

Learning Graph Algorithms With Recurrent Graph Neural Networks

Florian Grötschla*,¹ Joël Mathys*,¹ Roger Wattenhofer¹

¹ ETH Zurich

fgroetschla@ethz.ch, jmathys@ethz.ch, wattenhofer@ethz.ch

Abstract

Classical graph algorithms work well for combinatorial problems that can be thoroughly formalized and abstracted. Once the algorithm is derived, it generalizes to instances of any size. However, developing an algorithm that handles complex structures and interactions in the real world can be challenging. Rather than specifying the algorithm, we can try to learn it from the graph-structured data. Graph Neural Networks (GNNs) are inherently capable of working on graph structures; however, they struggle to generalize well, and learning on larger instances is challenging. In order to scale, we focus on a recurrent architecture design that can learn simple graph problems end to end on smaller graphs and then extrapolate to larger instances. As our main contribution, we identify three essential techniques for recurrent GNNs to scale. By using (i) skip connections, (ii) state regularization, and (iii) edge convolutions, we can guide GNNs toward extrapolation. This allows us to train on small graphs and apply the same model to much larger graphs during inference. Moreover, we empirically validate the extrapolation capabilities of our GNNs on algorithmic datasets.

Introduction

We believe that extrapolation is an important milestone for understanding machine learning. Being able to extrapolate to much larger inputs than seen in training is a convincing exhibit for a deeper understanding. For many learning architectures, this is a challenge, because the input size of the architecture is fixed. This is certainly true for multi-layer perceptrons, but even transformers have inputs bounded by a maximum number of tokens. A prominent exception are Graph Neural Networks (GNNs). In GNNs, each node is operating individually by merely exchanging messages with its neighbors. As such, in principle, GNNs can be trained on small graphs but then run on much larger graphs. Consequently, GNNs seem to be a natural match for studying extrapolation.

However, GNNs are usually just trained for a fixed and small number of message passing rounds. Nodes can learn a different behavior in each round, which gives them a lot of flexibility to learn even complicated functions. Nevertheless,

this flexibility also prevents GNNs from being able to handle problems where information has to be propagated through the whole graph, since these cannot be solved with a fixed number of rounds.

Instead, we want to mimic classical algorithms and be able to adapt the number of rounds a GNN can execute.

To do so, we use a recurrent GNN architecture to learn graph algorithms end-to-end. The core idea is to learn only on small graphs during training. Then, for inference, we can adapt the number of convolutions due to the recurrent design and apply more rounds for larger graphs. While a recurrent architecture seems to be necessary to achieve extrapolation, we show that it is not sufficient. In this paper, we propose measures to guide the model toward extrapolation and get a stable output from the network:

- We identify three techniques that lead GNNs towards more stable extrapolation: (i) skip connections to the problem input, (ii) state regularization through L2 loss, and (iii) edge convolutions.
- We show that using our approach, it is possible to extrapolate to larger graph instances, compare our models to existing baselines, and experimentally validate our findings.

Related Work

Graph Neural Networks: Initially proposed by Scarselli et al., GNNs have seen a significant popularity boost, and many new architectures have been established (Kipf and Welling 2016; Veličković et al. 2017). A key question regards the theoretical expressiveness of such models. It has been shown that the expressiveness of traditional GNNs is limited by the WL color refinement algorithm (Xu et al. 2018a; Leman and Weisfeiler 1968; Papp and Wattenhofer 2022). To achieve maximal expressiveness, message-passing GNNs have to use enough rounds to be able to match the computational power of WL. Moreover, increasing the number of rounds was proven to be necessary to solve graph problems (Loukas 2020). In fact, a GNN can probably not solve or, in some cases, even approximate a problem without executing the minimum amount of required rounds (Sato, Yamada, and Kashima 2019). Unfortunately, increasing the number of rounds has been shown to be difficult in practice. The training is more unstable and complex, leading to

*These authors contributed equally.

problems such as oversmoothing and oversquashing (Oono and Suzuki 2019; Alon and Yahav 2020). Therefore, most GNNs limit themselves to executing a constant number of rounds (Xu et al. 2018a).

