# Curriculum Learning and Other Improvements to Solving Maximum Independent Set with Reinforcement Learning

Chieu Le Heng* Dr Chai Kian Ming Adam†
Mr Wee Liang Chi†

December 19, 2019

## 1 Abstract

## 2 Introduction

Finding a Maximum Independent Set (MIS) is an NP-hard problem. An independent set of a graph is a set of nodes such that no two nodes in the set are connected. Finding an independent set of the maximum size is NP-hard. Exact algorithms can currently solve MISes in O(1.1996nnO(1)) time, where n is the number of nodes. [https://arxiv.org/pdf/1312.6260.pdf] [https://arxiv.org/pdf/1905.11623.pdf] introduced using Reinforcement Learning to solve the problem, using a combination of Monte Carlo Tree Search (MCTS) and Graph Neural Networks (GNN). 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 cardinality of the maximal independent set). Reinforcement learning does not make for exact algorithms, though it runs in polynomial time. The MISes predicted by [] were optimal for 100-nodes-250-edges Erdős-Rényi graphs, but diverged form optimality with larger graphs. Specifically, the testing runs in O() time, and training in O(mS) time. The paper trained each model for 9 hours with 18 cores, and tested each model for 10 minutes on each graph. While the paper showed that the models' performance plateaued after less than 50 epochs on 100-nodes-250-edges graphs, we found that the models had room for improvement with respect to their performance on larger and different graphs. Given that time and performance are both concerns, we explore various improvements to the training, chief of which

*NUS High School of Math and Science, 20 Clementi Ave 1, Singapore 129957
†DSO National Laboratories, 12 Science Park Dr, Singapore 118225

are curriculum learning and adding weights in the algorithm, and try to improve the models' performance while keeping track of training and testing time.

# 3 Modifications

## 3.1 Decreasing 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 $_{t}heperformance(seeappendix_{)}<graphofperformanceagainsttimefordifferentN_{i}ter\backslash percentages">$<tableofperformanc

## 3.2 Curriculum Learning: Number of Nodes

The paper suggested that the models might be better trained with larger training graphs, but that the training has time complexity of O(mS), where m is the number of edges, and S is the MIS size. Hence, we tried training the model on smaller graphs first, before gradually increasing the size of the training graphs. First, we tried slowly increasing node size. In particular, we trained on graphs of 100 nodes, 105 nodes, 110 nodes ..., 295 nodes. This proved $_{t}hanmodelstrainedwiththesametimebutongraphsof100nodes,200nodesor300nodesonly.$

## 3.3 Curriculum Learning: Graph Density

Second, we tried moving graph density slowly toward the phase transition mentioned in [Subsection: Testing]. We randomly added and removed edges to form sparser and denser versions of the hard dataset. This proved $_{t}hannormal.$

insert table

## 3.4 Simultaneously testing (failed; perhaps omit)

We tried combining 2 GNNs, but perhaps because it is hard to backpropagate results with 2 GNNs, this combination was conclusively worse than its 2 parents ¡ref $\mathrm{m200}_0 - n200_1 columninGoogleSheets/appendix_>$

## 3.5 Dynamically testing

We tried training on graphs before testing on them. Ran 10 epochs, and tested after every epoch. It did not help $_{v}eryconfusing_{.Prelimsshowithelps,butattheexpenseoftime.Giventhesametime,willtrainingsom}$

## 3.6 Weighting Q (of UCB)

Given a graph, the GNN predicts 2 vectors for each node: the probability P and the value V. The Upper Confidence Bound (UCB) is the formula that decides which node is sampled in rollouts to balance exploration and exploitation. The paper set UCB = U + Q, where U(state, action) favours less-ex. GNN to

evaluate new nodes, and set Q(state, action) to V(state, action). The choice of UCB was then based entirely on Q. Hence, the GNN would have no influence on nodes that have already been explored This UCB formulation does not utilise what the GNN has learned. Hence, we weight 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, we found a Q:V ratio of around 10:1 to be optimal I.e. UCB = U + (10/11 Q + 1/11 V). Henceforth, we define the Q:V ratio to be the weights assigned to Q and V. I.e. A Q:V ratio of n:m UCB = U + (n/(n+m)U + m/(n+m)V)

¡Revise labels and titles¿ We then tried starting each model with a higher Q:V ratio, and decreasing the Q:V ratio with later epochs. At first, the GNN might not have learnt much, and we should not let it influence the node exploration-exploitation that much. Hence, we use a higher Q:V ratio (i.e. weight the GNN's prediction, V, less). As we proceed, we lower the Q:V ratio (weight V more). This approach trains the model to better standards.

# 4 Setup

## 4.1 Environment

All time mentioned below is in terms of total computing time (across threads). The experiments were run on an Intel(R) Xeon(R) computing system (CPU: E5-2620v4, 2.1 GHz; 32 core; GPU: NVIDIA-SMI 418.67, GeForce GTX). GPU time 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 unless otherwise specified. Our model and code is available at https://github.com/c-j-lh/temporary-project. Testing: Harder and More Varied Graphs Throughout this paper, we used harder graphs, which might take more time. We used the dataset [http://sites.nlsde.buaa.edu.cn/ kexu/benchmarks/graph-benchmarks.htm], and generated larger testing graphs of 200 nodes and above. We found that the GNNs trained did not plateau after fewer than 50 epochs, as was found in the original paper, once larger graphs were used for evaluation.

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. 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. ¡phase transition of MIS?¿ Following Results

# 5 Characterisation

We attempted to study what the GNN learns, though this has not managed to inform our improvements. After sufficient training, the GNN mostly chooses nodes of low degree, and rates them with higher probability of . There is an

inverse correlation between node degree/eigenvector centrality and probability of being in an MIS, P, predicted by the GNN.

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.

# 6 Applications and Generalisations

MIS has applications The approaches above can be generalised to other graph problems, e.g. maximum clique (discounting minimum vertex cover, which is just a complement of MIS).

# 7 Extensions

-