Report

1 Introduction

An independent set of a graph is a set of nodes such that no two nodes in the set are neighbours (i.e., no two nodes in the set should have an edge connecting them). Finding a Maximum Independent Set (MIS), with the largest number of nodes possible, is NP-hard; that is, there is no known polynomial-time solution. Independent sets are applicable in market graphs, coding theory, wireless networks [1], cryptography and scheduling (where conflicts need to be avoided). For example, consider the problem of choosing a set of non-intersecting events to timetable from many possible events. We can represent each event as a node, and draw an edge between two events if they cannot occur together. Any valid set of events must be an independent set, and a set with the most number of nodes possible is an MIS.

Exact algorithms can currently solve the MIS problem in $O(1.1996^n n^{O(1)})$ time [2], where n is the number of nodes. For near-optimal solutions, researchers have recently tried using machine learning on NP-hard problems. Li *et al.* [3] has tried supervised learning on the same problem of MIS, while many other papers have explored using various machine learning methods to solve various NP-hard problems, like Maximum Clique, Maximum Cut, Travelling Salesman Problem etc. While machine learning does not make for exact algorithms and provides no guarantees of optimality, it runs in polynomial time. In particular, Abe *et al.* [4] has shown that its models trained on small graphs scale well to larger graphs.

Abe *et al.* introduced using Reinforcement Learning (RL) to solve the problem, using a combination of Monte Carlo Tree Search (MCTS) and Graph Neural Networks (GNNs). The GNN predicts the probability of each node being in a maximum independent set, and the tree search then explores and removes nodes until the graph is empty. The GNN guides the search's choice of nodes to explore, and is in turn trained by the tree search's results, the reward being the size of the maximal independent set.

Finding a MIS can be framed as a Markov Decision Process, where each action is selecting a node and removing it and its neighbors, each state is the graph with some nodes removed. This greedy process returns a *maximal* (but not necessarily maximum) independent set when all nodes have been removed, and thus MIS can be estimated with a tree search. Using MCTS allows the model to explore the search space by itself, without need for labelled data, which can be hard to obtain for NP-hard problems like the maximum independent set. Thus, the methods in this work and [4] can be applied to similar problems that can also be expressed as Markov Decision Problems. While RL is currently outperformed by the exact algorithm in

[2] and the supervised model in [3], we hope that RL can be competitive enough in the future, and provide solutions for other NP-hard problems without domain knowledge, handcrafted heuristics or training data.

While Abe *et al.* showed that the models' performance plateaued after less than 50 epochs on 100-nodes-250-edges and 500-nodes-1250-edges graphs most of the time, and tried training the network with different parameters, it did not explore the effect such changes had extensively. Our contribution is to run the experiments on harder and more diverse graphs than Abe *et al.* Throughout this work, we measure models' performance on graphs by the predicted independence number for the graph, $\alpha(G)$ (the size of an MIS).

2 Preliminaries

Given a graph, the GNN predicts two values, P and V for each node in the graph, an estimate for the probability of the node being in an MIS and the resultant $\alpha(G)$ if the node were to be chosen respectively. For unexplored nodes, the MCTS then initialises Q, the estimated $\alpha(G)$, with V.

The tree search then chooses the node with the highest probability, π, of being in the MIS, and removes it and its neighbors from the graph state. The above process repeats until the graph is empty (i.e., no more nodes can be added to the independent set)

To estimate π and Q for every node in a graph state, the model rolls-out a number of times from the current graph state, and π is set based on the number of times the node is explored.

3 Modifications to Training

Abe *et al.* observed that training on larger graphs improves performance but also takes much more time. While Abe *et al.* took only 9 hours for training with 18 cores, if larger training graphs were to be used, training time would have to be optimised. Given that training time and performance are both concerns, we explore various improvements to the training, chief of which are curriculum learning and adding weights in the algorithm, and try to improve performance while decreasing or maintaining training time.

3.1 Decreasing Number of Rollouts

We tried decreasing the number of rollouts for every node sampled, so that each node could be more quickly sampled and each epoch could train faster.