Recurrence and Residual Connections: To incorporate more layers, residual connections and recurrent neural networks (RNN) have been proposed (Xu et al. 2018b; Huang and Carley 2019). Residual connections are a common technique for building deeper architectures (He et al. 2016). Recurrent neural networks are designed to work with variable length input and can keep internal state over long sequences, two examples being LSTMs (Hochreiter and Schmidhuber 1997) and GRUs (Cho et al. 2014). Following this line of work, several GNN architectures have included mechanisms such as gating, residual connections, or reusing the same layer to build deeper models (Li et al. 2021; Tang et al. 2020; Li et al. 2015; Huang and Carley 2019; Liu, Gao, and Ji 2020).

Algorithmic Learning and Extrapolation: The main aim of extrapolation is to solve problems not encountered during training. As such, the differentiable Neural Turing machine (Graves, Wayne, and Danihelka 2014) or RNNs which generalize to arbitrary input lengths (Gers and Schmidhuber 2001) have been proposed. One notable example is the work by Schwarzschild et al. that presents a recurrent architecture for Convolutional Neural Networks with residual connections. The model can then solve a series of tasks, including mazes, prefix sums, and chess problems. They show that executing more computation steps enables their networks to achieve excellent extrapolation capabilities on problem instances up to orders of magnitude beyond what was encountered during training.

GNNs have been used to tackle algorithmic problems before. Prominent examples include SAT and TSP or shortest paths through algorithmic alignment (Selsam et al. 2018; Veličković and Blundell 2021; Palm, Paquet, and Winther 2018; Joshi et al. 2022). Moreover, recent approaches also focus on extrapolation capabilities to larger graphs on algorithmic reasoning problems (Tang et al. 2020; Xu et al. 2020; Veličković et al. 2022; Ibarz et al. 2022). However, they only test on graphs slightly larger than those in the training set, resulting in architectures that have difficulty scaling to arbitrary sizes. Contrarily, our work focuses on architectures that can extrapolate to graphs up to 1000 times larger than the ones in the training set.

Model Architecture

The recurrent message passing layer is at the core of our architecture, which is illustrated in Figure 1. It is repeatedly executed on every node of the graph. Therefore, during training, we can use fewer layers and increase the number of layers for larger graphs during inference. The recurrent layer uses a skip connection to the original input, combining the current embedding h_v^t of a node v with the input features and transforming them to a node embedding through a multilayer perceptron (MLP). Then, the recurrent graph convolution is applied and uses one round of message passing to derive h_v^{t+1} . The computed embedding is then fed back to the recurrent layer to initiate the next round of computation.

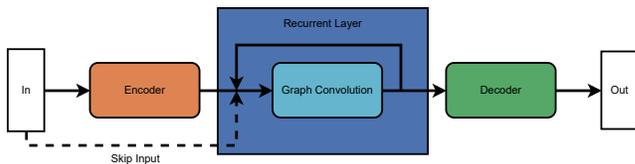


Figure 1: Overview of the recurrent architecture. The recurrent layer consists of a skip connection to the input, and the graph convolution is applied repeatedly. The encoder maps the input to the embedding dimension, while the decoder generates node predictions from the final embedding. The number of executions of the recurrent layer can be varied.

Furthermore, we use an encoder and decoder MLP to match the input and output dimensions to the recurrent layer’s node embedding dimension.

