# RELATIONAL ATTENTION: GENERALIZING TRANSFORMERS FOR GRAPH-STRUCTURED TASKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Transformers flexibly operate over sets of real-valued vectors representing task-specific entities and their attributes, where each vector might encode one word-piece token and its position in a sequence, or some piece of information that carries no position at all. As set processors, transformers are at a disadvantage in reasoning over more general graph-structured data where nodes represent entities and edges represent relations *between* entities. To address this shortcoming, we generalize transformer attention to consider and update edge vectors in each transformer layer. We evaluate this *relational transformer* on a diverse array of graph-structured tasks, including the large and challenging CLRS Algorithmic Reasoning Benchmark. There, it dramatically outperforms state-of-the-art graph neural networks expressly designed to reason over graph-structured data. Our analysis demonstrates that these gains are attributable to relational attention's inherent ability to leverage the greater expressivity of graphs over sets.

## 1 INTRODUCTION

While transformers are not fully understood, they have produced impressive results in a widening variety of domains, starting with machine translation (Vaswani et al., 2017), then quickly impacting language modeling (Devlin et al., 2018) and text generation (Brown et al., 2020). They are revolutionizing image processing (Dosovitskiy et al., 2020) and are being applied to a growing variety of settings including reinforcement learning, both online (Loynd et al., 2019; Parisotto et al., 2019) and offline (Chen et al., 2021; Janner et al., 2021). This cross-domain success is possible because the transformer fundamentally consumes *unordered sets* of real-valued vectors, injecting no other assumptions. This allows entities carrying domain-specific attributes (like position) to be encoded as vectors for input to the same transformer architecture applied to other domains.

By contrast, transformers have not had the same success in graph-structured domains. While many application areas consist of array-structured data, such as text or images, graph data is centrally concerned with pairwise relations between entities represented as edges and edge attributes, such as the bond structure between atoms in a molecule. Graphs are more general and expressive than sets, in the sense that a set is a special case of a graph, one without edges. So there is no generally applicable way of passing edge feature vectors (representing entity relations) into a transformer without losing the graph's connectivity information. We discuss ad-hoc methods that encode relations into sets in task-specific ways in Section 4, along with other prior works applying transformers to relational data.

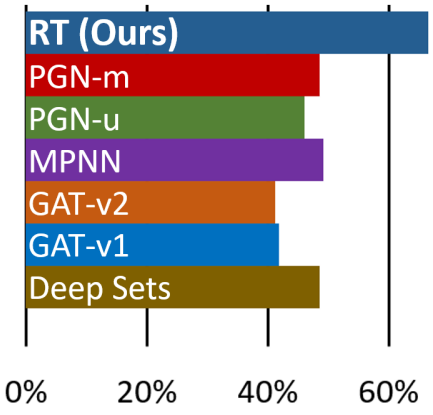


Figure 1: The relational transformer (RT) outperforms baseline GNNs on a set of 30 distinct graph-structured tasks from the CLRS Benchmark, averaged by algorithm class.

Graph Neural Networks (GNNs), on the other hand, are specifically designed to leverage graph-structured data, including the graph's (possibly directed) adjacency matrix, and (in some cases) features associated with the edges. Standard transformers (which we will refer to as

| | No Transformer Machinery | Some Transformer Machinery | Full Transformer Machinery |
|---|---|---|---|
| Full Edge Usage | Some GNNs | A Few Models | **RT: Relational Transformer** |
| Some Edge Usage | **Most Common GNNs** | Some GNNs & Transformer Variants | PRTs: Partially Relational Transformers |
| No Edge Vectors | | | **Transformers** |

Figure 2: Categories of GNNs and Transformers, compared in terms of transformer machinery (top labels) and edge vector incorporation (left-side labels). Model categories tested in our experiments are marked in bold.

simply *transformers* in this work) lack the relational inductive biases (Dwivedi & Bresson, 2020; Kreuzer et al., 2021; Ying et al., 2021) that are explicitly built into the most commonly used GNNs.

***We conjecture that a version of the transformer architecture, appropriately generalized to operate over graphs instead of sets, would be a competitive general-purpose model for graph-structured tasks.*** To investigate this conjecture, we design the *relational transformer* (RT, Section 3) by incorporating edge vectors to represent arbitrary learned relations, and generalizing transformer attention (as *relational attention*) to utilize them. As a native graph-to-graph model, RT does not rely on any special encodings or ad-hoc schemes to input or output graph data.