3.2 Weighting Q

The Upper Confidence Bound (UCB) [5] is the formula that decides which node is sampled in rollouts to balance exploration and exploitation. Abe *et al.* set UCB $= U + Q$, where $U(\text{state}, \text{action})$ favours less-explored nodes, and $Q(\text{state}, \text{action})$ favours promising nodes that have been explored before relative to their alternatives. For explored nodes, Abe *et al.* set $Q(\text{state}, \text{action})$ to the average $\alpha(G)$ thus far. Hence, the GNN would have no influence on nodes that have already been explored. try to improve the model, we replaced Q with a weighted average of Q and V, to let the model rely partly on the GNN too, instead of relying only on the MCTS's past explorations on the graph.

However, the GNN might not have learnt much initially, and we should not let it influence the node exploration-exploitation that much. Hence, we tried starting each model with a higher Q-to-V ratio, and decreasing the Q-to-V ratio with later epochs (see Subsection 5.2 for details). Initially, the higher Q-to-V ratio will weight the GNN's prediction more. As we proceed, we lower the Q-to-V ratio and weight V more.

3.3 Curriculum Learning

In machine learning, curriculum learning [6] refers to exposing a model to easier, simpler examples before moving on to harder ones, similar to how humans learn. We tried changing the training graphs the models were trained on to incorporate curriculum learning, by exposing the model to increasingly harder graphs. We considered two aspects of graph difficulty: number of nodes and graph density.

Varying Number of Nodes The original paper used graphs with the same number of nodes and edges for testing. It claimed that larger training graphs (with more nodes) were more helpful, but estimated that the training has a time complexity of $O(mS)$, where m is the number of edges, and S is the MIS size. Hence, larger graphs can take much more time to train on. Thus, we increased the number of nodes for the training graphs over time, to first let the graph learn from easier graphs that we hoped would prove easier, before training on larger graphs, as a compromise on training efficiency.

We then tried varying the density of the training graphs (how well connected the nodes of the graph are) over the epochs, since graphs' difficulty and characteristics may vary based on their density.

Varying Graph Density Another aspect of difficulty is graph density. In dense graphs, choosing a node to be in an Independent Set eliminates its many neighbours, and the MIS is small and more easily solved. In sparse graphs, the GNN can learn efficient reduction rules like its exact algorithm counterparts, and both exact algorithms and [4] are found to perform

well. Hence, the MIS problem has a phase transition midway with respect to graph density; i.e. graphs that are neither too dense nor too sparse are hardest to solve. Coja-Oghlan *et al.* [7] showed the phase transition occurs when $\alpha(G) = n\ln(k)/k$, where $k = 2m/n$ and n, m are the number of nodes and edges respectively. Thus, we tried varying the density of the training graphs with time, alternating between easier sparse and dense graphs before approaching the density at which MIS is harder.

4 Experiments

4.1 Environment

The results in Figure 1 and 3 were run on an Intel(R) Core(TM) computing system (CPU: i7-7700, 3.60 GHz, 4 cores). All other experiments were run on an Intel(R) Xeon(R) computing system (CPU: E5-2620v4, 2.1 GHz; 32 cores). GPU usage is negligible, due to the small neural networks being used on only one graph's data at a time. All random graphs below are Erdős-Rényi random graphs [8]. Unless otherwise specified, all time mentioned below is in terms of total computing time (across threads), and all testing below was run for 10 minutes. Our models and code are available online at https://github.com/c-j-lh/MIS_solver.

4.2 Dataset

Abe *et al.* only used two sets of training graphs: those with 100 nodes and 250 edges, 200 nodes and 500 edges and 500 nodes and 1250 edges. Throughout this paper, we generated larger testing graphs of 200 nodes or more. We also used harder graphs from the dataset [9], generated from a specific algorithm [10] meant to hide optimum solutions. Note that this dataset comprises larger graphs that take more time to train on. We found that GNNs trained with different parameters could be differentiated by their performance on larger graphs.

5 Results and Discussion

5.1 Decreasing Number of Rollouts

We tried decreasing the number of rollouts for every node sampled, so that each node could be more quickly sampled and each epoch could train faster. We then increased the number of epochs so that total training time remained the same. This only worsened or maintained