Graph Convolutions

In our work, we use two different graph convolutions: The GIN convolution (Xu et al. 2018a), a widely used message passing layer in GNNs, and a GRU convolution (Huang and Carley 2019), tailored towards the recurrent setting. The two convolutions allow us to compare to what extent a recurrent convolution alone is sufficient or if a more specialized convolution, such as the GRU, is required. Note that at every timestep, the GRU takes two inputs, the output of the convolution and its previous state. We use variations of both convolutions by adding an MLP on the edges before the aggregation. We refer to the versions without MLP as RecGIN and RecGRU and the ones with edge convolution as RecGIN-E and RecGRU-E. We define the RecGIN-E update as:

$$h_v^{t+1} = \Theta_1 \left((1 + \epsilon) \cdot h_v^t + \sum_{w \in N(v)} \Theta_2(h_v^t \| h_w^t) \right)$$

where $\|$ denotes concatenation and RecGRU-E is defined as:

$$h_v^{t+1} = \text{GRU} \left(\left(\sum_{w \in N(v)} \Theta(h_v^t \| h_w^t) \right), h_v^t \right)$$

The only difference to RecGIN and RecGRU is the additional MLP for the edges.

Extrapolation Techniques

The recurrent architecture allows us to vary the number of computation steps. This is not yet sufficient to achieve extrapolation. The predictions should also stabilize, so the GNN can use the information propagated through the additional layers without digressing. To guide the model toward extrapolation, we identify three essential techniques.

First, we introduce skip connections from the original input to the beginning of each recurrent layer as Schwarzschild et al. proposed. They observed that when the number of recurrent layers increases, the network tends to “forget” its initial features. By explicitly passing the input features to every execution of the recurrent layer, the network can recall its initial state and the problem it has to solve.

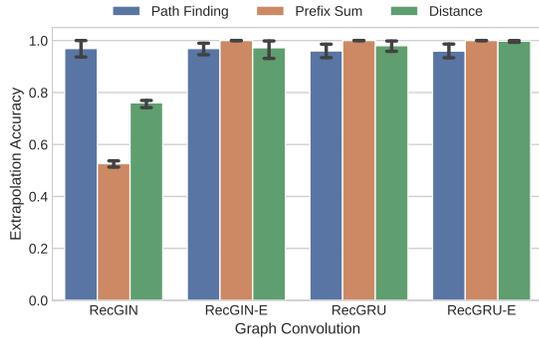


Figure 2: Comparison of extrapolation accuracy for different graph convolutions. Adapting the convolution on the edges or using a GRU are beneficial for extrapolation.

Another problem concerns the stability of computed embeddings. Once a solution is found by the GNN, executing more rounds should not change the prediction anymore. However, even slight deviations of the embedding in one layer can magnify in the following layers. We found that adding regularization in the form of an L2 loss on the embeddings improves the stability of the computation.

Finally, we add an MLP on pairs of node embeddings for every edge. The MLP allows the GNN to differentiate between neighbors and choose what messages to include in the aggregation step over all neighbors. This enables better control over information propagation through the graph and improves extrapolation capabilities.

Tasks

We test our model on multiple synthetic datasets that are specifically tailored to evaluate the ability of a GNN to gather and combine information over long distances.

Path Finding: Given a tree, predict if a node lies on the path between two marked nodes.

Prefix Sum: Paths are given where every node either has a one or zero as its initial feature. For each node, the sum of all initial features to its left modulo two has to be predicted.

Distance: Given a sparse graph with a marked starting node, for each node, predict the distance of the shortest path to the starting node modulo 2.

Experimental Evaluation

All models were trained for 100 epochs on graphs of size 10 using 12 instances of the recurrent layer. To evaluate extrapolation, we use graphs of size 100 and execute 120 rounds. For comparisons, we consider the model with the best loss on a validation dataset with graphs of size 10 that were not used for training. All values are averaged over 5 runs. The code to reproduce the experiments has been made available online ¹.

