SALSA-CLRS: A Sparse and Scalable Benchmark for Algorithmic Reasoning

Julian Minder
ETH Zurich
jminder@ethz.ch

Florian Grötschla*
ETH Zurich
fgroetschla@ethz.ch

Joël Mathys*
ETH Zurich
jmathys@ethz.ch

Roger Wattenhofer
ETH Zurich
wattenhofer@ethz.ch

Abstract

We introduce an extension to the CLRS algorithmic learning benchmark, prioritizing scalability and the utilization of sparse representations. Many algorithms in CLRS require global memory or information exchange, mirrored in its execution model, which constructs fully connected (*not sparse*) graphs based on the underlying problem. Despite CLRS's aim of assessing how effectively learned algorithms can generalize to larger instances, the existing execution model becomes a significant constraint due to its demanding memory requirements and runtime (*hard to scale*). However, many important algorithms do not demand a fully connected graph; these algorithms, primarily distributed in nature, align closely with the message-passing paradigm employed by Graph Neural Networks. Hence, we propose SALSA-CLRS, an extension of the current CLRS benchmark specifically with *scalability* and *sparseness* in mind. Our approach includes adapted algorithms from the original CLRS benchmark and introduces new problems from distributed and randomized algorithms. Moreover, we perform a thorough empirical evaluation of our benchmark. Code is publicly available at https://github.com/jkminder/salsa-clrs.

1 Introduction

Neural algorithmic reasoning combines the learning power of neural networks with the principles of algorithmic thinking. This fusion aims to promote logical reasoning and the ability to extrapolate. This is widely considered a weak spot for neural methods. Algorithms take various shapes and deal with sets, strings, images, or geometry. Several prominent and beautiful algorithms are concerned with graphs and networks. Graph algorithms usually take up a significant portion of algorithmic textbooks such as the CLRS textbook [1] and the CLRS benchmark [2], which is based on that textbook. Interestingly, the CLRS benchmark translates *every* algorithmic problem into a common graph-based format. This approach yields the significant advantage of utilizing a single architecture across various scenarios. However, the emphasis on algorithmic diversity and unification in CLRS introduces significant constraints that hinder scalability.

The CLRS-30 dataset contains 30 algorithms operating within a centralized execution model that facilitates global information exchange, which is essential for numerous algorithms. This global information exchange is enabled by enforcing all problems to operate on a complete graph – each node can communicate with every other node, resulting in quadratic communication costs. To maintain information on the original topology, the CLRS framework augments these complete graphs with flags on each edge to indicate whether the edge exists in the input. This strategy has several limitations. While CLRS highlights its proficiency in assessing *out-of-distribution* (OOD) capabilities, the reliance on a fully connected graph execution model imposes significant memory and computation constraints. This challenge is particularly pronounced as *graph* algorithms are often designed with sparse graphs in mind [1].

*Equal contribution.

J. Minder et al., SALSA-CLRS: A Sparse and Scalable Benchmark for Algorithmic Reasoning (Extended Abstract). Presented at the Second Learning on Graphs Conference (LoG 2023), Virtual Event, November 27–30, 2023.

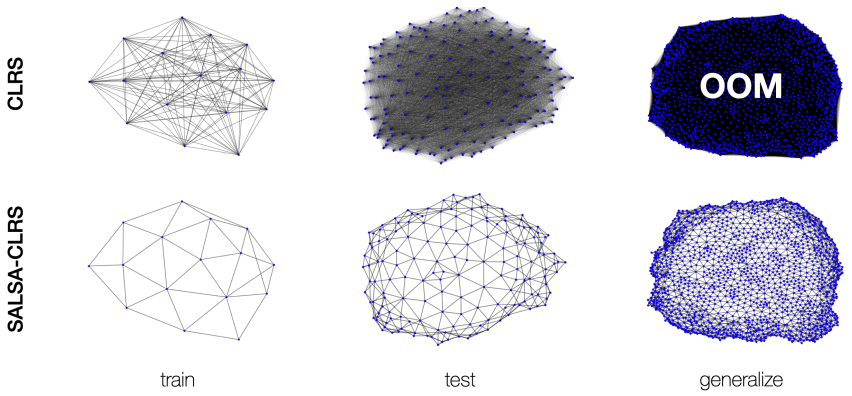

Figure 1: A visualization of the difference between the graph representation in SALSA-CLRS and CLRS. When dealing with large complete graphs, the memory demands become exceedingly impractical, leading to occurrences of *Out-Of-Memory* (OOM) errors. SALSA-CLRS enables the evaluation of scalable architectures on graphs up to 100 times the size of the training graphs.

Furthermore, when learning algorithms that guarantee correctness for any input size, evaluating models across a diverse range of large-scale inputs is crucial, as many studies have highlighted [3–8]. Apart from considering large-scale test graphs, relying solely on a single graph generation mechanism can yield false conclusions about OOD performances [7]. The CLRS library in principle allows more flexibility and a custom generation. However, the default CLRS-30 dataset used for benchmarking provides OOD test graphs, limited to only four times the size of the training graphs, and both training and test graphs stem from the same graph generation mechanism. While under the CLRS execution model, moderately larger graphs (10x) might still be feasible on modern hardware, much larger graphs – in the order of 100-fold scaling – become impossible to run due to their demanding memory requirements (Figure 1).

To address these challenges, we propose a more concise strategy. We focus solely on *graph* algorithms, which can follow a distributed execution model, thus reducing reliance on global memory and information flow. This allows a transition to a sparse execution model. Furthermore, building upon the findings presented in [6], which underscore the superior learning and OOD performance of parallelized algorithms compared to their sequential counterparts, we also emphasize the importance of encompassing problems from the realm of distributed and randomized algorithms. Towards this end, we introduce SALSA-CLRS, a **S**parse **A**lgorithmic **L**earning benchmark for **S**calable **A**rchitectures. Extending CLRS, our benchmark *i)* leverages a sparse execution mode to enable OOD test sets that cover graphs 100 times the size of training sets, *ii)* adds new graph generators for sparse and diverse graphs, thus enabling a more thorough OOD evaluation and *iii)* incorporates distributed and randomized algorithms that align more closely with the execution models used by Graph Neural Networks (GNNs).

2 SALSA-CLRS Benchmark

The SALSA-CLRS Benchmark follows the structure of CLRS [2]. Each data point comprises a graph with n nodes and an algorithm trajectory. Each trajectory comprises a set of input, intermediate, and output features. Specifically, the input features capture the input state of the algorithm, along with positional identifiers for nodes to resolve tie-breaking scenarios. The intermediate features, referred to as hints, correspond to interim values of algorithm variables. These hints provide insight into the algorithm's inner workings and act as a means to encourage models to adhere closely to the algorithm's execution. It is worth noting that execution without hints is possible and may even be beneficial, as demonstrated in Section 3. Lastly, the output features directly relate to the solution of the given problem. Moreover, each data point contains a trajectory length, defining the number of steps required to solve the algorithm. Every feature is associated with a location – either a *node*, an *edge*, or the entire *graph* – and possesses a corresponding type. SALSA-CLRS provides both pre-defined train-validation-test splits, facilitating model comparison and the capability to generate new data tailored to individual requirements. Beyond what CLRS-30 offers, SALSA-CLRS comes

with diverse graph types to explore OOD capabilities further. For comprehensive information, see Appendix A.3.2. The benchmark is implemented in PyG [9] and built with extendability in mind.

2.1 Algorithms

SALSA-CLRS encompasses a set of six algorithms, adapting four from the original CLRS paper and introducing two novel additions from the field of distributed and randomized algorithms. The four CLRS algorithms were selected to ensure the representation of input, hint, and output features on the sparse graph: Breadth-first search (BFS), Depth-first search (DFS), Dijkstra, and Maximum Spanning Tree (MST). Please refer to Appendix A.2 for more details. While the algorithms introduced by CLRS-30 are inspired by sequential algorithms in the CLRS textbook, although in some cases heavily parallelized, the message-passing paradigm – essentially the driving mechanism behind GNNs – aligns closely with distributed computing principles. To encompass this perspective, we extend our benchmark by introducing two new distributed algorithms, drawn from *Mastering Distributed Algorithms* [10]. Numerous distributed algorithms incorporate randomness as a crucial component of their computation. In light of this, we enhance the CLRS framework by including the concept of randomness. In cases where an algorithm necessitates randomness, we precompute random values and treat them as regular input to the algorithm. We introduce two new algorithms: Distributed Maximal Independent Set (MIS) and Distributed Eccentricity. A description of both can be found in Appendix A.2.

2.2 Graph Types

Building upon investigations [7] of CLRS and different graph types, we enrich the diversity of graph types compared to CLRS-30. While CLRS-30 works exclusively on Erdös-Renyi (ER) random graphs, the study by Mahdavi et al. [7] underscores the limitation of relying solely on ER graphs to assess the OOD capabilities of architectures. Recognizing this, we propose that broadening the spectrum of graph types is pivotal for a more comprehensive OOD evaluation. SALSA-CLRS comes with three distinct graph generation mechanisms: Erdös-Renyi graphs (ER) [11], Watts Strogatz graphs (WS) [12] — and Delaunay Graphs. In contrast to CLRS-30, we reduce the ER edge probability to just above the minimum to maintain graph connectivity. WS graphs belong to the category of small-world graphs and exhibit a low clustering coefficient [13]. While still sparse, WS graphs show a very different structure. Delaunay graphs are planar and hence inherently sparse. We refer to Appendix A.3.1 for associated graph parameters.

3 Empirical Evaluation

In this section, we undertake an empirical evaluation by comparing three baseline models. Our analysis involves a comparison of training scenarios with and without hints, followed by comprehensive testing across all SALSA-CLRS test sets. This evaluation sheds light on deficiencies in the models on OOD test sets and therefore affirms the importance of the SALSA-CLRS benchmark.

Architectures. We use the same Encode-Process-Decode [14] from CLRS, but propose a slight simplification. We omit the re-encoding of decoded hints to update the node hidden states. This results in a simplification of the computational graph, making the architectures more scalable. We compare three baseline processors, a GRU [15] adapted GIN[2] module [17], RecGNN [3], a recurrent message-passing GNN for algorithmic tasks and PGN [18], which has shown promising performance on the original CLRS benchmark. All architectures incorporate skip connections, implemented by forwarding the encoded input and the two most recent hidden states to the processor. This mechanism aids in mitigating vanishing gradient [19] issues. For a comprehensive overview of the Encode-Process-Decode architecture, our proposed changes, and the baselines, please see Appendix A.4.1.

Experiments. Each baseline model is trained for each algorithm with and without the inclusion of hints. Every run is confined to 100 epochs with early stopping. The batch size is eight graphs. All reported values are means over five runs. For more details on the metrics and the experiments, see Appendix A.4.

[2]Dijkstra and MST require edge weights, so we use GINE [16].

Table 1: Scores for both models on all algorithms, reported as percentages. The used metric for algorithms is denoted under the algorithm name. Models are trained only on ER graphs of size up to $n = 16$ without hints (first column) and evaluated on larger sizes and different distributions (other columns).

