Multi-Head Attention and Dual Message-Passing Neural Network for Neural Algorithmic Reasoning

Felipe A. L. Reis[1], **Zenilton K. G. do Patrocínio Jr.**[1]

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Avenida Dom José Gaspar, 500, Coração Eucarístico – Belo Horizonte – MG – Brazil

falreis@sga.pucminas.br, zenilton@pucminas.br

Abstract. *Neural Algorithmic Reasoning (NAR) is the research area that aims to build artificial neural networks that can mimic (classical) algorithms, reproducing intermediary steps from their execution traces. This paper introduces MHAD-MPNN (Multi-Head Attention and Dual Message-Passing Neural Network), a Triplet-GMPNN-inspired method that integrates feature representations with a multi-head attention memory block and a streamlined message-passing procedure. Our solution achieved results comparable to the current state-of-the-art. Furthermore, we conducted a comprehensive ablation study to evaluate key parameter modifications, thereby deepening our understanding of Neural Algorithmic Reasoning.*

1. Introduction

Neural Algorithmic Reasoning (NAR) is the research field that builds artificial neural networks to mimic *(classical)* algorithms, reproducing their steps from intermediary abstract inputs (traces) [Veličković and Blundell 2021] [Mahdavi et al. 2023]. This research area emerged to enhance the generalization of neural networks and develop faster, more efficient algorithms, particularly those focused on algorithmic representations [Mahdavi et al. 2023] [Veličković and Blundell 2021].

Learning algorithms must seem inappropriate at first, once classical deterministic algorithms use a precise, finite, and effective approach, using a set of instructions to transform an input into an output [Cormen et al. 2022] [Arora and Barak 2009] [Robert Sedgewick 2011]. Nevertheless, traditional metrics like output accuracy and correctness lose their conventional meaning when the input data is imprecise. Using a precise algorithm on an imprecise input will not necessarily produce an optimal value.

Neural Algorithmic Reasoning (NAR) is particularly promising for dynamic systems, especially those in real-world, uncontrolled environments. It excels in these contexts because they contain imprecise and noisy metrics, subject to interactive updates (feedback loops) that continuously alter the system's state. Real-time routing problems are an example of a practical application suitable for NAR methods. In such problems, input data, such as distance, speed, and traffic conditions, are imprecise and constantly changing, often forcing conventional algorithms to rely on simplified averages [Rahemi and Mosavi 2021] [Sirish Kumar and Srilatha Indira Dutt 2020] [Zhang and Fink 2026].

Inspired by the Triplet-GMPNN architecture [Ibarz et al. 2022], this research presents a new method to mimic algorithms named MHAD-MPNN (*Multi-Head Attention*

and Dual Message-Passing Neural Network). The main contributions of this paper are *(i)* a dual message passing mechanism for MPNN architectures that operates on node, edge, and graph pair; *(ii)* a multi-head attention memory block integrated with feature representations; *(iii)* the evaluation of the minimum aggregation function for dimensionality reduction; and *(iv)* a detailed ablation study evaluating the individual impact of architectural changes, allowing for an analysis through the lens of algorithmic alignment.

This work is organized as follows. Section 2 contains a brief background to understand this paper, while related works are presented in Section 3. Section 4 outlines our method. The experiments and their result analysis are presented in Section 5. Finally, Section 6 draws some conclusions and proposes future work.

2. Background

Neural Algorithmic Reasoning (NAR) aims to design neural network models to emulate the execution of algorithms with robust performance on novel and unseen training data distribution (OOD – *out of distribution*), solving instances with higher complexity than those used in the learning phase [Veličković and Blundell 2021] [Xu et al. 2021].

In NAR, an algorithm τ should be learned by a neural network ϕ. Considering that the algorithm τ admits the existence of an abstracted input $\overline{x}$, after the execution, τ produces an output $\overline{y}$, summarized by $\overline{y} = \tau(\overline{x})$. NAR model expects that ϕ receives an abstracted input $\overline{x}$, producing an output $\overline{y} \approx \phi(\overline{x})$ [Veličković and Blundell 2021] [Ibarz et al. 2022].

Considering that NAR is suitable for imprecise or noisy input data, it is possible to consider that the abstracted input data $\overline{x} \approx x$, where x is the real non-abstracted input. Then, the output y for the real problem is modeled as $y \approx \tau(\overline{x})$, which can be modeled as $y \approx \phi(\overline{x})$, once the differentiable neural network can supposedly adjust the inherited modeled input error.

In NAR, a common paradigm to mimic algorithms is the *encode-process-decode* [Hamrick et al. 2018]. In that, ϕ represents an encoder-process-decoder neural network composed of an encoder function $\overline{f}$, a processor ρ, and a decoder function $\overline{g}$. The processor ρ operates within a latent space, learning a differentiable function that algorithmically maps an input $\overline{x}$ to an output $\overline{y}$.

