Benchmarks for Parity Games

Jeroen J.A. Keiren[1,2]

[1] Jeroen.Keiren@ou.nl
Open University of the Netherlands
Faculty of Management, Science & Technology
Heerlen, The Netherlands
[2] VU University Amsterdam
Theoretical Computer Science
Amsterdam, The Netherlands

Abstract We propose a benchmark suite for parity games that includes the benchmarks that have been used in the literature, and make it available online. We give an overview of the parity games, including a description of how they have been generated. We also describe structural properties of parity games, and using these properties we show that our benchmarks are representative. With this work we provide a starting point for further experimentation with parity games.

1 Introduction

Parity games (see, *e.g.*, [24,55,78]) play an important role in model checking research. The μ-calculus model checking problem is polynomial time reducible to the problem of deciding the winner in parity games [73]. Other problems that are expressible in parity games are equivalence checking of labelled transition systems [73], as well as synthesis, satisfiability and validity of temporal logics [66].

Besides their practical interest for verification, solving (deciding the winner of) parity games is known to be in the complexity class $\mathsf{NP} \cap \mathsf{co} - \mathsf{NP}$, and more specifically in $\mathsf{UP} \cap \mathsf{co} - \mathsf{UP}$ [42]. Parity game solving is one of the few problems in this complexity class that is not known to be in P, yet there is hope that a polynomial time algorithm exists. In recent years this has led to the development of (1) a large number of algorithms for solving parity games, such as [44,67,68], all of which were recently shown to be exponential, and (2) the study of (polynomial time) reduction techniques for parity games [30,47,21,22].

So far, practical evaluation of parity game algorithms has been based on ad-hoc benchmarks, mainly consisting of random games or synthetic benchmarks. Friedmann and Lange observed in 2009 [30] that no standard benchmark set for parity games was available. They introduced a small benchmark set in the context of their comprehensive comparison of parity game solving algorithms and their related heuristics [30]. The set of benchmarks was extended in [47,21,22] using model checking and equivalence checking cases. To the best of our knowledge, the situation has not improved since then, and the benchmarks in these

papers still are the most comprehensive benchmarks included in a single paper. The number of games and the diversity of parity games in each set in isolation are however limited. The lack of standard benchmarks makes it hard to compare the different tools and algorithms presented in the literature.

To improve the current situation, in this paper we propose a set of parity games for benchmarking purposes that (1) is diverse, (2) contains games that originate from different verification problems, and (3) includes those games that have been used to experimentally evaluate algorithms in the literature.

In general, parity game examples in the literature can be classified as follows (we indicate their origins):

1. Encodings of problems such as model checking, equivalence checking and complementation of Büchi automata to parity games [53,54,75,47,21,22,30].
2. Synthetic parity games for which a certain solving algorithm requires exponential time [53,43,58,26,31,29,35].
3. Random games [6,49,68,69,30,31].

Our benchmarks include games from each of these categories.

Additionally, inspired by the properties for explicit state spaces in [60] we introduce a set of structural properties for parity games, and in the spirit of [61,62] we analyse our benchmarks. Among others, we introduce a novel notion of alternation depth for parity games.

The structure of the paper is as follows. We first introduce parity games and their structural properties in Section 2. Next we describe the benchmarks (Section 3) and the way in which they have been generated (Section 4). Finally we illustrate diversity of our benchmarks with respect to the structural properties in Section 5. This paper is based on the PhD thesis of the author [45, Chapter 5]; an extended version of this paper, including more detailed descriptions and analyses is available as [46]. We plan to keep [46] up-to-date when new benchmarks are added, and we invite the community to contribute benchmarks.

2 Parity Games and Their Structural Properties

A parity game is a two-player game played on a finite, directed graph by two players, *even* and *odd*, denoted $\diamond$ and $\square$, respectively. We use $\bigcirc \in \{\diamond, \square\}$ to denote an arbitrary player. Formally, a parity game is a structure $(V_\diamond, V_\square, \rightarrow, \Omega)$, where $V_\diamond$ and $V_\square$ are disjoint sets of vertices. We say that $\bigcirc$ owns v if $v \in V_\bigcirc$, we write V for $V_\diamond \cup V_\square$; $\rightarrow \subseteq V \times V$ provides the total edge relation—hence each vertex has a successor—and $\Omega \colon V \rightarrow \mathbb{N}$ assigns a non-negative integer priority to every vertex. The parity game is played by placing a token on some initial vertex, and then the players take turns moving the token: if the token is on a vertex $v \in V_\bigcirc$ then $\bigcirc$ plays the token to one of the successors of v. This way, an infinite play through the game is constructed. If the largest priority that occurs infinitely often on this play is *even* (resp. *odd*) then $\diamond$ (resp. $\square$) wins the play.

The time required for parity game solving and reduction algorithms depends on the structure of the game. Typically the algorithmic complexity of parity

game algorithms are expressed in terms of the size of the game graph, *i.e.* the number of vertices and edges, and the number of priorities in the game. Although other structural properties may not affect the asymptotic running times of the algorithms, in general they do affect the actual running time. We therefore describe a number structural properties that could be used for the further study of parity games.