¹<https://github.com/floriangroetschla/Recurrent-GNNs-for-algorithm-learning>

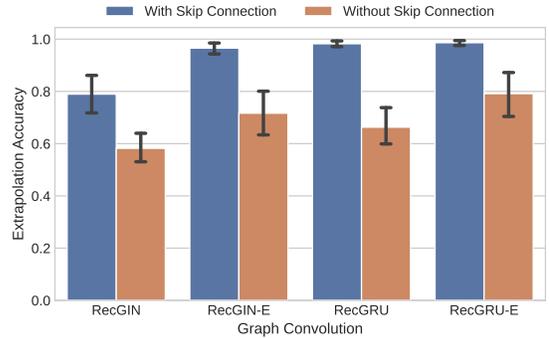


Figure 3: Extrapolation accuracy for each graph convolution for the Prefix Sum task. We hypothesize that the ability to recall the initial features is crucial for extrapolation.

Model Architecture

First, we compare the different graph convolutions and the effect of including an edge convolution. We train models with skip connections and regularization on graphs of size 10 and evaluate their accuracy on graphs of size 100. As illustrated in Figure 2, we observe that models using a GRU in the convolution perform better than the GIN variants. Adding the edge convolution results in better generalization accuracy overall, although only marginally for RecGRU. Models that use the edge convolution can reach perfect accuracy, while this is not the case for RecGIN. Therefore, we conclude that the MLP on edges is helpful, albeit not completely necessary for all convolutions.

Regarding the use of skip connections, we can observe a meaningful difference, shown exemplarily on the prefix task in Figure 3. Even models that use edge convolutions are not able to reach good accuracy without them. This confirms the findings of Schwarzschild et al., that access to the input features is a crucial part of gaining extrapolation abilities. We conclude that adding the skip connection to the inputs is also an excellent tool for extrapolation for GNNs.

Stabilization and Extrapolation

Before, we tested extrapolation to graphs of size 100. We want to extend this to even larger graphs. In order for a model to extrapolate, it also needs to be able to stabilize predictions when many convolutions are applied. We evaluate the stabilization on graphs of size 10 and then increase the number of layers from 10 up to 10,000. Figure 4 shows the effect of using L2 regularization for the distance task. Within the first 10 rounds, the accuracy increases as the information can propagate further through the graph. Then, both versions output correct predictions until approximately 100 layers. Afterwards, the models trained with L2 regularization still stay stable while accuracy declines rapidly without regularization. We hypothesize that slight deviations in the node embeddings can lead to the performance degradation over time. The additional regularization loss incentivizes the model to keep embeddings compact. We conclude that without regularization, the model can still exhibit extrapolation

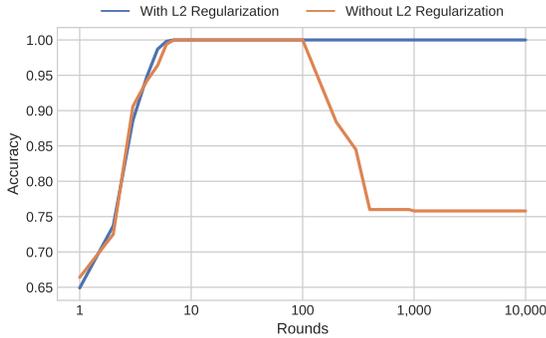


Figure 4: Accuracy for the Distance task on graphs of size 10 for RecGRU-E over the number of rounds the GNN executes. Initially, the accuracy increases as information flows through the graph. Then, predictions stabilize for models using regularization while it decreases for models without it.

