Triplet Edge Attention for Algorithmic Reasoning

Yeonjoon Jung
Pohang University of Science and Technology
jeffjung01@postech.ac.kr

Sungsoo Ahn
Pohang University of Science and Technology
sungsoo.ahn@postech.ac.kr

Abstract

This work investigates neural algorithmic reasoning to develop neural networks capable of learning from classical algorithms. The main challenge is to develop graph neural networks that are expressive enough to predict the given algorithm outputs while generalizing well to out-of-distribution data. In this work, we introduce a new graph neural network layer called Triplet Edge Attention (TEA), an edge-aware graph attention layer. Our algorithm works by precisely computing edge latent, aggregating multiple triplet messages using edge-based attention. We empirically validate our TEA layer in the CLRS benchmark and demonstrate a 5% improvement on average. In particular, we achieve a 30% improvement for the string algorithms compared to the state-of-the-art model.

1 Introduction

Neural networks are undergoing rapid development and unprecedented performance in tackling intricate tasks across a wide range of domains. However, a critical vulnerability of neural networks lies in their robustness, which is the capacity to maintain consistent performance on all inputs [1]. Despite their high accuracy on the training distribution, neural networks often exhibit subpar performance during the inference stage, particularly when confronting input data lying outside the training data distribution. Neural algorithmic reasoning [2] is a promising avenue to address this robustness problem by combining neural network models with classical algorithms, which entails training the algorithm itself to the neural network.

In contrast to traditional algorithmic solutions for real-world tasks, which embed the task into a known domain and then select an algorithm to solve the proxy-domain problem, neural algorithmic reasoning allows direct input without the proxy step [2]. Furthermore, neural algorithmic networks can be repurposed as a pretrained model without losing their generality. This versatility enables the execution of complex tasks by synthesizing multiple algorithmic reasoning layers and even discovering new algorithms from unconventional perspectives compared to human analyses [3].

Algorithmic reasoning tasks are commonly accomplished using graph neural networks (GNNs) [4], which excel in terms of expression power compared to models using sequential or grid-based inputs. Corresponding attribute facilitates the efficient expression of algorithms [5] such as Breadth-First Search [6] or Bellman-Ford [7]. This research aims to develop a graph-based algorithmic learner, a GNN capable of learning various algorithms involving diverse input types. The previous state-of-the-art (SOTA) model, i.e., Triplet-GMPNN [8], displayed remarkable enhancements across the CLRS benchmark [3] thanks to its capability to reason over triplet of vertices.

Contribution. In this work, we focus on refining neural architectures to better align with the algorithmic reasoning tasks. To this end, we devise a novel edge-attention method tailored for reasoning, coined **T**riplet **E**dge **A**ttention (**TEA**), an efficient method for computing edge latent suitable for both node-based and edge-based algorithms. We use the newly proposed attention to construct **T**riplet **E**dge **A**ttention **M**essage passing neural network (**TEAM**), which we combine with the encoder-processor-decoder network to enhance the computation of algorithmic outputs. We evaluate our algorithm on the CLRS-30 benchmark [3] and achieve state-of-the-art results with an average rank of 1.63.

2 Preliminaries

Problem setting. Algorithms solve a particular problem in a finite amount of time with unambiguous and executable steps. Several algorithms in the CLRS-30 benchmark, such as the Matrix-Chain-Order algorithm [9] or the Floyd-Warshall algorithm [10], require edge-based reasoning to determine the next states of algorithms. Thus, given graph $G = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E}$, handling a $|\mathcal{V}|^3$ message is beneficial for edge-level reasoning by aggregating intermediate node effects [11].

