $i \in \{0, 1\}$ then we can intuitively think that we are making a lot of progress: such recursive calls cannot have a depth of more than $2k$ before we are able to solve the problem (this is where the $O^*(4^k)$ running time comes from). Thus, our high-level strategy is to avoid branching in cases where the removed set does not simplify the graph in this way.

Let us begin by arguing that the algorithm's second recursive call always makes significant progress in the sense described above. In the remainder we assume, as in the previous section, that the graph has been simplified using Fact 3, so each player must at some point pass the token to her opponent to avoid losing. We now need a helpful lemma.

Lemma 3. *Let M be a non-sink module of G and suppose that $W_i \cap M \neq \emptyset$. Then, $V_i \cap M \subseteq W_i$. Furthermore, there exists a vertex $v \in W_i \setminus M$ such that there is an edge from M to v.*

Proof. Let $M' \subseteq V \setminus M$ be the set of vertices that have an incoming edge from M (and, therefore, have incoming edges from all vertices of M, since M is a module). By assumption, $M' \neq \emptyset$.

If $M' \cap W_i \neq \emptyset$ then we are done, because all vertices of $V_i \cap M$ have an out-neighbor in W_i, so they must also belong in W_i by Fact 1.

Let us then show that it cannot be the case that $M' \cap W_i = \emptyset$. In that case we have $M' \subseteq W_{1-i}$. Observe now that $M \cap V_{1-i} \subseteq W_{1-i}$ by Fact 1 and the fact that M' is non-empty. Any winning play for player i starting from a vertex $u \in M$ must eventually visit either $M \cap V_{1-i}$ (because player i does not control a winning cycle) or M'. However, both of these sets are subsets of W_{1-i}, which is a contradiction. □

An informal way to interpret Lemma 3 is that, if a player wins some vertex of a module, then she must be winning all the vertices she controls in that module. We now have the following:

Corollary 1. *Let D be an i-dominion, for some $i \in \{0, 1\}$ and M a non-sink module such that $D \cap M \neq \emptyset$. Then $\mathrm{attr}_i(D) \supseteq M \cap V_i$.*

Proof. Similarly to the proof of Lemma 3, if the graph contains no trivial cycles (which it does not, thanks to Fact 3) then D cannot be contained in M. □

We now know that the second recursive call of McNaughton's algorithm will always remove all the vertices owned by one of the two players in one of the basic modules. What remains is to deal with the first recursive call, where we remove an attractor of the maximum priority vertices. In this case we cannot guarantee that the removal of the attractor will necessarily produce a much simpler graph. However, we will argue that, if the graph is not simplified, the removed vertices were “irrelevant”: their presence does not change the winning status of any other vertex. We will then be able to calculate their winning status *without* making the second recursive call using Lemma 3. The key idea is contained in the following lemma.

Lemma 4. *Let p be the maximum priority in G, v be a vertex with priority p and $i := p \bmod 2$. Let $A := \mathrm{attr}_i(\{v\})$. Suppose that $A \subseteq M_j$ for some basic non-sink module M_j and that if A contains a vertex controlled by some player, then $M_j \setminus A$ also contains a vertex controlled by that player. Then we have $W_i(G) \setminus A = W_i(G \setminus A)$.*

Proof. Observe that for any $u \in W_{1-i}(G \setminus A)$ we also have $u \in W_{1-i}(G)$, by standard properties of attractors. Contrapositively, $u \in W_i(G) \setminus A$ implies that $u \in W_i(G \setminus A)$. Therefore, it suffices to show that for all $u \in W_i(G \setminus A)$ we have $u \in W_i(G)$. In other words, we need to describe a strategy for player i which wins in G at least the same vertices i can win in $G \setminus A$.