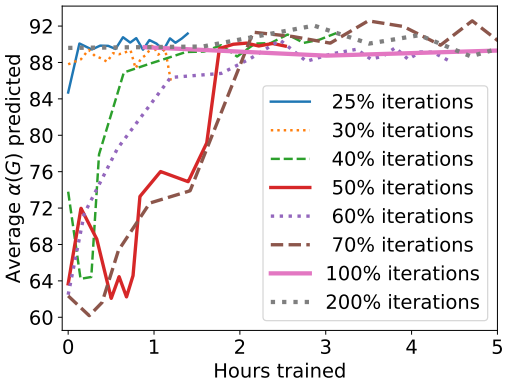

Figure 1: Graph of performance against time for different number of iterations

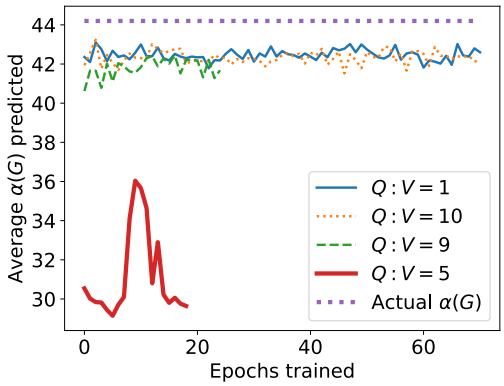

Figure 2: Graph of performance against time for different $Q : V$ ratios

the performance, as seen in Figure 1. Hence, we postulate that the paper's original formula of number of rollouts $= \min(500, \max(50, 2n))$ is rather ideal.

5.2 Weighting Q

We weighted Q to include the GNN-predicted V, so that the GNN can influence the MCTS more. Since each graph is unique, we still weight Q more heavily and weight V lightly. From a bit of testing (see Figure 2), we found that a Q-to-V ratio of 10 (i.e., UCB $= U + (\frac{10}{11}Q + \frac{1}{11}V)$) does not impede training, while training deteriorates under an overly small ratio.

We then tried training various models with dynamically weighted Q. Note that no weighted-Q model performed better than the model run with default parameters. Hence, weighting Q does not help. We postulate that weighting Q does not benefit the model, and that weighting $Q : V$ too low will cause V to disrupt the training, as seen in the "$Q : V = 5$" model in Figure 1.

5.3 Curriculum Learning: Varying Number of Nodes

Train100, Train200 and Train300 were trained only on graphs of 100, 200 and 300 nodes respectively. Curriculum was trained on graphs of 100 nodes to 295 nodes, in increments of 5 nodes. This proved not to be better than models trained on graphs of 100 nodes, 200 nodes or 300 nodes only.

Comparing Train200 and Train300 with Train100, we observe that training on 200-nodes or 300-nodes graphs right from the start did not help and only slowed down training, possibly because the 200 and 300-nodes graphs are too large for the model to learn from initially. We

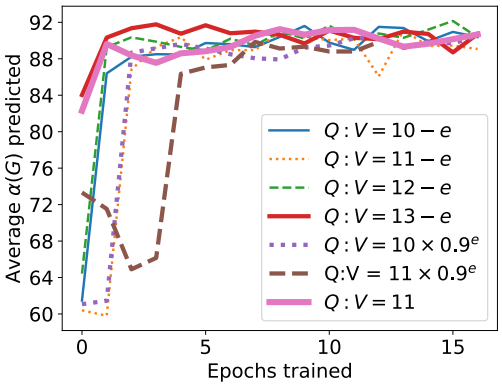

Figure 3: Graph of performance against epochs for many dynamically weighted $Q : V$

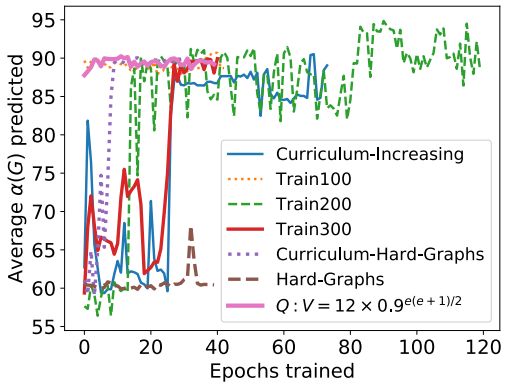

Figure 4: Graph of performance against epochs for curriculum models and a model with weighted $Q : V$

In the line labels above, $e = \lfloor \frac{\text{epoch}}{20} \rfloor$

postulate that 100-nodes graphs are quite ideal for training, and that changing the node size without changing the density does not improve the model much because the tree search is already training on smaller subgraphs with similar density when a node is removed.