Hints. Hints are time series data that represent algorithm states, which contain the necessary information needed to replicate the logical, step-by-step execution of algorithms [3]. The type of hints depends on the task and can be stored at node, edge, or graph level. In the absence of monitoring the model with ground-truth hints, it is difficult to confirm that the intended algorithm is being learned during the training. For example, when an insertion sort algorithm is given as input, the model might learn other behaviors such as heap sort [12] or quicksort [13] algorithm without the help of hints. To address this issue, we adopted an architecture that anticipates the algorithm's hint trajectory through an iterative process before generating the final output.

Encode-process-decode paradigm. We embrace the encode-process-decode paradigm as presented in the CLRS benchmark [3]. For a given task τ, at each t-th time step, the encoder f_τ performs linear encoding for input and hint, embedding them as high-dimensional vectors. These embeddings of inputs and hints located in the nodes all have the same dimension and are added together; the same happens with hints and inputs located in the edges, and in the graph. Thus, at the end of the encoding step for a time step t of the algorithm, we have a single set of embeddings $\{\mathbf{x}_i^{(t)}, \mathbf{e}_{ij}^{(t)}, \mathbf{g}^{(t)}\}$ with $\mathbf{x}_i^{(t)} \in \mathbb{R}^{n \times h}$, $\mathbf{e}_{ij}^{(t)} \in \mathbb{R}^{n \times n}$, $\mathbf{g}^{(t)} \in \mathbb{R}^h$, in the nodes, edges, and graphs, respectively. Note that this process remains independent of the size or type of the inputs and hints inherent to a particular algorithm, allowing us to share this latent space across all the thirty algorithms in CLRS.

3 Triplet Edge Attention

We propose a novel **T**riplet **E**dge **A**ttention (**TEA**) which captures multiple node features to compute the edge latent values. Previous attempts for graph attention were made in both node-based and edge-based level [14–19]. Our TEA computes the edge latent representation $\mathbf{h}_{ij}$ as follows:

$$\mathbf{h}_{ij} = \mathrm{ReLU}\left(\sum_{k \in \mathcal{N}_i \cup \mathcal{N}_j} \boldsymbol{\alpha}_{ijk}\left(\mathbf{W}'\mathbf{e}_{ik}\right)\right), \tag{1}$$

where $\boldsymbol{\alpha}_{ijk}$ is the newly proposed edge-based triplet attention computed as follows:

$$\mathbf{t}'_{ijk} = \mathbf{a}^T\,\mathrm{LeakyReLU}(\mathbf{t}_{ijk}), \qquad \boldsymbol{\alpha}_{ijk} = \frac{\exp(\mathbf{t}'_{ijk})}{\sum_{k' \in \mathcal{N}_i \cup \mathcal{N}_j} \exp(\mathbf{t}'_{ijk'})}, \tag{2}$$

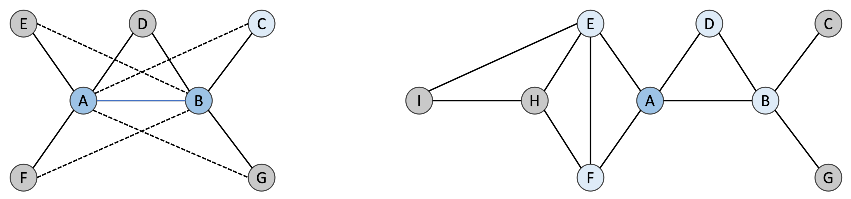
$$\mathbf{t}_{ijk} = \mathbf{W}[\mathbf{x}_i \parallel \mathbf{x}_j \parallel \mathbf{x}_k \parallel \mathbf{e}_{ij} \parallel \mathbf{e}_{ik} \parallel \mathbf{e}_{jk} \parallel \mathbf{g}]. \tag{3}$$