Sizes. As basic parity game properties, we consider the numbers of vertices $|V|$, $|V_\diamond|$ and $|V_\square|$, and the number of edges $|\rightarrow|$. We write $\Omega(V)$ for the set of priorities $\{\Omega(v) \mid v \in V\}$, and denote the number of priorities in the game by $|\Omega(V)|$. The number of vertices with priority k is represented by $|\Omega^{-1}(k)|$. The complexity of most parity game algorithms is expressed in these quantities. For parity games in which either $|V_\square| = 0$ or $|V_\diamond| = 0$, special polynomial time solving algorithms are available, see [30].

Degrees. Typical structural properties in the graph are the in- and out-degrees of vertices, *i.e.*, the number of incoming and outgoing edges of vertices. Formally, for vertex $v \in V$, $\mathsf{indeg}(v) = |\{u \in V \mid u \rightarrow v\}|$, $\mathsf{outdeg}(v) = |\{w \in V \mid v \rightarrow w\}|$, and $\mathsf{deg}(v) = |\{w \in V \mid v \rightarrow w \vee w \rightarrow v\}|$ are the in-degree, out-degree and degree of v. We consider the minimum, maximum and average of these values.

The degrees of vertices might have an effect on, *e.g.*, algorithms that use lifting strategies to propagate information between vertices. Examples of such algorithms are small progress measures [43] and the strategy improvement algorithm [68].

Strongly Connected Components. The strongly connected components (SCCs) of a graph are the maximal strongly connected subgraphs. More formally, a strongly connected component is a maximal set $\mathcal{C} \subseteq V$ for which, for all $u, v \in \mathcal{C}$, $u \rightarrow^* v$, *i.e.*, each vertex in $\mathcal{C}$ can reach every other vertex in $\mathcal{C}$.

The strongly connected components in a graph induce a quotient graph. Let $\mathsf{sccs}(G)$ denote the strongly connected components of the graph. The quotient graph is the graph $(\mathsf{sccs}(G), \rightarrow')$ and for $\mathcal{C}_1, \mathcal{C}_2 \in \mathsf{sccs}(G)$, there is an edge $\mathcal{C}_1 \rightarrow' \mathcal{C}_2$ if and only if $\mathcal{C}_1 \neq \mathcal{C}_2$ and there exist $u \in \mathcal{C}_1$ and $v \in \mathcal{C}_2$ such that $u \rightarrow v$. Observe that the quotient graph is a directed acyclic graph.

We say that an SCC $\mathcal{C}$ is *trivial* if $|\mathcal{C}| = 1$ and $\mathcal{C} \not\rightarrow \mathcal{C}$, *i.e.*, it only contains one vertex and no edges, and we say that $\mathcal{C}$ is *terminal* if $\mathcal{C} \not\rightarrow'$, *i.e.*, its outdegree in the quotient graph is 0. The *SCC quotient height* of a graph is the length of the longest path in the quotient graph.

Parity game algorithms and heuristics can benefit from a decomposition into strongly connected components (SCCs). One prominent example of this is the global parity game solving algorithm presented by Friedmann and Lange [30], for which it was shown that SCC decomposition generally works well in practice.

Properties of Search Strategies. Given some initial vertex $v_0 \in V$, breadth-first search (BFS) and depth-first search (DFS) are search strategies that can be used to systematically explore all vertices in the graph. The fundamental difference

between BFS and DFS is that the BFS maintains a queue of vertices that still need to be processed, whereas the DFS maintains a stack of vertices. We record the queue and stack sizes during the search.

Breadth-first search induces a natural notion of levels, where a vertex is at level k if it has least distance k to v_0. The *BFS height* of a graph is k if k is the maximal non-empty level of the BFS. For each level the number of vertices at that level is recorded. During a BFS, three kinds of edges can be detected, *viz.* edges that go to a vertex that was not yet seen, edges that go to a vertex that was seen, but has not yet been processed (*i.e.*, vertices in the queue) and edges that go back to a vertex on a previous level. This last type of edges is also referred to as a *back-level edge*. Formally it is an edge $u \to v$ where the level of u, say k_u is larger than the level of v, say k_v. The length of a back-level edge $u \to v$ is $k_u - k_v$.

Graph algorithms are typically based on a search strategy like BFS or DFS, given some initial vertex $v_0 \in V$. The characteristics of these search strategies are therefore likely to affect the performance of such graph algorithms.

Width-measures on Graphs. Width-measures of graphs are based on cops-and-robbers games [56,63], where different measures are obtained by varying the rules of the game. For various measures, specialised algorithms are known that can solve games polynomially if their width is bounded. Most of the measures have an alternative characterisation using graph decompositions.

The classical width notion for *undirected graphs* is *treewidth* [64,11]. Intuitively, the treewidth of a graph expresses how tree-like the graph is—the treewidth of a tree is 1. This corresponds to the idea that some problems are easier to solve for trees, or graphs that are almost trees, than for arbitrary graphs. For directed graphs, the treewidth is defined as the treewidth of the graph obtained by forgetting the direction of the edges. The complexity for solving parity games is bounded in the treewidth [57]; this means that, for parity games with a small, constant treewidth, parity game solving is polynomial.

