Predicting Winning Regions in Parity Games via Graph Neural Networks

Tobias Hecking*
German Aerospace Center (DLR)
Institute for Software Technology
Cologne, Germany
tobias.hecking@dlr.de

Swathy Muthukrishnan*
University of Stuttgart
Stuttgart, Germany
swathy.muthukrishnan.24@gmail.com

Alexander Weinert*
German Aerospace Center (DLR)
Institute for Software Technology
Cologne, Germany
alexander.weinert@dlr.de

Abstract

Solving parity games is a major building block for numerous applications in reactive program verification and synthesis. While they can be solved efficiently in practice, no known approach has a polynomial worst-case runtime complexity. We present a incomplete polynomial-time approach to determining the winning regions of parity games via graph neural networks.

Our evaluation on 900 randomly generated parity games shows that this approach is effective and efficient in practice. It correctly determines the winning regions of ~60% of the games in our data set and only incurs minor errors in the remaining ones. We believe that this approach can be extended to efficiently solve parity games as well.

ACM Reference Format:
Tobias Hecking, Swathy Muthukrishnan, and Alexander Weinert. 2023. Predicting Winning Regions in Parity Games via Graph Neural Networks. In *Proceedings of First International Workshop on Deep Learning-aided Verification (DAV '23).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Parity games are infinite arena-based games between Player 0 and Player 1 that capture all ω-regular languages. They are the canonical model for specifications of reactive systems, where Player 0 and Player 1 represent the system and its environment, respectively. Solving them is a major building block for reactive program verification and synthesis. [3, 16] Solving comprises determining the winning regions of both players as well as a winning strategy for both players.

The problem of solving parity games is known to be in UP∩coUP [9] and to be solvable in quasi-polynomial time [2]. There exist parity game solvers that are efficient in practice [4, 15, 17], all of which suffer from a non-polynomial worst-case runtime complexity.

Here, we trade completeness for efficiency in determining the winning regions. Classically, parity games are solved by building a correct-by-construction strategy, which yields the winning regions as a by-product. We instead train graph neural networks (GNNs) [1] to determine the winning regions. Applying the trained GNN to a parity game $\mathcal{G}$ then yields a prediction of the winning region of $\mathcal{G}$.

In a preliminary evaluation we constructed two GNNs and trained these on 2 100 randomly generated parity games. We then predicted the winning region of another 900 randomly generated parity games using these GNNs. This approach yields correct winning regions in 537 (433) cases, as well as regions that differ by one vertices from the correct result in 245 (279) cases.

2 Preliminaries

We write $\mathbb{R}$ and $\mathbb{R}^{k,l}$ to denote the real numbers and the set of $k \times l$-matrices over the reals, respectively. Moreover, given a set S, we write S^n to denote the n-ary cartesian power of S.

Parity Games. An arena $\mathcal{A} = (V, V_0, V_1, E)$ comprises a finite set of vertices V, a partition (V_0, V_1) of V, as well as a set of edges $E \subseteq V \times V$ with $\forall v \in V. (\{v\} \times V) \cap E \neq \emptyset$. We call V_i the vertices of Player i. A play $\rho = v_0 v_1 v_2 \cdots$ of $\mathcal{A}$ is an infinite path through (V, E). A parity game $(\mathcal{A}, \textsc{Parity}(\Omega))$ comprises an arena $\mathcal{A}$ with vertex set V and a coloring $\Omega \colon V \longrightarrow \mathbb{N}$. The play ρ is winning for Player i if $(\max \{\Omega(v_i) \mid i \in \mathbb{N}\}) \mod 2 = i$.

A strategy $\sigma \colon V_i \longrightarrow V$ for Player i is a mapping with $\forall v \in V_i. (v, \sigma(v)) \in E$. A play $\rho = v_0 v_1 v_2 \cdots$ is consistent with σ if $\forall j \in \mathbb{N}. v_j \in V_i \implies v_{j+1} = \sigma(v_j)$. We say that Player i wins $\mathcal{G}$ from vertex $v \in V$ if she has a strategy σ such that all plays starting in v and consistent with σ are winning for her. The winning region $W_i(\mathcal{G})$ of Player i in $\mathcal{G}$ is the set of all vertices from which Player i wins $\mathcal{G}$.

