249 Project Proposal: Training Schedule for Graph Representation of Neural Algorithmic Reasoning

Alexander Taylor
Department of Computer Science
UCLA
ataylor2@cs.ucla.edu

Ted Zadouri
Department of Computer Science
UCLA
tedzadouri@g.ucla.edu

Jingdong Gao
Department of Computer Science
UCLA
mxuan@ucla.edu

Yihang Guo
Department of Computer Science
UCLA
yihangguo@g.ucla.edu

1 Problem and goal

Neural algorithmic reasoning applied to graph algorithms is a recent extension of the developing neural algorithmic reasoning field. Recent works [1; 2] have shown promising results in capturing the behavior of classical graph-based computer science algorithms by taking advantage of the additional information provided by measuring intermediate steps taken by a given algorithm, or subroutines, as opposed to simply measuring the accuracy of the model's prediction for the last step. However, contrary to the informing intuition[1; 2], achieved suboptimal results across transfer learning in both parallel and sequential algorithm tasks, leading to the authors' hypothesis that the expressiveness of the Message-Passing Neural Networks (MPNNs) used negatively impacted the performance on this task and that the initial random weightings were too far from an optimal solution.
We believe that it is possible to improve upon the baseline set by [1; 2] by developing a novel training scheduling approach based on [3] that would allow us to dynamically adjust the sequence of algorithms and accompanying graphs used in training as well as the weight of each instances updates on the model's weights. Solving the problem of leveraging shared subroutines across graph-based problems is of great importance due to its potential downstream applications as well as expanding the field of neural algorithmic reasoning, by establishing an improved baseline method for the execution of graph-based algorithms utilizing intermediate steps. We also believe that making use of intermediate steps in the learning process will improve the explainability of neural algorithmic reasoning tasks and will make it easier for future work to further extend this task.

2 Data plan

- We will utilize the DeepMind CLRS benchmark [4], which was built by the authors of [1; 2]. The CLRS automatically generates input/output pairs for each algorithm, alongside execution trajectories that expose the internal states of the algorithms, which we will use to produce the training, validation, and testing instances for each algorithm training or validation/testing task. We will use the following graph algorithms for our tasks:
  1. Breadth-First Search (Parallel)
  2. Bellman-Ford (Parallel)
  3. Widest Path (Parallel)
  4. Most Reliable-Path (Parallel)
  5. Prims (Sequential)
  6. Dijkstra (Sequential)
  7. Depth-First Search (Sequential)
  8. Widest Path (Sequential)
  9. Most Reliable-Path (Sequential)

For learning tasks under parallel algorithms, nodes can exchange messages with their neighbors until convergence. For learning tasks under sequential algorithms, the nodes will be greedily popped from the priority queue and the neighbor nodes will be updated.

3 Solution Plan

- Our experimental settings will be based on [1; 2]. Our training setting will consist of tasks to train on each algorithm $A = \{a_1, ..., a_n\}$ and tasks to validate the performance of the model on a set of algorithms $B = \{b_1, ..., b_m\}$ under settings where A and B are equivalent and where they are disjoint. Our testing setting will consist of inference tasks given algorithms $C = \{c_1, ..., c_l\}$ under the same conditions as outlined in [1; 2] and further specified below. We will utilize the same model architecture as in the previous work, i.e. Max-MPNNs, under a multi-task learning, encode-process-decode pipeline where each algorithm learning task has its encoder f_A and decoder d_A and each algorithm learning task updates the same processor P_A.
  The training phase will begin with the initialization of our scheduler. Our scheduler consists of a transfer matrix $W_t^{K \times K}$ that will be initialized randomly or based on the results of warm-up training epochs for each algorithm. We will then train the processor network P_A under the same settings outlined by [1; 2] and update $W_t^{K \times K}$ using the aggregation of the product validation loss and the feature vector gained from the validation algorithm tasks, which will be used to determine the order and frequency of the training examples for each algorithm learning task.
- We believe this solution addresses the shortcomings of [1; 2] outlined above and stated by the authors by leveraging the expressiveness of MPNNs more effectively by dynamically increasing the frequency of relevant training examples to improve the generalizability of the model. Additionally, we hope to account for the concern in [1; 2] of initial weightings being too distant from the optimal solution by incorporating experiments utilizing both random initialization of our scheduler and initialization based on the results of warm-up training epochs for each algorithm in the training set. Our intuition is that this weighting will improve the scheduler performance at earlier epochs and mitigate the effects of random initialization distant from the optimal solution.

4 Evaluation Plan

- We intend to evaluate our solution by utilizing the Next node, Predecessor, Key [2] error rates. The Next node measures whether the next node is the correct one to pick demonstrating whether the correct algorithm is executed. Key measures whether the key of the current node is picked correctly and Predecessor measure whether the task at hand is completed with success; they both measure the correctness of the solution.
- The baselines will be neural execution (NE) [1] and NE++ [2], where each task has its own encoder and decoder, with their associated variations such as the four distinct transfer methods: Freeze weights, Fine-tune weights, 2-Processor, and transfer via multi-task learning. Additionally, each experiment will be tested on a combination of different number of nodes, for instance, 20, 50, and 100.

5 Schedule

Week 4: Proposal writing (All), Code Request set to authors of the original papers
Week 5-6.5:

- Reproduce relevant results from the original papers [1; 2] (All)
- Implement adaptive scheduling for NE and NE++ (All)

week 6.5-9: Conduct experiments and complete write-up (All)

References

[1] P. Veličković, R. Ying, M. Padovano, R. Hadsell, and C. Blundell, "Neural execution of graph algorithms," 2019. [Online]. Available: https://arxiv.org/abs/1910.10593

[2] L. A. C. Xhonneux, A. Deac, P. Velickovic, and J. Tang, "How to transfer algorithmic reasoning knowledge to learn new algorithms?" *CoRR*, vol. abs/2110.14056, 2021. [Online]. Available: https://arxiv.org/abs/2110.14056

[3] S. Jean, O. Firat, and M. Johnson, "Adaptive scheduling for multi-task learning," 2019. [Online]. Available: https://arxiv.org/abs/1909.06434

[4] P. Veličković, A. P. Badia, D. Budden, R. Pascanu, A. Banino, M. Dashevskiy, R. Hadsell, and C. Blundell, "The clrs algorithmic reasoning benchmark," 2021.