Graph-structured problems turn up in many problem domains, including knowledge bases (Hu et al., 2021; Bordes et al., 2013), communication networks (Leskovec et al., 2010; 2008; Bai et al., 2019), molecules (Debnath et al., 1991; Zhang et al., 2020a), proteins (Borgwardt et al., 2005), and citation networks (Mccallum et al., 2000; Giles et al., 1998). One example of a graph-structured task is predicting which molecules inhibit HIV replication (Wu et al., 2018). More generally, Veličković et al. (2022) and Battaglia et al. (2018) state that graphs can represent any set of discrete entities with relations between them. Along with their ubiquity, graph-structured problems vary widely in difficulty. For example, certain graph problems can be solved with an *untrained* Graph Convolutional Network (Li et al., 2018), while other graph-structured problems are quite challenging and require explicit modeling of relational characteristics, which RT is designed to provide.

We find that RT outperforms baseline GNNs on a large and diverse set of difficult graph-structured tasks. In particular, RT establishes dramatically improved state-of-the-art performance (Figure 1) over baseline GNNs on the challenging CLRS Algorithmic Reasoning Benchmark (Veličković et al., 2022), which comprises 30 different algorithmic tasks in a framework for probing the reasoning abilities of graph-to-graph models. Through our experimental analysis, we shed light on how RT's gains are attributable to relational attention's native ability to leverage the greater expressivity of graphs over sets. Our contributions are as follows:

- We introduce the relational transformer as a general-purpose graph-to-graph model, and make the implementation available as open-source code at (link withheld during review).
- We evaluate the reasoning power of RT on a wide range of challenging graph-structured tasks, including new state-of-the-art results on the CLRS Benchmark.
- We enhance the CLRS Benchmark framework to support evaluation of a broader array of models (Section 5.1.2).
- We improve the performance of the CLRS Benchmark baseline models by adding multi-layer functionality, and tuning their hyperparameters (Section 5.1.1).

## 2 GRAPH NEURAL NETWORKS

We introduce the graph-to-graph model formalism used in the rest of this paper, inspired by Battaglia et al. (2018). The input graph is a directed, attributed graph $G = (\mathbf{g}, \mathcal{V}, \mathcal{E})$, where $\mathbf{g}$ is the graph's global vector and $\mathcal{V}$ is an unordered set of node vectors $\mathbf{n}_i \in \mathbb{R}^{d_v}$, and $i$ denotes the $i$-th node. $\mathcal{E}$ is a set of edge vectors $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$, where directed edge $(i, j)$ points from node $j$ to node $i$. Each layer $l$ in the model accepts a graph $G^l$ as input, processes the graph's features, then outputs graph $G^{l+1}$ with the same structure as $G$, but with potentially updated node, edge and global vectors.

Each layer $l$ of a graph-to-graph model is comprised of three update functions $\phi$ and three aggregation functions $\bigoplus$.

$$\text{Node vector} \quad \mathbf{n}_i^{l+1} = \phi_v \left( \mathbf{n}_i^l, \bigoplus_{j \in \mathcal{N}_i} \psi^m \left( \mathbf{e}_{ij}^l, \mathbf{n}_i^l, \mathbf{n}_j^l, \mathbf{g}^l \right) \right) \tag{1}$$

$$\text{Edge vector} \quad \mathbf{e}_{ij}^{l+1} = \phi_e \left( \mathbf{e}_{ij}^l, \mathbf{n}_i^{l+1}, \mathbf{n}_j^{l+1}, \mathbf{g}^l \right) \tag{2}$$

$$\text{Global vector} \quad \mathbf{g}^{l+1} = \phi_g \left( \bigoplus_{(i,j)} \mathbf{e}_{ij}^{l+1}, \bigoplus_{i} \mathbf{n}_i^{l+1}, \mathbf{g}^l \right) \tag{3}$$

where $\mathcal{N}_i$ denotes the set of node $i$'s neighbors (optionally including node $i$), and $\psi^m$ denotes a message function. Graph neural networks comprise a subset of graph-to-graph models. Thus, we can describe the specific GNNs used to benchmark RT's performance in graph-to-graph model terms. These baseline GNNs (listed below) let $\phi_e$ and $\phi_g$ be identity functions such that $\mathbf{e}_{ij}^{l+1} = \mathbf{e}_{ij}^l$ for all edges $(i, j)$ and $\mathbf{g}^{l+1} = \mathbf{g}^l$. Furthermore, they use permutation-invariant aggregation functions $\bigoplus$. We now provide specific details for each baseline GNN, following Veličković et al. (2022):

In **Deep Sets** (Zaheer et al., 2017), the only edges are self-connections so $\mathcal{N}(i)$ is the singleton set containing node $i$.