Neural Network Layers. A k, l-neural network layer is a function $h \colon \mathbb{R}^k \longrightarrow \mathbb{R}^l$. In this work we use the linear layer $h_{A,b}(x) = xA^T + b$, where $A \in \mathbb{R}^{l \times k}$ and $b \in \mathbb{R}^{1 \times l}$, the rectified

*Authors are listed in alphabetical order and have contributed equally to this work

DAV '23, July 18, 2023, Paris, France
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

linear unit (ReLU) layer

$$h((x_1,\ldots,x_k)^T) = (\max(0,x_1),\ldots,\max(0,x_k))^T \ ,$$

and the softmax layer

$$h((x_1,\ldots,x_k)^T) = ({e^{x_1}}/{\Sigma_{1\leq i\leq j} e^{x_i}},\ldots,{e^{x_n}}/{\Sigma_{1\leq i\leq j} e^{x_i}})^T \ .$$

Graph Neural Networks. A k-attributed graph $G = (V, E, X)$ comprises a graph (V, E) and a vertex-labeling $X\colon V \to \mathbb{R}^k$. A graph neural network (GNN) in its general form maps a k-attributed input graph G to an isomorphic l-attributed output graph G' by passing information through several layers. Intuitively, each layer of an GNN can be seen as a function $h\colon \mathbb{R}^k \to \mathbb{R}^l$ that for each node aggregates the state of its neighbours and updates its state according to some rule.

Formally, for $v \in V$ we define $\mathcal{N}(v) = \{v' \in V \mid (v, v') \in E\}$ and $\deg(v) = |\mathcal{N}(v)|$. Let $\mathcal{G}_k$ denote the set of k-attributed graphs. A k,l-message-passing layer is a function $h\colon \mathcal{G}_k \to (V \to \mathbb{R}^l)$ where for all k-attributed graphs $G = (V, E, X)$, $G' = (V, E, X')$ and all vertices $v \in V$ we have

$$(\forall v' \in \{v\} \cup \mathcal{N}(v). X(v') = X'(v')) \Rightarrow h(G)(v) = h(G')(v) \ .$$

Intuitively, we require that h only updates the attributes of a vertex v based on the attributes of its neighbors.

In this work we evaluate two well-established message passing layers that differ in the way neighbourhood information about nodes is updated. The first model uses a k,l-graph convolutional (GCN) layer, which was described by Kipf and Welling [20]:

$$h_{\mathbf{W}}((V, E, X))\colon v \mapsto \mathbf{W}(\sum_{v' \in \{v\}\cup\mathcal{N}(v)} ({X(v')}/{\sqrt{\deg(v)\deg(v')}})) \ ,$$

where $\mathbf{W}$ is a $l \times k$-matrix. The operator aggregates the attributes in the neighborhood of v and weights them by their degree.

In contrast, a k,l-graph attention layer (GAT) also aggregates the attributes in the neighborhood of v, but weight these attributes by so-called attention weights:

$$h_{\mathbf{W},\alpha_{v,v'}}((V, E, X))\colon v \mapsto \mathbf{W}(\sum_{v' \in \{v\}\cup\mathcal{N}(v)} \alpha_{v,v'} X(v')) \ ,$$

where $\mathbf{W}$ is again a $l \times k$-matrix and where $\alpha_{v,v'}$ is the attention score for the pair (v, v'). We omit the definition of the $\alpha_{v,v'}$ for brevity and refer the interested reader to work by Veličković, Cucurull, Casanova, et al. for details [18].

3 Method

We aim to compare the performance of the GCN layer and the GAT layer in predicting winning regions. Since both layers take attributed graphs as input and produce node labelings, we need to encode the problem of determining winning regions as a vertex-labeling problem. In particular, we need to encode the color as well as the owner of a vertex. Experience shows that the naïve encoding $x(v) = (\Omega(v), p)$ with $p = 0$ if $v \in V_0$ and $p = 1$ otherwise and using a $2,1$-message passing layer does not yield satisfactory results.