3. Related Work

Before the rise of Graph Neural Networks (GNNs), Reinforcement Learning (RL) stood out as a differentiable architecture to reproduce algorithms. Even before the definition of Neural Algorithm Reasoning, some authors [Vinyals et al. 2015] [Graves et al. 2014] [Reed and de Freitas 2016] built methods and architectures to replicate part of the algorithms, such as loops, internal states, recursion, and other structures, especially for tasks, such as addition, multiplication, if, loops, and variable assignment.

The sequence of NAR research was characterized by efforts to reproduce the algorithms themselves or their core components, including basic structures, if-structures, and loops [Veličković et al. 2020] [Veličković et al. 2020] [Yan et al. 2020]. Individual work reproduced a set of algorithms, but without a standardized evaluation framework. After

2019, Graph Neural Networks (GNNs) emerged as the prevailing methodology for algorithmic reasoning. The introduction of the CLRS-30 benchmark [Veličković et al. 2022] accelerated this trend, creating a standard that biased the use of graph-based architectures.

Using the CLRS-30 benchmark [Veličković et al. 2022], multiple new model architectures were developed. Ibarz et al. [Ibarz et al. 2022] built Triplet-GMPNN, a generalist learner using triplet messages. Triplet Learning was also used as a base method for Triplet Edge Attention [Jung and Ahn 2023], which included an attention mechanism with edge features, and Triplet-GMPNN with No-Hint [Rodionov and Prokhorenkova 2023], which removed intermediate supervision, producing a new graph representation without a hint regime.

Different approaches were evaluated in the CLRS-30 benchmark, as ForgetNet [Bohde et al. 2024], which uses historical embeddings to realign the graph with the algorithms. Neural Priority Queues [Jain et al. 2023] introduced memory-enhanced GNNs by using priority queues as external memory. In addition to GNNs, Transformer-based architectures have been used to emulate algorithmic execution [Diao and Loynd 2023] [Mahdavi et al. 2023] [Li et al. 2024].

The literature also provides the theoretical foundation for using neural networks to mimic algorithms. Dudzik and Veličković [Dudzik and Veličković 2022] established a formal relationship between Graph Neural Networks and Dynamic Programming, suggesting that Message-Passing Neural Networks can represent relational structures required for algorithmic execution. In related work, Luca and Fountoulakis [Back De Luca and Fountoulakis 2024] demonstrated that Transformer Networks are able to reproduce algorithms and their intermediate steps.

4. Method

The following sections provide a detailed exposition of the proposed method.

4.1. Processor

Our model follows the Triplet-GMPNN [Ibarz et al. 2022] architecture by combining node, $\mathrm{x}_i^{(t)}$, edge, $\mathrm{e}_{ij}^{(t)}$, and graph, $\mathrm{g}^{(t)}$, embeddings. However, it departs from the baseline approach by omitting the concatenation of node embeddings with previous intermediate inputs, $\mathrm{h}_i^{(t-1)}$. Instead, node embeddings are processed independently from previous computational steps

The set of the embeddings is defined as $\left\{\mathrm{x}_i^{(t)}\ ,\ \mathrm{h}_i^{(t-1)}\ ,\ \mathrm{e}_{ij}^{(t)}\ ,\ \mathrm{g}^{(t)}\right\}$. To incorporate generalization and memory-based perspectives, a memory block (e.g., Multi-head Attention, LSTM, GRU, etc) was integrated with the inputs. This block, $\mathrm{m}_{\{ij\}}^{(t-1)}$, where $\mathrm{m}^{(0)} = 0$, is fused individually with each node and edge embedding, excluding graph-level embeddings, to produce a refined set of features. Subsequently, node, edge, and graph embeddings, along with historical intermediate inputs, undergo independent linear transformations of the input, according to Equation 1.

$$\left\{\mathrm{xa}_i^{(t)}\ ,\ \mathrm{ha}_i^{(t-1)}\ ,\ \mathrm{ea}_{ij}^{(t)}\ ,\ \mathrm{g}^{(t)}\right\} = \left\{\mathcal{L}\left(\mathrm{x}_i^{(t)} + \mathrm{m}_i^{(t-1)}\right),\ \mathcal{L}\left(\mathrm{h}_i^{(t-1)} + \mathrm{m}_i^{(t-1)}\right),\ \mathcal{L}\left(\mathrm{e}_{ij}^{(t)} + \mathrm{m}_{ij}^{(t-1)}\right),\ \mathcal{L}\left(\mathrm{g}^{(t)}\right)\right\} \quad (1)$$

The combined embeddings are processed by a Multi-layer Perceptron (MLP), acting as a *message function*, f_m. This representation is then scaled (multiplied) by the "hints" before undergoing a permutation-invariant reduction, as defined by Equation 2.

$$\mathrm{m}_i^{(t)} = \underset{1 \le j \le n}{\square} f_m \left(\mathrm{xa}^{(t)},\ \mathrm{ha}_j^{(t-1)},\ \mathrm{ea}_{ij}^{(t)},\ \mathrm{g}^{(t)} \right), \quad \text{where } \square = max,\ min,\ avg,\ \ldots \tag{2}$$