Graph Type		ER					WS					Delaunay				
n		16	80	160	800	1600	16	80	160	800	1600	16	80	160	800	1600
BFS	GIN(E)	100.0	99.6	99.3	98.0	98.0	99.9	92.9	86.7	70.4	75.3	100.0	94.3	84.6	52.7	45.9
(Node Acc.)	PGN	100.0	99.8	99.5	99.0	98.9	100.0	95.5	88.7	75.9	80.6	100.0	98.2	90.4	53.6	40.3
	RecGNN	100.0	99.8	99.5	99.3	99.2	100.0	97.8	94.2	82.2	82.1	100.0	98.5	92.0	67.1	55.6
DFS	GIN(E)	49.3	30.6	19.7	18.1	16.5	29.7	15.9	16.8	22.3	20.1	46.7	28.0	25.1	23.4	23.2
(Node Acc.)	PGN	74.2	41.2	29.9	27.8	25.8	58.8	17.9	17.7	23.6	21.3	72.7	41.7	38.2	35.8	35.4
	RecGNN	33.4	28.0	18.7	18.2	16.8	22.7	15.9	16.8	21.5	19.5	32.3	26.8	25.2	24.1	24.0
Dijkstra	GIN(E)	98.0	89.8	84.3	75.8	72.8	95.4	85.0	79.9	61.4	52.6	97.4	81.6	70.4	46.5	39.9
(Node Acc.)	PGN	99.6	98.6	97.2	94.1	92.2	98.3	97.1	95.4	81.8	72.5	99.5	97.6	92.4	62.7	51.0
	RecGNN	98.5	86.8	76.0	63.7	60.6	95.8	89.2	83.9	71.4	67.3	98.0	90.4	85.0	60.2	50.0
Eccentricity	GIN(E)	57.3	77.1	72.3	51.3	36.7	78.0	27.6	3.6	0.0	0.0	84.8	0.0	0.0	0.0	0.0
(Graph Acc.)	PGN	100.0	100.0	100.0	100.0	64.6	100.0	93.8	100.0	25.6	5.2	100.0	100.0	76.9	0.0	0.0
	RecGNN	75.8	80.5	75.0	72.7	63.0	86.7	60.8	57.4	27.6	15.2	89.9	25.2	8.3	0.0	0.0
MIS	GIN(E)	61.2	48.2	51.7	29.5	41.1	57.5	63.5	61.7	52.7	53.2	62.7	60.5	58.1	56.5	55.2
(Node F1)	PGN	99.6	99.1	98.9	96.4	97.3	99.4	99.0	97.7	90.8	86.1	99.8	99.6	99.6	99.1	98.6
	RecGNN	87.7	76.5	78.5	61.8	70.6	84.0	85.6	83.9	77.5	77.9	89.3	86.5	85.5	84.3	83.4
MST	GIN(E)	92.6	79.1	77.6	74.5	72.9	89.6	75.3	74.4	73.0	72.8	92.8	77.4	75.8	74.8	74.7
(Node Acc.)	PGN	97.3	89.1	84.6	75.7	71.9	96.8	82.5	77.6	67.4	65.1	97.4	85.2	78.5	68.7	66.8
	RecGNN	94.2	70.7	66.6	58.9	56.0	92.8	67.4	62.8	53.5	52.5	94.7	69.9	62.6	52.5	50.6

3.1 Evaluation

In Table 1, we showcase the performance of two baseline models on all SALSA-CLRS algorithms. Note, as we increase the graph size, all models show a clear decline in performance. Furthermore, we observe significant performance disparities among different graph types. Remarkably, different algorithms show varying degrees of sensitivity to different graph types. For example, BFS shows stability when applied to larger ER graphs, but its performance drops on large Delaunay graphs. DFS shows the opposite behaviour. Similarly, the architectures show sensitivity to algorithms. For example, RecGNN shows the best extrapolation performance on BFS, while PGN is clearly the best on MIS. In general, the PGN model is often the best performer, in particular for DFS and Eccentricity, and for MIS we even see a very strong performance up to the largest graph sizes. It is worth mentioning that, consistent with previous findings [7], the incorporation of hints does not lead to performance improvements across the board (see Tables 2 and 3). More details can be found in Appendix A.4. It is important to emphasize the pivotal role of metrics selection. An example: Despite seemingly excellent Node Accuracy scores of both baselines on BFS, the graph accuracy shows a completely different picture (see Table 4). For larger graph instances, almost all graphs are predicted incorrectly, despite achieving a near-perfect Node Accuracy. These findings underscore SALSA-CLRS's effectiveness in comprehensively evaluating architectural vulnerabilities in terms of both scalability and graph diversity.

4 Conclusion

As traditional algorithms are invariant to input size, scalability and extrapolation are important when evaluating learned algorithmic reasoning models. Thus, we introduce SALSA-CLRS, an extension to CLRS designed for scalable architectures and sparse graph representations. Addressing the limitations of the original CLRS benchmark, SALSA-CLRS focuses on graph problems that align with distributed execution models. This orientation fosters scalability and improved assessment of generalization capabilities, particularly for larger graph instances. In addition to four CLRS problems, SALSA-CLRS incorporates two additional algorithms rooted in distributed and randomized paradigms. By including diverse *out-of-distribution* (OOD) test sets, which involve graphs up to 100 times the scale of the training set and encompass various graph types, our empirical evaluation underscores the critical role of such extrapolation for a comprehensive assessment of algorithmic reasoning. These OOD tests unveil several limitations that might remain concealed when examined solely within the CLRS dataset's confines. SALSA-CLRS serves as a tool for advancing Neural Algorithmic Reasoning, facilitating the evaluation of scalable architectures on sparse graphs.

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022. 1

[2] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR, 2022. 1, 2, 8

[3] Florian Grötschla, Joël Mathys, and Roger Wattenhofer. Learning graph algorithms with recurrent graph neural networks. 2022. URL https://arxiv.org/abs/2212.04934. 2, 3, 8, 11

[4] Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020. 8

[5] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *arXiv preprint arXiv:2202.05826*, 2022.

[6] Valerie Engelmayer, Dobrik Georgiev, and Petar Veličković. Parallel algorithms align with neural execution. *arXiv preprint arXiv:2307.04049*, 2023. 2, 8

[7] Sadegh Mahdavi, Kevin Swersky, Thomas Kipf, Milad Hashemi, Christos Thrampoulidis, and Renjie Liao. Towards better out-of-distribution generalization of neural algorithmic reasoning tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=xkrtvHlp3P. 2, 3, 4, 8

[8] Beatrice Bevilacqua, Kyriacos Nikiforou, Borja Ibarz, Ioana Bica, Michela Paganini, Charles Blundell, Jovana Mitrovic, and Petar Veličković. Neural algorithmic reasoning with causal regularisation. *arXiv preprint arXiv:2302.10258*, 2023. 2, 8

[9] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, May 2019. URL https://github.com/pyg-team/pytorch_geometric. 3

[10] Roger Wattenhofer. Mastering distributed algorithms, 2020. 3, 8, 9

[11] Paul Erdős and Alfréd Rényi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959. 3, 9

[12] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998. 3, 10

[13] Marc Barthélémy and Luis A Nunes Amaral. Small-world networks: Evidence for a crossover picture. *Physical Review Letters*, 82(15):3180, 1999. 3, 10

[14] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 3, 11

[15] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. 3, 11

[16] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019. 3

[17] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 3

[18] Petar Veličković, Lars Buesing, Matthew Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. *Advances in Neural Information Processing Systems*, 33: 2232–2244, 2020. 3, 11

[19] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998. 3

[20] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 8

[21] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE transactions on neural networks*, 12(6):1333–1340, 2001. 8

[22] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021. 8

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 8

[24] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018. 8

[25] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021.

[26] Rasmus Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *Advances in neural information processing systems*, 31, 2018.

[27] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020. 8

[28] Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. *Advances in Neural Information Processing Systems*, 35:20635–20647, 2022. 8

[29] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019. 8

[30] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020. 8

[31] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner. In Bastian Rieck and Razvan Pascanu, editors, *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pages 2:1–2:23. PMLR, 09–12 Dec 2022. URL https://proceedings.mlr.press/v198/ibarz22a.html. 8

[32] Gleb Rodionov and Liudmila Prokhorenkova. Neural algorithmic reasoning without intermediate supervision. *arXiv preprint arXiv:2306.13411*, 2023. 8

[33] Cameron Diao and Ricky Loynd. Relational attention: Generalizing transformers for graph-structured tasks. In *ICLR 2023*, May 2023. URL https://www.microsoft.com/en-us/research/publication/relational-attention-generalizing-transformers-for-graph-structured-tasks/.

[34] Dobrik Georgiev, Danilo Numeroso, Davide Bacciu, and Pietro Liò. Neural algorithmic reasoning for combinatorial optimisation. *arXiv preprint arXiv:2306.06064*, 2023. 8

[35] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960. 9

[36] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 11

[37] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax. 12

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy,

Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. 12

A Appendix

A.1 Related Work

Algorithmic Learning. In recent years, the field of Algorithmic Learning has witnessed significant advancements, driven by a convergence of ideas from neural network architecture and algorithmic reasoning. To effectively tackle algorithmic tasks, models must incorporate a notion of variable computation length that enables extrapolation to handle larger input states. Consequently, proposals have emerged such as the differentiable Neural Turing machine [20] or RNNs with the capacity to generalize across varying input lengths [21]. Schwarzschild et al. [22] successfully showed the extrapolation capabilities of a recurrent architecture based on CNNs [23] with residual connections on a series of algorithmic tasks, including mazes, prefix sums, and chess problems.

Recently, graph-based methods have gained traction, because of their capability to model different sizes of inputs. In particular, the use of Graph Neural Networks (GNNs) has become a focus for algorithmic learning. Notable instances of this trend include applications to problems like SAT, TSP, and shortest path computations through algorithmic alignment [24–27]. Theoretical investigations have established links between GNNs and dynamic programming algorithms, along with the parallel computing paradigm [6, 28, 29]. Recent efforts have turned to improving the extrapolation capabilities on extending extrapolation capabilities to handle larger graph instances in the context of algorithmic reasoning problems [4, 30]. Grötschla et al. [3] present architectures capable of scaling up to sizes 1000 times that of the training data.

CLRS Benchmark. Introducing the CLRS benchmark, Veličković et al. [2] offer a comprehensive benchmark featuring CLRS-30 a dataset with over 30 algorithms designed for algorithmic reasoning tasks. This benchmark represents algorithms as graphs with task-specific inputs, outputs, and intermediate states called hints. The CLRS has triggered a wide range of follow-up work. Ibarz et al. [31] propose a generalist algorithmic learner, a single model capable of simultaneously tackling all CLRS-30 algorithms. CLRS-30 evaluations include simple *Out-Of-Distribution* (OOD) tests with graphs four times the size of the training graphs. Mahdavi et al. [7] provide an in-depth exploration of OOD generalization within the CLRS framework, emphasizing the need for diversified test sets consisting of more varied graphs. Several studies suggest that the inclusion of hints, as suggested by the CLRS, is not necessarily beneficial [7, 8, 32]. Bevilacqua et al. [8] introduce Causal Regularization, a data augmentation technique applied to hints, which enhances OOD generalization capabilities. Their work indicates the effectiveness of hints when employed correctly. Other architectural approaches to tackling the CLRS benchmark include [32–34]. Notably, it has been shown that parallel counterparts of the sequential algorithms implemented in CLRS prove to be more efficient to learn and execute of neural architectures, subsequently also leading to improved OOD predictions [6].

A.2 Algorithms

Breadth-first search (BFS). The input is a pointer to the starting node. The output is the directed BFS tree pointing from node to parent. Refer to Figure 6 for an example.

Depth-first search (DFS). The search starts at node 0, and the output is again the directed DFS tree, pointing from leaf to root. Refer to Figure 7 for an example.

Dijkstra. The Dijkstra shortest path algorithm on a weighted graph. As input, the source node is given. The output is the directed tree that corresponds to the shortest path from all nodes to the source node, again pointing from the leaf to the source node. Refer to Figure 8 for an example.

Maximum Spanning Tree (MST). Prim's algorithm for finding the Maximum Spanning Tree (MST) of a weighted graph. As input, we are given a source node, and the output is the directed MST pointing from leaf to root. Refer to Figure 9 for an example.