Note that the multi-head attention can also be implemented as follows:

$$\mathbf{h}_{ij} = \Big\Vert_{m=1}^{M}\left(\mathrm{ReLU}\left(\sum_{k \in \mathcal{N}_i \cup \mathcal{N}_j} \boldsymbol{\alpha}_{ijk}^{m}\left(\mathbf{W}^m\mathbf{e}_{ik}\right)\right)\right) \tag{4}$$

where, $\|$ indicates concatenation, and the activation function from Equation (1), (4) may be changed depending on the task. The major difference of previous triplet reasoning and our TEA is that, we efficiently map representations for all additional nodes through edge-level attention, instead of maximization over triplet messages. The computational complexity of a single attention head TEA layer is expressed as $\mathbf{O}\left(|\mathcal{V}|^2h^2 + |\mathcal{V}|^3h\right)$ which matches the time complexity of triplet reasoning method, where h is the hint dimension. This enables fair comparison between TEA and triplet reasoning based models respective to its computational cost. Practical ratios of the number of parameters are stated in 3.

Our TEA method exhibits effectiveness not only for edge-based reasoning problems but also for node-based reasoning problems. Compared to the traditional message passing with two steps, where



(a) TEA for edge AB **(b)** Final Computation for node A

Figure 1: Triplet Edge Attention structure for edge latent AB and final computation process for Node A's Output. A solid line represents an edge, and a dotted line implies that the nodes are not directly connected but are still considered for TEA. Colored nodes from 1b are A's neighbors which are considered for the output computation.

the effects of 2-hop neighbors are indirectly computed through 1-hop nodes, TEA directly calculates the influence of 2-hop neighbors relative to the target node's state. This direct calculation ensures accurate attention score computation. The TEA method is also beneficial when dealing with fully-connected graphs, which aligns with the practical condition of our experiment. Contrary to the two-step message passing or two-step graph attention, which stores information as node latents, the TEA method preserves computed information as edge latents. This distinction becomes critical within a fully-connected graph, where all node latents are used to predict the output of each node. In contrast, our edge latent solely affects the two connected nodes during the final step. This difference in the influence scope allows precise reasoning for individual nodes and the aggregation of multiple conditions to determine the next execution state of the algorithm.

We demonstrate the effectiveness of our TEA in the example of Figure 1a. Here, the effect of node C is embedded in the edge latent $\mathbf{h}_{AB}$ considering features of nodes A, B, C, edges AB, BC, AC, and graph G. It is noteworthy that edge AC could be a zero vector if the two nodes are not connected. In the case we are computing the output for node A, as shown in Figure 1b, the traditional message passing method with two steps would embed the effect of node C into node B, then propagate it to node A. On the other hand, even when nodes A and C are not directly linked, our TEA method computes the influence of node C relative to the state of node A, enabling accurate computation of attention score.

4 Experiment

4.1 Implementation

Our experiments were conducted using the CLRS benchmark [3], which includes 30 classical algorithms categorized into 8 distinct groups, reformatted as graph structure enabling training on GNNs. During the training process, we applied model improvement techniques [8] to enhance the model's robustness. Additionally, we replaced the in-distribution validation set of size 16 into a OOD data of size 32, to prevent over-fitting on a particular graph size and assure the model is learning the general algorithm. We also applied OOD validation to the previous SOTA model to make comparison of its effect. The effect of changing validation set is examined in the results section. For the test set, OOD data of size 64 were used.

We use the Triplet-GMPNN architecture [8] as a baseline model for this research. The model is designed with encoder-processor-decoder architecture for hint-based tasks introduced from the previous work [3]. The processor network for Triplet-GMPNN is a fully connected MPNN [20] with triplet reasoning and gating method [8].