Subsequently, the outputs $\mathrm{xa}^{(t)}$, $\mathrm{ha}_j^{(t-1)}$, and $\mathrm{m}_i^{(t)}$ undergo independent linear transformations before merging into a single representation, defined in Equation 3.

$$\mathrm{h}_i^{(t)} = \left(\mathcal{L}\left(\mathrm{xa}^{(t)}\right),\ \mathcal{L}\left(\mathrm{ha}_j^{(t-1)}\right),\ \mathcal{L}\left(\mathrm{m}_i^{(t)}\right) \right) \tag{3}$$

Finally, the output $\hat{h}_i^{(t)}$ is defined in Equation 4:

$$\hat{\mathrm{h}}_i^{(t)} = \mathrm{h}_i^{(t)} + \mathrm{h}_i^{(t-1)} \tag{4}$$

4.2. Reasoning model

Our preliminary evaluation of the triplet reasoning mechanism indicated that the high-dimensional embeddings generated by the Triplet GMPNN led to excessive complexity in the fully connected layers. To mitigate this, we transitioned from $b \times n \times n \times n \times h$ to a simplified $b \times n \times n \times h$ architecture, where n denotes the number of nodes, b is the batch size, and h signifies the hints, a time series of intermediate states [Ibarz et al. 2022].

However, this reduction in triplet embeddings led to a decline in performance. Our analysis revealed that the issue stemmed from insufficient informational variability rather than the underlying data format. To address this, we duplicated the node and edge embeddings to enhance data diversity. This adjustment successfully improved model performance and served as the foundation for further architectural refinements.

The final messaging architecture incorporates dual representations of node, edges, and intermediate layer embeddings. A single graph representation complete the messaging model. To enhance feature diversity, each embedding is processed through two distinct readout architectures: f_1, a single-layer configuration consisting of a solitary linear transformation, and f_2, a dual-layer configuration comprising two linear layers interspersed with a ReLU activation. These operations are formally defined by Equation 5.

$$\begin{aligned} \mathrm{x}_1 &= f_1(\mathrm{x}_i), \quad \mathrm{h}_1 = f_1(\mathrm{h}_i), \quad \mathrm{e}_1 = f_1(\mathrm{e}_{ij}), \quad \mathrm{g}_1 = f_1(\mathrm{g}) \\ \mathrm{x}_2 &= f_2(\mathrm{x}_j), \quad \mathrm{h}_2 = f_2(\mathrm{h}_j), \quad \mathrm{e}_2 = f_2(\mathrm{e}_{ij}) \end{aligned} \tag{5}$$

The reasoning module computes representations that are subsequently reduced to a consolidated latent vector, as formally defined in Equation 6.

$$\mathrm{t}_{ij} = \psi_t(\mathrm{x}_1,\ \mathrm{x}_2,\ \mathrm{h}_1,\ \mathrm{h}_2,\ \mathrm{e}_1,\ \mathrm{e}_2,\ \mathrm{g}_1) \tag{6}$$

Since the output data maintains the same dimensionality as the edge inputs ($b \times n \times n \times h$), an explicit aggregation function is unnecessary. This preserves the full

embedding dimensionality and mitigates potential information loss typically associated with reduction operations. Consequently, the final output is derived through a singular activation function, defined in Equation 7.

$$\mathrm{h}_{ij} = \phi_t(\mathrm{t}_{ij}) \tag{7}$$

5. Experiments and Results

5.1. Dataset

The CLRS Algorithmic Reasoning Benchmark [Veličković et al. 2022] features 30 algorithms from the textbook "Introduction to Algorithms", by Cormen, Leiserson, Rivest, and Stein [Cormen et al. 2022], categorized into sorting, searching, dynamic programming, graphs, strings, and geometry. In CLRS-30, all algorithms are modeled as graph problems, regardless of their original domain. Each task's data includes an input, an output, and a set of "hints" – intermediate steps that capture the algorithm's state – enabling a step-by-step replication of its execution [Veličković et al. 2022] [Diao and Loynd 2023].

5.2. Experimental Setup and Parameters

The experiments were developed using the CLRS-30 benchmark suite [Veličković et al. 2022], running on a GeForce RTX 5060 16GB GPU. The settings applied to all models, including in the ablative analysis, were a learning rate $lr = 0.001$, a batch size of 16, and no dropout probability. To mitigate experimental variance, all trials – including those in the ablative analysis – were executed three times.

5.3. Results

To evaluate MHAD-MPNN, we adopted a parameterization consistent with the baseline Triplet-GMPNN algorithm [Ibarz et al. 2022], setting the hidden dimension to $h = 128$. The model was trained for $10,000$ epochs, and all experiments were repeated three times to mitigate bias and ensure stability. Additionally, the evaluation utilizes two message-passing steps to run per hint, followed by an LSTM message-passing block and the minimum aggregation function defined in Equation 2.