In **Graph Attention Networks (GAT)** (Veličković et al., 2017; Brody et al., 2021), $\bigoplus$ is self-attention, and the message function $\psi^m$ merely extracts the sender features: $\psi^m(\mathbf{e}_{ij}^l, \mathbf{n}_i^l, \mathbf{n}_j^l, \mathbf{u}^l) = \mathbf{W}_s \mathbf{n}_j^l$, where $\mathbf{W}_s$ is a shared weight matrix.

In **Message Passing Neural Networks (MPNN)** (Gilmer et al., 2017), edges lie between any pair of nodes and $\bigoplus$ is the max pooling operation.

In **Pointer Graph Networks (PGN)** (Veličković et al., 2020), edges are constrained by the adjacency matrix and $\bigoplus$ is the max pooling operation.

## 3 RELATIONAL TRANSFORMER

We design the relational transformer (RT) to test our conjecture that a version of the transformer architecture, appropriately generalized to operate over graphs instead of sets, will be a competitive general-purpose model for graph-structured tasks. This goal leads us to the following design criteria:

1. Preserve all of the transformer's original machinery (though still not fully understood), for its proven advantages.
2. Introduce directed edge vectors to represent relations between entities.
3. Condition transformer attention on the edge vectors.
4. Extend the transformer layer to input edge vectors and output updated edge vectors.
5. Preserve the transformer's $O\left(N^2\right)$ computational complexity.

### 3.1 RELATIONAL ATTENTION

(See Appendix A for a mathematical overview of transformers.) In addition to accepting node vectors representing entity features (as do all transformers), RT also accepts edge vectors representing relation

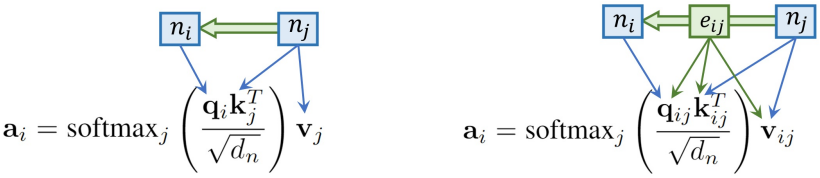


Figure 3: (Left) Standard transformer attention conditions the QKV computation on node vectors. (Right) Relational attention conditions this computation on the intervening edge vector as well.

features, which may include an adjacency matrix. But RT's graph is fully connected, unconstrained by any input adjacency matrix.

Transformer attention begins by projecting QKV vectors from each node vector (equation 18), then performing a dot-product between each pair of vectors $\mathbf{q}_i$ and $\mathbf{k}_j$ (equation 17). This dot-product determines the degree to which node $i$ attends to node $j$. Relational attention's central innovation (illustrated in Figure 3) is to condition the QKV vectors on the directed edge $\mathbf{e}_{ij}$ between the nodes, by concatenating that edge vector to each node vector prior to the linear transformations:

$$\mathbf{q}_{ij} = [\mathbf{n}_i, \mathbf{e}_{ij}]\mathbf{W}^Q \qquad \mathbf{k}_{ij} = [\mathbf{n}_j, \mathbf{e}_{ij}]\mathbf{W}^K \qquad \mathbf{v}_{ij} = [\mathbf{n}_j, \mathbf{e}_{ij}]\mathbf{W}^V \tag{4}$$

where each weight matrix $\mathbf{W}$ is now of size $\mathbb{R}^{(d_n+d_e)\times d_n}$, where $d_e$ is the edge vector size. To implement this efficiently, we split each weight matrix $\mathbf{W}$ into two separate matrices for projecting node and edge vectors, project the edge vector to three embeddings, then add those to the node's usual attention vectors:

$$\mathbf{q}_{ij} = \left(\mathbf{n}_i \mathbf{W}_n^Q + \mathbf{e}_{ij} \mathbf{W}_e^Q\right) \qquad \mathbf{k}_{ij} = \left(\mathbf{n}_i \mathbf{W}_n^K + \mathbf{e}_{ij} \mathbf{W}_e^K\right) \qquad \mathbf{v}_{ij} = \left(\mathbf{n}_i \mathbf{W}_n^V + \mathbf{e}_{ij} \mathbf{W}_e^V\right) \tag{5}$$

While we have described relational attention in terms of fully connected self-attention, it applies equally to restricted forms of attention such as causal attention, cross-attention, or even GAT-like attention that passes messages only over the edges present in a graph's adjacency matrix. Relational attention is orthogonal to multi-head attention, and leaves the transformer's $O\left(N^2\right)$ complexity unchanged. Our implementation maintains the high GPU utilization that makes transformers efficient.