Distributed Maximal Independent Set (MIS). A Maximal Independent Set within a graph refers to a maximal set of nodes where no two nodes are adjacent. Our implementation is derived from the *Fast MIS* algorithm [10]. The algorithm relies on *randomness*, enabling a $\mathcal{O}(\log n)$ distributed runtime. The randomness is supplied as an input. The output is a mask over the nodes representing the MIS. Refer to Figure 10 for a visualized example and to Algorithm 1 for an algorithm outline.

Distributed Eccentricity. The eccentricity algorithm accepts a graph and a source node as input and produces the source node's eccentricity (also known as radius) as a scalar output. The eccentricity is the maximum distance from the given node to any other node in the graph. It can be solved by a combination of flooding a message through the graph and echoing the maximum value back to the source node. See Algorithm 2 for an outline and Figure 11 for a visualized example.

Most of the chosen algorithms for SALSA-CLRS do not explicitly rely on the learned models to perform value generalization by expressing the solution as topological encodings rather than scalar values. However, computing the diameter in the Distributed Eccentricity task requires the models to perform value generalization to larger scalar values for larger graphs. This might be one reason for worse empirical performance compared to other tasks of the SALSA-CLRS benchmark. However, we deem it important for the field of Neural Algorithmic Reasoning overall to consider the challenge of value generalization, which is one of the motivations for expressing the Eccentricity task in this form.

Algorithm 1 Fast MIS 2 [10]

The algorithm operates in synchronous rounds, grouped into phases.
A single phase is as follows:
1) Each node v takes its precomputed random value $r(v) \in [0, 1]$ and sends it to its neighbors.
2) If $r(v) < r(w)$ for all neighbors w of v, node v enters the MIS and informs its neighbors.
3) If v or a neighbor of v entered the MIS, v terminates (v and all edges adjacent to v are removed from the graph), otherwise v enters the next phase.

Algorithm 2 Eccentricity, adapted from [10]

The algorithm is a combination of flooding and echoing. Each node can be either dead or alive.
Flooding:
1) The source node sends the initial flooding message 1 to all neighbors and marks itself dead.
2) Each other node v, upon receiving the message the first time, increases the message by 1 and forwards it to all alive neighbors. Node v also remembers its parent, the node it got the flooding message from. Once the messages are sent, it marks itself dead.
Echoing:
3) If a node receives a flooding message and all of its neighbors are dead, it echos the message back to all of its dead neighbors and removes itself from the graph.
4) If a dead node v receives an echo message, it waits until it got an echo from all of its neighbors besides its parent. Then v echos the maximum of the received echos back to its parent. Finally, it removes itself from the graph.
5) Once the source node has received an echo from all its children, the maximum received value is its eccentricity.

A.3 Dataset

A.3.1 Graph Types

Erdös-Renyi Graphs (ER) [11]. An ER graph is generated by choosing each of the $\frac{n^2-n}{2}$ edges with probability p. In CLRS-30 this p is sampled from a range of numbers between 0.01 and 0.81[3]. In ER-graphs the degree of a node grows linearly with the number of nodes in the graph for fixed p. This means that when choosing a static p, the larger the graph is, the higher its connectivity. Different from CLRS-30, we ensure connectedness on all graphs. As we are interested in sparse graphs, we choose p to be a function of the number of nodes n. Hence, we require p to be as low as possible while the graph still remains connected with a high probability. Erdös and Renyi showed that for $c \leq \frac{1}{2}$ for ER graphs G with number of edges $E(G) \approx c\frac{\ln n}{n}$ the graph G is *almost surely connected* [35]. As the average number of edges in an ER graph is $E(G) \approx \binom{n}{2}p$ we choose p to be

$$p = c\frac{\ln n}{n}$$

where c is a scalar that is randomly sampled out of the interval $(1, 2)$ to increase the diversity in dataset. See Figure 2 for examples.

[3]They sample for the range $[0.1, 0.2, ..., 0.9]$ and square it.

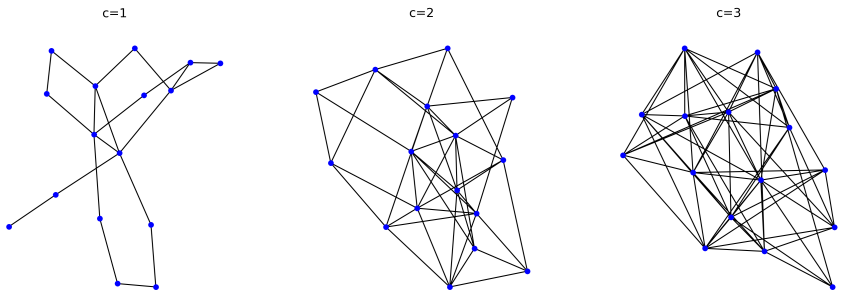

Figure 2: Examples of ER graphs with $n = 16$ and $p = c\frac{\ln n}{n} \approx 0.173c$.

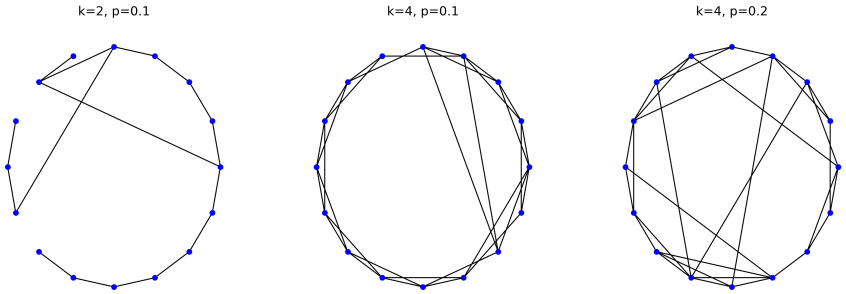

Figure 3: Examples of WS graphs with $n = 16$.

Watt Strogatz Graphs (WS) [12]. A WS graph, a Small World graph [13], is created by taking a ring lattice – a ring where each node is connected to k neighbors ($\frac{k}{2}$ neighbors on each side) – and rewire each edge with probability p to a random other node. WS graphs have the characteristic that they are globally connected but show local clustering. We also enforce connectivity on WS graphs. As $p \to \infty$ WS graphs approach ER graphs with $p \approx \binom{n}{2}^{-1} 2nk$, so we keep p relatively small. We randomly sample k from $[4, 6, 8]$ and p from the interval of $(0.05, 0.2)$. See Figure 3 for examples.

Delaunay Graphs. Delaunay graphs are created by sampling n points in the plane and computing the Delaunay triangulation. As the graph of a Delaunay triangulation is planar, its average degree is below 6. See Figure 4 for examples.

A.3.2 Dataset Statistics

SALSA-CLRS provides a training set, a validation set, and 15 different test sets. The training and validation sets follow closely the datasets CLRS-30, ER graphs with n sampled randomly from $[4, 7, 11, 13, 16]$. We slightly modify this and choose p as described in Section A.3.1 and increase the size of the training and validation sets to 10000 and 1000, respectively.

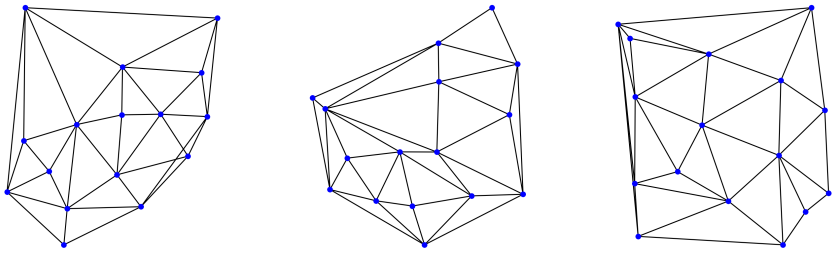

Figure 4: Examples of Delaunay graphs with $n = 16$.

The SALSA-CLRS dataset test extrapolation for graphs of size $5\times, 10\times, 50\times$ and $100\times$ the size of the training graphs, resulting in graphs of size 16, 80, 160, 800, and 1600. This results in 15 different test sets across the three different graph types to measure different facets of OOD performance.

A.4 Evaluation

A.4.1 Architectures

We use the same Encode-Process-Decode framework [14] that CLRS builds on. An encoder – specific to every input feature – produces a latent representation of the input features. These latent representations are aggregated (max) to yield a 128-dimensional node hidden state. Similarly, a per-feature decoder computes the features from the node's hidden state. Notably, when dealing with hints, our model diverges from the CLRS approach. While CLRS decodes the predicted hints in each algorithmic step, calculates a loss, and then re-encodes these hints, our approach streamlines this process. We only decode the hint predictions to calculate the loss and do not re-encode the decoded hints afterward. In the process step, a message-passing layer updates the node embeddings. As part of the process step, node hidden states undergo an update via a message-passing layer. In scenarios necessitating randomness, the precomputed randomness is concatenated to the processor input. Hints and outputs are decoded from the last two hidden states as well as the input state.

The following three processor modules are evaluated, all employing maximum aggregation and layer normalization [36]. We define h_v^t to be the hidden state of node v at timestep t and $\mathcal{F}$ to be the aggregation function.

GIN(E). Standard GIN module with a two-layer Multi-Layer Perceptron (MLP) with ReLu activations and batch norm. We also add a GRU (Gated Recurrent Unit) Cell [15] after the message passing to improve training stability. The update without edge weights is defined as:

$$h_v^{t+1} = \text{GRU}\left[\Theta_1\left((1+\epsilon)\cdot h_v^t + \underset{w\in N(v)}{\mathcal{F}} h_w^t\right), h_v^t\right]$$

RecGNN. The architecture proposed by [3], originally named RecGRU-E. Before the message passing step, an MLP is applied on the edges – the concated node embedding. After each message passing update all node embeddings are passed through a GRU cell. For algorithms that requsire edge weights, we concat the edge weight to each message before we pass it through the edge MLP Θ. The update without edge weights is defined as:

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

PGN. The PGN architecture is introduced in [18]. It defines the following components: The source node linear layer Θ_s, the target node linear layer Θ_t, the two layer message MLP Θ_{msg}, the skip connection linear layer Θ_{skip}, the output linear layer Θ_{out} and a ReLu activation σ. The update without edge weights is defined as

$$h_v^{t+1} = \sigma\left(\Theta_{skip}(h_v^t) + \Theta_{out}\left[\underset{w\in N(v)}{\mathcal{F}} \Theta_{msg}\left(\Theta_s(h_v^t) + \Theta_t(h_w^t)\right)\right]\right)$$

A.4.2 Experiments

We use early stopping with patience 30. The patience is kept this high because we observed that some training runs dip quite strongly before finding a new optimum. Further, we use a plateau scheduler with patience 10 and factor 0.1. The seeds selected are $42-46$. To combat exploding gradients we apply gradient clipping on the 2-norm of the weights. Adding 2-norm regularization on the hidden node states also helps with training stability. We employ different learning rates for the baseline models, determined by a hyperparameter sweep on the Dijkstra algorithm. For GIN(E) and PGN, we use a learning rate of 0.0004239, and for RecGNN we use 0.0008.

A.4.3 Metrics

For all problems, we report Graph Accuracy, referring to whether a graph was entirely solved correctly or not. On the problems BFS, DFS, MIS, Dijkstra, and MST we report Node Accuracy, and for MIS

we also report Node F1. As for Eccentricity, we predict a single scalar for the whole graph, the notion of Node Accuracy and Node F1 is not applicable. To compute the Graph Accuracy for Eccentricity, we round the predicted scalar and check whether it was correctly predicted. Additionally, we report the Mean-Squared-Error (MSE) of the unrounded scalar prediction. For all problems predicting a tree with Node Pointers[4] (BFS, DFS, Dijkstra, MST) each node either predicts its parent correctly or not. An F1 score is not of interest here. It is important to highlight the reporting of all of these metrics and their differences. The Node metrics can be misleading, as is apparent when comparing, e.g., the Node Accuracy performance to the Graph Accuracy. If we predict a node mask, like in MIS, the Node F1 score is the most indicative, as the Node Accuracy does not consider class imbalance.