Minimum aggregation introduces an inductive bias aligned with algorithmic reasoning by emphasizing constraint propagation and optimal (smallest) values, as seen in classical methods like Dijkstra's shortest path, Bellman–Ford, and dynamic programming. Unlike max aggregation, which highlights dominant features, min aggregation better supports stable and accurate reasoning in graph-based algorithmic tasks, as confirmed by the experimental results.

The experimental results are presented in Figure 1. Using the best overall setup, the algorithm achieved an average accuracy of $75.13 \pm 0.74\%$. When incorporating the best individual parameter settings described in Section 5.4 (Ablative Analysis), our method reached 78.71% accuracy, demonstrating strong performance despite its simplicity. Table 1 compares the proposed method against existing literature, categorized by algorithm class, while a comprehensive evaluation of all algorithms is provided in Appendix A.

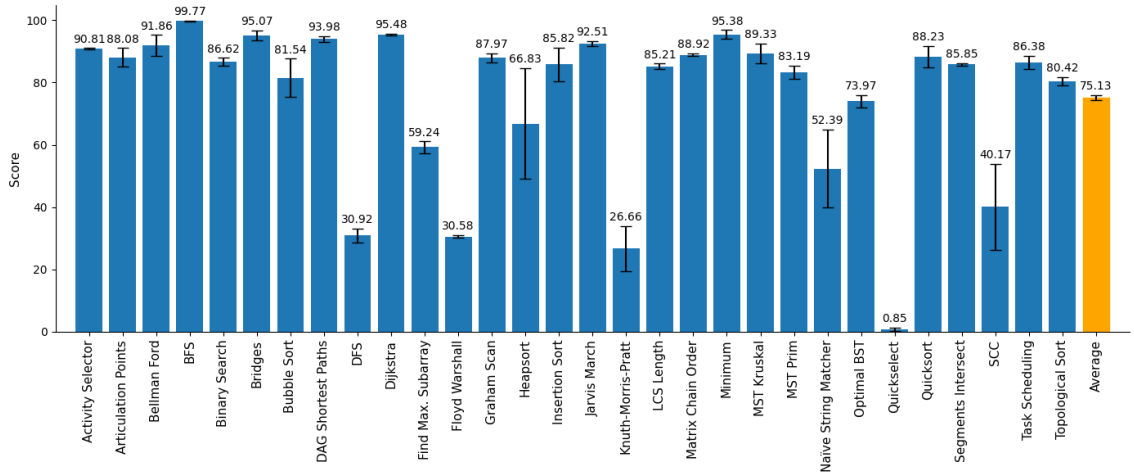


Figure 1. Test scores of our method MHAD-MIN on the CLRS-30 benchmark.

Table 1. Comparison of our method with the literature by group of algorithms in the CLRS-30 benchmark. Best results are marked in bold. Complete and individual results are available in Appendix A.

	Neural Network Architecture											
Algorithm	T-GMPNN [Ibarz et al. 2022]	TEAM [Jung and Ahn 2023]	T-GMPNN-LSR [Mirjanic et al. 2024]	T-GMPNN-NH [Rodionov and Prokhorenkova 2023]	G-ForgetNet [Bohde et al. 2024]	N. Priority Queues [Jain et al. 2023]	Rel. Transformers [Diao and Loynd 2023]	2-WL GNN [Mahdavi et al. 2023]	Open-Book NAR [Li et al. 2024]	Recurrent NAR [Xu and Veličković 2024]	**MHAD-MIN (ours)**	**MHAD-BEST (ours)**
Divide & Conquer	76.36	69.79	52.77	-	78.97	19.94	66.52	75.70	74.52	**83.53**	59.24	65.80
Dynamic Programming	81.99	83.61	86.67	-	**86.70**	68.71	83.20	85.92	82.14	42.69	82.70	83.40
Geometry	94.09	94.03	89.20	-	95.09	82.67	84.55	-	**96.55**	88.44	88.77	93.80
Graphs	81.41	81.86	69.44	83.34	**88.80**	48.62	65.33	-	85.37	74.15	76.57	79.83
Greedy	91.22	91.80	88.42	-	91.79	80.58	85.33	88.22	**93.40**	91.66	88.59	92.54
Search	58.61	62.79	54.73	-	63.84	34.81	65.31	64.85	63.15	**82.90**	60.95	61.11
Sorting	60.38	68.75	44.99	90.67	78.09	15.01	50.01	-	83.65	**94.15**	80.61	85.20
String	49.09	**81.25**	15.38	-	54.74	3.32	32.52	-	72.41	49.13	39.53	48.42
Average	75.98	79.82	65.51	-	82.89	46.32	66.18	-	**82.91**	75.78	75.13	78.71

Table 1 shows that our method achieved results comparable to other approaches in the literature, outperforming several models, despite the architectural simplicity. Al-

though overall benchmarks were competitive with existing literature, performance attenuated when capturing certain algorithms, such as Depth-First Search, Floyd-Warshall, Knuth-Morris-Pratt, Quickselect, and Strongly Connected Components. This outcome parallels known limitations in Graph Neural Network (GNN) frameworks, underscoring a broader challenge regarding their algorithmic alignment.