4.2 Results

We evaluated OOD performance over CLRS-30 benchmark [3], with 5 repetitions. Table 1, 2 and Figure 2 present the micro-F1 score for OOD test data, comparing the baseline Triplet-GMPNN [8], the baseline with OOD validation set, and our TEAM model. TEAM model exhibited the highest performance, showing a 5.09% improvement compared to the previous SOTA, while the appliance of OOD validation in the baseline model yielded a 3.51% improvement. Additionally, our TEAM model

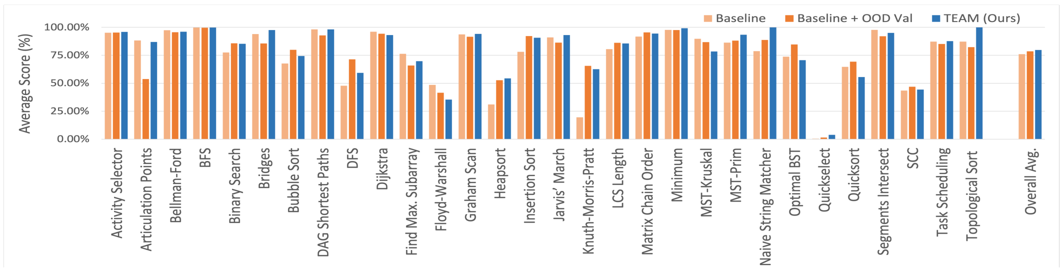


Figure 2: Bar chart for OOD micro-F1 scores of baseline (Triplet-GMPNN), baseline with OOD validation set, and our best model TEAM, after 10,000 training steps on each algorithm.

Table 1: Average OOD micro-F1 scores of Memnet, MPNN, PGN, baseline (Triplet-GMPNN), baseline with OOD validation set, and our best TEAM model, after 10,000 training steps on each algorithm category.

Algorithm	Memnet [3]	MPNN [3]	PGN [3]	Baseline [8]	Baseline + OOD Val	TEAM (Ours)
Div.&C.	$13.05\% \pm 0.14$	$20.30\% \pm 0.85$	$65.23\% \pm 4.44$	$\mathbf{76.36\% \pm 1.34}$	$65.70\% \pm 3.72$	$69.79\% \pm 1.60$
DP	$67.94\% \pm 8.20$	$65.10\% \pm 6.44$	$70.58\% \pm 6.48$	$81.99\% \pm 4.98$	$\mathbf{88.83\% \pm 5.06}$	$83.61\% \pm 10.57$
Geometry	$45.14\% \pm 11.95$	$73.11\% \pm 17.19$	$61.19\% \pm 7.01$	$\mathbf{94.09\% \pm 2.30}$	$88.02\% \pm 6.63$	$94.03\% \pm 1.57$
Graphs	$24.12\% \pm 5.30$	$62.79\% \pm 8.75$	$60.25\% \pm 8.42$	$81.41\% \pm 6.21$	$77.89\% \pm 21.49$	$\mathbf{81.6\% \pm 23.33}$
Greedy	$53.42\% \pm 20.82$	$82.39\% \pm 3.01$	$75.84\% \pm 6.59$	$91.21\% \pm 2.95$	$90.08\% \pm 5.06$	$\mathbf{91.80\% \pm 4.68}$
Search	$34.35\% \pm 21.67$	$41.20\% \pm 19.87$	$56.11\% \pm 21.56$	$58.61\% \pm 24.3$	$61.89\% \pm 45.48$	$\mathbf{62.79\% \pm 43.66}$
Sorting	$71.53\% \pm 1.41$	$11.83\% \pm 2.78$	$15.45\% \pm 8.46$	$60.37\% \pm 12.7$	$\mathbf{72.08\% \pm 21.31}$	$68.75\% \pm 18.64$
Strings	$1.51\% \pm 0.46$	$3.21\% \pm 0.94$	$2.04\% \pm 0.20$	$49.09\% \pm 23.5$	$75.33\% \pm 22.57$	$\mathbf{81.24\% \pm 25.05}$
Average	38.88%	44.99%	50.84%	74.14%	77.65%	**79.23%**
Rank Avg.	5.38	4.75	4.63	2.38	2.25	**1.63**
$> 90\%$	0/30	6/30	3/30	11/30	11/30	**15/30**
$> 50\%$	10/30	17/30	18/30	24/30	26/30	**27/30**

outperformed the baseline across 6 algorithm categories, displaying the highest rank among models, suggesting that TEAM shows the most robust performance overall in all algorithm categories.