A.4.4 Scalability

In Figure 5, we assess the scalability of SALSA-CLRS and CLRS by examining their GPU VRAM utilization when used with the BFS algorithm. The figure shows the clear **asymptotic** advantage brought by the sparsification in SALSA-CLRS. SALSA-CLRS manages inference of graphs as large as 32768 nodes with less than 8GB of VRAM.

To obtain our results, we generated 10 graphs for each graph size using the default settings for each benchmark. For more details regarding these graph types, please refer to Section 2.2. For SALSA-CLRS, we employed the GIN architecture, while for CLRS, we compared the performance of both the "trippled-mpnn" and "pgn" models. We processed all 10 graphs individually with a batch size of 1. These measurements were conducted on an Nvidia A100 GPU boasting 80GB of VRAM.

Notably, CLRS is implemented in JAX [37], whereas SALSA-CLRS is based on PyTorch [38], each of which reports memory usage differently. For CLRS, we report memory usage as `jax.local_devices()[gpu_id].memory_stats()['peak_bytes_in_use']`, while for SALSA-CLRS, we use `torch.cuda.max_memory_allocated()`. These reported values might be notably lower than what is indicated by the *nvidia-smi* tool, as the latter includes memory that is reserved but not actively allocated.

It's also worth noting that CLRS uses in-memory datasets, meaning that the entire dataset is stored in RAM. While this approach offers runtime advantages, it can become a bottleneck when dealing with large graphs and datasets due to high RAM usage. For example, a dataset with 1000 graphs of size 1600 does not fit on a machine with 64GB of RAM. In contrast, SALSA-CLRS offloads the dataset to disk and dynamically loads the datapoints into RAM. As long as a single datapoint fits into RAM, the dataset size is not a constraint.

A.5 Implementation of Node Pointers

A Node Pointer is used to encode the reference from one node to another and is often used to represent the solutions of the algorithms, i.e., the BFS tree. However, not all fields that are of type `(*, Node, Pointer)` behave exactly the same way. In the algorithms we have selected (BFS, DFS, etc.), these fields share one property: The node pointer always points to a neighboring node. Therefore, in order to derive these node pointers, the computational cost is proportional to the amount of edges in the graph. However, for other algorithms that were part of CLRS-30 but are not yet incorporated into SALSA-CLRS (Toposort, MST Kruskal, etc.), these node pointers are no longer restricted in the same way. They could and must, in certain cases, point to arbitrary nodes in the graph (and not just immediate neighbors). For this, all potential edges that could exist in the graph must be considered – which is again in order of $\mathcal{O}(n^2)$ and clashes with the idea of sparse computation on the original topology.
In the example of topological sort, the `topo` feature, a node pointer, represents the output of the algorithm. Each node points to the next element in the topological sort, and the last element points to itself. This definition does not guarantee that the pointers are also part of the sparse graph. On the other hand, in DFS, as it is implemented in CLRS, we compute the DFS tree starting at node 0. The mentioned output feature `pi`, also of type node pointer, represents this tree by node pointers,

[4] A Node Pointer is a data type that serves as a reference to a single neighbor from among the various neighbors connected to a node. To illustrate this concept, consider the case of a BFS tree (the result of the BFS algorithm) in a graph, where a Node Pointer can be used to denote the edge leading to the parent node. We refer to the original CLRS paper for more details.

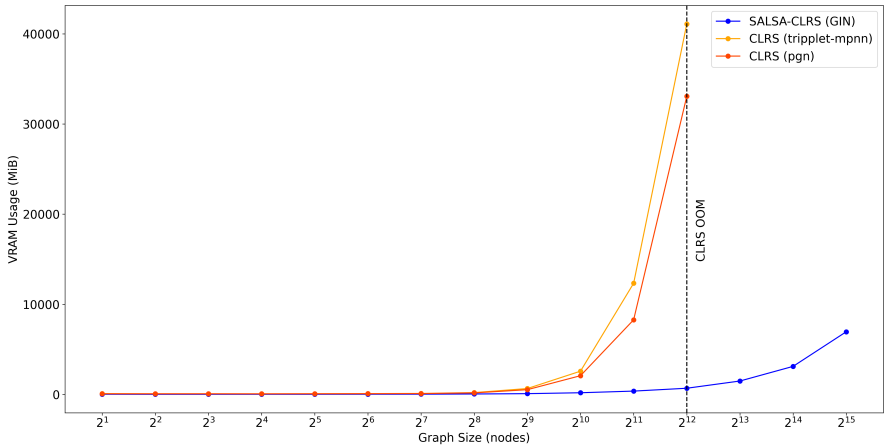

Figure 5: Scalability analysis of SALSA-CLRS and CLRS. Shown is the peak VRAM usage on a NVIDIA A100 GPU with 80GB VRAM across 10 randomly sampled graphs with batch size 1 on the BFS algorithm.

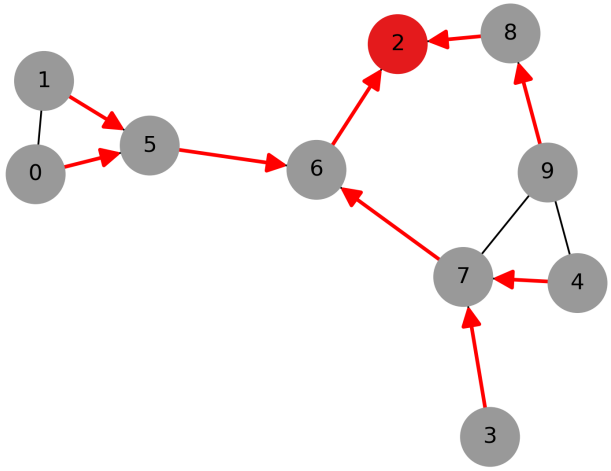

Figure 6: Example of BFS output. The red node corresponds to the source node, and the red edges to the directed BFS tree.

pointing from child to parent. By definition, these nodes are neighboring, so `pi` can be encoded on a sparse graph. Single Source Shortest Path uses topsort as a subroutine, so the hint `topo_h` encodes the output of this topsort. Hence, we run into the same problem as described above. Kruskal uses the `pi` variable in the union subroutine, which is not sparsely representable. For strongly connected components, the `scc_id` output feature is a node pointer, pointing to the node with the lowest id in each component. This, again, does not require that this edge exists in the sparse graph.

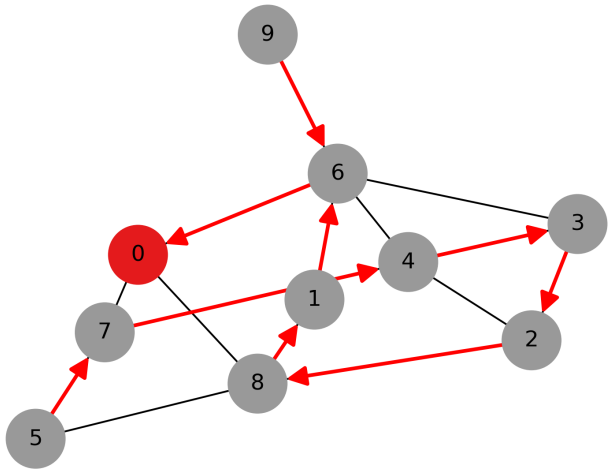

Figure 7: Example of DFS output. The red node 0 corresponds to the source node, and the red edges to the directed DFS tree.

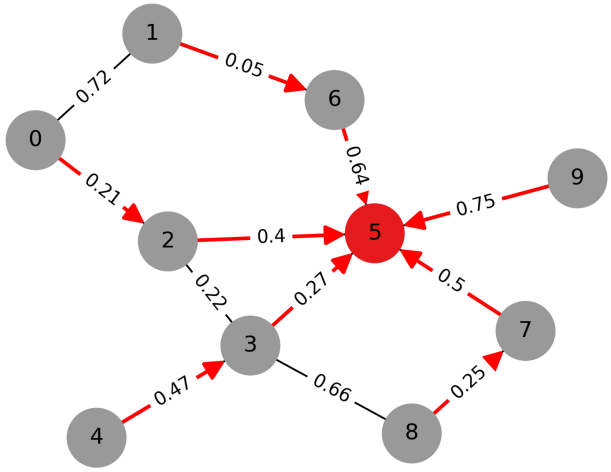

Figure 8: Example of Dijkstra output. The red node corresponds to the source node, and the red edges to the directed shortest path tree.

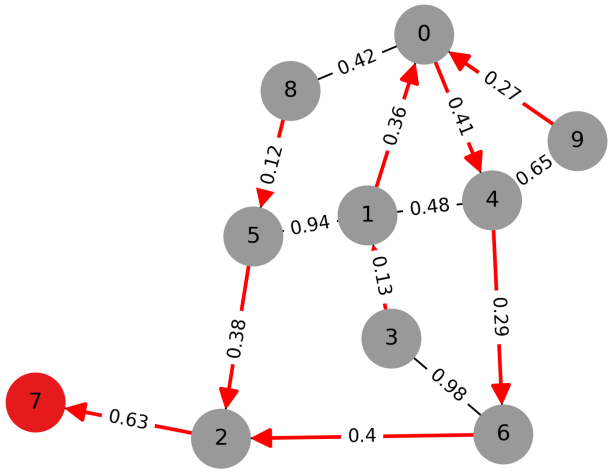

Figure 9: Example of MST output. The red node corresponds to the source node for Prim's algorithm, and the red edges to directed MST.

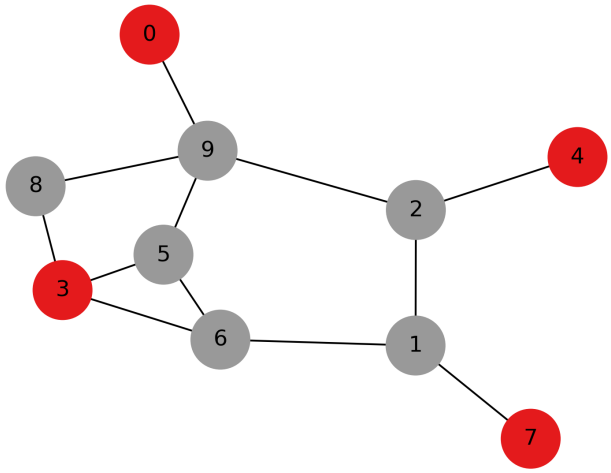

Figure 10: Example of MIS output. The red nodes correspond to the found Maximal Independent Set.