### 3.2 Edge Updates

To update edge vectors in each layer of processing, we follow the general pattern used to update node vectors: first aggregate messages into one, then use the result to perform a local update. In self-attention each node attends to all nodes in the graph. Having each of the $N^2$ edges attend to $N$ nodes would raise computational complexity to $O\left(N^3\right)$. Instead, we restrict the edge's aggregation function to gather messages only from its immediate locale, which consists of its two adjoining nodes, itself, and the directed edge running in the opposite direction:

$$\mathbf{e}_{ij}^{l+1} = \phi_e\left(\mathbf{e}_{ij}^l, \mathbf{e}_{ji}^l, \mathbf{n}_i^{l+1}, \mathbf{n}_j^{l+1}\right) \tag{6}$$

Since these four input vectors are not interchangeable, their message projections can be aggregated through summation instead of attention. To implement this efficiently (and exactly), we compute the aggregated message $\mathbf{m}_{ij}^l$ by first concatenating the four neighbor vectors into one, then applying a single linear transformation to the concatenated vector, followed by a ReLU non-linearity:

$$\mathbf{m}_{ij}^l = \text{ReLU}\left(\text{concat}\left(\mathbf{e}_{ij}^l, \mathbf{e}_{ji}^l, \mathbf{n}_i^{l+1}, \mathbf{n}_j^{l+1}\right)\mathbf{W}_4\right) \tag{7}$$

where $\mathbf{W}_4 \in \mathbb{R}^{d_e \times d_{eh1}}$, and the non-linear ReLU operation takes the place of the non-linear softmax in regular attention.

The remainder of the edge update function is essentially identical to the transformer node update function (equation 21):

$$\mathbf{u}_{ij}^l = \text{LayerNorm}\left(\mathbf{m}_{ij}^l \mathbf{W}_5 + \mathbf{e}_{ij}^l\right) \qquad \mathbf{e}_{ij}^{l+1} = \text{LayerNorm}\left(\text{ReLU}(\mathbf{u}_{ij}^l \mathbf{W}_6)\mathbf{W}_7 + \mathbf{u}_{ij}^l\right) \tag{8}$$

where $\mathbf{W}_5 \in \mathbb{R}^{d_{eh1} \times d_e}$, $\mathbf{W}_6 \in \mathbb{R}^{d_e \times d_{eh2}}$, $\mathbf{W}_7 \in \mathbb{R}^{d_{eh2} \times d_e}$, and $d_{eh1}$ and $d_{eh2}$ are the hidden layer sizes of the edge feed-forward networks. Node vectors are updated before edge vectors within each RT layer, so that a node aggregates information from the entire graph before its adjoining edges use that information in their local updates. This is another instance of the aggregate-then-update pattern employed by both transformers and GNNs for node vectors.

## 4 Prior Work

A number of prior works have applied transformers to relational information, such as by using relative position encodings. To compare these formulations with RT, we expand relational attention's dot-product into four terms as follows:

$$\mathbf{q}_{ij}\mathbf{k}_{ij}^T = \left([\mathbf{n}_i, \mathbf{e}_{ij}]\mathbf{W}^Q\right)\left([\mathbf{n}_j, \mathbf{e}_{ij}]\mathbf{W}^K\right)^T \tag{9}$$

$$= \left(\mathbf{n}_i\mathbf{W}_n^Q + \mathbf{e}_{ij}\mathbf{W}_e^Q\right)\left(\mathbf{n}_i\mathbf{W}_n^K + \mathbf{e}_{ij}\mathbf{W}_e^K\right)^T \tag{10}$$

$$= \left(\mathbf{n}_i\mathbf{W}_n^Q + \mathbf{e}_{ij}\mathbf{W}_e^Q\right)\left(\left(\mathbf{W}_n^K\right)^T\mathbf{n}_i^T + \left(\mathbf{W}_e^K\right)^T\mathbf{e}_{ij}^T\right) \tag{11}$$