We will say that a vertex $u \in G \setminus A$ is *problematic* if $u \in W_i(G \setminus A) \cap W_{1-i}(G)$, that is, u is a counterexample to the lemma. If a problematic vertex u exists, then there must exist a problematic vertex that has an edge to A in G. To see this, consider a play starting from u where player i uses her optimal strategy for $G \setminus A$. If player $1-i$ is using her optimal strategy in G, then the token must eventually leave $W_i(G \setminus A)$ (otherwise player i would win). This can only be done through a vertex of V_{1-i}, since player i is following a strategy that keeps the token in $W_i(G \setminus A)$. Such vertices have no edges to $W_{1-i}(G \setminus A)$, therefore the only way out is through A.

Suppose then that u is a problematic vertex that has an edge to A. Therefore, $u \in V_{1-i}$, otherwise u would be in the attractor. Suppose that $u \notin M_j$. It must then be the case that $M_j \setminus A \subseteq W_i(G \setminus A)$, by Fact 1, since u has edges to all of M_j. If, on the other hand, $u \in M_j$ then for any $u' \notin M_j$ that has an incoming edge from M_j we have $u' \in W_i(G \setminus A)$. This again implies that $M_j \setminus A \subseteq W_i(G \setminus A)$, using Lemma 3.

So, if there exists a problematic vertex u then player i is winning all of $M_j \setminus A$ in $G \setminus A$. If $v \in V_i$ then player i can follow the following strategy in G: for vertices of $G \setminus A$ follow an optimal strategy for $G \setminus A$, for vertices of $A \setminus \{v\}$ follow a strategy that leads the token to v, and from v push the token to a vertex of $V \setminus M_j$ that belongs in $W_i(G \setminus A)$ (such a vertex exists by Lemma 3). If the token enters A infinitely often player i wins, otherwise the last time the token leaves A it is trapped in $W_i(G \setminus A)$, where player i wins. Thus, no problematic vertex can exist in this case.

Finally, suppose that $v \in V_{1-i}$. By the conditions of the lemma there exists a vertex $w \in V_{1-i} \cap (M_j \setminus A)$. As before, if there exists a problematic vertex, player i wins all of $M_j \setminus A$, including w, in $G \setminus A$. Therefore, player i wins (in $G \setminus A$) every vertex $w' \notin M_j$ that has an incoming edge from M_j. Player i now follows the same strategy as in the previous paragraph. Observe that once again, if the token visits A infinitely often the maximum priority is p, otherwise, when the token leaves v it will go to a vertex that player i wins in $G \setminus A$. $\square$

We are now ready to put everything together to obtain the promised algorithm.

Theorem 7. *Consider a parity game on a strongly connected graph $G(V, E)$ where V is partitioned into $k > 1$ non-empty modules. There exists an algorithm that decides the winning regions of the two players and their winning strategies from these regions in time $O^*(4^k)$.*

Proof. We will run a version of McNaughton's algorithm, so at each step we will be looking at a subgraph of G. The measure of progress for our algorithm will be the number of pairs i, j for $i \in \{0, 1\}$ and $j \in \{1, \dots, k\}$ such that $V_i \cap M_j \neq \emptyset$ in the current graph. Initially the measure has value at most $2k$ and we want to prove that any non-trivial recursive step decreases its value.

We want to maintain the following invariant: for each basic module M_j of the original graph, in any considered subgraph the vertices of M_j which remain form a module which is either controlled by a single player or has some outgoing edge. The invariant is initially satisfied, since the graph is strongly connected, therefore all modules have outgoing edges.

Let us argue that, as long as we produce the considered subgraphs by removing attractors, the invariant will always hold. Indeed, suppose that at some point in the algorithm's execution one of the basic modules M_j has lost all its outgoing arcs. That means that we removed an attractor that contained a vertex v such that all vertices of M_j were pointing to v. Thus, for some $i \in \{0, 1\}$ all vertices of $V_i \cap M_j$ must have also been placed in the same attractor as v.

The algorithm is now the following: first, suppose that there exists a module M_j without outgoing edges. We solve the problem on $G[M_j]$; this can be done in polynomial time since all its vertices belong to one player. We can now simplify G using Fact 2. We then continue with the remainder of the graph.