Table 2: Node Accuracy scores of both models on all algorithms that support Node Accuracy. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

		ER					WS					Delaunay				
		16	80	160	800	1600	16	80	160	800	1600	16	80	160	800	1600
BFS	GIN(E)	100.0± 0.1	99.6± 0.4	99.3± 0.6	98.0± 1.6	98.0± 1.5	99.9± 0.3	92.9± 4.2	86.7± 5.5	70.4± 10.8	75.3± 6.1	100.0± 0.1	94.3± 5.6	84.6± 10.6	52.7± 17.2	45.9± 15.8
	PGN	100.0± 0.0	99.8± 0.1	99.5± 0.3	99.0± 0.2	98.9± 0.2	100.0± 0.0	95.5± 0.7	88.7± 1.5	75.9± 3.3	80.6± 0.7	100.0± 0.0	98.2± 0.7	90.4± 4.5	53.6± 7.0	40.3± 6.5
	RecGNN	100.0± 0.0	99.8± 0.1	99.5± 0.3	99.3± 0.4	99.2± 0.4	100.0± 0.0	97.8± 1.1	94.2± 2.0	82.2± 4.7	82.1± 2.3	100.0± 0.0	98.5± 0.8	92.0± 5.5	67.1± 11.8	55.6± 10.0
BFS (H)	GIN(E)	98.8± 2.4	95.3± 9.2	95.1± 8.9	86.9± 26.1	86.5± 27.2	99.2± 1.4	83.0± 25.0	77.5± 24.9	60.6± 28.8	64.4± 32.4	98.1± 4.0	79.5± 32.3	68.9± 32.6	42.8± 16.4	34.2± 10.0
	PGN	100.0± 0.0	99.8± 0.1	99.6± 0.1	98.7± 0.3	98.5± 0.3	100.0± 0.0	96.1± 0.5	90.8± 0.8	76.4± 1.6	80.6± 1.0	100.0± 0.0	97.5± 0.8	89.4± 1.6	53.2± 2.4	40.8± 3.2
	RecGNN	100.0± 0.0	99.6± 0.2	99.3± 0.5	99.0± 0.5	98.6± 0.6	100.0± 0.1	96.7± 0.8	92.5± 2.0	77.6± 3.8	79.3± 1.7	100.0± 0.0	95.3± 2.3	83.6± 6.0	51.5± 4.1	42.9± 5.0
DFS	GIN(E)	49.3± 8.1	30.6± 4.0	19.7± 3.9	18.1± 3.8	16.5± 3.5	29.7± 4.9	15.9± 0.9	16.8± 0.8	22.3± 0.6	20.1± 0.5	46.7± 7.3	28.0± 3.1	25.1± 3.1	23.4± 2.9	23.2± 2.9
	PGN	74.2± 14.0	41.2± 3.8	29.9± 2.6	27.8± 2.1	25.8± 2.1	58.8± 20.8	17.9± 1.7	17.7± 0.8	23.6± 0.6	21.3± 0.6	72.7± 13.1	41.7± 3.9	38.2± 2.8	35.8± 2.1	35.4± 2.1
	RecGNN	33.4± 14.5	28.0± 6.5	18.7± 4.1	18.2± 4.4	16.8± 4.3	22.7± 8.2	15.9± 1.5	16.8± 1.4	21.5± 1.6	19.5± 1.4	32.3± 14.9	26.8± 5.8	25.2± 5.3	24.1± 5.2	24.0± 5.2
DFS (H)	GIN(E)	41.5± 7.5	30.4± 2.3	20.0± 3.1	19.5± 2.6	17.8± 2.5	25.0± 3.7	15.8± 0.6	16.8± 0.4	22.7± 0.7	20.6± 0.6	39.6± 9.1	28.3± 3.1	26.1± 3.7	25.3± 2.9	25.2± 2.9
	PGN	82.0± 9.2	38.4± 2.7	26.9± 2.5	24.9± 2.3	23.1± 2.3	57.6± 17.6	17.0± 1.6	17.2± 0.5	22.9± 1.3	20.7± 1.1	79.9± 8.8	38.3± 3.9	34.7± 3.7	31.9± 3.7	31.5± 3.7
	RecGNN	48.3± 19.1	22.8± 4.7	13.5± 4.6	13.1± 4.1	12.0± 3.6	35.3± 17.7	13.5± 2.6	14.7± 1.9	19.4± 2.1	17.9± 1.7	50.2± 21.7	21.8± 3.2	19.4± 3.8	18.7± 3.6	18.5± 3.5
Dijkstra	GIN(E)	98.0± 0.2	89.8± 1.1	84.3± 1.6	75.8± 2.2	72.8± 2.3	95.4± 0.7	85.0± 1.4	79.9± 1.9	61.4± 4.0	52.6± 4.1	97.4± 0.4	81.6± 1.3	70.4± 2.6	46.5± 3.7	39.9± 3.6
	PGN	99.6± 0.1	98.6± 0.3	97.2± 0.5	94.1± 0.6	92.2± 0.7	98.3± 0.4	97.1± 0.2	95.4± 0.3	81.8± 1.2	72.5± 6.0	99.5± 0.1	97.6± 0.3	92.4± 0.7	62.7± 1.2	51.0± 3.9
	RecGNN	98.5± 1.6	86.8± 15.4	76.0± 22.1	63.7± 27.7	60.6± 27.7	95.8± 4.2	89.2± 14.1	83.9± 18.9	71.4± 20.4	67.3± 17.7	98.0± 1.9	90.4± 9.7	85.0± 10.0	60.2± 4.4	50.0± 3.6
Dijkstra (H)	GIN(E)	95.2± 1.8	62.4± 7.0	53.3± 6.2	40.4± 8.1	36.9± 7.6	91.2± 3.5	55.3± 9.3	48.1± 8.3	38.6± 5.2	35.6± 4.4	94.2± 1.8	54.4± 7.8	45.2± 5.4	37.2± 4.1	36.0± 4.1
	PGN	99.3± 0.1	94.2± 2.5	92.0± 2.3	87.1± 2.7	84.5± 3.4	97.8± 0.2	85.8± 6.0	80.9± 7.0	60.5± 8.3	52.4± 8.3	99.2± 0.1	84.9± 6.8	72.8± 8.9	50.8± 4.6	46.4± 3.1
	RecGNN	98.0± 0.1	32.9± 21.6	25.0± 17.4	17.7± 12.2	16.4± 10.7	95.5± 1.0	36.3± 16.4	29.4± 16.1	27.3± 12.3	26.6± 11.7	97.4± 0.4	35.6± 17.8	29.5± 17.0	26.7± 14.4	26.3± 14.1
MIS	GIN(E)	82.2± 2.5	81.6± 1.9	80.8± 2.4	83.6± 1.5	80.8± 2.5	84.2± 2.1	82.0± 2.6	82.3± 2.4	84.3± 1.9	83.4± 2.6	82.5± 3.2	82.4± 3.0	81.5± 3.2	80.9± 3.7	80.3± 4.0
	PGN	99.8± 0.1	99.6± 0.2	99.5± 0.2	98.8± 0.6	98.9± 0.5	99.8± 0.1	99.4± 0.3	98.8± 0.6	95.8± 2.6	93.3± 4.4	99.9± 0.1	99.8± 0.1	99.8± 0.1	99.5± 0.2	99.3± 0.3
	RecGNN	93.6± 2.2	90.0± 2.3	90.1± 2.5	87.9± 1.9	88.2± 2.6	93.3± 2.2	92.6± 2.6	92.2± 2.9	91.8± 3.3	91.4± 3.5	94.3± 2.0	93.4± 2.0	93.0± 2.5	92.5± 3.0	92.1± 3.4
MIS (H)	GIN(E)	79.9± 2.9	79.9± 2.2	78.2± 2.7	83.4± 0.8	79.2± 1.6	83.1± 1.9	79.5± 3.4	79.8± 3.3	83.2± 2.2	81.8± 2.6	80.6± 3.1	80.6± 3.5	79.8± 3.6	78.9± 3.7	78.2± 3.7
	PGN	99.8± 0.1	99.4± 0.1	99.4± 0.2	98.8± 0.5	98.9± 0.7	99.7± 0.1	99.5± 0.2	99.1± 0.3	98.6± 0.8	98.2± 1.3	99.8± 0.1	99.7± 0.1	99.7± 0.2	99.4± 0.7	99.1± 1.2
	RecGNN	92.2± 0.7	88.7± 1.7	88.3± 2.8	85.6± 3.5	84.7± 5.5	92.9± 0.5	92.2± 1.9	91.8± 2.6	90.5± 4.3	88.8± 6.0	93.5± 0.7	92.4± 1.4	91.7± 2.2	89.8± 4.6	88.0± 6.5
MST	GIN(E)	92.6± 0.8	79.1± 1.3	77.6± 1.7	74.5± 2.0	72.9± 2.2	89.6± 1.4	75.3± 1.0	74.4± 1.4	73.0± 2.4	72.8± 2.3	92.8± 0.8	77.4± 0.6	75.8± 1.1	74.8± 1.7	74.7± 1.7
	PGN	97.3± 0.4	89.1± 1.6	84.6± 1.7	75.7± 2.0	71.9± 2.1	96.8± 1.0	82.5± 2.4	77.6± 2.6	67.4± 3.1	65.1± 3.3	97.4± 0.5	85.2± 1.5	78.5± 1.4	68.7± 1.0	66.8± 0.9
	RecGNN	94.2± 2.3	70.7± 27.8	66.6± 28.2	58.9± 29.0	56.0± 28.5	92.8± 2.8	67.4± 22.9	62.8± 23.2	53.5± 20.9	52.5± 17.1	94.7± 2.1	69.9± 22.1	62.6± 20.9	52.5± 13.4	50.6± 11.3
MST (H)	GIN(E)	89.6± 1.7	51.6± 4.5	49.5± 4.3	45.0± 4.2	43.2± 4.0	86.0± 2.1	54.9± 6.2	52.7± 6.5	50.9± 6.4	54.1± 6.9	91.1± 1.5	58.4± 5.9	56.4± 5.6	55.0± 5.6	54.9± 5.5
	PGN	96.4± 0.6	79.7± 3.8	75.6± 4.5	69.5± 5.5	66.8± 5.1	96.1± 1.0	74.5± 3.9	72.5± 4.5	69.2± 4.4	68.8± 5.9	96.7± 0.5	77.7± 4.1	74.3± 5.0	71.4± 6.5	71.0± 6.7
	RecGNN	87.5± 2.4	29.0± 6.7	25.7± 6.6	21.3± 6.4	20.1± 6.3	82.0± 4.0	32.0± 7.3	29.6± 6.0	24.9± 7.3	28.8± 8.7	88.2± 2.1	34.2± 8.4	31.9± 7.1	28.0± 7.2	27.8± 7.3

Table 3: Graph Accuracy scores of both models on all algorithms. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