$$= \mathbf{n}_i\mathbf{W}_n^Q\left(\mathbf{W}_n^K\right)^T\mathbf{n}_i^T + \mathbf{n}_i\mathbf{W}_n^Q\left(\mathbf{W}_e^K\right)^T\mathbf{e}_{ij}^T + \mathbf{e}_{ij}\mathbf{W}_e^Q\left(\mathbf{W}_n^K\right)^T\mathbf{n}_i^T + \mathbf{e}_{ij}\mathbf{W}_e^Q\left(\mathbf{W}_e^K\right)^T\mathbf{e}_{ij}^T \tag{12}$$

$$= \mathbf{n}_i\mathbf{W}_n^Q\left(\mathbf{n}_i\mathbf{W}_n^K\right)^T + \mathbf{n}_i\mathbf{W}_n^Q\left(\mathbf{e}_{ij}\mathbf{W}_e^K\right)^T + \mathbf{e}_{ij}\mathbf{W}_e^Q\left(\mathbf{n}_i\mathbf{W}_n^K\right)^T + \mathbf{e}_{ij}\mathbf{W}_e^Q\left(\mathbf{e}_{ij}\mathbf{W}_e^K\right)^T \tag{13}$$

The transformer of Vaswani et al. (2017) employs only the first term. The transformer of Shaw et al. (2018) adds part of the second term, leaving out one weight matrix $\mathbf{W}_e^K$. The Transformer-XL (Dai et al., 2019) uses parts of all four terms, but leaves out two of the eight weight matrices $\mathbf{W}_e^Q$ and $\mathbf{W}_e^K$. Each prior work uses edge vectors only for relative positional information. RT employs all four terms, allows the edge vectors to represent arbitrary information depending on the task, and updates the edge vectors in each layer of computation.

We categorize, then discuss relevant works as types of graph-to-graph models. Each work is discussed in terms of transformer machinery and edge vector incorporation. See Figure 2 for visual comparisons.

**Graph Neural Networks** - GAT (Veličković et al., 2017) and GATv2 (Brody et al., 2021) use some transformer machinery, such as the self-attention mechanism. In (Veličković et al., 2022), the two GNNs partially incorporate edge information in the form of attention biases conditioned on edges. MPNN (Gilmer et al., 2017) accepts entire edge vectors as input, as do some other works such as MXMNet (Zhang et al., 2020b) and G-MPNN Yadati (2020). Even fewer works update the edge vectors themselves, for example EGNN (Gong & Cheng, 2018) and Censnet (Jiang et al., 2019). None of the above GNNs use the full transformer machinery, and in general many GNNs are designed for specific settings, such as quantum chemistry.

**Transformers** - We only discuss transformers applied to graph-structured problems. Since transformers, as set processors, do not natively consider graph structure, several works encode relations into the transformer's input features. These works make varying levels of modifications to the transformer architecture itself. For example, TokenGT (Kim et al., 2022) and Graphformer (Ying et al., 2021) use positional encoding schemes, while ReFormer (Yang et al., 2021), GraphTrans (Wu et al., 2022), and SAT (Chen et al., 2022) use initial convolutions or message passing to encode node features with either local, global, or relative graph information. In general, encoding schemes are largely task- and data-dependent (Rampášek et al., 2022). Other transformer works rely *primarily* on transformer modifications to consider graph structure. EGT (Hussain et al., 2022), Graphformer (Ying et al., 2021), and GRAT (Yoo et al., 2020) are some of the most similar to the original transformer, since they use fully connected, dot-product attention. They do not use full edge vectors to update node vectors, so we call them *partially relational transformers* in Figure 2. GPS (Rampášek et al., 2022), The Graph Transformer (Dwivedi & Bresson, 2020), and SAN (Kreuzer et al., 2021) perform edge updates; but only GPS performs round-trip updating in which full edge vectors condition node updates and full node vectors condition edge updates. It is unclear whether GPS's modifications to the transformer architecture interfere with the transformer's full functionality.

Most similar to the spirit of our work, Bergen et al. (2021) introduce the Edge Transformer which replaces standard transformer attention with a triangular attention mechanism that takes edge vectors into account and updates the edge vectors in each layer. This differs from RT in three important

respects. First, triangular attention is a completely novel form of attention, unlike relational attention which is framed as a natural generalization of standard transformer attention. Second, triangular attention ignores node vectors altogether, and thereby requires node input features and node output predictions to be somehow mapped onto edges. And third, triangular attention's computational complexity is $O\left(N^3\right)$ in the number of nodes, unlike RT's relational attention which maintains the $O\left(N^2\right)$ transformer complexity.

