Analyzing sequential Graph Generation with Graph Convolutional Policy Networks

Ruxandra Lasowski

Furtwangen University
ruxandra.lasowski@hs-furtwangen.de

Abstract. One property of Graph Convolutional Neural Networks (GCN) is to be permutation equivariant with respect to the node ordering. When GCN are used in combination with Reinforcement Learning (RL) and the action space depends on the node labeling then the action space is not equivariant by default. In this work we empirically show on very small graphs that the Graph Convolutional Policy Network introduced in [1] cannot generalize to symmetries introduced by node permutations. We extend the method to deal with permutation symmetries by using representatives of an isomorphic class of valid constructed subgraphs for a desired graph structure.

Keywords: Graph Generative Method, Graph Convolutional Networks, Deep Reinforcement Learning, Permutation Symmetries

1 Introduction

Symmetries in the data are a challenge for deep learning systems if the underlying neural network architecture doesn't account for it. In [2] the authors unify deep learning architectures under the hood of geometric deep learning and show how those deal with symmetries inherent in our world. Moreover they emphasize the success of several architectures because they are able to deal with certain kind of symmetries. For example, graph convolutional neural networks are by design permutation equivariant when computing node embeddings. This property ensures that if the labeling is permuted then the node embeddings will permute in the same way. When the graph is represented by a diagram, it means that the embeddings will be attached to the same position in the graph regardless of the label/ordering. We illustrate the equivariance property on a path graph with three nodes in Fig. 1.

In a recent work [1] the authors propose a graph convolutional policy network for drug generation and optimization towards a desired chemical property. The method is able to start generating graph molecules from scratch, i.e. from one node, or to continue from a subgraph making the method very attractive also for other applications like mesh generation or mesh completion. But before trying to use the method in other applications we need to understand it's generalization capabilities. In this work we focus first on how this method will generalize to the symmetries introduced by the symmetric group S_n, i.e. all possible permutations of node labelings. All those graphs are isomorphic. That means that there is a bijection between the vertex sets of the involved graphs that preserves the adjacency of the vertices from one graph to the other. The permutations can be represented as matrices and the mapping between two isomorphic graphs G_1, G_2 can be written as $G_1 = P^T G_2 P$. We consider a reinforcement learning setting where the agent hasn't seen all possible permutations and our research question is whether the method is capable to account for them. The involved complexity is in general n factorial and therefore we analyze a toy problem which consists of the sequential generation of a 2×2 grid graph. As an introductory example lets consider that the agent is presented a path graph of four nodes as illustrated in Fig. 2. By taking into account the topology of a path graph, there are $4!/2 = 12$ distinct node labelings that the agent has to deal with. All those 4-node path graphs with different labeling represented as diagrams are isomorphic. In this representation the graphs look to us humans equivalent and we would connect the ends regardless of the labelings to obtain a 2×2 grid graph. This visual equivalence disappears also for humans when the graph is represented by an adjacency matrix or node embeddings (there are also graphs represented by diagrams that are isomorphic yet difficult to recognize as isomorphic by visual inspection). All those isomorphic path graphs have different adjacency matrices and also permuted node embeddings that are generated by a graph convolution. So for an agent in a reinforcement learning setting these are different states. This means that we need to ensure that either the action space is also equivariant or all equivalent states, i.e. isomorphic graphs, can be mapped to one state, i.e. a representative. Homomorphic Markov Decision Process Networks(Homomorphic MDP Networks) [3] achieve equivariance under actions. In the latter work the authors propose to construct equivariant layers in the input and action space by identifying the involved group structured symmetries. Equivariance is thus hardwired in the network architecture. This approach would work for our application for very small examples due to the involved complexity of n factorial. Path graphs and grid graphs do not have $n!$ distinct adjacency matrices however all those permutations that generate distinct

adjacency matrices do not form in general a subgroup of S_n because they are not closed under composition (see Appendix A for the number of distinct adjacency matrices of different graphs and the discussion related to the only subgroup of S_4 with 12 elements called A_4). Here we propose to use representatives of a valid subgraph that would lead to the construction of the desired grid graphs. Our approach shifts the complexity that is present on the algorithmic side by performing a graph isomorphism test between the constructed subgraph and representatives that we identify. We map then valid constructed subgraphs to those representatives. We use the algorithm that is implemented in NetworkX [4] that is based on the paper [5]. The graph isomorphism is a problem for which no polynomial algorithm is known. And it is still open if it is NP-complete. The latest achievement [6] in this regard showed that it can be solved in quasi-polynomial time. For certain classes of graphs like planar graphs, as are grid graphs, there exist polynomial time algorithms [7], [8]. Instead of using representatives of valid subgraphs we can cast the problem as subgraph isomorphism: Given a grid graph and a constructed subgraph test if the given grid graph contains a subgraph that is isomorphic to the constructed subgraph. Unlike graph isomorphism, subgraph isomorphism is proven to be an NP-complete problem.