So the interesting case is when all basic modules have outgoing edges. Now we do the following: first locate a vertex v with maximum priority p, and let $i := p \bmod 2$. Let $A := \mathrm{attr}_i(\{v\})$. If A fulfills the requirements of Lemma 4 then we solve the problem on $G \setminus A$ recursively. We then know the winning status of all vertices of $G \setminus A$ in G, since by Lemma 4 this does not change when we put the attractor back. We can also infer the winning status of vertices of A: if $u \in A \cap V_l$ for $l \in \{0, 1\}$ then there exists a vertex $u' \in (M_j \setminus A) \cap V_l$ and we know the winning status of u'. By Lemma 3 u has the same winner. Thus, in this case we are done.

Finally, if A does not fulfill the requirements of Lemma 4 then we solve (recursively) the problem on $G \setminus A$ and then proceed as in McNaughton's algorithm. That is, we know that $W_{1-i}(G \setminus A)$ is a $(1 - i)$-dominion in G, so we use Fact 2 to remove it and solve recursively in the rest of the graph. We will argue that in both of these cases the graph we recurse on is simpler.

If A does not fulfill the requirements of Lemma 4, then we have two cases: In the first case, A contains some vertex u of another module $M_{j'}$. If $u \in V_i$ then A contains all vertices of $M_{j'} \cap V_i$ (which was non-empty) so the complexity measure is reduced. If $u \in V_{1-i}$ then A must contain all of M_j, so again the complexity measure is reduced. In the second case, A contains all of the vertices of M_j controlled by some player, so deleting A eliminates $V_l \cap M_j$ for some $l \in \{0, 1\}$. So, the graph obtained after removing A is simpler.

For the second recursive call, observe that if we know the winning status of a vertex u, we know the winning status of all vertices which are in the same module and controlled by the same player as u (by Lemma 3). So we can assume that for the second recursive call we remove a region that contains all of $V_i \cap M_j$ for some i, j.

We are now ready to conclude that the running time of the recursive algorithm we have described is $O^*(4^k)$. To see this, note that if the algorithm makes two recursive calls, each is on a graph where we have just removed all of $V_i \cap M_j$ for some i, j. Suppose that we started with a graph where all modules have outgoing edges. Then, after a depth of at most $2k-1$ operations where $V_i \cap M_j$ is removed for some i, j, we have a graph that contains a module M_j without outgoing edges. Furthermore, by the stated invariant, this module is controlled by a single player and can be solved in polynomial time. □

It's now easy to apply the above algorithm to the case of modular width.

Corollary 2. *There exists a $O^*(4^k)$ algorithm that decides parity games on graphs with modular width k.*

Proof. Recall that a graph has modular width k if it is K_1 or it can be partitioned into at most k modules such that each module induces a graph with modular width k. This recursive structure is usually described by a tree, called a modular decomposition, which can be calculated in linear time [18].

If the input graph is strongly connected then we can invoke Theorem 7. Otherwise, we find a strongly connected component without outgoing edges. This part of the graph also has modular width k, so we can apply to it Theorem 7 and, after calculating its winning regions, remove them from the original graph and repeat. □

4.2 Sub-exponential FPT algorithm

Let us now present an improved version of the algorithm of Theorem 7. The idea is inspired by the improved version of McNaughton's algorithm from [22]: we will first look for a dominion that is confined inside at most $\sqrt{k}$ of the k basic modules. If such a dominion is found we can simplify the graph. Otherwise, we know that the second recursive call (when made) will touch at least $\sqrt{k}$ dominions, thus making significant progress.

Theorem 8. *Consider a parity game on a strongly connected graph $G(V, E)$ where V is partitioned into $k > 1$ non-empty modules. There exists an algorithm that decides the winning regions of the two players and their winning strategies from these regions in time $k^{O(\sqrt{k})}n^{O(1)}$.*

Proof. We use the same invariant and the same measure of progress as in the proof of Theorem 7. The difference now is the following: suppose that each of the (remaining) basic modules has outgoing edges. Also, suppose that we know that removing that attractor of the maximum priority does not make progress (otherwise we handle this case as in Theorem 7 with a single recursive call).