# 5 EXPERIMENTS

We hypothesize that by generalizing the transformer to operate over graphs instead of sets, RT should prove to be a competitive general-purpose model for graph-structured tasks. To test this conjecture, we evaluate RT against common GNNs on a diverse set of graph-structured tasks that measure reasoning power and generalization. RT outperforms these GNNs by wide margins, especially on tasks that require processing of node relations (see Section 5.1.5 for analysis).

## 5.1 STEP-BY-STEP REASONING

In the CLRS Algorithmic Reasoning Benchmark, each step in a task is framed as a graph-to-graph problem, even for algorithms that may seem unrelated to graphs. To give an example, for list sorting algorithms, each input list element is treated as a separate node and predecessor links are added to order the elements. Task data is organized into task inputs, task outputs, and 'hints', which are intermediate inputs and outputs for the intervening steps of an algorithm. Data is comprised of combinations of node, edge, and/or graph features, which can be of five possible types: scalars, categoricals, masks, binary masks, or pointers.

The CLRS Benchmark employs an encode-process-decode framework for model evaluation. Input features are encoded using linear layers, then passed to the model (called a processor) being tested. The model performs one step of computation on the inputs, then it outputs a set of node vectors, which are passed back to the model as input on the next step. On each step, the model's output node vectors are decoded by the framework (using linear layers) and compared with the targets (either hints or final outputs) to compute training losses or evaluation scores.

### 5.1.1 BASELINE GNNS

We began by reproducing the published results of key baseline models on the eight representative tasks (one per algorithm class) listed in Figure 3 of Veličković et al. (2022), and here in Table 16. For several of the following experiments, we refer to these as the *8 core tasks*. Our results on these tasks agree closely with the published CLRS results (Table 11). See Appendix C for details on our train/test protocol.

We observed that the CLRS benchmark results show sharp drops in OOD performance for all models. For instance, MPNN's average evaluation score dropped from 96.63% on the validation set to 51.02% on the test set. We note that small training datasets can induce overfitting even in models that are otherwise capable of generalizing to OOD test sets. To mitigate this spurious form of overfitting, we expanded the training datasets by 10x to 10,000 examples (generated from the same canonical random seed of 1), and evaluated the effects on the 8 core tasks. As shown in Table 9, expanding the training set significantly boosts the performance of all baseline models. For all of the other experiments in this work, we use the larger training sets of 10,000 examples.

Following Veličković et al. (2022), we compute results for two separate PGN models, masked (PGN-m) and unmasked (PGN-u), then select the best result on each task to compute the average shown for the combination PGN-c model (which is called PGN in the CLRS results). Note therefore that PGN-c does not represent a single model. But it does represent the performance that would be achievable by a PGN model that adaptively learned when to use masking.

We found in early experiments that RT obtained far better results than those of the CLRS baseline GNNs. So to further enhance the performance of the baseline GNN models, we extended them to support multiple layers (rounds of message passing) per algorithmic step, and thoroughly tuned their hyperparameters (see Appendix B). This significantly improved the baseline results (see Table 10).

Table 1: Average test scores of all tuned models across all algorithm classes.

| Class | Deep Sets | GAT-v1 | GAT-v2 | MPNN | PGN-u | PGN-m | PGN-c | RT (Ours) |
|---|---|---|---|---|---|---|---|---|
| Divide & Conquer | 12.29% | 15.19% | 14.80% | 16.14% | 16.71% | 51.30% | 51.30% | **66.52%** |
| Dynamic Programming | 68.29% | 63.88% | 59.22% | 68.81% | 68.56% | 71.07% | 71.13% | **83.20%** |
| Geometry | 65.47% | 68.94% | 67.80% | 83.03% | 67.77% | 66.63% | 67.78% | **84.55%** |
| Graphs | 42.09% | 52.79% | 55.53% | 63.30% | 59.16% | 64.36% | 64.59% | **65.33%** |
| Greedy | 77.83% | 75.75% | 75.03% | **89.40%** | 75.30% | 76.72% | 76.72% | 85.32% |
| Search | 50.99% | 35.37% | 38.04% | 43.94% | 50.98% | 54.21% | 60.39% | **65.31%** |
| Sorting | **68.89%** | 21.25% | 17.82% | 27.12% | 28.93% | 2.48% | 28.93% | 50.01% |
| Strings | 2.92% | 1.36% | 1.57% | 2.09% | 1.61% | 1.17% | 1.82% | **32.52%** |
| Average | 48.60% | 41.82% | 41.23% | 49.23% | 46.13% | 48.49% | 52.83% | **66.60%** |