5.4. Ablative Analysis

The results of our ablation study are presented in Table 2. For clarity, the method variations are assigned names that reflect their specific parameter settings, as described below.

1. **MHAD-MIN-NL**: remove the LSTM layer after each message-passing block;
2. **MHAD-MAX**: the original parametrization was maintained, but the aggregation function (Equation 2) was switched to *max*;
3. **GRUD-MIN**: the Multi-Head Attention (MHA) memory block was replaced by a Gated Recurrent Unit (GRU);
4. **D-MIN**: removed the Multi-Head Attention (MHA) mechanism to isolate the contribution of the memory block.

As shown in Table 2, removing the LSTM layer after GNN message passing yielded an unexpected performance boost for most algorithms, often resulting in the highest individual scores. However, this modification considerably limited the learning capacity of some algorithms, such as Bubble Sort, Insertion Sort, Quicksort, and Minimum Search. Because of these performance drops, the overall average score declined sharply, falling well below the other parameterizations.

Ablation results showed that maximum aggregation underperformed compared to minimum aggregation, offered no significant algorithmic improvement, and was slightly behind the baseline model without Attention. Interestingly, removing the Attention mechanism yielded performance nearly identical to the top-performing model. Achieving an average training time of 3:22:18 ± 0:00:31–a 16.25% speedup–with only a marginal 0.58% decrease in performance, omitting Attention may be recommended for reducing training time and memory usage. Unlike MHA, the variant using a GRU memory layer underperformed compared to the memoryless configuration, suggesting that this mechanism actively hindered learning.

Our ablation study reveals that parameter selection significantly impacts algorithmic performance. While customized parameterizations could optimize individual models, we maintained a uniform setup to respect the generalist nature of NAR. However, to highlight the potential performance, we added two evaluation columns in Table 2: MHAD-BEST (the best performance using Multi-Head Attention) and D-BEST (the top performance across all variations, including GRU and memoryless configurations). Both optimized baselines yield results competitive with the literature, validating our approach.

6. Conclusion

This work introduces MHAD-MPNN, an evolution of the Triplet-GMPNN [Ibarz et al. 2022] featuring an improved message-passing mechanism for MPNN networks [Gilmer et al. 2017]. The proposed mechanism reduces both the dimensionality and the information content of embeddings during message passing. Also, it uses

Table 2. Ablation study: Accuracy of our model variations on the CLRS-30 benchmark. Best results are marked in bold.