Our contribution is to explore and go towards a direction that considers special classes of graphs where the intractable symmetry permutations on the nodes could be handled by a tractable graph isomorphism test.

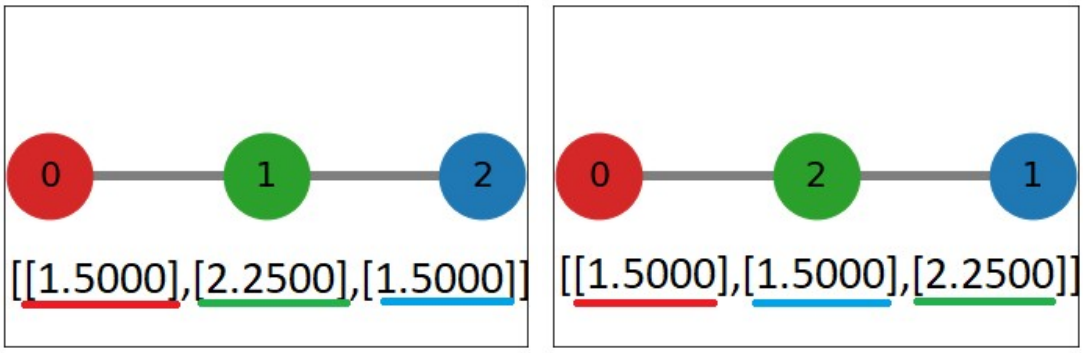

Fig. 1. Equivariance property illustrated on a path graph with three nodes. Nodes 1 and 2 are permuted. Node features are here scalar values. Visually, the node features (red, green, blue) stay at the same position in the diagram regardless of the labeling. The internal representation in terms of a vector will permute the values according to the node permutations. Notice that when only the topology is considered for the embeddings, the end nodes have the same embedding values.

2 Background

2.1 Deep Reinforcement Learning

In a reinforcement learning setting an agent learns how to act in an environment by receiving a positive or negative feedback called reward. The agent is trained such that it maximizes the cumulative reward. Formally, the environment is represented by a Markov Decision Process (MDP) which is a tuple $M = (S, A, R, T, \gamma)$, where $s \in S$ is a Markov state, $a \in A$ an action that an agent can take, $R : S \times A \to \mathbb{R}$ is a reward function that returns a scalar, γ is a discount factor to weigh recent rewards more than future rewards and $T : S \times A \times S \to [0, 1]$ is a transition function that assigns a probability for a pair of states and action of transitioning from the first to the second state. The goal of an agent is to find a policy $\pi : S \times A \to [0, 1]$, a function assigning probabilities for taking an action in a state, that maximizes the cumulative reward over an infinite time horizon. In deep reinforcement learning the policy is parametrized by a deep neural network.

2.2 MDP Homomorphism

In RL many tasks exhibit various types of redundancies and symmetries [9], [10] and researchers formalized these symmetries through the concept of Markov Decision Process Homomorphism (MDP Homomorphism) [10] that captures equivalent state-action pairs. Informally, two states are equivalent if for every action in the first state there is some valid possibly different action in the other state that produces similar results. Going back to our application, consider again two isomorphic 4-node path graphs $0 - 1 - 2 - 3$ and $1 - 0 - 3 - 2$. A valid action in the first graph is to connect node 0 and node 3 and the result is a 2×2 grid graph. A different action in the second graph

connecting node 1 and node 2 leads also to the construction of a 2×2 grid graph isomorphic to the first result. The goal of an MDP homomorphism is to find a minimal number of admissible state-action pairs, solve the problem in this reduced MDP and transfer the solution back to the original MDP. This is possible since under an MDP Homomorphism the equivalent state-action pairs share the same optimal Q-value and optimal value function [10]. When an MDP exhibits symmetries these can be treated as a special case of MDP homomorphism [10], [3]. In GCPN the action space is directly connected to the state space (see 3) so we are only interested in state equivalence (and not any more in equivalent state-action pairs). As the method is graph generation, the number of nodes and/or edges increases over time, so we need to identify for each number of nodes and edges valid subgraphs of the desired graph structure.

3 Graph Environment

We adopt the design of the environment as in [1]. The state of the MPD is represented by the adjacency matrix of the graph to be constructed. The agent can add a node with an id greater than the actual number of nodes and connect it to the actual graph or it can connect two nodes in the actual graph. During training it learns to respect the topology of a grid graph. Thus the action space consists of a pair denoting the node ids that we vectorize. The size is $maxnode * maxnode$ where maxnode is 4 for 2×2 grid graph. We note that for example if the state consists of a subgraph containing a node with id id then actions $[id, id1]$ and $[id, id2]$ with $id1 \neq id2$, $id1, id2 > id$ and $id1, id2 \notin$ subgraph then both actions map to the same state. This invariance, as we will see in the results section, has some impact on what actions an agent learns.