Treewidth has been lifted to directed graphs in a number of different ways. For instance, *Directed treewidth* [41] is bounded by the treewidth [1]. *DAG-width* [7] describes how much a graph is like a directed acyclic graph. DAG-width bounds the directed tree width of a graph from above, and is at most the treewidth. The *Kelly-width* [40] is yet another generalitation of treewidth to directed graphs. If the Kelly-width of a graph is bounded, then also a bound on its directed treewidth can be given, however, classes of directed graphs with bounded directed treewidth and unbounded Kelly-width exist. *Entanglement* [9,10] is a graph measure that aims to express how much the cycles in a graph are intertwined. If an undirected graph has bounded treewidth or bounded DAG-with, then it also has bounded entanglement. Finally, *clique-width* [19] measures how close a graph is to a complete bipartite graph. For every directed graph with bounded treewidth an exponential upper bound on its clique-width can be given. Unlike the other width measures that we discussed clique-width does not have a characterisation in terms of cops-and-robbers games.

If a parity game is bounded to a constant in any of the measures introduced above, it can be solved in polynomial time.

Alternation Depth. Typically, the complexity of parity game algorithms is expressed in the number of vertices, the number of edges, and the number of priorities in the game. If we look at other verification problems, such as μ-calculus model checking, or solving Boolean equation systems, the complexity is typically expressed in terms of the *alternation depth*. Different versions of alternation depth (with varying precision) have been coined, see [14]. Intuitively, the alternation depth of a formula captures the number of alternations between different fixed point symbols.

Analogous to the definition of alternation depth for modal equation systems by Cleaveland *et al.* [18], our definition consists of two parts. First we define the nesting depth of a strongly connected component within a parity game, next we define the alternation depth of the parity game as the maximum of the nesting depths of its strongly connected components.

Definition 1. *Let* $G = (V_\diamondsuit, V_\square, \rightarrow, \Omega)$ *be a parity game, and let* $\mathsf{sccs}(G)$ *be the set of strongly connected components of* G. *Let* $\mathcal{C} \in \mathsf{sccs}(G)$ *be a strongly connected component. The nesting depth of* v_i *in* $\mathcal{C}$ *is given by*

$$\mathsf{nd}(v_i, \mathcal{C}) \triangleq \max\{1,$$
$$\max\{\mathsf{nd}(v_j, \mathcal{C}) \mid v_j \rightarrow^*_{\mathcal{C}, \Omega(v_i)} v_i, v_j \neq v_i \text{ and } \Omega(v_i) \equiv_2 \Omega(v_j)\},$$
$$\max\{\mathsf{nd}(v_j, \mathcal{C}) + 1 \mid v_j \rightarrow^*_{\mathcal{C}, \Omega(v_i)} v_i \text{ and } \Omega(v_i) \not\equiv_2 \Omega(v_j)\}$$
$$\}$$

where $v_j \rightarrow_{\mathcal{C}, k} v_i$ *if* $v_j \rightarrow v_i$ *is an edge in the SCC* $\mathcal{C}$ *with* $\Omega(v_j) \leq k$ *and* $\Omega(v_i) \leq k$. *Intuitively, the nesting depth of a vertex* v *counts the number of alternations between even and odd priorities on paths of descending priorities in the SCC of* v. *Note that this is well-defined since we forbid paths between identical nodes.*

The nesting depth of an SCC $\mathcal{C} \in \mathsf{sccs}(G)$ is defined as the maximum nesting depth of any vertices in $\mathcal{C}$, *i.e.*, $\mathsf{nd}(\mathcal{C}) \triangleq \max\{\mathsf{nd}(v, \mathcal{C}) \mid v \in \mathcal{C}\}$. The *alternation depth* of a parity game is defined as the maximal nesting depth of its SCCs.

Definition 2. *Let* $G = (V_\diamondsuit, V_\square, \rightarrow, \Omega)$ *be a parity game, and let* $\mathsf{sccs}(G)$ *be the set of strongly connected components of* G. *Then the* alternation depth *of* G *is defined as* $\mathsf{ad}(G) \triangleq \max\{\mathsf{nd}(\mathcal{C}) \mid \mathcal{C} \in \mathsf{sccs}(G)\}$.

There are reasonable translations of the μ-calculus model checking problem into parity games, such that the alternation depth of the resulting parity game is at most the fixed point alternation depth of the μ-calculus formula as described by Emerson and Lei [25], see [45, Proposition 5.4]. Note that the alternation depth of a game can be smaller than the number of priorities in the game, and could provide an interesting alternative to the number of priorities in computing the complexity of parity game algorithms.

Other measures. When studying structural properties for labelled transition systems, other global measures such as *diameter* and *girth* have been considered. The diameter is the maximal length of a shortest path between any pair of vertices. The girth is the length of the shortest cycle in the graph. Also, local properties such as the *number of diamonds* and the *k-neighbourhood* were studied. These measures could be considered for parity games as well, but currently there is no clear indication that they are related to the performance of parity game algorithms. These measures and their analysis with respect to the games presented in the next section have been described in more detail in the extended version [46].