Model	$n = 10$	$n = 50$	$n = 100$	$n = 1,000$	$n = 10,000$	
Path Finding	GIN	1.00 ± 0.00	0.70 ± 0.10	0.52 ± 0.11	0.22 ± 0.07	0.08 ± 0.04
	IterGNN	0.86 ± 0.09	0.51 ± 0.12	0.41 ± 0.08	0.15 ± 0.06	0.06 ± 0.02
	RecGIN-E	1.00 ± 0.00	0.91 ± 0.04	0.88 ± 0.09	0.66 ± 0.26	0.33 ± 0.31
	RecGRU-E	1.00 ± 0.00	0.96 ± 0.09	0.92 ± 0.16	0.86 ± 0.30	0.81 ± 0.39
Prefix Sum	GIN	0.95 ± 0.12	0.54 ± 0.11	0.48 ± 0.16	0.41 ± 0.22	0.40 ± 0.22
	IterGNN	1.00 ± 0.00	0.58 ± 0.11	0.51 ± 0.17	0.40 ± 0.25	0.40 ± 0.26
	RecGIN-E	1.00 ± 0.00				
	RecGRU-E	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.03
Distance	GIN	1.00 ± 0.00	0.89 ± 0.05	0.83 ± 0.08	0.59 ± 0.20	0.52 ± 0.23
	IterGNN	1.00 ± 0.00	0.99 ± 0.01	0.98 ± 0.04	0.81 ± 0.05	0.57 ± 0.17
	RecGIN-E	1.00 ± 0.00	0.98 ± 0.02	0.91 ± 0.04	0.74 ± 0.07	0.57 ± 0.08
	RecGRU-E	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.02	0.97 ± 0.04

Table 1: We report the F1 score for the evaluation of extrapolation abilities. All models were only trained on graphs of size 10. The non-recurrent GIN can not extrapolate. The RecGRU-E outperforms IterGNN and can extrapolate well to graphs that are 1,000 times larger than the graphs encountered during training.

to some degree. However, for further extrapolation, it is essential to use regularization.

Lastly, we evaluate our model’s extrapolation capabilities by testing it on even larger graphs and comparing it against existing baselines as illustrated in Table 1. All listed models were trained on graphs of size 10. We compare against a non-recurrent GIN baseline model that uses 10 separate GIN convolutions with skip connections and was trained with L2 regularization. As the architecture is not recurrent, it always uses exactly 10 rounds of message passing. We observe that GIN can still solve the task on graph sizes it has encountered during training. However, even for slightly larger graphs, it struggles to extrapolate as it can not propagate the relevant information far enough to solve the task. Furthermore, we compare our models to IterGNN (Tang et al. 2020). IterGNN is also based on a recurrent architecture and can vary the number of iterations. In addition, it learns a stopping criterion. Even though it can extrapolate to some degree, it can not extend to very large graphs. Both the RecGIN-E and RecGRU-E can extrapolate to graphs that are up to 1000 times larger than the graphs encountered during training, with the GRU variant achieving the best results overall.

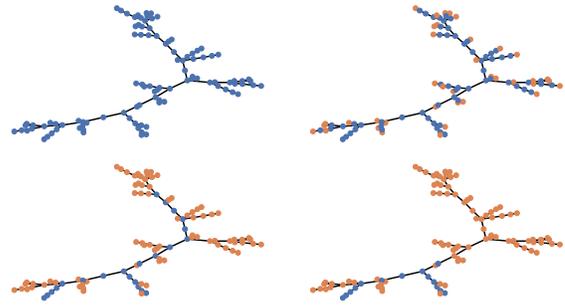


Figure 5: Development of the predictions for the Path Finding task after executing more rounds, from top left to bottom right (row by row). The GNN scales to different graph sizes by successively removing nodes that are not part of the path.

Learned Algorithms

To further evaluate to what extent our model could learn algorithmic behavior, we plot predictions after several iterations of the recurrent layer for the Path Finding task. By visually inspecting the outputs, we can observe that the GNN uses a dead-end-filling algorithm that “removes” nodes that can not be part of the path until it reaches the path between the endpoints. The algorithm is correct and scales to arbitrary graph sizes. Similarly, on the Distance task, the GNN emulates a Breadth First Search while information for the total sum is propagated from left to right for the Prefix Sum task.

Conclusion

Classic graph algorithms can scale to any graph size by allowing the algorithm to execute more computation steps for larger instances. We want to mimic this ability by training on small graphs and then extrapolate to larger instances. Traditional GNN architectures can only execute a fixed number of rounds. To make up for this shortcoming, we use a recurrent architecture that can vary this number.