3.1 Reward design 2×2 grid graph

We adapt the reward design according to our 2×2 grid graph: a positive step reward is assigned for constructing a path graph and a final positive reward is assigned when the grid is constructed. A negative step reward is assigned if there is a deviation from a path graph, like a star graph, and a negative final reward is assigned for not connecting the ends of the path graph.

3.2 Learning

For training the agent we use the Proximal Policy Optimization (PPO)[11] algorithm as provided in [12]. The state, i.e. the adjacency matrix is used some agents to perform an one layer GCN and pass the node embeddings to the policy model. Input features for the GCN layer are vectors of ones for each node of the size of the nodes of the graph. The embeddings and policy are trained jointly. We train following agents:

- **Agent1** is trained for constructing a 2×2 grid graph where the state is represented as an adjacency matrix and is passed directly to PPO.
- **Agent2** is trained for constructing a 2×2 grid graph where an one layer GCN that computes 2-dimensional node embeddings is introduced and the features/node embeddings are passed to PPO.
- **Agent3** works like **Agent2** but states belonging to valid subgraphs are mapped to a representative. This state is going through the GCN layer and then passed to PPO.

The agents are trained with default hyperparameters.

What are the valid subgraph representatives? For the 2×2 grid graph, valid subgraphs are $2, 3, 4$-nodes path graphs.

4 Results

Our test scenarios for the agents are to build the desired grid graphs from one node and from any valid subgraph. We summarize the results in Table 1:

4.1 Results Agent1

Agent1 could construct a 2×2 grid graph starting from one node. When starting from a 4-node path graph with $4!/2 = 12$ states only 4 have been seen during training and only those could be reconstructed.

Table 1. Results table.

Agents	Grid construction from one node	Generalization to S_4 for path graphs
Agent1	1.0	0.33
Agent2	1.0	0.42
Agent3	1.0	1.0

4.2 Results Agent2

Agent2 could also construct a 2×2 grid graph starting from one node. This agent could identify one additional unseen state compared to Agent1 when starting from a 4-node path graph due to the graph convolutional layer that introduced an invariance induced by the topology of a path graph. Path graph $0-1-2-3$ has identical node embeddings with path graph $0-2-1-3$. We illustrate it in Fig. 2.

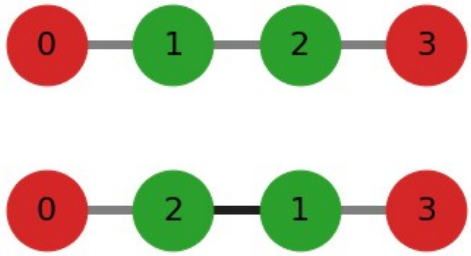

Fig. 2. Invariance introduced by GCN: 4-node path graphs with different adjacency matrices but equal embeddings. Upper graph: state that has been seen during training. Down graph: state that has not been seen during training. Node 1 and node 2 are permuted. Equal features or node embeddings are colored with the same color. The optimal action is to connect node 0 with node 3 to obtain a 2×2 grid graph.

4.3 Results Agent3

Agent3 learned to build the 2×2 grid graph with only one action $[0, 3]$ starting from a graph with one node, id = 0. This is because of the invariance that we mentioned in section 3 and the mapping of the valid path graphs to one representative. We lay out the sequence and we omit the action since it is the same:

1. Node id 1 and path graph $0-1$ is created.
2. Node id 2 and path graph $2-0-1$ is created.
3. Path graph is mapped to representative $0-1-2$.
4. Node id 3 and path graph $3-0-1-2$ is created.
5. Path graph is mapped to representative $0-1-2-3$.
6. Node id's 0 and 3 are connected and episode ends successfully.

When starting from a 4-node path graph all 12 states have been recognized due to the mapping to one representative of an isomorphic class.

5 Discussion and Conclusion

In this toy examples we have seen that in the context of RL also the action space has to be taken into account if the considered application needs to generalize to all equivalent state-action pairs that are not seen during training. We enumerated all valid subgraphs of equivalence classes that could occur during the sequential construction of the grid graphs. We think that even for the small graphs considered it was a better option to perform isomorphism graph tests than to incorporate the equivariance in the action space through the permutation group. For the 2×2 grid graph there were only 2 representatives to check against compared to 24 different permutations. In the analyzed method for Graph Convolutional Policy Networks [1] the authors used imitation learning and generated valid subgraphs of the desired distribution. This approach could be used to generate representatives of an equivalence class of a subgraph. This may not include all representatives but it would scale.