Instead, experience and preliminary evaluation leads us to the following architecture: Given a parity game $\mathcal{G} = (\mathcal{A}, \textsc{Parity}(\Omega))$ with $\mathcal{A} = (V, V_0, V_1, E)$ we define $G = (V, E, X_0)$ with $X_0(v) = (\Omega(v), x_0, x_1)$ where $x_i = 1$ if $v \in V_i$ and $x_i = 0$ otherwise. We then pass this 2-attributed graph to a stack of 10 message-passing layers, each of which yields a 256-attributed graph, i.e., for each vertex we obtain a vector of size 256. For each vertex, we pass this vector to a stack of neural network layers comprising a $256, 256$ linear layer, a $256, 256$ ReLU layer, a $256, 2$ linear layer, and a $2, 2$ softmax layer. Thus, for each vertex $v \in V$ we obtain a vector (x_0, x_1). We interpret this vector such that the architecture predicts $v \in W_0(\mathcal{G})$ if $x_0 > x_1$ and $v \in W_1(\mathcal{G})$ otherwise.

Recall that most layers in our architecture are parameterized. Hence, it remains to determine optimal parameters for these layers for the task of predicting winning regions. As with most machine learning tasks it is infeasible to determine these parameters analytically. Instead, we approximate them as follows: Let $\{\mathcal{G}_1,\ldots,\mathcal{G}_n\}$ be a set of parity games with arenas $\mathcal{A}_1,\ldots,\mathcal{A}_n$ and let $W_i^j(\mathcal{G})$ be the winning region of Player i in game j. For each arena $\mathcal{A}_j = (V, V_0, V_1, E)$ we construct the k-attributed graph $G_j = (V, E, X)$ with $X(v) = (\Omega(v), p_0, p_1)$ with $p_i = 1$ if $v \in V_i$ and $p_i = 0$ otherwise. Moreover, for each $\mathcal{G}_j$ we define the "target" vertex attributes $X'_j(v) = (w_0, w_1)$, where $w_i = 1$ if $v \in W_i(\mathcal{G}_j)$ and $w_i = 0$ otherwise. Using these pairs of inputs and desired outputs we approximate the optimal parameters of the architectures using the Adam algorithm [10]. For technical reasons, we additionally use an so-called Dropout layer [8] after the ReLU-layer. This improves the performance of the optimization, but does not change the function computed by the neural networks.

Having obtained approximations of the optimal parameters on the input data described above, we can apply either model to a previously unseen game $\mathcal{G} = (\mathcal{A}, \textsc{Parity}(\Omega))$ and obtain vertex-attributes $X'_V\colon V \to \mathbb{R}^2_+$. Given these attributes, we define the predicted winning regions $W_0^p(X'_V) = \{v \in V \mid X'_V(v) = (w_0, w_1), w_0 > w_1\}$ and $W_1^p = V \setminus W_0^p(X'_V)$.

The application of the model on $\mathcal{G}$ comprises the application of the ten message-passing layers as well as the application of each of the neural network layers for each vertex. Applying a single message-passing layer to takes constant time per vertex, i.e., linear time in the number of vertices of the arena. Similarly, the application of each neural network layer takes constant time. Hence, we are able to obtain the predicted winning regions in linear time in the number of vertices of the arena of $\mathcal{G}$.

4 Evaluation

To evaluate our approach we have developed a prototypical implementation [6] and generated 3 000 parity games and their solutions [7] using `PGSolver` [4, 5]. We sampled the

Table 1. Results of the evaluation. Acc. denotes the share of correctly identified vertices of the total number of vertices in the training set, while $n_{\text{err}} = 0$, $n_{\text{err}} = 1$, $n_{\text{err}} \geq 2$ denote the number of games with zero, one, and at least two misclassified vertices in the test set, respectively.