In addition to recurrence, we take additional measures to make our models scale. Skip connections to the input features, the addition of MLPs on edges and the regularization of node embeddings help GNNs to extrapolate well. With our approach, we can train on small graph instances and extrapolate to graphs outside the training regime that are orders of magnitudes larger. We evaluate our models on algorithmic datasets and compare them to existing baselines. Using the techniques above, our model can extrapolate to graphs 1,000 times larger than during training. Moreover, the computed solutions stabilize and do not change after executing more rounds.

While we indicate essential tools to help extrapolation, more work is required toward algorithmic learning on graphs. Possible directions include discrete embeddings to imitate state machine or symbolic algorithms that go beyond just defining state transitions. This could lead to interpretable algorithmic behavior and close the gap between human and artificial decision-making.

References

- Alon, U.; and Yahav, E. 2020. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*.
- Cho, K.; Van Merriënboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
- Gers, F. A.; and Schmidhuber, J. 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6): 1333–1340.
- Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural Turing machines. *arXiv preprint arXiv:1410.5401*.
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.
- Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.
- Huang, B.; and Carley, K. M. 2019. Residual or gate? towards deeper graph neural networks for inductive graph representation learning. *arXiv preprint arXiv:1904.08035*.
- Ibarz, B.; Kurin, V.; Papamakarios, G.; Nikiforou, K.; Ben-nani, M.; Csordás, R.; Dudzik, A.; Bošnjak, M.; Vitvitskiy, A.; Rubanova, Y.; et al. 2022. A Generalist Neural Algorithmic Learner. *arXiv preprint arXiv:2209.11142*.
- Joshi, C. K.; Cappart, Q.; Rousseau, L.-M.; and Laurent, T. 2022. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 1–29.
- Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- Leman, A.; and Weisfeiler, B. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tekhnicheskaya Informatsiya*, 2(9): 12–16.
- Li, G.; Müller, M.; Ghanem, B.; and Koltun, V. 2021. Training graph neural networks with 1000 layers. In *International conference on machine learning*, 6437–6449. PMLR.
- Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.
- Liu, M.; Gao, H.; and Ji, S. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 338–348.
- Loukas, A. 2020. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*.
- Oono, K.; and Suzuki, T. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*.
- Palm, R.; Paquet, U.; and Winther, O. 2018. Recurrent relational networks. *Advances in Neural Information Processing Systems*, 31.
- Papp, P. A.; and Wattenhofer, R. 2022. A Theoretical Comparison of Graph Neural Network Extensions. *arXiv preprint arXiv:2201.12884*.
- Sato, R.; Yamada, M.; and Kashima, H. 2019. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32.
- Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.
- Schwarzschild, A.; Borgnia, E.; Gupta, A.; Huang, F.; Vishkin, U.; Goldblum, M.; and Goldstein, T. 2021. Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks. In *Advances in Neural Information Processing Systems*, volume 34, 6695–6706. Curran Associates, Inc.
- Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2018. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.
- Tang, H.; Huang, Z.; Gu, J.; Lu, B.-L.; and Su, H. 2020. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33: 15811–15822.
- Veličković, P.; Badia, A. P.; Budden, D.; Pascanu, R.; Bano, A.; Dashevskiy, M.; Hadsell, R.; and Blundell, C. 2022. The CLRS Algorithmic Reasoning Benchmark. *arXiv preprint arXiv:2205.15659*.
- Veličković, P.; and Blundell, C. 2021. Neural algorithmic reasoning. *Patterns*, 2(7): 100273.
- Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018a. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.
- Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018b. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 5453–5462. PMLR.
- Xu, K.; Zhang, M.; Li, J.; Du, S. S.; Kawarabayashi, K.-i.; and Jegelka, S. 2020. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*.