Now, before doing anything else we attempt to find a dominion that touches a small number of these modules. Specifically, for each set $S \subseteq \{1, \dots, k\}$ such that $|S| \leq \sqrt{k}$ we want to see if the current graph contains a dominion confined in $\cup_{j\in S} M_j$. To see if such an i-dominion exists, for some $i \in \{0, 1\}$, we remove a $(1 - i)$-attractor of the set $\cup_{j\notin S} M_j$ and recursively solve the remaining graph using Theorem 7. The complexity of this step is dominated by trying all sets S, and is therefore $k^{O(\sqrt{k})}$.

If the above step finds a dominion, we remove it from the graph using Fact 2 and repeat the process. Otherwise, we run an iteration of McNaughton's algorithm, knowing that the second recursive call (if made) will touch at least $\sqrt{k}$ modules. The complexity is therefore given by the recurrence $T(n, k) \leq k^{O(\sqrt{k})} + T(n, k-1) + T(n, k-\sqrt{k})$. This is the same recurrence as in Theorem 2. □

Corollary 3. *There exists a $k^{O(\sqrt{k})}n^{O(1)}$ algorithm that decides parity games on graphs with modular width k.*

5 Hardness Results

5.1 Finding Dominions

In this section we consider the parameterized complexity of the following problem: given an instance of parity games, does there exist a dominion consisting of at most k vertices? This problem arises very naturally in the course of examining the sub-exponential time algorithm given in [22]. The first step of this algorithm is to look for a "small" dominion, that is, a dominion that has size k, where k is a parameter to be optimized. The approach proposed in [22] is simply to try out all $\binom{n}{k}$ sets of vertices of size k and solve parity games on the resulting subgraph. Since solving parity games in a graph with k vertices can be done in time sub-exponential in k, the $\binom{n}{k}$ factor dominates the running time of this process. It is therefore natural to ask whether this can be sped up, which in turn would imply that a different value should be chosen for k to get the best worst-case bound on the algorithm. One could plausibly hope to try improving the running time to something like $2^{\sqrt{n}}$ (from $n^{\sqrt{n}}$) using such an approach.

Unfortunately, Theorem 9 establishes that improving significantly upon this brute force method is likely to be very hard. In particular, we give a parameterized reduction from MULTI-COLORED CLIQUE, showing that finding a dominion of size at most k cannot be done in time $n^{o(\sqrt{k})}$ (under standard assumptions).

Theorem 9. *There is no algorithm which, given an instance of parity games decides if there exists a dominion on at most k vertices in time $n^{o(\sqrt{k})}$, unless the ETH fails.*

Proof. We will use the results given in [9, 10], which show that MULTI-COLORED CLIQUE cannot be solved in time $O(n^{o(k)})$ under ETH, by giving a quadratic fpt-reduction, i.e., the parameter of the constructed instance is quatratic in the parameter of the originial instance, from MULTI-COLORED CLIQUE.

The reduction is from MULTI-COLORED CLIQUE: we are given a graph $G(V, E)$ with V partitioned into k independent sets and are asked if there exists a clique of size k. We construct a directed graph $G'(V', E')$. We set $V' = V \cup E \cup \{1, \dots, k\} \cup \{(i, j) \mid 1 \le i < j \le k\}$. In other words, we create a vertex for each vertex and for each edge of the original graph, as well as a vertex for each color and each pair of colors.

We then add a directed edge from each vertex i, for $i \in \{1, \dots, k\}$ to each vertex u that has color i in G. Similarly, for each (i, j) with $1 \le i < j \le k$ we add edges from (i, j) to the vertex representing $(u, v) \in E$ where u, v have colors i, j in G respectively. For each vertex representing $(u, v) \in E$ we add edges to the vertices representing $u, v \in V$. Finally, for each $u \in V$ and for each $(u, v) \in E$ we add edges from the vertices representing u and (u, v) to all $i \in \{1, \dots, k\}$ and all (i, j), $1 \le i < j \le k$. All vertices are given priority 1. Player 1 controls vertices i and (i, j) and player 0 the vertices corresponding to $V \cup E$.