3 Benchmarks

For benchmarking parity game algorithms, it makes sense to distinguish three classes of parity games, (1) the games that are the result of encoding a problem into parity games, (2) games that represent hard cases for certain algorithms, and (3) random games. All three classes of games occur in the literature, and our benchmark set contains games from each of these classes. In the rest of this section we discuss our benchmarks. In the next section we briefly discuss these games with respect to the properties described in Section 2.

3.1 Encodings

A broad range of verification problems can be encoded as a parity game. The most prominent examples of these are the μ-calculus model checking problem—does a model satisfy a given property?—, equivalence checking problems—are two models equivalent?—, decision procedures—is a formula valid or satisfiable?—and synthesis—given a property, give a model that satisfies the property.

Model Checking. The model checking problems we consider are mainly selected from the literature. All of the systems are encodings that, given a model L of a system, and a property φ, encode the model checking problem $L \models \varphi$, *i.e.*, does L satisfy property φ. Most sensible encodings of model checking problems typically lead to a low number of priorities, corresponding to the low alternation depths of these properties. We verify fairness, liveness and safety properties. This set includes, but is not limited to, the model checking problems described in [54,75,30,21,22].

We take a number of communication protocols from the literature, see, *e.g.*, [4,15,48,38]: two variations of the *Alternating Bit Protocol* (ABP), the *Concurrent Alternating Bit Protocol* (CABP), the *Positive Acknowledgement with Retransmission Protocol* (PAR), the *Bounded Retransmission Protocol* (BRP), the *Onebit* sliding window protocol, and the *Sliding Window Protocol* (SWP). All protocols are parameterised with the number of messages that can be sent, and the sliding window protocol is parameterised by the window size. For these

protocols a number of properties of varying complexity was considered, ranging from alternation free properties, *e.g.* deadlock freedom, to fairness properties.

A *Cache Coherence Protocol* (CCP) [76] and a *wait-free handshake register* (Hesselink) [39] are considered. For the cache coherence protocol we consider a number of properties from [59] and for the register we consider properties from [39]. Additionally we consider a *leader election protocol* for which we verify whether it eventually stabilises.

To obtain parity games with a high degree of alternation between vertices owned by different players we also consider a number of two-player board games, *viz. Clobber* [2], *Domineering* [34], *Hex*, see *e.g.* [5,52], *Othello*, also known as reversi, see *e.g.* [65], and *Snake*. For these games we check for each of the players whether the player has a winning strategy starting from the initial configuration of the game. The games are parameterised by their board size.

Additionally, we consider a number of industrial model checking problems. The first is a system for lifting trucks (Lift) [37], of which we consider both a correct and an incorrect version. We verify the liveness and safety properties described in [37]. For the *IEEE 1394 Link Layer Protocol* (1394) we verify the properties from [51]. We translated the ACTL properties from [71] to the μ-calculus.

Finally, we check the *Elevator* described by Friedmann and Lange, in a version in which requests are treated on a first-in-first-out basis (FIFO), and on a last-in-first-out basis (LIFO). We then check whether, globally, if the lift is requested on the top floor, then it is eventually served. This holds for the FIFO version, but does not hold for the LIFO version of the model. The elevator model is parameterised by the strategy and the number of floors. Furthermore we consider the parity games generated using an encoding of an LTS with a μ-calculus formula, as well as the direct encoding presented in [30]. In a similar way we consider the Hanoi towers from [30] as well as our own version of this problem.

Equivalence Checking. Given two processes L_1, L_2, the problem whether $L_1 \equiv L_2$, for relations $\equiv$, denoting that L_1 and L_2 are equivalent under some process equivalence, can be encoded as a parity game [50,77]. We consider strong bisimulation, weak bisimulation, branching bisimulation and branching simulation equivalence in our benchmarks, using the approach described in [17]. The number of different priorities in these parity games is limited to 2, but they do include alternations between vertices owned by different players.

Here we again use the specifications of the communication protocols that we also used for model checking, *i.e.*, two ABP versions, CABP, PAR, Onebit and SWP. In addition we include a model of a buffer. We vary the capacity of the buffer, the number of messages that can be transmitted, and the window size in the sliding window protocol. We compare each pair of protocols using all four equivalences, resulting in both positive and negative cases. These cases are a superset of the ones described in [21,22].

In addition, we include a comparison of the implementation of the wait-free handshake register with a possible specification. The implementation is trace

equivalent to the specification, but it is not equivalent with respect to the equivalences that we consider here.

Decision Procedures. Parity games can also be obtained from decision procedures for temporal logics such as LTL, CTL, CTL*, PDL and the μ-calculus. Friedmann *et al.* presented a decision procedure that is based on a combination of infinite tableaux in which the existence of a tableau is coded as a parity game [33]. For a given formula, it is checked whether it is (1) *valid*, *i.e.*, whether the formula holds in all models, or (2) *satisfiable*, *i.e.*, whether the formula is satisfiable in some model.