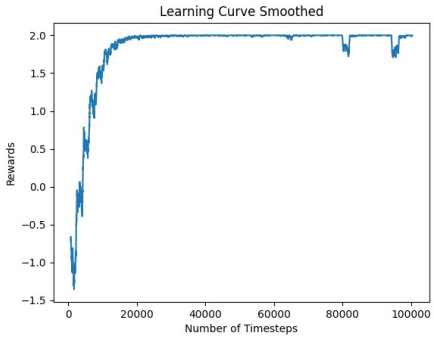

Fig. 3. Rewards for Agent1 training.

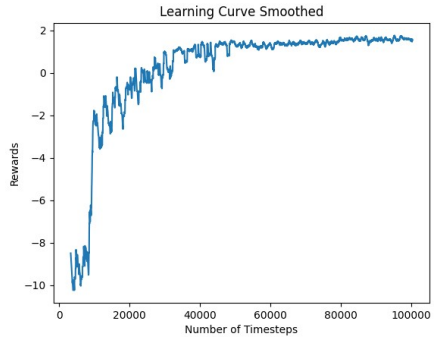

Fig. 4. Rewards for Agent2 training.

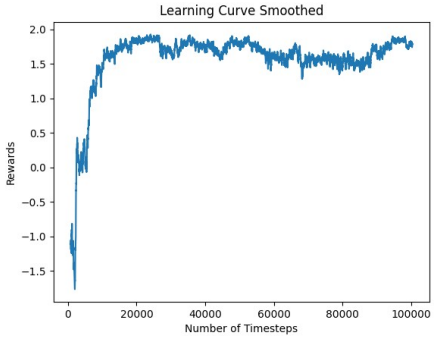

Fig. 5. Rewards for Agent3 training.

References

1. You, J., Liu, B., Ying, R., Pande, V.S., Leskovec, J.: Graph convolutional policy network for goal-directed molecular graph generation. CoRR **abs/1806.02473** (2018)
2. Bronstein, M.M., Bruna, J., Cohen, T., Velickovic, P.: Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. CoRR **abs/2104.13478** (2021)
3. van der Pol, E., Worrall, D.E., van Hoof, H., Oliehoek, F.A., Welling, M.: MDP homomorphic networks: Group symmetries in reinforcement learning. In: Advances in Neural Information Processing Systems. (2020)
4. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., Millman, J., eds.: Proceedings of the 7th Python in Science Conference, Pasadena, CA USA (2008) 11 – 15
5. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen. (2001) 149–159
6. Babai, L.: Graph isomorphism in quasipolynomial time [extended abstract]. In: Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing. STOC '16, New York, NY, USA, Association for Computing Machinery (2016) 684–697
7. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: STOC '74. (1974)
8. Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: Planar graph isomorphism is in log-space. In: 2009 24th Annual IEEE Conference on Computational Complexity. (2009) 203–214
9. Zinkevich, M., Balch, T.: Symmetry in markov decision processes and its implications for single agent and multi agent learning. In: In Proceedings of the 18th International Conference on Machine Learning, Morgan Kaufmann (2001) 632–640
10. Ravindran, B.: An Algebraic Approach to Abstraction in Reinforcement Learning. PhD thesis, University of Massachusetts, Amherst MA (2004)
11. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017)
12. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268) (2021) 1–8

A Appendix

Due to the symmetric topology of path and grid graphs, not all permutations of the labeling generate distinct adjacency matrices. The question for a graph with n nodes is: Do those restricted set of permutations form a subgroup of S_n? Group structure is required for hardwiring equivariant action spaces in Homomorphic MDP's [3]. By Lagrange theorem, the order of a subgroup divides the order of the group. We identified computationally that the number of distinct adjacency matrices that are generated by a restricted set of permutations is a divisor of the group order and the cardinality matches the order of the subgroup. However as we will see the restricted set does not always form a subgroup. The intermediate steps leading to a 2×2 grid graph are path graphs with $2, 3, 4$ nodes. Starting from 3 nodes, permutations of the labeling generate isomorphic graphs with different adjacency matrices. There are $n!/2$ permutations that generate distinct adjacency matrices of path graphs. For S_3 we can find $3!/2 = 3$ of them forming a subgroup: $\{(1, 2, 3), (2, 3, 1), (3, 1, 2)\}$. For S_4 there exists only one subgroup of order $4!/2 = 12$, the alternating group A_4, in cycle notation:

$$\{e, (12)(34), (13)(24), (14)(23), (123), (132), (124), (142), (134), (143), (234), (243)\}$$

The identity permutation $e = (1, 2, 3, 4)$ and $(14)(23) = (4, 3, 2, 1)$ generate the same adjacency matrix. The missing permutation to generate the 12-th distinct adjacency matrix would destroy the group structure by violating the closeness property.