5.4 Curriculum Learning: Varying Graph Density

The hard dataset [9] has 8 graph types with different number of nodes and edges and $\alpha(G)$, with 5 instances for each type. For Hard-Graphs, we trained the model on all 40 instances of the hard dataset, on increasingly large graph types. For Curriculum-Hard-Graphs, we randomly added and removed 1/10 of the number of edges to form sparser and denser versions of the hard dataset. For each graph type, ordered increasingly, we took the first instance, and trained on the graph with 2/10 fewer edges, with 2/10 more edges, with 1/10 fewer edges, with 1/10 more edges, and finally on the original graph.

This proved more effective than normal, as can be seen from comparing Hard-Graphs and Curriculum-Hard-Graphs in Figure 4. Hence, graph density may be the more appropriate indicator of difficulty, or more suited for curriculum learning in this case.

6 Characterisation of GNN

Lastly, we attempted to further study what the GNN learns, though this has not managed to inform our improvements. After sufficient training, the GNN mostly chooses nodes of low

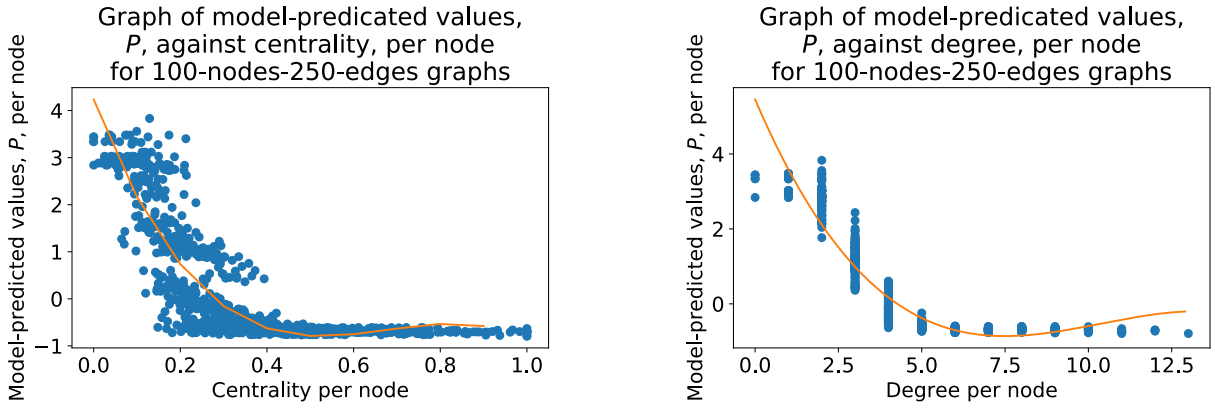

Figure 5: Graphs of GNN-predicted P against centrality and degree

degree, and rates them with higher probability of being in an MIS. There is also an inverse correlation between the probability of being in an MIS, P, predicted by the GNN and the two node attributes of degree and eigenvector centrality (see Figure 5). This makes sense, since low-degree and less central nodes exclude fewer other nodes from being in a maximal independent set, and are more likely to be in the MIS. (However, note that nodes with lower degree are not *always* favoured over nodes with higher degree, and that the GNN cannot be summarised by this relationship.)

In addition, we postulate that the GNN further observes the second-order neighbours of nodes. For two nodes of the same degree, the node with neighbours of lower degree seems to be favoured. Further exploring how the GNN's internal workings shift with different graph types (number of nodes and graph density) may provide insight for improving the model.

7 Conclusions and Extensions

We have shown that curriculum learning on graph density may improve training, and explored miscellaneous methods like weighting Q and changing the number of iterations in an epoch.

Comparing convergence rate and final performance for different training datasets and curricula (e.g., with different density or number of nodes and edges) may also reveal what an optimal training dataset or curriculum might be. While Table 4 in Abe *et al.* has shown that the models trained generalise well to larger graphs, more thorough testing might reveal whether models trained on one graph type (e.g., sparse graphs) adapt well to other graph types (e.g., dense graphs). Testing on the Cora, Citeseer and Pubmed citation network datasets [11] used in Abe *et al.* may also give a more conclusive evaluation of the changes.

References

[1] S. Butenko, "Maximum independent set and related problems, with applications", PhD thesis, University of Florida, 2003.

[2] M. Xiao and H. Nagamochi, "Exact algorithms for maximum independent set", *Information and Computation*, vol. 255, pp. 126–146, 2017.

[3] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search", in *Advances in Neural Information Processing Systems*, 2018, pp. 539–548.