Our benchmark set includes a number of scalable satisfiability and validity problems that are provided as examples for the MLSolver tool [32]. In particular, we include the benchmarks used in [32]: encoding that a deterministic parity condition is expressible as a nondeterministic Büchi condition, and nesting Kleene stars in different logics. Additionally we consider formulas that involve encodings of a binary counter in various logics.

Synthesis. Another problem that involves solving parity games is the LTL synthesis problem. Traditional synthesis approaches convert a formula into a nondeterministic Büchi automaton, which is, in turn, transformed into a deterministic parity automaton using Safra's construction [66]. Emptiness of this deterministic parity automaton can then be checked using parity games with three priorities. Synthesis tools have been implemented that employ parity games internally, most notably GOAL [74] and Gist [16]. All synthesis tools that we are aware of, however, are research quality tools, of which we have not been able to obtain working versions on current computing platforms. As a result, our benchmark set currently does not include parity games obtained from the synthesis problem. We plan to extend our benchmarks with such games, and update [46] accordingly.

3.2 Hard Games

The interesting complexity of solving parity games, and its link to the model checking problem, have led to the conception of a large number of parity game solving algorithms. For most of these algorithms it has long been an open problem whether they have exponential lower bounds.

We consider the games described by Jurdziński that shows the exponential lower bound for small progress measures [43], the *ladder games* described by Friedmann [28] defeating strategy guessing heuristics, *recursive ladder games* that give a lower bound for the recursive algorithms, and *model checker ladder games* [27] for which the algorithm by Stevens and Stirling [72] behaves exponentially.

3.3 Random Games

The final class of games that is typically used in publications that empirically evaluate the performance of algorithms on parity games are random parity games

[6,69,68,49,30]. We study three classes of random games. We expect that the structural properties of random games are, typically, different from parity games obtained in the previous classes. This class is, therefore, unlikely to give insights in the performance of parity game algorithms on practical problems.

4   Implementation

All games were generated on a 1TB main memory, 56-core Linux machine, where each core was running at 2.27GHz. Executions of tools generating and solving parity games, and tools collecting statistics about parity games, were limited to running times of 1 hour and their memory usage was limited to 32GB.

To systematically generate the benchmarks, we have implemented tooling that allows the parallel execution of individual cases. Here a case is either generating or solving a game, or collecting a single measure. Each individual case only uses a single core. The tools are implemented in an extensible way, *i.e.*, additional parity games, additional encodings, as well as additional measures can be added straightforwardly. The tools are available for download from https://github.com/jkeiren/paritygame-generator.

4.1   Generating Parity Games

For the generation of our benchmarks we rely on a number of external tools: version 3.3 of PGSolver [31] for generating random games, and games that prove to be hard for certain algorithms; version 1.2 of MLSolver to generate the games for satisfiability and validity problems [32]; and revision 11703 of the mCRL2 toolset [20] for the model checking and equivalence checking problems. For all games we have collected the information described in Section 2 to the extent in which this is feasible.

4.2   Collecting Statistics

We developed the tool `pginfo` for collecting structural information from parity games. The tool is available from https://github.com/jkeiren/pginfo and accepts parity games in the file format used by PGSolver. The tool reads a parity game, and writes statistics to a file in a structured way.

The implementation is built on top of the Boost Graph library [70], which provides data structures and basic algorithms for manipulating graphs. Computing the exact value for the width-measures is problematic: it is known to be NP-complete [3]. Approximation algorithms are known that compute upper- and lower bound for these measures; especially for treewidth these have been thoroughly studied [12,13]. To determine feasibility of computing width-measures for our benchmarks we have implemented three approximation algorithms. For computing upper and lower bounds on treewidth we implemented the greedy degree algorithm [12] and the minor min-width algorithm [36], respectively. For computing an upper bound of the Kelly-width we implemented the elimination

ordering described in [40]. Even these approximation algorithms have proven to be impractical due to their complexity. Computing (bounds) on the other width measures is equally complex.

4.3 Availability of Parity Games

All parity games that are described in this paper are available for download from http://www.github.com/jkeiren/paritygame-generator in bzip2 compressed PG-Solver format [31]. The dataset is approximately 10GB in size, and includes the structural information that was collected from these games.

5 Analysis of Benchmarks

We have presented benchmarks originating from different problems. Next we analyse them with respect to the measures described in Section 2. This analysis illustrates that our benchmarks exhibit a wide variety of properties. Furthermore, this gives us some insights in the characteristics of typical parity games. For each of the statistics, we only consider games for which that specific statistic could be computed within an hour, and we only include those statistics that can feasibly be computed for the majority of games, as a consequence the width measure are excluded from the analysis we present here. We used this selection to avoid timeouts for computing the measures that are expensive to compute. All graphs in this section are labelled by their class. Note that the satisfiability and validity problems are labelled by "mlsolver" and the games that are hard for some solving algorithms are labelled by "specialcases". The full data presented in this chapter is also available from http://www.github.com/jkeiren/paritygame-generator. Due to lack of space, we cannot present an analysis of all measures.