Of course, it is clear that player 1 wins everywhere in this game, since all priorities are odd. The question is if she can trap player 0 in a small dominion. We argue that there exists a dominion of size at most $2(k + \binom{k}{2})$ if and only if G has a k-clique. The theorem then follows.

Suppose that G has a k-clique. Then the dominion for player 1 contains all her vertices, as well as the vertices corresponding to vertices and edges of the clique.

For the converse direction, suppose that player 1 has some dominion D in G'. It must contain some vertices of player 0, therefore it must contain all vertices of player 1. For player 1 to construct a dominion she must select an out-neighbor of each vertex she controls, and since these vertices have disjoint out-neighborhoods a dominion of the presribed size can only exist if she selects exactly one out-neighbor for each of her vertices. This selection corresponds to a selection of k vertices and $\binom{k}{2}$ edges in the original graph. If one of the selected edges is incident on an unselected vertex in G, then D is not a dominion because player 0 could escape through the vertex representing that edge in G'. $\square$

5.2 Rabin Games

A Rabin game is another type of infinite game played on a graph, which in some sense generalizes parity games. In this section we present a simple reduction which shows that the algorithmic results we have obtained for parity games are unlikely to be extendible to Rabin games. Viewed another way, this reduction shows that the restricted graph classes we have considered are still quite non-trivial.

First, let us define a Rabin game. We are again given a directed graph $G(V, E)$, where V is partitioned into V_0 and V_1, the vertices controlled by each player. In addition, we are given a set of k pairs $(I_i, F_i) \subseteq V \times V$. The winning condition is the following: Player 0 wins if there exists an $i \in \{1, \ldots, k\}$ such that the token visits I_i infinitely often but only visits the vertices of F_i a finite number of times. Otherwise, Player 1 wins.

Let us note that it is not hard to see that this game generalizes parity games. Given an instance of parity games with maximum priority p we can set $I_i = \mathrm{Pr}^{-1}(2i)$ and $F_i = \cup_{j \geq i} \mathrm{Pr}^{-1}(2j+1)$ for each $i \leq p/2$. In fact, a more complicated reduction in the opposite direction is possible: it is known that parity games parameterized by the maximum priority p are FPT-equivalent to Rabin games parameterized by the number of winning conditions k [3]. Nevertheless, this reduction requires super-polynomial time to run. In the context of polynomial time solvability the two problems are quite different. While parity games are in NP$\cap$coNP, Rabin games are NP-complete.

Our reduction is from Multi-Colored Clique. It is essentially a simple tweak of known reductions for parity games (see e.g. the reduction in [3]). The new feature is that we are careful that the graph produced has some very restricted structure. Below, we will say that a vertex is non-trivial if its out-degree is more than 1.

Theorem 10. *There is a polynomial-time reduction from* Multi-Colored Clique *to Rabin games which produces an instance $G(V, E)$ with the following properties:*

- *G can be turned into a DAG by deleting one vertex.*
- *G has modular width $2k + 1$.*
- *Player 0 controls k non-trivial vertices.*
- *Player 1 controls 1 non-trivial vertex.*

Proof. Recall that Rabin games, unlike parity games, are not symmetric. If Player 0 has a winning strategy, she also has a positional winning strategy. However, the same is not true for Player 1, who may have a winning strategy that alternates the out-going arc selected from a vertex between rounds [3].

Let $G'(V', E')$ be an instance of Multi-Colored Clique. We construct an instance of Rabin games as follows. First, $V = V' \cup \{1, \ldots, k\} \cup \{s\}$. We have arcs (s, i) for all $i \in \{1, \ldots, k\}$. For each $i \in \{1, \ldots, k\}$ we also have all arcs from the vertex i to all vertices of V' with color i. Finally, all vertices of V' have an out-going arc to s. Vertex s is controlled by Player 1, while vertices $i \in \{1, \ldots, k\}$ are controlled by Player 0. The remaining vertices have out-degree 1, so their owner is irrelevant.