ALGORITHM	MHAD-MIN	MHAD-MIN-NL	MHAD-MAX	GRUD-MIN	D-MIN	MHAD-BEST	D-BEST
Activity Selector	90.81±0.23	**93.16±2.16**	89.37±1.87	83.57±2.67	89.89±4.05	93.16±2.16	93.16±2.16
Articulation Points	88.08±2.98	**94.92±0.80**	89.11±1.80	80.63±9.71	88.54±2.72	94.92±0.80	94.92±0.80
Bellman Ford	91.86±3.43	89.73±3.79	89.19±2.15	87.34±4.23	**94.14±1.08**	91.86±3.43	94.14±1.08
BFS	99.77±0.07	**99.89±0.12**	99.79±0.20	99.19±0.99	99.35±0.54	99.89±0.12	99.89±0.12
Binary Search	**86.62±1.31**	82.41±4.21	83.85±6.89	76.42±3.77	85.89±3.83	86.62±1.31	86.62±1.31
Bridges	95.07±1.63	**97.45±1.83**	94.53±1.47	82.15±8.28	90.65±3.88	97.45±1.83	97.45±1.83
Bubble Sort	81.54±6.08	1.51±0.00	71.57±11.97	72.74±7.35	**83.48±5.86**	81.54±6.08	83.48±5.86
DAG shortest paths	93.98±0.96	**95.31±2.24**	91.36±1.75	92.29±1.85	92.11±0.51	95.31±2.24	95.31±2.24
DFS	30.92±2.19	**38.49±4.12**	36.28±4.10	34.85±3.50	24.30±8.95	38.49±4.12	38.49±4.12
Dijkstra	95.48±0.27	**97.51±0.42**	94.48±1.91	93.49±0.34	94.16±1.09	97.51±0.42	97.51±0.42
Find Max. Subarray	59.24±1.85	**65.80±4.16**	56.62±6.12	59.03±2.46	59.59±1.47	65.80±4.16	65.80±4.16
Floyd-Warshall	30.58±0.47	**32.30±1.60**	28.35±1.00	29.99±0.71	31.37±2.21	32.30±1.60	32.30±1.60
Graham Scan	87.97±1.42	89.52±5.49	91.21±0.87	89.70±0.95	**91.60±2.07**	91.21±0.87	91.60±2.07
Heapsort	66.83±17.71	85.03±11.79	57.06±18.25	**88.77±2.00**	80.66±9.81	85.03±11.79	88.77±2.00
Insertion sort	85.82±5.47	1.64±0.24	86.00±6.26	77.23±16.91	**87.63±3.74**	86.00±6.26	87.63±3.74
Jarvis' March	92.51±0.78	44.21±24.58	**93.02±1.26**	92.62±0.94	92.13±0.79	93.02±1.26	93.02±1.26
Knuth-Morris-Pratt	26.66±7.20	**44.45±7.76**	42.42±4.14	10.81±7.65	24.24±7.21	44.45±7.76	44.45±7.76
LCS Length	85.21±0.87	82.69±0.16	85.92±0.72	**86.22±0.62**	84.91±0.22	85.92±0.72	86.22±0.62
Matrix Chain Order	88.92±0.39	37.91±28.28	**89.17±2.71**	87.85±0.48	88.99±0.39	89.17±2.71	89.17±2.71
Minimum Search	**95.38±1.44**	29.20±47.49	93.07±0.85	95.35±2.24	94.95±1.62	95.38±1.44	95.38±1.44
MST-Kruskal	89.33±3.13	80.60±13.65	**89.74±1.29**	89.45±0.62	89.08±1.57	89.74±1.29	89.74±1.29
MST-Prim	83.19±2.14	85.60±2.45	87.42±3.42	85.12±3.72	**87.71±4.55**	87.42±3.42	87.71±4.55
Naïve String Matcher	**52.39±12.56**	1.81±0.00	35.81±6.80	1.12±0.72	12.08±8.35	52.39±12.56	52.39±12.56
Optimal BST	73.97±2.04	75.12±5.54	73.55±1.72	74.49±3.10	**75.97±2.50**	75.12±5.54	75.97±2.50
Quickselect	0.85±0.51	**1.32±0.60**	0.13±0.12	0.46±0.42	1.14±1.06	1.32±0.60	1.32±0.60
Quicksort	88.23±3.44	1.51±0.00	87.01±2.91	81.05±10.14	**89.39±3.24**	88.23±3.44	89.39±3.24
Segments Intersect	85.85±0.36	**97.16±0.18**	84.79±0.91	85.02±0.97	85.58±0.91	97.16±0.18	97.16±0.18
SCC	40.17±13.80	37.34±3.64	**50.31±4.38**	43.86±3.84	50.13±5.33	50.31±4.38	50.31±4.38
Task Scheduling	86.38±2.12	**91.92±3.26**	85.97±2.34	86.17±1.79	85.32±2.59	91.92±3.26	91.92±3.26
Topological Sort	80.42±1.39	**82.78±2.15**	78.87±0.67	78.88±2.65	81.49±2.53	82.78±2.15	82.78±2.15
AVERAGE	**75.13±0.74**	61.94±3.02	74.53±1.09	71.53±0.72	74.55±0.81	78.71	79.13

multi-head attention to add a small set of memory, which helps to learn algorithms, despite our empirical results showing this yields only a subtle gain in performance. To analyze the influence of each modification, a comprehensive ablation study was conducted, containing parameter variations that influence the algorithm's performance, which helps to advance the understanding of Neural Algorithmic Reasoning.

For future work, we plan to integrate improvements existing in the literature, using the training protocol of OpenBook [Li et al. 2024] and focus on a deeper analysis of the generated networks, evaluating their similarity to the original solution to enhance the interpretability of the results.

Reproducibility Statement

The source code for this project is publicly available at `https://github.com/IMScience-PPGINF-PucMinas/mhad-clrs`.

Acknowledgments

The authors thank the Pontifícia Universidade Católica de Minas Gerais – PUC-Minas, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Grant CAPES/COFECUB 88881.188646/2025-01, Grant PROAP 88887.842889/2023-00 – PUC/MG and Finance Code 001), the Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (Grants 407242/2021-0, 306573/2022-9, 442950/2023-3) and Fundação de Apoio à Pesquisa do Estado de Minas Gerais – FAPEMIG (Grants APQ-01079-23, APQ-05058-23).

LLM-Assistance Declaration

This paper was drafted in Portuguese, translated to English, and then polished using the Gemini LLM. We used the prompt "*rewrite: '<sentence>'*" to refine the language. The AI-generated text was manually reviewed and integrated by the authors.

Disclosure of Interests

The authors declare no conflicts of interest relevant to the content of this paper.

References

Arora, S. and Barak, B. (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 edition.

Back De Luca, A. and Fountoulakis, K. (2024). Simulation of graph algorithms with looped transformers. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F., editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 2319–2363. PMLR.