Our data set contains 1037 parity games that range from 2 vertices to 40 million vertices, and on average they have about 95,000 vertices. The number of edges ranges from 2 to 167 million, with an average of about 3.1 million. The 59 parity games are games in which all vertices are owned by a single player, the so-called solitaire games [8], the rest are parity games in which both players own non-empty sets of vertices. The parity games that we consider have differing degrees. There are instances in which the average degree is 1, the average degree is maximally 9999, but it is typically below 10. The ratio between the number of vertices and the number of edges is, therefore, relatively small in general. This can also be observed from Figure 1a, which displays the correlation between the two. The games in which these numbers coincide are on the line $x = y$, the other games lie around this line due to the log scale that we use. Our parity games generally contain a vertex with in-degree 0, which is the starting vertex. Most of the games contain vertices with a high in-degree—typically representing vertices that are trivially won by either of the players—, and vertices with a high out-degree.

In general, the SCC quotient height ranges up to 513 for the parity games that we consider with an average of around 14. The number of non-trivial SCCs can grow large, up to 1.4 million for our games.
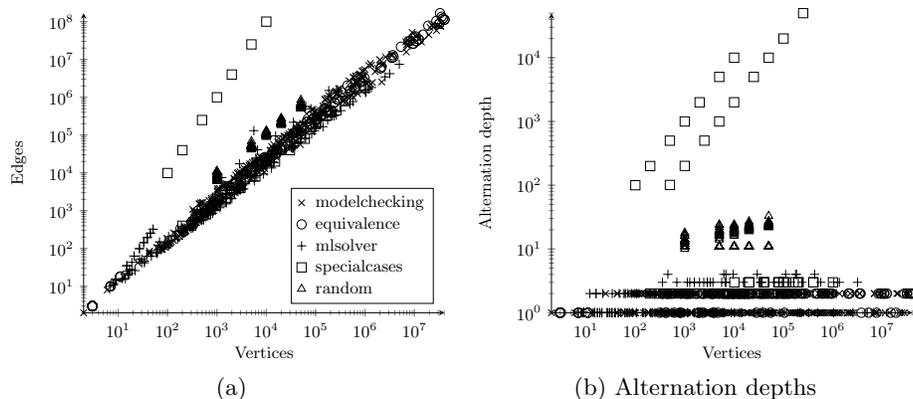
Figure 1: Relation between number of vertices and (a) number of edges (b) alternation depth. The legend is the same for both plots.

We have included parity games with alternation depths up to 50,000 as shown in Figure 1b. Observe that the games for model checking and equivalence checking included in our benchmarks all have alternation depth at most 2. Model checking problems could be formulated that have a higher alternation depth—up to arbitrary numbers—however, in practice properties have limited alternation depth because they become too hard to understand otherwise. The satisfiability and validity properties have alternation depths between 1 and 4. The alternation depths of the random games are between 10 and 15. All parity games with more than 50 priorities represent special cases. Closer investigation shows that these special cases are the clique games and recursive ladder games.

To summarise, we have presented a large set of parity games. For a selection of the structural properties introduced in Section 2 we have shown that the games cover a large range of values. Due to space restrictions we have not covered all structural properties here, we refer to [46] for a more detailed analysis. Also observe that, for parity game specific properties such as alternation depth, higher values are only available for smaller games due to generation times. Unsurprisingly, the random games considered in this paper are not structurally similar to parity games that represent encodings of verification problems.

6 Closing Remarks

No standard benchmarks for parity game algorithms existed. As a consequence, it was virtually impossible to make a good comparison between algorithms and applications described in the literature. In this paper we have addressed this issue by presenting a comprehensive set of parity game benchmarks. Our benchmarks include the games that appear in the literature, and provides a first step towards

11

standardising experimental evaluation of parity game algorithms. All games have been generated in an extensible way, and are available on-line.

We also presented a set of structural properties for parity games, and analysed our benchmarks with respect to these properties. Of particular interest is a new notion of alternation depth for parity games, that is always at most the number of priorities in a parity game, and that is bounded also by the alternation depth of μ-calculus formulae given a reasonable translation of the model checking problem.

Future work. Some of the structural properties, such as treewidth, cannot be computed for all games in the benchmark suite due to their complexity. An interesting algorithmic question is, therefore, whether algorithms or heuristics can be devised that can compute or approximate these measures for large graphs.

Additionally, we have presented a selection of structural properties in this paper. One can wonder whether there are other structural properties of parity games that are relevant to the practical performance of parity game algorithms. The question whether the theoretical complexity of existing parity game algorithms can be made tighter using structural properties, such as our notion of alternation depth is left open.

We believe our work also paves the way for a full-scale comparison of parity game algorithms and the effect of heuristics in the spirit of [30], including the comparison of alternative implementations of algorithms [20,23]. Here also the impact of the structural properties on the performance of implementations should be studied, since we have only scratched the surface of this aspect in this paper.

Finally, we welcome the addition of problems and properties to our benchmark suite to establish and maintain a corpus for experimentation with parity game algorithms. In particular parity games with a large number of priorities and a high alternation depth stemming from encodings of, *e.g.*, verification and synthesis problems form a welcome addition.