Significant performance increases were observed in sorting algorithms (insertion sort, bubble sort, heapsort [12], quicksort [13]) and string algorithms (Naive string matching, Knuth-Morris-Pratt (KMP) string matcher [21]). In particular, our TEAM model achieved an 81.24% performance for string algorithms requiring multiple comparison capability, a considerable improvement over the previous best of 49.09%.

The appliance of the new training framework with the OOD validation set improved the baseline model in terms of overall average score and average rank. It also increased the number of algorithms solvable exceeding 50% performance compared to the baseline. Whereas, our TEAM model, also using the OOD validation set, showed the best result for all standards including average performance, average rank, and the number of algorithms over 50% & 90% OOD performance. A notable observation is the large intersection of algorithms surpassing the baseline between the baseline with OOD validation and our TEAM model. This implies that the effectiveness of the OOD validation method might be dependent on the target task's characteristics.

5 Conclusion

In this work, we present a new **TEA** (**T**riplet **E**dge **A**ttention) method, designed to compute edge latent during the algorithmic reasoning process. Based on the method, we propose the **TEAM** (**T**riplet **E**dge **A**ttention **M**essage Passing Neural Network) model. Our TEAM model with OOD validation set exhibited improvements across the CLRS benchmark algorithms and demonstrated a particularly significant enhancement with string algorithms, as compared to the baseline model [8]. In conclusion, we have developed a novel model and training methodology for algorithmic reasoning tasks and we are confident that additional developments within the reasoning field can lead even to higher performance levels, eventually unlocking the potential for a generally effective model.

References

[1] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy (SP)*, 2017. 1

[2] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021. 1

[3] 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, 3, 4

[4] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 1

[5] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019. 1

[6] Edward F Moore. The shortest path through a maze. In *Proc. of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, 1959. 1

[7] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958. 1

[8] Borja Ibarz, Vitaly Kurin, George Papamakario, and Kyriacos Nikiforou. A Generalist Neural Algorithmic Learner. In *PMLR*, 2022. 1, 3, 4, 6, 8, 9

[9] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974. 2

[10] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962. 2

[11] Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. *Advances in Neural Information Processing Systems*, 35:20635–20647, 2022. 2

[12] J Williams. Heapsort. *Commun. ACM*, 7(6):347–348, 1964. 2, 4

[13] Charles AR Hoare. Quicksort. *The computer journal*, 5(1):10–16, 1962. 2, 4

[14] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017. 2

[15] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*, 2019.

[16] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Multi-hop attention graph neural network. *arXiv preprint arXiv:2009.14332*, 2020.

[17] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[18] Jun Chen and Haopeng Chen. Edge-featured graph attention network, 2021.

[19] Soo Yong Lee, Fanchen Bu, Jaemin Yoo, and Kijung Shin. Towards deep attention in graph neural networks: Problems and remedies. *arXiv preprint arXiv:2306.02376*, 2023. 2

[20] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 3, 5

[21] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977. 4

A Implementation details

A.1 Processor

The processor is the key component in the encoder-processor-decoder architecture for algorithmic reasoning. We designed a new **TEAM** (**T**riplet **E**dge **A**ttention **M**essage Passing Neural Network) network for training algorithms, combining the triplet edge attention (TEA) layer to a fully connected MPNN (Message Passing Neural Network) [20] as Figure 3.