Model	Acc.	$n_{\text{err}} = 0$	$n_{\text{err}} = 1$	$n_{\text{err}} \geq 2$
GCN	0.60	537	245	118
GAT	0.98	433	279	188

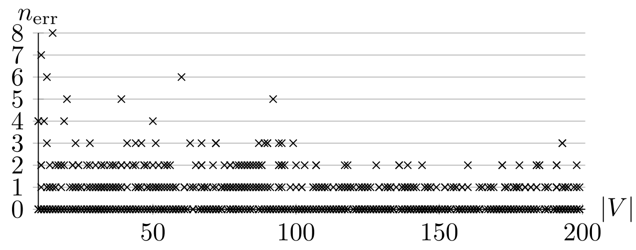

Figure 1. Misclassified vertices in relation to the size of the arena for GCN.

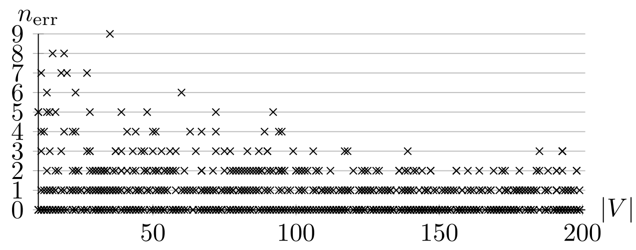

Figure 2. Misclassified vertices in relation to the size of the arena for GAT.

number of vertices uniformly from $\{10, \ldots, 200\}$. In each game with n vertices, for each vertex v, we sampled the number of outgoing edges and $\Omega(v)$ from $\left\{\frac{n}{100}, \ldots, \frac{n}{2}\right\}$ and $\{1, \ldots, n\}$, respectively. We trained both models on 2 100 of these games (the training set) and evaluated their performance on the remaining 900 games (the test set). We show the results in Table 1.

During training we observed that both models converge very fast, namely after processing the first 50 games. Our results show that GCN cannot characterize vertices in games it has not seen during training, it assigns vertices to the winning regions correctly for around 60% of previously unseen games. GAT, in contrast, shows better performance in this aspect. However, when it comes to solving entire games correctly, it only correctly predicts winning regions entirely for around half of previously unseen games. This in the first view contradicting results can be explained by examining the relation between the size of games and the number of incorrectly identified vertices in Figure 1 and Figure 2. GAT fails more often in small games compared to GCN but performes better on larger ones. Consequently, the overall accuracy on vertex level is better for GAT while solving less games entirely correctly than GCN. We leave further research on this, however, to future research.

5 Conclusion and Future Work

We have described a novel method for predicting the winning regions in parity games and we have evaluated a prototypical implementation of this method using randomly generated parity games. This evaluation showed that our method correctly identifies ~60% of the winning regions for Player 0 in our data set. We strongly believe that these results indicate the feasibility of our approach to determining the winner of parity games in practice. There are, however, open questions left for future research.

One direction of future work is to apply GNNs trained on smaller games to larger games. The correlation between game size and quality of prediction shown in Figure 1 and in Figure 2 indicates that we may be able to retain the prediction quality on larger games as well. If this is the case, it might be possible to use games of sizes that can be handled by exact solvers to create training data, which can then be used to train graph neural networks that determine winning regions of games of sizes intractable by classical methods.

Another avenue for future research concerns our architecture. In particular, the number of message-passing layers, hidden features, and a possible relationship with game size need further investigations.

Moreover, we are currently only predicting the winner of a parity game for a given vertex, but obtain no witness of this prediction. While the GNNs can easily be extended to produce edge attributes as well as node attributes, preliminary experiments show that the former cannot be used to construct winning strategies for the player predicted to win from a given vertex. In future work, we will look into predicting winning strategies as well as winning regions.

We also aim to evaluate our approach in a more realistic setting. This includes, e.g., solving parity games to construct controllers satisfying specifications given in LTL [11, 13].