References

1. I. Adler. Directed tree-width examples. *Journal of Combinatorial Theory, Series B*, 97(5):718–725, 2007.
2. M.H. Albert, J.P. Grossman, R.J. Nowakowski, and D. Wolfe. An introduction to clobber. *Integers*, 5(2), 2005.
3. S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

4. K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

5. A. Beck, M.N. Bleicher, and D.W. Crowe. *Excursions into Mathematics: The Millennium Edition*. CRC Press, 2000.

6. E. Beffara and S.G. Vorobyov. Adapting Gurvich-Karzanov-Khachiyan's algorithm for parity games. Technical report, Uppsala University, Sweden, Uppsala, 2001.

7. D. Berwanger, A. Dawar, P.W. Hunter, S. Kreutzer, and J. Obdržálek. The DAG-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.

8. D. Berwanger and E. Grädel. Fixed-point logics and solitaire games. *Theory of Computing Systems*, 37(6):675–694, 2004.

9. D. Berwanger and E. Grädel. Entanglement — a measure for the complexity of directed graphs with applications to logic and games. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3452 of *LNCS*, pages 209–223. Springer, 2005.

10. D. Berwanger, E. Grädel, L. Kaiser, and R. Rabinovich. Entanglement and the complexity of directed graphs. *Theoretical Computer Science*, 463:2–25, 2012.

11. H.L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997.

12. H.L. Bodlaender and A.M.C.A. Koster. Treewidth computations I. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

13. H.L. Bodlaender and A.M.C.A. Koster. Treewidth computations II. lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.

14. J.C. Bradfield and C. Stirling. Modal logics and mu-calculi: an introduction. In *Handbook of Process Algebra*, chapter 4, pages 293–330. Elsevier, 2000.

15. V. Cerf and R.E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, 1974.

16. K. Chatterjee, T.A. Henzinger, B. Jobstmann, and A. Radhakrishna. GIST: A solver for probabilistic games. In *CAV 2012*, volume 6174 of *LNCS*, pages 665–669. Springer, 2010.

17. T. Chen, S.C.W. Ploeger, J.C. van de Pol, and T.A.C. Willemse. Equivalence checking for infinite systems using parameterized Boolean equation systems. In *CONCUR'07: Concurrency Theory*, volume 4703 of *LNCS*, pages 120–135. Springer, 2007.

18. R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In *Computer Aided Verification*, volume 663 of *LNCS*, pages 410–422. Springer, 1993.

19. B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

20. S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, J.W. Wesselink, and T.A.C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *TACAS'13*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013.

21. S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. Stuttering mostly speeds up solving parity games. In *NASA Formal Methods*, volume 6617 of *LNCS*, pages 207–221. Springer, 2011.

22. S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. A cure for stuttering parity games. In *Theoretical Aspects of Computing — ICTAC 2012*, volume 7521 of *LNCS*, pages 198–212. Springer, 2012.

23. A. Di Stasio, A. Murano, V. Prignano, and L. Sorrentino. Solving parity games in Scala. In *FACS*, 2014.

24. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy. In *SFCS '91: Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 368–377. IEEE Computer Society, 1991.

25. E.A. Emerson and C.L.L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of LICS 1986*, pages 267–278. IEEE Computer Society, 1986.

26. O. Friedmann. A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it. *2009 24th Annual IEEE Symposium on Logic In Computer Science*, 7:145–156, 2009.

27. O. Friedmann. The Stevens-Stirling-algorithm for solving parity games locally requires exponential time. *International Journal of Foundations of Computer Science*, 21(03):277–287, 2010.

28. O. Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Logical Methods in Computer Science*, 7:1–42, 2011.

29. O. Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO - Theoretical Informatics and Applications*, 45(4):449–457, 2011.

30. O. Friedmann and M. Lange. Solving parity games in practice. In *Automated Technology for Verification and Analysis 7th International Symposium, ATVA 2009*, volume 5799 of *LNCS*, pages 182–196. Springer, 2009.

31. O. Friedmann and M. Lange. The PGSolver collection of parity game solvers. Technical report, Institut für Informatik, Ludwig-Maximilians-Universität München, Germany, 2010.

32. O. Friedmann and M. Lange. A solver for modal fixpoint logics. In *Electronic Notes in Theoretical Computer Science*, volume 262, pages 99–111. Elsevier, 2010.

33. O. Friedmann, M. Latte, and M. Lange. A decision procedure for CTL* based on tableaux and automata. In *Automated Reasoning*, volume 6173 of *LNCS*, pages 331–345. Springer, 2010.

34. M. Gardner. Mathematical games: Cram, crosscram and quadraphage: New games having elusive winning strategies. *Scientific American*, 230:106–108, 1974.

35. M.W. Gazda and T.A.C. Willemse. Zielonka's recursive algorithm: dull, weak and solitaire games and tighter bounds. In *Proceedings GandALF 2013*, volume 119 of *EPTCS*, pages 7–20. 2013.

36. V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, UAI '04, pages 201–208. AUAI Press, 2004.

37. J.F. Groote, J. Pang, and A.G.G. Wouters. Analysis of a distributed system for lifting trucks. *The Journal of Logic and Algebraic Programming*, 55(1-2):21–56, 2003.