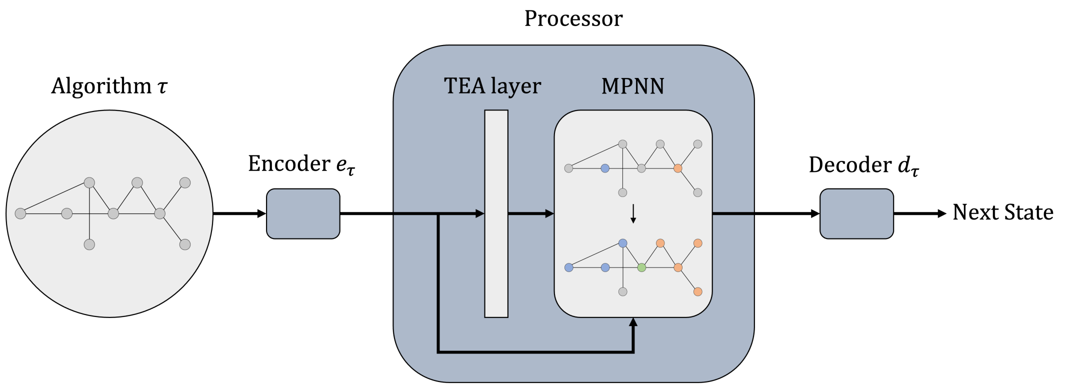


Figure 3: The **TEAM** Processor Network composed of Triplet Edge Attention Layer and Message Passing Neural Network.

The embedded results from the encoder are processed through the TEA layer to compute the edge latents. Afterward, the computed edge latents and the embeddings from the encoder are given as input for the fully connected MPNN. The MPNN is responsible for predicting the algorithm's hints, which are then fed to the decoder to get the current state. In contrast to the previous triplet reasoning approach [8], which fed the edge latent directly to the decoder, we have replaced the edge feature to our TEA-derived edge latent for MPNN computation. This change permits reasoning on nodes utilizing our edge latents.

A.2 Encoder and Decoder

The encoder and decoder from the baseline were adopted in our approach. For an algorithm τ, encoder $\mathbf{f}_\tau$ and decoder $\mathbf{g}_\tau$ are defined tasks specifically since each algorithm requires different hints to predict the algorithm's subsequent state. The encoder performs linear embeddings of node features, edge features, graph features, and the hints of the preceding state into a higher-dimensional space. The embedded results are then fed to our processor network to predict the hints for the next stage of the algorithm. Finally, the decoder estimates the algorithmic reasoning for the current state by linearly decoding the processor network's output.