		ER					WS					Delaunay				
		16	80	160	800	1600	16	80	160	800	1600	16	80	160	800	1600
BFS	GIN(E)	99.4± 0.8	84.3± 13.9	57.5± 15.3	2.2± 4.1	0.1± 0.2	98.0± 4.2	5.7± 8.7	0.2± 0.5	0.0± 0.0	0.0± 0.0	99.3± 1.0	25.1± 28.6	0.7± 1.4	0.0± 0.0	0.0± 0.0
	PGN	100.0± 0.0	88.7± 5.9	54.9± 21.5	0.2± 0.1	0.0± 0.0	100.0± 0.0	13.1± 3.3	0.1± 0.1	0.0± 0.0	0.0± 0.0	100.0± 0.0	35.1± 8.3	0.3± 0.4	0.0± 0.0	0.0± 0.0
	RecGNN	99.9± 0.2	87.9± 8.8	55.8± 24.8	4.6± 6.5	0.4± 0.6	100.0± 0.0	32.5± 18.3	1.0± 1.2	0.0± 0.0	0.0± 0.0	100.0± 0.0	53.4± 11.5	1.7± 1.2	0.0± 0.0	0.0± 0.0
BFS (H)	GIN(E)	92.5± 13.9	59.4± 38.3	37.8± 37.9	0.9± 1.4	0.0± 0.1	92.8± 12.0	10.2± 13.8	0.4± 0.7	0.0± 0.0	0.0± 0.0	85.2± 28.9	17.5± 17.7	0.2± 0.3	0.0± 0.0	0.0± 0.0
	PGN	100.0± 0.0	88.1± 3.8	66.3± 8.7	0.2± 0.3	0.0± 0.0	100.0± 0.0	14.2± 3.6	0.2± 0.2	0.0± 0.0	0.0± 0.0	100.0± 0.0	26.2± 11.5	0.1± 0.1	0.0± 0.0	0.0± 0.0
	RecGNN	99.9± 0.1	81.7± 13.0	49.6± 25.2	1.8± 2.3	0.0± 0.1	99.4± 1.3	20.7± 13.5	1.3± 2.3	0.0± 0.0	0.0± 0.0	99.9± 0.2	18.7± 8.4	0.0± 0.0	0.0± 0.0	0.0± 0.0
DFS	GIN(E)	0.1± 0.1	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	18.4± 37.7	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	9.5± 21.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	13.9± 29.4	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	RecGNN	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
DFS (H)	GIN(E)	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	19.9± 30.7	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	3.2± 7.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	13.8± 23.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	RecGNN	4.5± 7.8	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	5.8± 11.5	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
Dijkstra	GIN(E)	73.4± 2.6	0.2± 0.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	51.6± 3.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	66.6± 4.3	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	94.6± 1.1	37.8± 6.9	5.2± 1.9	0.0± 0.0	0.0± 0.0	76.4± 4.0	17.2± 2.8	0.9± 0.8	0.0± 0.0	0.0± 0.0	93.0± 1.4	19.2± 4.2	0.1± 0.0	0.0± 0.0	0.0± 0.0
	RecGNN	81.7± 16.1	6.8± 6.1	0.3± 0.5	0.0± 0.0	0.0± 0.0	60.4± 22.7	8.4± 7.4	0.2± 0.2	0.0± 0.0	0.0± 0.0	74.4± 19.9	4.4± 4.5	0.0± 0.0	0.0± 0.0	0.0± 0.0
Dijkstra (H)	GIN(E)	49.8± 10.8	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	28.7± 9.9	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	40.3± 10.4	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	89.5± 1.0	3.3± 3.7	0.0± 0.1	0.0± 0.0	0.0± 0.0	70.8± 2.4	0.4± 0.6	0.0± 0.0	0.0± 0.0	0.0± 0.0	87.6± 0.7	0.4± 0.8	0.0± 0.0	0.0± 0.0	0.0± 0.0
	RecGNN	73.8± 1.6	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	50.9± 5.6	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	66.4± 3.3	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
Eccentricity	GIN(E)	57.3± 21.2	77.1± 17.5	72.3± 18.0	51.3± 34.2	36.7± 17.6	78.0± 18.7	27.6± 19.5	3.6± 8.0	0.0± 0.0	0.0± 0.0	84.8± 12.4	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0	64.6± 14.9	100.0± 0.0	93.8± 2.1	100.0± 0.1	25.6± 7.5	5.2± 3.3	100.0± 0.0	100.0± 0.0	76.9± 19.8	0.0± 0.0	0.0± 0.0
	RecGNN	75.8± 26.2	80.5± 35.0	75.0± 39.1	72.7± 27.9	63.0± 24.8	86.7± 25.7	60.8± 29.1	57.4± 38.7	27.6± 29.4	15.2± 13.7	89.9± 19.4	25.2± 37.6	8.3± 11.9	0.0± 0.0	0.0± 0.0
Eccentricity (H)	GIN(E)	25.3± 41.0	23.8± 39.0	26.1± 36.8	17.1± 32.9	16.0± 21.7	25.3± 42.2	19.0± 18.8	18.6± 18.9	4.6± 8.9	9.8± 10.2	24.8± 42.5	17.0± 12.5	3.0± 5.8	0.0± 0.0	0.0± 0.0
	PGN	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0	83.0± 6.5	100.0± 0.0	88.3± 1.8	100.0± 0.1	34.8± 7.2	9.2± 4.8	100.0± 0.0	99.7± 0.3	64.4± 14.2	0.0± 0.0	0.0± 0.0
	RecGNN	95.0± 6.3	96.6± 3.6	95.8± 4.6	93.4± 10.3	72.1± 20.9	99.0± 1.2	66.4± 22.4	46.2± 40.9	14.1± 6.3	8.3± 4.9	99.6± 0.8	51.0± 36.0	19.4± 11.7	0.0± 0.0	0.0± 0.0
MIS	GIN(E)	6.2± 3.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	6.5± 2.8	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	6.1± 3.8	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	98.8± 0.2	89.2± 4.6	74.1± 10.1	10.7± 10.5	2.0± 2.5	98.1± 0.6	84.4± 8.4	58.3± 14.1	4.6± 4.3	0.5± 0.6	98.9± 0.6	93.9± 2.2	87.2± 4.9	41.2± 8.9	17.4± 7.4
	RecGNN	56.1± 13.1	5.5± 7.1	0.8± 1.6	0.0± 0.0	0.0± 0.0	52.6± 14.6	9.0± 9.8	2.0± 2.9	0.0± 0.0	0.0± 0.0	56.0± 13.3	9.6± 7.8	1.7± 2.2	0.0± 0.0	0.0± 0.0
MIS (H)	GIN(E)	3.3± 2.5	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	4.4± 2.3	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	3.3± 2.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	98.6± 0.4	88.9± 3.1	76.2± 7.3	18.0± 8.6	5.2± 4.3	98.2± 0.3	82.2± 7.4	54.1± 6.6	2.3± 1.7	0.1± 0.0	98.6± 0.4	92.2± 2.2	85.1± 3.4	40.1± 9.1	15.1± 6.5
	RecGNN	44.1± 5.8	2.6± 1.5	0.1± 0.2	0.0± 0.0	0.0± 0.0	46.5± 5.7	4.2± 1.1	0.4± 0.4	0.0± 0.0	0.0± 0.0	46.9± 6.2	4.8± 1.4	0.3± 0.3	0.0± 0.0	0.0± 0.0
MST	GIN(E)	43.2± 4.6	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	30.0± 4.1	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	43.0± 5.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	79.2± 4.3	2.0± 1.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	73.2± 9.1	0.3± 0.3	0.0± 0.0	0.0± 0.0	0.0± 0.0	78.8± 4.1	0.6± 0.4	0.0± 0.0	0.0± 0.0	0.0± 0.0
	RecGNN	56.8± 15.9	0.6± 0.8	0.0± 0.0	0.0± 0.0	0.0± 0.0	44.4± 18.0	0.1± 0.1	0.0± 0.0	0.0± 0.0	0.0± 0.0	58.7± 15.8	0.1± 0.2	0.0± 0.0	0.0± 0.0	0.0± 0.0
MST (H)	GIN(E)	29.7± 5.6	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	20.4± 5.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	34.6± 6.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	PGN	69.9± 6.1	0.0± 0.1	0.0± 0.0	0.0± 0.0	0.0± 0.0	65.7± 8.8	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	72.6± 5.2	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
	RecGNN	24.5± 7.5	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	14.8± 5.4	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	26.0± 7.5	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0

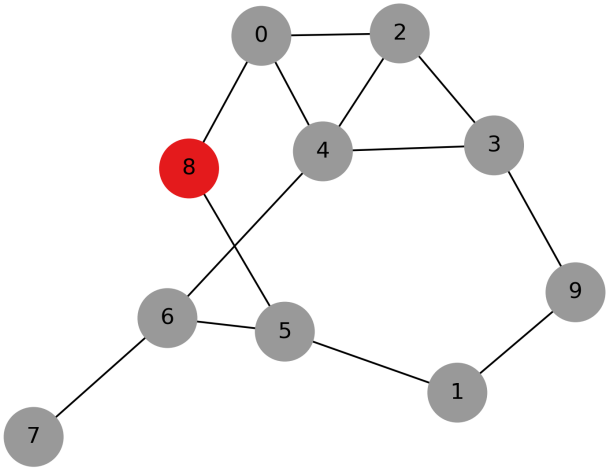

Figure 11: Example of an eccentricity output. The eccentricity of node 8 – the distance to the furthest away node – is 3.

Table 4: BFS Results. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

(a) Graph Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	99.4 ± 0.8	92.5 ± 13.9	100.0 ± 0.0	100.0 ± 0.0	99.9 ± 0.2	99.9 ± 0.1
	80	84.3 ± 13.9	59.4 ± 38.3	88.7 ± 5.9	88.1 ± 3.8	87.9 ± 8.8	81.7 ± 13.0
	160	57.5 ± 15.3	37.8 ± 37.9	54.9 ± 21.5	66.3 ± 8.7	55.8 ± 24.8	49.6 ± 25.2
	800	2.2 ± 4.1	0.9 ± 1.4	0.2 ± 0.1	0.2 ± 0.3	4.6 ± 6.5	1.8 ± 2.3
	1600	0.1 ± 0.2	0.0 ± 0.1	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.6	0.0 ± 0.1
WS	16	98.0 ± 4.2	92.8 ± 12.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	99.4 ± 1.3
	80	5.7 ± 8.7	10.2 ± 13.8	13.1 ± 3.3	14.2 ± 3.6	32.5 ± 18.3	20.7 ± 13.5
	160	0.2 ± 0.5	0.4 ± 0.7	0.1 ± 0.1	0.2 ± 0.2	1.0 ± 1.2	1.3 ± 2.3
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Delaunay	16	99.3 ± 1.0	85.2 ± 28.9	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	99.9 ± 0.2
	80	25.1 ± 28.6	17.5 ± 17.7	35.1 ± 8.3	26.2 ± 11.5	53.4 ± 11.5	18.7 ± 8.4
	160	0.7 ± 1.4	0.2 ± 0.3	0.3 ± 0.4	0.1 ± 0.1	1.7 ± 1.2	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

(b) Node Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	100.0 ± 0.1	98.8 ± 2.4	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	80	99.6 ± 0.4	95.3 ± 9.2	99.8 ± 0.1	99.8 ± 0.1	99.8 ± 0.1	99.6 ± 0.2
	160	99.3 ± 0.6	95.1 ± 8.9	99.5 ± 0.3	99.6 ± 0.1	99.5 ± 0.3	99.3 ± 0.5
	800	98.0 ± 1.6	86.9 ± 26.1	99.0 ± 0.2	98.7 ± 0.3	99.3 ± 0.4	99.0 ± 0.5
	1600	98.0 ± 1.5	86.5 ± 27.2	98.9 ± 0.2	98.5 ± 0.3	99.2 ± 0.4	98.6 ± 0.6
WS	16	99.9 ± 0.3	99.2 ± 1.4	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.1
	80	92.9 ± 4.2	83.0 ± 25.0	95.5 ± 0.7	96.1 ± 0.5	97.8 ± 1.1	96.7 ± 0.8
	160	86.7 ± 5.5	77.5 ± 24.9	88.7 ± 1.5	90.8 ± 0.8	94.2 ± 2.0	92.5 ± 2.0
	800	70.4 ± 10.8	60.6 ± 28.8	75.9 ± 3.3	76.4 ± 1.6	82.2 ± 4.7	77.6 ± 3.8
	1600	75.3 ± 6.1	64.4 ± 32.4	80.6 ± 0.7	80.6 ± 1.0	82.1 ± 2.3	79.3 ± 1.7
Delaunay	16	100.0 ± 0.1	98.1 ± 4.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	80	94.3 ± 5.6	79.5 ± 32.3	98.2 ± 0.7	97.5 ± 0.8	98.5 ± 0.8	95.3 ± 2.3
	160	84.6 ± 10.6	68.9 ± 32.6	90.4 ± 4.5	89.4 ± 1.6	92.0 ± 5.5	83.6 ± 6.0
	800	52.7 ± 17.2	42.8 ± 16.4	53.6 ± 7.0	53.2 ± 2.4	67.1 ± 11.8	51.5 ± 4.1
	1600	45.9 ± 15.8	34.2 ± 10.0	40.3 ± 6.5	40.8 ± 3.2	55.6 ± 10.0	42.9 ± 5.0