Finally, in our current approach we only use a binary payoff function: Either Player 0 wins from a vertex, or she does not. In recent years more granular metrics for strategies have emerged. These include, e.g., evaluating the robustness of strategies against external disturbances [12] or the number of steps until an odd number is followed by a larger even number [14, 19]. In future work, we aim to use graph neural networks to evaluate vertices according to these metrics.

References

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani,

Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. (June 2018). arXiv:1806.01261 [cs.LG]

[2] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. 2020. Deciding Parity Games in Quasi-polynomial Time. *SIAM J. Comput.* 51, 2 (jan 2020), STOC17–152–STOC17–188. https://doi.org/10.1137/17m1145288

[3] John Fearnley. 2017. Efficient Parallel Strategy Improvement for Parity Games. In *Computer Aided Verification*. Springer International Publishing, 137–154. https://doi.org/10.1007/978-3-319-63390-9_8

[4] Oliver Friedmann and Martin Lange. 2009. Solving Parity Games in Practice. In *Automated Technology for Verification and Analysis*. Springer Berlin Heidelberg, 182–196. https://doi.org/10.1007/978-3-642-04761-9_15

[5] Oliver Friedmann and Martin Lange. 2014. *The PGSolver Collection of Parity Game Solvers*. Technical Report. Ludwig-Maximilians-Universität München.

[6] Tobias Hecking and Alexander Weinert. 2022. GNN Parity Game Solver 1.0.0. Zenodo. https://doi.org/10.5281/zenodo.7198473

[7] Tobias Hecking and Alexander Weinert. 2022. Randomly Generated Parity Games 1.0.0. Zenodo. https://doi.org/10.5281/zenodo.7128973

[8] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. (2012). https://doi.org/10.48550/ARXIV.1207.0580 arXiv:1207.0580

[9] Marcin Jurdziński. 1998. Deciding the winner in parity games is in UP ∩ co-UP. *Inform. Process. Lett.* 68, 3 (nov 1998), 119–124. https://doi.org/10.1016/s0020-0190(98)00150-1

[10] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR 2015*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[11] Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. 2019. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica* 57, 1-2 (nov 2019), 3–36. https://doi.org/10.1007/s00236-019-00349-3

[12] Daniel Neider, Alexander Weinert, and Martin Zimmermann. 2019. Synthesizing optimally resilient controllers. *Acta Informatica* 57, 1-2 (oct 2019), 195–221. https://doi.org/10.1007/s00236-019-00345-7

[13] A. Pnueli and R. Rosner. 1989. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '89*. ACM Press. https://doi.org/10.1145/75277.75293

[14] Sven Schewe, Alexander Weinert, and Martin Zimmermann. 2018. Parity Games with Weights. *Logical Methods in Computer Science, Volume 15, Issue 3 (August 23, 2019) lmcs:5705* (April 2018). https://doi.org/10.23638/LMCS-15(3:20)2019 arXiv:1804.06168 [cs.GT]

[15] Antonio Di Stasio, Aniello Murano, Vincenzo Prignano, and Loredana Sorrentino. 2021. Improving parity games in practice. *Annals of Mathematics and Artificial Intelligence* 89, 5-6 (jan 2021), 551–574. https://doi.org/10.1007/s10472-020-09721-3

[16] Wolfgang Thomas. 2002. Infinite Games and Verification. In *Computer Aided Verification*. Springer Berlin Heidelberg, 58–65. https://doi.org/10.1007/3-540-45657-0_5

[17] Tom van Dijk. 2018. Oink: An Implementation and Evaluation of Modern Parity Game Solvers. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer International Publishing, 291–308. https://doi.org/10.1007/978-3-319-89960-2_16

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[19] Alexander Weinert and Martin Zimmermann. 2016. Easy to Win, Hard to Master: Optimal Strategies in Parity Games with Costs. *Logical Methods in Computer Science, Volume 13, Issue 3 (September 19, 2017) lmcs:3938* (April 2016). https://doi.org/10.23638/LMCS-13(3:29)2017 arXiv:1604.05543 [cs.LO]

[20] Max Welling and Thomas N Kipf. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.