A.3 Additional Experiment Results

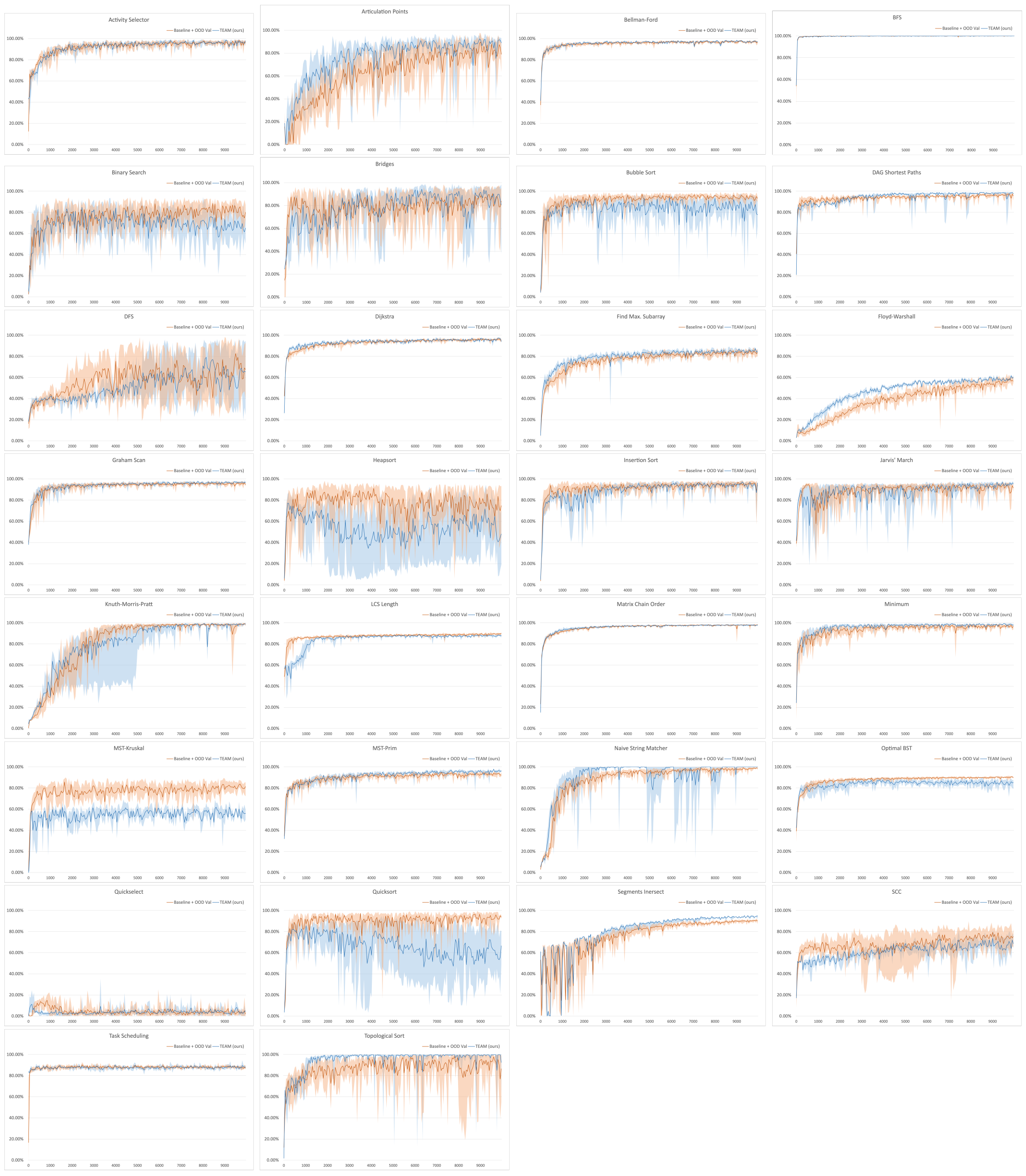


Figure 4: The learning curve of each algorithm, where the blue line indicates the learning curve of our TEAM model, and the orange line indicates the learning curve of the baseline model with OOD validation. The algorithms are ordered in alphabetical order, the same as the Table 2

Table 2: Average OOD micro-F1 scores of the baseline (Triplet-GMPNN), baseline with OOD validation set, and our best TEAM model, after 10,000 training steps on single algorithm tasks.