Table 5: DFS Results. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

(a) Graph Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	0.1 ± 0.1	0.0 ± 0.0	18.4 ± 37.7	19.9 ± 30.7	0.0 ± 0.0	4.5 ± 7.8
	80	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
WS	16	0.0 ± 0.0	0.0 ± 0.0	9.5 ± 21.2	3.2 ± 7.2	0.0 ± 0.0	0.0 ± 0.0
	80	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Delaunay	16	0.0 ± 0.0	0.0 ± 0.0	13.9 ± 29.4	13.8 ± 23.0	0.0 ± 0.0	5.8 ± 11.5
	80	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

(b) Node Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	49.3 ± 8.1	41.5 ± 7.5	74.2 ± 14.0	82.0 ± 9.2	33.4 ± 14.5	48.3 ± 19.1
	80	30.6 ± 4.0	30.4 ± 2.3	41.2 ± 3.8	38.4 ± 2.7	28.0 ± 6.5	22.8 ± 4.7
	160	19.7 ± 3.9	20.0 ± 3.1	29.9 ± 2.6	26.9 ± 2.5	18.7 ± 4.1	13.5 ± 4.6
	800	18.1 ± 3.8	19.5 ± 2.6	27.8 ± 2.1	24.9 ± 2.3	18.2 ± 4.4	13.1 ± 4.1
	1600	16.5 ± 3.5	17.8 ± 2.5	25.8 ± 2.1	23.1 ± 2.3	16.8 ± 4.3	12.0 ± 3.6
WS	16	29.7 ± 4.9	25.0 ± 3.7	58.8 ± 20.8	57.6 ± 17.6	22.7 ± 8.2	35.3 ± 17.7
	80	15.9 ± 0.9	15.8 ± 0.6	17.9 ± 1.7	17.0 ± 1.6	15.9 ± 1.5	13.5 ± 2.6
	160	16.8 ± 0.8	16.8 ± 0.4	17.7 ± 0.8	17.2 ± 0.5	16.8 ± 1.4	14.7 ± 1.9
	800	22.3 ± 0.6	22.7 ± 0.7	23.6 ± 0.6	22.9 ± 1.3	21.5 ± 1.6	19.4 ± 2.1
	1600	20.1 ± 0.5	20.6 ± 0.6	21.3 ± 0.6	20.7 ± 1.1	19.5 ± 1.4	17.9 ± 1.7
Delaunay	16	46.7 ± 7.3	39.6 ± 9.1	72.7 ± 13.1	79.9 ± 8.8	32.3 ± 14.9	50.2 ± 21.7
	80	28.0 ± 3.1	28.3 ± 3.1	41.7 ± 3.9	38.3 ± 3.9	26.8 ± 5.8	21.8 ± 3.2
	160	25.1 ± 3.1	26.1 ± 3.7	38.2 ± 2.8	34.7 ± 3.7	25.2 ± 5.3	19.4 ± 3.8
	800	23.4 ± 2.9	25.3 ± 2.9	35.8 ± 2.1	31.9 ± 3.7	24.1 ± 5.2	18.7 ± 3.6
	1600	23.2 ± 2.9	25.2 ± 2.9	35.4 ± 2.1	31.5 ± 3.7	24.0 ± 5.2	18.5 ± 3.5

Table 6: Dijkstra Results. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

(a) Graph Accuracy

		GINE	GINE (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	73.4 ± 2.6	49.8 ± 10.8	94.6 ± 1.1	89.5 ± 1.0	81.7 ± 16.1	73.8 ± 1.6
	80	0.2 ± 0.2	0.0 ± 0.0	37.8 ± 6.9	3.3 ± 3.7	6.8 ± 6.1	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	5.2 ± 1.9	0.0 ± 0.1	0.3 ± 0.5	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
WS	16	51.6 ± 3.0	28.7 ± 9.9	76.4 ± 4.0	70.8 ± 2.4	60.4 ± 22.7	50.9 ± 5.6
	80	0.0 ± 0.0	0.0 ± 0.0	17.2 ± 2.8	0.4 ± 0.6	8.4 ± 7.4	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.9 ± 0.8	0.0 ± 0.0	0.2 ± 0.2	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Delaunay	16	66.6 ± 4.3	40.3 ± 10.4	93.0 ± 1.4	87.6 ± 0.7	74.4 ± 19.9	66.4 ± 3.3
	80	0.0 ± 0.0	0.0 ± 0.0	19.2 ± 4.2	0.4 ± 0.8	4.4 ± 4.5	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

(b) Node Accuracy

		GINE	GINE (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	98.0 ± 0.2	95.2 ± 1.8	99.6 ± 0.1	99.3 ± 0.1	98.5 ± 1.6	98.0 ± 0.1
	80	89.8 ± 1.1	62.4 ± 7.0	98.6 ± 0.3	94.2 ± 2.5	86.8 ± 15.4	32.9 ± 21.6
	160	84.3 ± 1.6	53.3 ± 6.2	97.2 ± 0.5	92.0 ± 2.3	76.0 ± 22.1	25.0 ± 17.4
	800	75.8 ± 2.2	40.4 ± 8.1	94.1 ± 0.6	87.1 ± 2.7	63.7 ± 27.7	17.7 ± 12.2
	1600	72.8 ± 2.3	36.9 ± 7.6	92.2 ± 0.7	84.5 ± 3.4	60.6 ± 27.7	16.4 ± 10.7
WS	16	95.4 ± 0.7	91.2 ± 3.5	98.3 ± 0.4	97.8 ± 0.2	95.8 ± 4.2	95.5 ± 1.0
	80	85.0 ± 1.4	55.3 ± 9.3	97.1 ± 0.2	85.8 ± 6.0	89.2 ± 14.1	36.3 ± 16.4
	160	79.9 ± 1.9	48.1 ± 8.3	95.4 ± 0.3	80.9 ± 7.0	83.9 ± 18.9	29.4 ± 16.1
	800	61.4 ± 4.0	38.6 ± 5.2	81.8 ± 1.2	60.5 ± 8.3	71.4 ± 20.4	27.3 ± 12.3
	1600	52.6 ± 4.1	35.6 ± 4.4	72.5 ± 6.0	52.4 ± 8.3	67.3 ± 17.7	26.6 ± 11.7
Delaunay	16	97.4 ± 0.4	94.2 ± 1.8	99.5 ± 0.1	99.2 ± 0.1	98.0 ± 1.9	97.4 ± 0.4
	80	81.6 ± 1.3	54.4 ± 7.8	97.6 ± 0.3	84.9 ± 6.8	90.4 ± 9.7	35.6 ± 17.8
	160	70.4 ± 2.6	45.2 ± 5.4	92.4 ± 0.7	72.8 ± 8.9	85.0 ± 10.0	29.5 ± 17.0
	800	46.5 ± 3.7	37.2 ± 4.1	62.7 ± 1.2	50.8 ± 4.6	60.2 ± 4.4	26.7 ± 14.4
	1600	39.9 ± 3.6	36.0 ± 4.1	51.0 ± 3.9	46.4 ± 3.1	50.0 ± 3.6	26.3 ± 14.1

Table 7: MST Results. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

(a) Graph Accuracy

		GINE	GINE (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	43.2 ± 4.6	29.7 ± 5.6	79.2 ± 4.3	69.9 ± 6.1	56.8 ± 15.9	24.5 ± 7.5
	80	0.0 ± 0.0	0.0 ± 0.0	2.0 ± 1.2	0.0 ± 0.1	0.6 ± 0.8	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
WS	16	30.0 ± 4.1	20.4 ± 5.0	73.2 ± 9.1	65.7 ± 8.8	44.4 ± 18.0	14.8 ± 5.4
	80	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.3	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Delaunay	16	43.0 ± 5.0	34.6 ± 6.0	78.8 ± 4.1	72.6 ± 5.2	58.7 ± 15.8	26.0 ± 7.5
	80	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.4	0.0 ± 0.0	0.1 ± 0.2	0.0 ± 0.0
	160	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

(b) Node Accuracy

		GINE	GINE (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	92.6 ± 0.8	89.6 ± 1.7	97.3 ± 0.4	96.4 ± 0.6	94.2 ± 2.3	87.5 ± 2.4
	80	79.1 ± 1.3	51.6 ± 4.5	89.1 ± 1.6	79.7 ± 3.8	70.7 ± 27.8	29.0 ± 6.7
	160	77.6 ± 1.7	49.5 ± 4.3	84.6 ± 1.7	75.6 ± 4.5	66.6 ± 28.2	25.7 ± 6.6
	800	74.5 ± 2.0	45.0 ± 4.2	75.7 ± 2.0	69.5 ± 5.5	58.9 ± 29.0	21.3 ± 6.4
	1600	72.9 ± 2.2	43.2 ± 4.0	71.9 ± 2.1	66.8 ± 5.1	56.0 ± 28.5	20.1 ± 6.3
WS	16	89.6 ± 1.4	86.0 ± 2.1	96.8 ± 1.0	96.1 ± 1.0	92.8 ± 2.8	82.0 ± 4.0
	80	75.3 ± 1.0	54.9 ± 6.2	82.5 ± 2.4	74.5 ± 3.9	67.4 ± 22.9	32.0 ± 7.3
	160	74.4 ± 1.4	52.7 ± 6.5	77.6 ± 2.6	72.5 ± 4.5	62.8 ± 23.2	29.6 ± 6.0
	800	73.0 ± 2.4	50.9 ± 6.4	67.4 ± 3.1	69.2 ± 4.4	53.5 ± 20.9	24.9 ± 7.3
	1600	72.8 ± 2.3	54.1 ± 6.9	65.1 ± 3.3	68.8 ± 5.9	52.5 ± 17.1	28.8 ± 8.7
Delaunay	16	92.8 ± 0.8	91.1 ± 1.5	97.4 ± 0.5	96.7 ± 0.5	94.7 ± 2.1	88.2 ± 2.1
	80	77.4 ± 0.6	58.4 ± 5.9	85.2 ± 1.5	77.7 ± 4.1	69.9 ± 22.1	34.2 ± 8.4
	160	75.8 ± 1.1	56.4 ± 5.6	78.5 ± 1.4	74.3 ± 5.0	62.6 ± 20.9	31.9 ± 7.1
	800	74.8 ± 1.7	55.0 ± 5.6	68.7 ± 1.0	71.4 ± 6.5	52.5 ± 13.4	28.0 ± 7.2
	1600	74.7 ± 1.7	54.9 ± 5.5	66.8 ± 0.9	71.0 ± 6.7	50.6 ± 11.3	27.8 ± 7.3

Table 8: MIS Results. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. All numbers are given as percentages.