#### 5.1.2 Enhancements of the CLRS Benchmark's framework

A layer in a fully general graph-to-graph models can consume and produce node vectors, edge vectors, and a global vector. Most GNNs consume and produce node vectors, and many also consume edge vectors or edge types. However, relatively few GNNs (and none of the CLRS baseline models) are designed to output edge or global vectors. Because of this, the CLRS framework does not support edge or global vector outputs from a processor network. To test models such as RT that have these abilities, we extended the CLRS framework to accept edge and global vectors from the processor at each step, and pass these vectors back to the processor as input in the next step. The framework handles node vectors as usual.

In casting algorithms as graph-to-graph tasks, the CLRS benchmark relies heavily on what it terms a *node pointer*, which is conceptually equivalent to a directed edge pointing from one node to another. Since the CLRS baseline models do not output edge vectors, a decoder in the CLRS framework uses the model's output node vectors to create node pointers. But for models like RT that output edge vectors, it is more natural to decode a node pointer from those edge vectors alone. To better support such models, we added a flag to enable this modified behavior in the CLRS framework.

#### 5.1.3 Main results

After tuning hyperparameters for all models (Appendix B), we evaluated RT against six baseline GNNs on all 30 CLRS Benchmark tasks, using 20 seeds. The full results are presented in Table 18. RT outperforms the top-scoring baseline model (MPNN) by **11%** overall. As bolded in the table, RT scores the highest on **12 out of 30** tasks. RT is also the best-performing model on **6 of 8** algorithmic classes (Table 1), and scores the highest when results are averaged over those classes (Figure **??**). See Table 8 for the algorithms-class mappings. In summary, RT significantly outperforms all baseline GNN models over the 30 diverse and challenging CLRS Benchmark tasks.

#### 5.1.4 Ablations

Using only the 8 core tasks (except where noted), we perform several ablations to analyze the factors behind RT's solid performance on the CLRS Benchmark.

**Transformer** - We compare RT to a standard, set-based transformer (Vaswani et al., 2017) by disabling edge vectors in RT. Table 12 shows that performance collapses by 40% without edge vectors and relational attention.

**Layers** - The tuned RT uses three layers of computation per algorithmic step. When restricted to a single layer, performance drops drastically (Table 13). However, single-layer RT still outperforms the top-scoring MPNN by **9.39%** on the 8 core tasks, suggesting that relational attention improves expressivity even when restricted to a single layer of computation.

**Global vector** - Many CLRS tasks provide a global feature vector as input to the processor model. We designed RT to handle this global vector by either concatenating it to each input node vector, or by passing it to a dedicated core node (Loynd et al., 2019; Guo et al., 2019). Hyperparameter tuning reported higher validation scores using concatenation instead of the core node option, so concatenation was used in all experiments. But in this ablation, the core-node method obtained slightly higher test scores on 7 tasks we selected that use global vectors as inputs to the processor

(Table 15). The score difference of 0.08% was marginal and not statistically significant, which suggests that both methods (core node and concatenation) are effective at handling global features.

**Node pointer decoding** - We assess the flag we added to the CLRS framework, which can be used to decode node pointers from edge vectors only. Compared to using the original decoding procedure, using our flag improved performance by a small amount (0.30%) (Table 14).

**Disabling edge updates** - We disable edge updates in RT such that RT relies solely on relational attention to process input features. Table 16 shows the resulting drop in performance, from **81.30%** to **53.99%**. This indicates that edge updates are crucial for learning relational characteristics in the graph data. As a final note, RT without edge updates still outperforms the transformer by **12.11%**, demonstrating the effectiveness of relational attention even without expressive edges.

#### 5.1.5 Algorithmic Analysis

We investigate the reasoning power of RT based on its test performance on specific algorithmic tasks.