Algorithm	Baseline [8]	Baseline + OOD Val	TEAM (Ours)
Activity Selector	$95.18\% \pm 0.45$	$94.60\% \pm 1.19$	$95.90\% \pm 1.61$
Articulation Points	$88.32\% \pm 2.01$	$54.35\% \pm 9.56$	$86.89\% \pm 7.03$
Bellman-Ford	$97.39\% \pm 0.19$	$95.35\% \pm 1.20$	$96.08\% \pm 0.48$
BFS	$99.73\% \pm 0.04$	$99.58\% \pm 0.19$	$99.85\% \pm 0.09$
Binary Search	$77.58\% \pm 2.35$	$86.27\% \pm 5.49$	$85.31\% \pm 4.42$
Bridges	$93.99\% \pm 2.07$	$91.87\% \pm 5.68$	$97.53\% \pm 1.39$
Bubble Sort	$67.68\% \pm 5.50$	$84.26\% \pm 10.30$	$74.35\% \pm 14.15$
DAG Shortest Paths	$98.19\% \pm 0.30$	$93.75\% \pm 1.33$	$98.13\% \pm 0.27$
DFS	$47.79\% \pm 4.19$	$61.97\% \pm 16.66$	$59.31\% \pm 27.75$
Dijkstra	$96.05\% \pm 0.60$	$93.87\% \pm 1.08$	$93.14\% \pm 0.93$
Find Max. Subarray	$76.36\% \pm 0.43$	$65.70\% \pm 3.72$	$69.79\% \pm 1.60$
Floyd-Warshall	$48.52\% \pm 1.04$	$42.56\% \pm 2.15$	$35.33\% \pm 2.82$
Graham Scan	$93.62\% \pm 0.91$	$90.77\% \pm 4.25$	$94.12\% \pm 1.61$
Heapsort	$31.04\% \pm 5.82$	$47.95\% \pm 23.13$	$54.42\% \pm 8.85$
Insertion Sort	$78.14\% \pm 4.64$	$91.11\% \pm 2.22$	$90.73\% \pm 2.89$
Jarvis' March	$91.01\% \pm 1.30$	$85.22\% \pm 10.84$	$93.02\% \pm 1.91$
Knuth-Morris-Pratt	$19.51\% \pm 4.57$	$62.85\% \pm 24.11$	$62.59\% \pm 23.28$
LCS Length	$80.51\% \pm 1.84$	$86.38\% \pm 0.26$	$85.67\% \pm 0.27$
Matrix Chain Order	$91.68\% \pm 0.59$	$95.48\% \pm 0.75$	$94.46\% \pm 0.88$
Minimum	$97.78\% \pm 0.55$	$97.67\% \pm 1.13$	$99.23\% \pm 0.13$
MST-Kruskal	$89.80\% \pm 0.77$	$86.96\% \pm 3.25$	$78.47\% \pm 2.36$
MST-Prim	$86.39\% \pm 1.33$	$89.24\% \pm 1.21$	$93.30\% \pm 1.74$
Naive String Matcher	$78.67\% \pm 4.99$	$87.81\% \pm 14.98$	$99.90\% \pm 0.15$
Optimal BST	$73.77\% \pm 1.48$	$84.64\% \pm 0.43$	$70.70\% \pm 5.45$
Quickselect	$0.47\% \pm 0.25$	$1.73\% \pm 0.83$	$3.84\% \pm 3.56$
Quicksort	$64.64\% \pm 5.12$	$64.98\% \pm 11.66$	$55.49\% \pm 15.60$
Segments Intersect	$97.64\% \pm 0.09$	$92.15\% \pm 0.13$	$94.96\% \pm 0.21$
SCC	$43.43\% \pm 3.15$	$47.14\% \pm 5.89$	$44.30\% \pm 3.43$
Task Scheduling	$87.25\% \pm 0.35$	$85.56\% \pm 1.18$	$87.70\% \pm 2.17$
Topological Sort	$87.27\% \pm 2.67$	$78.08\% \pm 24.27$	$100.00\% \pm 0.01$
Overall Avg.	75.98%	78.00%	79.82%

Table 3: The ratio of the number of parameters between the baseline model (Triplet-GMPNN) and our TEAM model.

Algorithm	TEAM (Ours) / Baseline [8]
Activity Selector	1.46
Articulation Points	1.17
Bellman-Ford	1.17
BFS	1.17
Binary Search	1.46
Bridges	1.16
Bubble Sort	1.17
DAG Shortest Paths	1.07
DFS	1.11
Dijkstra	1.17
Find Max. Subarray	1.46
Floyd-Warshall	1.14
Graham Scan	1.26
Heapsort	1.11
Insertion Sort	1.17
Jarvis' March	1.46
Knuth-Morris-Pratt	1.26
LCS Length	1.44
Matrix Chain Order	1.14
Minimum	1.46
MST-Kruskal	1.26
MST-Prim	1.17
Naive String Matcher	1.46
Optimal BST	1.14
Quickselect	1.26
Quicksort	1.17
Segments Intersect	1.46
SCC	1.11
Task Scheduling	1.46
Topological Sort	1.11
Overall Avg.	1.25