(a) Graph Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	6.2 ± 3.2	3.3 ± 2.5	98.8 ± 0.2	98.6 ± 0.4	56.1 ± 13.1	44.1 ± 5.8
	80	0.0 ± 0.0	0.0 ± 0.0	89.2 ± 4.6	88.9 ± 3.1	5.5 ± 7.1	2.6 ± 1.5
	160	0.0 ± 0.0	0.0 ± 0.0	74.1 ± 10.1	76.2 ± 7.3	0.8 ± 1.6	0.1 ± 0.2
	800	0.0 ± 0.0	0.0 ± 0.0	10.7 ± 10.5	18.0 ± 8.6	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	2.0 ± 2.5	5.2 ± 4.3	0.0 ± 0.0	0.0 ± 0.0
WS	16	6.5 ± 2.8	4.4 ± 2.3	98.1 ± 0.6	98.2 ± 0.3	52.6 ± 14.6	46.5 ± 5.7
	80	0.0 ± 0.0	0.0 ± 0.0	84.4 ± 8.4	82.2 ± 7.4	9.0 ± 9.8	4.2 ± 1.1
	160	0.0 ± 0.0	0.0 ± 0.0	58.3 ± 14.1	54.1 ± 6.6	2.0 ± 2.9	0.4 ± 0.4
	800	0.0 ± 0.0	0.0 ± 0.0	4.6 ± 4.3	2.3 ± 1.7	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.5 ± 0.6	0.1 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Delaunay	16	6.1 ± 3.8	3.3 ± 2.2	98.9 ± 0.6	98.6 ± 0.4	56.0 ± 13.3	46.9 ± 6.2
	80	0.0 ± 0.0	0.0 ± 0.0	93.9 ± 2.2	92.2 ± 2.2	9.6 ± 7.8	4.8 ± 1.4
	160	0.0 ± 0.0	0.0 ± 0.0	87.2 ± 4.9	85.1 ± 3.4	1.7 ± 2.2	0.3 ± 0.3
	800	0.0 ± 0.0	0.0 ± 0.0	41.2 ± 8.9	40.1 ± 9.1	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	17.4 ± 7.4	15.1 ± 6.5	0.0 ± 0.0	0.0 ± 0.0

(b) Node Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	82.2 ± 2.5	79.9 ± 2.9	99.8 ± 0.1	99.8 ± 0.1	93.6 ± 2.2	92.2 ± 0.7
	80	81.6 ± 1.9	79.9 ± 2.2	99.6 ± 0.2	99.4 ± 0.1	90.0 ± 2.3	88.7 ± 1.7
	160	80.8 ± 2.4	78.2 ± 2.7	99.5 ± 0.2	99.4 ± 0.2	90.1 ± 2.5	88.3 ± 2.8
	800	83.6 ± 1.5	83.4 ± 0.8	98.8 ± 0.6	98.8 ± 0.5	87.9 ± 1.9	85.6 ± 3.5
	1600	80.8 ± 2.5	79.2 ± 1.6	98.9 ± 0.5	98.9 ± 0.7	88.2 ± 2.6	84.7 ± 5.5
WS	16	84.2 ± 2.1	83.1 ± 1.9	99.8 ± 0.1	99.7 ± 0.1	93.3 ± 2.2	92.9 ± 0.5
	80	82.0 ± 2.6	79.5 ± 3.4	99.4 ± 0.3	99.5 ± 0.2	92.6 ± 2.6	92.2 ± 1.9
	160	82.3 ± 2.4	79.8 ± 3.3	98.8 ± 0.6	99.1 ± 0.3	92.2 ± 2.9	91.8 ± 2.6
	800	84.3 ± 1.9	83.2 ± 2.2	95.8 ± 2.6	98.6 ± 0.8	91.8 ± 3.3	90.5 ± 4.3
	1600	83.4 ± 2.6	81.8 ± 2.6	93.3 ± 4.4	98.2 ± 1.3	91.4 ± 3.5	88.8 ± 6.0
Delaunay	16	82.5 ± 3.2	80.6 ± 3.1	99.9 ± 0.1	99.8 ± 0.1	94.3 ± 2.0	93.5 ± 0.7
	80	82.4 ± 3.0	80.6 ± 3.5	99.8 ± 0.1	99.7 ± 0.1	93.4 ± 2.0	92.4 ± 1.4
	160	81.5 ± 3.2	79.8 ± 3.6	99.8 ± 0.1	99.7 ± 0.2	93.0 ± 2.5	91.7 ± 2.2
	800	80.9 ± 3.7	78.9 ± 3.7	99.5 ± 0.2	99.4 ± 0.7	92.5 ± 3.0	89.8 ± 4.6
	1600	80.3 ± 4.0	78.2 ± 3.7	99.3 ± 0.3	99.1 ± 1.2	92.1 ± 3.4	88.0 ± 6.5

(c) Node F1

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	61.2 ± 7.0	52.1 ± 14.2	99.6 ± 0.1	99.6 ± 0.1	87.7 ± 4.3	84.9 ± 1.0
	80	48.2 ± 11.1	26.8 ± 17.7	99.1 ± 0.4	98.8 ± 0.3	76.5 ± 6.8	73.5 ± 5.2
	160	51.7 ± 11.1	29.4 ± 18.1	98.9 ± 0.5	98.7 ± 0.5	78.5 ± 6.6	74.4 ± 8.0
	800	29.5 ± 17.2	9.5 ± 8.1	96.4 ± 1.8	96.4 ± 1.6	61.8 ± 10.9	51.8 ± 19.0
	1600	41.1 ± 14.6	18.3 ± 13.7	97.3 ± 1.3	97.4 ± 1.7	70.6 ± 9.4	60.1 ± 19.3
WS	16	57.5 ± 7.2	50.8 ± 13.1	99.4 ± 0.3	99.3 ± 0.2	84.0 ± 5.0	82.4 ± 1.1
	80	63.5 ± 7.1	50.5 ± 17.4	99.0 ± 0.6	99.0 ± 0.4	85.6 ± 5.3	85.2 ± 3.5
	160	61.7 ± 8.6	45.1 ± 20.1	97.7 ± 1.1	98.2 ± 0.5	83.9 ± 6.7	83.7 ± 5.2
	800	52.7 ± 12.8	34.0 ± 19.9	90.8 ± 5.3	96.3 ± 1.9	77.5 ± 10.5	75.2 ± 11.5
	1600	53.2 ± 13.1	32.5 ± 19.9	86.1 ± 8.7	95.7 ± 2.9	77.9 ± 10.4	72.1 ± 15.8
Delaunay	16	62.7 ± 8.3	56.5 ± 14.0	99.8 ± 0.1	99.7 ± 0.1	89.3 ± 3.6	87.5 ± 1.0
	80	60.5 ± 7.8	46.3 ± 20.0	99.6 ± 0.2	99.5 ± 0.2	86.5 ± 4.5	84.9 ± 2.9
	160	58.1 ± 9.7	40.9 ± 22.4	99.6 ± 0.2	99.4 ± 0.3	85.5 ± 5.9	83.4 ± 4.7
	800	56.5 ± 11.1	37.0 ± 23.1	99.1 ± 0.4	98.8 ± 1.4	84.3 ± 7.3	79.0 ± 10.3
	1600	55.2 ± 11.6	35.0 ± 22.9	98.6 ± 0.7	98.2 ± 2.5	83.4 ± 8.2	75.1 ± 15.0

Table 9: Eccentricity Results. The table shows also the standard deviation across the 5 runs. Runs marked with *(H)* are trained with hints. Graph Accuracy is given as percentages and Graph MSE is the *Mean-Squared-Error* (lower is better).

(a) Graph Accuracy

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	57.3 ± 21.2	25.3 ± 41.0	100.0 ± 0.0	100.0 ± 0.0	75.8 ± 26.2	95.0 ± 6.3
	80	77.1 ± 17.5	23.8 ± 39.0	100.0 ± 0.0	100.0 ± 0.0	80.5 ± 35.0	96.6 ± 3.6
	160	72.3 ± 18.0	26.1 ± 36.8	100.0 ± 0.0	100.0 ± 0.0	75.0 ± 39.1	95.8 ± 4.6
	800	51.3 ± 34.2	17.1 ± 32.9	100.0 ± 0.0	100.0 ± 0.0	72.7 ± 27.9	93.4 ± 10.3
	1600	36.7 ± 17.6	16.0 ± 21.7	64.6 ± 14.9	83.0 ± 6.5	63.0 ± 24.8	72.1 ± 20.9
WS	16	78.0 ± 18.7	25.3 ± 42.2	100.0 ± 0.0	100.0 ± 0.0	86.7 ± 25.7	99.0 ± 1.2
	80	27.6 ± 19.5	19.0 ± 18.8	93.8 ± 2.1	88.3 ± 1.8	60.8 ± 29.1	66.4 ± 22.4
	160	3.6 ± 8.0	18.6 ± 18.9	100.0 ± 0.1	100.0 ± 0.1	57.4 ± 38.7	46.2 ± 40.9
	800	0.0 ± 0.0	4.6 ± 8.9	25.6 ± 7.5	34.8 ± 7.2	27.6 ± 29.4	14.1 ± 6.3
	1600	0.0 ± 0.0	9.8 ± 10.2	5.2 ± 3.3	9.2 ± 4.8	15.2 ± 13.7	8.3 ± 4.9
Delaunay	16	84.8 ± 12.4	24.8 ± 42.5	100.0 ± 0.0	100.0 ± 0.0	89.9 ± 19.4	99.6 ± 0.8
	80	0.0 ± 0.0	17.0 ± 12.5	100.0 ± 0.0	99.7 ± 0.3	25.2 ± 37.6	51.0 ± 36.0
	160	0.0 ± 0.0	3.0 ± 5.8	76.9 ± 19.8	64.4 ± 14.2	8.3 ± 11.9	19.4 ± 11.7
	800	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	1600	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

(b) Graph MSE

		GIN	GIN (H)	PGN	PGN (H)	RecGNN	RecGNN (H)
ER	16	47.7 ± 27.6	186.8 ± 154.6	0.0 ± 0.0	0.0 ± 0.0	14.8 ± 9.6	6.2 ± 5.2
	80	26.0 ± 20.5	248.5 ± 192.5	0.0 ± 0.0	0.0 ± 0.0	11.1 ± 11.8	5.2 ± 3.7
	160	43.9 ± 34.2	274.6 ± 211.9	0.1 ± 0.1	0.1 ± 0.0	13.7 ± 11.5	5.0 ± 3.6
	800	47.3 ± 40.1	442.7 ± 390.5	1.4 ± 2.3	0.2 ± 0.0	15.0 ± 11.2	6.3 ± 5.6
	1600	93.4 ± 43.7	357.5 ± 210.3	22.7 ± 5.9	12.2 ± 0.8	33.7 ± 13.9	29.0 ± 21.9
WS	16	18.8 ± 11.0	220.6 ± 181.6	0.0 ± 0.0	0.0 ± 0.0	10.2 ± 9.4	2.9 ± 2.2
	80	501.0 ± 301.6	524.5 ± 277.9	153.3 ± 102.0	87.7 ± 28.9	98.9 ± 64.1	306.6 ± 207.0
	160	348.4 ± 249.7	382.3 ± 581.6	2.1 ± 1.7	0.4 ± 0.2	31.2 ± 23.1	84.8 ± 70.7
	800	2381.8 ± 1355.2	1815.2 ± 1315.6	1699.8 ± 1411.3	578.5 ± 435.4	527.5 ± 391.6	1797.0 ± 918.4
	1600	2555.7 ± 1518.2	1894.6 ± 1421.7	1706.9 ± 1415.3	632.4 ± 468.3	573.9 ± 251.9	1761.0 ± 1114.8
Delaunay	16	11.4 ± 6.8	217.6 ± 173.1	0.0 ± 0.0	0.0 ± 0.0	8.4 ± 9.2	2.0 ± 1.3
	80	590.4 ± 403.6	524.7 ± 754.2	2.8 ± 1.3	1.0 ± 0.5	73.1 ± 54.1	58.1 ± 74.8
	160	1706.9 ± 1025.6	1523.9 ± 1956.7	131.0 ± 132.2	100.0 ± 93.4	238.9 ± 218.4	552.6 ± 456.0
	800	11779.4 ± 6340.3	12865.6 ± 7419.7	11461.5 ± 4652.8	8983.4 ± 4977.5	6091.6 ± 2161.0	13097.1 ± 2412.4
	1600	24314.6 ± 13344.1	34168.0 ± 13313.6	26683.1 ± 7749.0	22210.4 ± 7224.4	18734.3 ± 1985.7	29640.9 ± 6097.9