Bohde, M., Liu, M., Saxton, A., and Ji, S. (2024). On the markov property of neural algorithmic reasoning: Analyses and methods. In *The Twelfth International Conference on Learning Representations*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to Algorithms, fourth edition*. MIT Press.

Diao, C. and Loynd, R. (2023). Relational attention: Generalizing transformers for graph-structured tasks. In *The Eleventh International Conference on Learning Representations*.

Dudzik, A. J. and Veličković, P. (2022). Graph neural networks are dynamic programmers. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20635–20647. Curran Associates, Inc.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines.

Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., and Battaglia, P. W. (2018). Relational inductive bias for physical construction in humans and machines. In *Proceedings of the 40th Annual Conference of the Cognitive Science Society*, Austin, TX.

Ibarz, B., Kurin, V., Papamakarios, G., Nikiforou, K., Bennani, M., Csordás, R., Dudzik, A. J., Bošnjak, M., Vitvitskyi, A., Rubanova, Y., Deac, A., Bevilacqua, B., Ganin, Y., Blundell, C., and Veličković, P. (2022). A generalist neural algorithmic learner. In *The First Learning on Graphs Conference*.

Jain, R., VeliČkoviĆ, P., and Liò, P. (2023). Neural priority queues for graph neural networks.

Jung, Y. and Ahn, S. (2023). Triplet edge attention for algorithmic reasoning. In *The Second Learning on Graphs Conference*.

Li, H., Peng, C., Xu, C., and Yang, Z. (2024). Open-book neural algorithmic reasoning. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C., editors, *Advances in Neural Information Processing Systems*, volume 37, pages 9819–9836. Curran Associates, Inc.

Mahdavi, S., Swersky, K., Kipf, T., Hashemi, M., Thrampoulidis, C., and Liao, R. (2023). Towards better out-of-distribution generalization of neural algorithmic reasoning tasks. *Transactions on Machine Learning Research*.

Mirjanic, V. V., Pascanu, R., and Veličković, P. (2024). Latent space representations of neural algorithmic reasoners. In Villar, S. and Chamberlain, B., editors, *Proceedings of the Second Learning on Graphs Conference*, volume 231 of *Proceedings of Machine Learning Research*, pages 10:1–10:24. PMLR.

Rahemi, N. and Mosavi, M. R. (2021). Positioning accuracy improvement in high-speed gps receivers using sequential extended kalman filter. *IET Signal Processing*, 15(4):251–264.

Reed, S. and de Freitas, N. (2016). Neural programmer-interpreters.

Robert Sedgewick, K. W. (2011). *Algorithms, 4th Edition: Essential Information about Algorithms and Data Structures*. Addison-Wesley.

Rodionov, G. and Prokhorenkova, L. (2023). Neural algorithmic reasoning without intermediate supervision. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Sirish Kumar, P. and Srilatha Indira Dutt, V. (2020). The global positioning system: Popular accuracy measures. *Materials Today: Proceedings*, 33:4797–4801. International Conference on Nanotechnology: Ideas, Innovation and Industries.

Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. (2022). The CLRS algorithmic reasoning benchmark. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22084–22102. PMLR.

Veličković, P. and Blundell, C. (2021). Neural algorithmic reasoning. *Patterns*, 2(7).

Veličković, P., Buesing, L., Overlan, M., Pascanu, R., Vinyals, O., and Blundell, C. (2020). Pointer graph networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2232–2244. Curran Associates, Inc.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. (2020). Neural execution of graph algorithms. In *International Conference on Learning Representations*.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Xu, K. and Veličković, P. (2024). Recurrent aggregators in neural algorithmic reasoning. In *The Third Learning on Graphs Conference*.

Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-I., and Jegelka, S. (2021). How neural networks extrapolate: From feedforward to graph neural networks. In *International Conference on Learning Representations*.

Yan, Y., Swersky, K., Koutra, D., Ranganathan, P., and Hashemi, M. (2020). Neural execution engines: Learning to execute subroutines. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17298–17308. Curran Associates, Inc.

Zhang, Z. and Fink, O. (2026). Algorithm-informed graph neural networks for leakage detection and localization in water distribution networks. *Reliability Engineering & System Safety*, 265:111494.

Appendix

A. Complete Results in CLRS-30

Table 3 presents the complete results of our algorithm compared with the literature. To ensure a fair comparison, we calculated the average performance using only those works that reported results for all algorithms. In cases where a paper presented several method variations, we chose the one that either performed best or was tested on a greater number of algorithms. Our methods were trained three times to mitigate bias and ensure stability.

Table 3. Accuracy of literature algorithms in the CLRS-30 Benchmark.