[4] K. Abe, Z. Xu, I. Sato, and M. Sugiyama, "Solving NP-Hard problems on graphs by reinforcement learning without domain knowledge", *arXiv preprint arXiv:1905.11623*, 2019.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[6] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning", in *Proceedings of the 26th Annual Iternational Conference on Machine Learning*, ACM, 2009, pp. 41–48.

[7] A. Coja-Oghlan and C. Efthymiou, "On independent sets in random graphs", *CoRR*, vol. abs/1007.1378, 2010. arXiv: 1007.1378. [Online]. Available: http://arxiv.org/abs/1007.1378.

[8] P. Erdős and A. Rényi, "On the evolution of random graphs", *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[9] K. Xu. (2004). BHOSLIB: Benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring) —— hiding exact solutions in random graphs, [Online]. Available: http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm (visited on 12/20/2019).

[10] ——, (2004). Forced satisfiable CSP and SAT benchmarks of model RB ——hiding solutions with growing domains, [Online]. Available: http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm (visited on 12/29/2010).

[11] LINQS group at University of California, Santa Cruz. (1999). LINQS, Statistical Relational Learning Group, [Online]. Available: https://linqs.soe.ucsc.edu/ (visited on 12/29/2010).

Appendices

A Modifications to Testing

We also tried to improving testing, to improve the results obtained given the same models.

A.1 Combining two GNNs

We tried combining the predictions of 2 differently-trained models for choosing nodes, to see if they could complement each other. To choose each node, we averaged the two GNNs' π values. [1] However, we did not try to backpropagate the results, which can be an area for future research. This combination was conclusively worse than its two parents, as seen in Figure 2.

A.2 Dynamically Testing

We also tried training on the test graphs before testing on them, in the hope that the backpropagation and updating of the GNN during training would improve the GNN's performance on the same graph. We logged the time taken for training and testing, to see if pre-training can be a viable strategy if enough time is allocated to solving a particular graph. We tried the above with two learning rates. As seen in Table 1, however, training on the test graphs did not improve performance.

B Dynamically testing: details

We ran a model pre-trained for 237 epochs for an additional 9 epochs on a 200-node-500-edges graph, and re-tested the model before each additional epoch. The learning rate for training is controlled by a parameter, τ. We kept to Abe *et al.*'s formula of $\tau = 10 \times 0.98^t$ for epoch t. We then tested the following for two learning rates (LRs), a higher LR assuming the model was new and had been trained for 0 epochs (i.e., $t = 0, 1, \ldots, 8$), and a lower LR taking the model as is and assuming it had been trained for 237 epochs (i.e., $t = 237, 238, \ldots, 245$).

π is set to $N^{1/\tau}$, where $N(\text{state}, \text{action})$ is the number of times a node has been explored so far in the MCTS (see Abe *et al.*, Appendix C for pseudocode). Hence, a higher learning rate (lower τ) would mean that frequented notes are not that much more likely to be explored

[1] In lines 3 and 4 of Abe *et al.*, Appendix C, Algorithm 3, the probability π is calculated for the two models, and the vertex a sampled is the vertex with the highest average π.

Epoch	Higher LR	Lower LR
0	85.5	84.7
1	84.8	84.3
2	85.2	84.0
3	83.0	85.2
4	83.4	84.4
5	84.4	83.7
6	84.5	83.9
7	82.2	83.8
8	83.8	82.8
9	85.4	85.2

Table 1: Average $\alpha(G)$ for different Learning Rates (LRs)

than ignored nodes, favouring exploration of unvisited nodes to the expense of exploiting frequented nodes, and allowing the model to learn faster from its more unconventional explorations. However, the different learning rates do not seem to affect the results significantly. Perhaps more extensive testing would reveal the effect of this change.

C General details

To be more specific than what was mentioned in Section 2, The vector V does not represent $\alpha(G)$ for each action, but the $\alpha(G)$ *normalised* with respect to other possible actions (i.e., after subtracting the mean $\alpha(G)$ and dividing by the standard deviation of $\alpha(G)$ for all possible actions at that point in time).

Test Graph			Testing Method		
n	m	ID	Train200	Train300	Combination
100	250	0	42	42	42
		1	42	42	44
		2	41	40	41
		3	45	44	45
500	1250	0	206	210	209
		1	213	211	215
		2	205	205	208
		3	212	215	213
		4	206	204	210
		5	208	206	210
		6	210	206	209
		7	207	205	207
		8	207	202	205

Table 2: Table of $\alpha(G)$ predicted on random graphs