Let us also define the winning conditions. For each $u \in V'$ we define $I_u = \{u\}$ and $F_u = V' \setminus N(u)$, that is, for each vertex u of the original graph Player 0 can win by visiting u infinitely often, provided she refrains from visiting its non-neighbors.

It is not hard to see that the graph satisfies the claimed properties. Let us therefore establish correctness. First, suppose there exists a multi-colored clique of size k in the original graph. Player 0 has the following positional strategy: for each vertex i she selects the outgoing arc that leads to the vertex of the clique. Then, for any strategy of Player 1, some vertex of V' will be visited infinitely often without visiting any of its non-neighbors.

Conversely, suppose that Player 0 has a winning strategy, which must be positional. Consider the vertices of V' which are the heads of the out-going arcs selected in this strategy. We argue that they must be a clique. For contradiction, suppose that two of them u, v are not connected, and they have colors i_1, i_2. Then Player 1 would be able to counter the strategy of Player 0 by alternating his moves as follows: each time the token arrives at s it is pushed either to i_1 or to i_2. In this case, the only vertices visited infinitely often would be $\{s, i_1, i_2, u, v\}$, which would mean that Player 1 wins. $\square$

Acknowledgements

We would like to thank Danupon Nanongkai for suggesting this problem and for our useful discussions.

References

[1] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdržálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

[2] D. Berwanger, E. Grädel, L. Kaiser, and R. Rabinovich. Entanglement and the complexity of directed graphs. *Theor. Comput. Sci.*, 463:2–25, 2012.

[3] H. Bjorklund, S. Sandberg, and S. Vorobyov. On fixed-parameter complexity of infinite games. In K. Sere and M. Waldén, editors, *The Nordic Workshop on Programming Theory (NWPT'03)*, number 34 in Åbo Akademi, Reports on Computer Science and Mathematics, pages 29–31. Citeseer, 2003.

[4] H. Björklund, S. Sandberg, and S. G. Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theor. Comput. Sci.*, 310(1-3):365–378, 2004.

[5] H. Björklund and S. G. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

[6] M. Bojanczyk, C. Dittmann, and S. Kreutzer. Decomposition theorems and model-checking for the modal μ-calculus. In T. A. Henzinger and D. Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, page 17. ACM, 2014.

[7] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997.

[8] K. Chatterjee and T. A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

[9] J. Chen, X. Huang, I. A. Kanj, and G. Xia. On the computational hardness based on linear fpt-reductions. *Journal of Combinatorial Optimization*, 11(2):231–247, 2006.

[10] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006.

[11] A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

[12] C. Dittmann, S. Kreutzer, and A. I. Tomescu. Graph operations on parity games and polynomial-time algorithms. *arXiv preprint arXiv:1208.1640*, 2012.

[13] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[14] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001.

[15] F. V. Fomin, M. Liedloff, P. Montealegre-Barba, and I. Todinca. Algorithms parameterized by vertex cover and modular width, through potential maximal cliques. In R. Ravi and I. L. Gørtz, editors, *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2014.

[16] J. Gajarský, M. Lampis, and S. Ordyniak. Parameterized algorithms for modular-width. In G. Gutin and S. Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

[17] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[18] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[19] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008.

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

[21] M. Jurdzinski. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

[22] M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

[23] R. Küsters. Memoryless determinacy of parity games. In Grädel et al. [17], pages 95–106.

[24] R. McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.

[25] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In W. A. H. Jr. and F. Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2003.

[26] J. Obdržálek. Clique-width and parity games. In J. Duparc and T. A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2007.

[27] S. Schewe. Solving parity games in big steps. In V. Arvind and S. Prasad, editors, *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2007.

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

[29] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.