	Neural Network Architecture												
Algorithm	T-GMPNN [Ibarz et al. 2022]	TEAM [Jung and Ahn 2023]	T-GMPNN-LSR [Mirjanic et al. 2024]	T-GMPNN-NH [Rodionov and Prokhorenkova 2023]	G-ForgetNet [Bohde et al. 2024]	N. Priority Queues [Jain et al. 2023]	Rel. Transformers [Diao and Loynd 2023]	2-WL GNN [Mahdavi et al. 2023]	Open-Book NAR [Li et al. 2024]	Recurrent NAR [Xu and Veličković 2024]	**MHAD-MIN (ours)**	**MHAD-BEST (ours)**	**D-BEST (ours)**
Activity Selector	95.18	95.90	90.50	-	99.03	83.36	87.72	92.31	95.86	95.23	90.81	93.16	93.16
Articulation Points	88.32	86.89	59.99	94.55	97.97	13.40	34.15	79.07	98.30	26.32	88.08	94.92	94.92
Bellman-Ford	97.39	96.08	98.96	95.25	99.18	87.50	94.24	98.16	95.18	96.00	91.86	91.86	94.14
BFS	99.73	99.85	99.50	99.95	99.96	99.64	99.14	100.00	99.99	100.0	99.77	99.89	99.89
Binary Search	77.58	85.31	66.49	85.29	85.96	28.79	81.48	94.45	87.44	64.71	86.62	86.62	86.62
Bridges	93.99	97.53	84.10	98.91	99.43	1.83	37.88	97.05	99.26	72.22	95.07	97.45	97.45
Bubble Sort	67.68	74.35	55.51	90.67	83.19	17.78	38.22	-	73.16	95.78	81.54	81.54	83.48
DAG Shortest Paths	98.19	98.13	82.56	98.18	99.37	96.19	96.61	99.40	97.79	96.40	93.98	95.31	95.31
DFS	47.79	59.31	20.65	34.52	74.31	6.03	39.23	26.11	40.79	100.0	30.92	38.49	38.49
Dijkstra	96.05	93.14	96.02	96.97	99.14	83.94	91.20	-	98.29	95.04	95.48	97.51	97.51
Find Max. Subarray	76.36	69.79	52.77	-	78.97	19.94	66.52	75.70	74.52	83.53	59.24	65.80	65.80
Floyd-Warshall	48.52	35.33	32.52	42.16	56.32	17.84	31.59	33.75	51.52	27.49	30.58	32.30	32.30
Graham Scan	93.62	94.12	87.21	-	97.67	61.95	74.15	95.80	96.19	76.20	87.97	91.21	91.60
Heapsort	31.04	54.42	16.28	90.67	57.47	16.37	32.96	-	85.71	93.07	66.83	85.03	88.77
Insertion Sort	78.14	90.73	61.69	90.67	98.40	8.16	89.43	-	92.61	93.00	85.82	86.00	87.63
Jarvis' March	91.01	93.02	81.57	-	88.53	92.88	94.57	-	94.74	91.83	92.51	93.02	93.02
Knuth-Morris-Pratt	19.51	62.59	20.59	-	12.45	3.23	0.03	-	71.24	4.54	26.66	44.45	44.45
LCS Length	80.51	85.67	87.75	-	85.43	67.50	83.32	87.53	85.54	66.91	85.21	85.92	86.22
Matrix Chain Order	91.68	94.46	90.45	-	91.08	78.74	91.89	92.54	90.85	25.12	88.92	89.17	89.17
Minimum	97.78	99.23	96.08	99.98	99.26	75.65	95.28	99.46	98.65	96.92	95.38	95.38	95.38
MST-Kruskal	89.80	78.47	85.49	92.89	91.25	43.24	64.91	90.42	91.26	67.29	89.33	89.74	89.74
MST-Prim	86.39	93.30	76.68	85.70	95.19	60.31	85.77	78.22	93.41	86.60	83.19	87.42	87.71
Naïve String Matcher	78.67	99.90	10.16	-	97.02	3.40	65.01	16.24	73.57	93.71	52.39	52.39	52.39
Optimal BST	73.77	70.70	81.82	-	83.58	59.89	74.40	77.68	70.04	36.04	73.97	75.12	75.97
Quickselect	0.47	3.84	1.62	-	6.30	0.00	19.18	0.63	3.37	87.08	0.85	1.32	1.32
Quicksort	64.64	55.49	46.46	90.67	73.28	17.71	39.42	15.33	83.13	94.73	88.23	88.23	89.39
Segments Intersect	97.64	94.96	98.81	-	99.06	93.19	84.94	97.55	98.71	97.30	85.85	97.16	97.16
SCC	43.43	44.30	32.53	66.38	53.53	22.50	28.59	71.47	65.83	48.43	40.17	50.31	50.31
Task Scheduling	87.25	87.70	86.34	-	84.55	77.79	82.93	84.12	90.93	88.08	86.38	91.92	91.92
Topological Sort	87.27	100.00	64.27	94.58	99.92	50.98	80.62	80.84	92.80	74.00	80.42	82.78	82.78
Average	75.98	79.82	65.51	-	82.89	46.32	66.18	-	82.91	75.78	75.13	78.71	79.13