38. J.F. Groote and J. van de Pol. A bounded retransmission protocol for large data packets. In *Algebraic Methodology and Software Technology*, volume 1101 of *LNCS*, pages 536–550. Springer, 1996.

39. W.H. Hesselink. Invariants for the construction of a handshake register. *Information Processing Letters*, 68:173–177, 1998.

40. P.W. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008.

41. T. Johnson, N. Robertson, P.D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

42. M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

43. M. Jurdziński. Small progress measures for solving parity games. In *STACS '00: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

44. M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06*, pages 117–123, 2006.

45. J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. PhD thesis, Eindhoven University of Technology, 2013.

46. J.J.A. Keiren. Benchmarks for parity games. *CoRR*, abs/1407.3121, 2014.

47. J.J.A. Keiren and T.A.C. Willemse. Bisimulation minimisations for Boolean equation systems. In *Proceedings of Haifa Verification Conference 2009 (HVC'09)*, volume 6405 of *LNCS*, pages 102–116. Springer, Heidelberg, 2011.

48. C.P.J. Koymans and J.C. Mulder. A modular approach to protocol verification using process algebra. In *Applications of Process Algebra*, number 17 in Cambridge Tracts in Theoretical Computer Science, pages 261–306. 1990.

49. M. Lange. Solving parity games by a reduction to SAT. In *Proc. of the Workshop on Games in Design and Verification, GDV'05*, 2005.

50. K.G. Larsen. Efficient local correctness checking. In *Computer Aided Verification, Fourth International Workshop, CAV '92, Proceedings*, volume 663 of *LNCS*, pages 30–43. Springer, 1993.

51. S.P. Luttik. Description and formal specification of the link layer of P1394. In *Workshop on Applied Formal Methods in System Design*, pages 43–56, 1997.

52. T. Maarup. Hex - everything you always wanted to know about hex but were afraid to ask. Master's thesis, 2005.

53. A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technische Universität München, 1997.

54. R. Mateescu. A generic on-the-fly solver for alternation-free Boolean equation systems. In *TACAS'03*, volume 2619 of *LNCS*, pages 81–96. Springer, 2003.

55. R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

56. R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3):235–239, 1983.

57. J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 80–92. Springer, 2003.

58. J. Obdržálek. *Algorithmic Analysis of Parity Games*. PhD thesis, Laboritory for Foundations of Computer Science, School of Informatics, University of Edinburgh, 2006.

59. J. Pang, W.J. Fokkink, R. Hofman, and R. Veldema. Model checking a cache coherence protocol of a Java DSM implementation. *The Journal of Logic and Algebraic Programming*, 71(1):1–43, 2007.

60. R. Pelánek. Typical structural properties of state spaces. In *Model Checking Software*, volume 2989 of *LNCS*, pages 5–22. Springer, 2004.

61. R. Pelánek. Web portal for benchmarking explicit model checkers. Technical Report FIMU-RS-2006-03, Faculty of Informatics Masaryk University Brno, 2006.

62. R. Pelánek. BEEM: benchmarks for explicit model checkers. In *Model Checking Software*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.

63. A. Quilliot. *Jeux et pointes fixes sur les graphes*. PhD thesis, Université de Paris VI, 1978.

64. N. Robertson and P.D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

65. B. Rose. *Othello: A Minute to Learn... A Lifetime to Master*. 2005.

66. S. Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327. IEEE, 1988.

67. S. Schewe. Solving parity games in big steps. volume 4855 of *LNCS*, pages 449–460. Springer, 2007.

68. S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Computer Science Logic*, volume 5213 of *LNCS*, pages 369–384. Springer, 2008.

69. S. Schewe. *Synthesis of Distributed Systems*. Phd thesis, Universität des Saarlandes, 2008.

70. J.G. Siek, L.Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.

71. M. Sighireanu and R. Mateescu. Verification of the link layer protocol of the IEEE-1394 serial bus (FireWire): An experiment with E-LOTOS. *STTT*, 2(1):68–88, 1998.

72. P. Stevens and C. Stirling. Practical model checking using games. In *TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

73. C. Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of IGPL*, 7(1):103124, January 1999.

74. Y.K. Tsay, Y.F. Chen, M.H. Tsai, W.C. Chan, and C.J. Luo. GOAL extended: Towards a research tool for omega automata and temporal logic. In *TACAS'08*, volume 4963 of *LNCS*, pages 346–350. Springer, 2008.

75. J.C. van de Pol and M. Weber. A multi-core solver for parity games. *Electronic Notes in Theoretical Computer Science*, 220(2):19–34, 2008.

76. R. Veldema, R.F.H. Hofman, R.A.F. Bhoedjang, C.J.H. Jacobs, and H.E. Bal. Source-level global optimizations for fine-grain distributed shared memory systems. *ACM SIGPLAN Notices*, 36(7):83–92, June 2001.

77. B. Vergauwen and J. Lewi. Efficient local correctness checking for single and alternating Boolean equation systems. In *Automata, Languages and Programming*, volume 820 of *LNCS*, pages 304–315. Springer, 1994.

78. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.