**Underperformance** - The only class of CLRS tasks where RT underperforms significantly is greedy algorithms. The two greedy tasks, activity selector and task scheduling, require selecting node entities that minimize some metric at each step. For example, in task scheduling, the optimal solution involves repeatedly selecting the task with the smallest processing time. We hypothesize that RT underperforms here because it performs soft attention pooling on nodes, which may not align with the discrete choices required for greedy solutions. MPNN, PGN-u, and PGN-m, on the other hand, perform max pooling, which is better suited to making discrete decisions over neighborhoods (Veličković et al., 2019). This may also explain RT's underperformance on Prim's, Articulation Points, and Bridges.

**Overperformance** - RT overwhelmingly beats baseline GNNs on dynamic programming (DP) tasks. We consider DP's general form (Xu et al., 2019):

$$\text{Answer}[l+1][i] = \text{DP-Update}(\{\text{Answer}[l][j]\}, j = 1, \ldots, n) \tag{14}$$

where $\text{Answer}[l+1][i]$ is the solution to the sub-problem indexed by iteration $l$ and state $i$, and DP-Update is a task-specific update function that computes $\text{Answer}[l+1][i]$ from $\text{Answer}[l][j]$'s. In the 1-D case, node states $\mathbf{n}_i^{l+1}$ are $\text{Answer}[l+1][i]$ and relational attention performs DP-Update. We posit that RT learns more expressive edges that relational attention uses to update $\text{Answer}[l+1][i]$ across nodes $i$. For 2-D DP, like the Matrix Chain Order task, RT can explicitly materialize the DP solution table as its edge vector matrix, and perform updates on each entry using edge updates.

### 5.2 End-To-End Algorithmic Reasoning

The CLRS Benchmark evaluates reasoning ability by examining how closely a model can emulate each step of an algorithm. But we may also evaluate reasoning ability by learning an algorithm end-to-end, based on how closely the model's output matches the algorithm's output. Many works such as Xu et al. (2019); Tang et al. (2020) take this approach. We use the tasks provided by Tang et al. (2020) to evaluate RT. Specifically, we task RT with finding a shortest path distance between two nodes in an undirected lobster graph. The node features are one-hot encoded (100 for source, 010 for destination, and 001 for other nodes). The edge features are binary values (1 for present, 0 for absent).

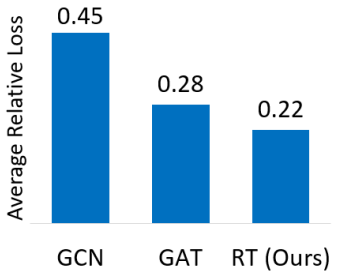


Figure 4: Average Relative Loss on Shortest Paths

**Main Results** We use the same experiment settings as Tang et al. (2020) for the shortest path task. Importantly, we use their method for testing graph size generalizability, by training models on graphs of size $[4, 34)$ and evaluating models on graphs of size 100. Furthermore, layer size for all models is set to 30. Results are reported using the relative loss metric introduced by the authors, defined as $|y - \hat{y}|/|y|$ given a label $y$ and a prediction $\hat{y}$. We compare RT to their two baselines, Graph Convolution Network or

GCN (Kipf & Welling, 2016) and Graph Attention Network or GAT (Veličković et al., 2017). Results were averaged across 20 random seeds. RT outperformed both baselines with an average relative loss of **0.22**, compared to GCN's **0.45** and GAT's **0.28** (Figure 4). These results were obtained without using the iterative module proposed by Tang et al. (2020) that introduces a stopping criterion to message passing computations.

### 5.3 Reasoning Over Hidden Graph Structure

In the experiments described so far, the model always received edge vectors as inputs. The question we pose here is whether RT's latent edge vectors can improve reasoning ability even on tasks with graph structures that are *hidden*, rather than passed to the model in the form of edge vectors. We use the Sokoban (Guez et al., 2019) reinforcement learning task to investigate. In Sokoban, the agent must push four yellow boxes onto the red targets within 120 time steps. Humans solving these puzzles tend to plan out which boxes will go onto which targets. Assuming that a successful agent will learn a similar strategy, representing the box-to-target pair as a directed relation, we hypothesize that RT is more capable than a plain transformer at reasoning over such pairwise relations. For evaluation, we use RT in place of the plain transformer originally used by the Working Memory Graph (WMG) RL agent (Loynd et al., 2019), then train both modified and unmodified agents on the Sokoban task for 10 million time steps.

**Main Results** We find that using RT reduces the agent's final error rate by a relative **15%**, from 34% to 29% of the puzzles. This improvement is much larger than the confidence intervals (0.3% and 0.7% standard error, respectively). Our results support the hypothesis that RT can learn even hidden graph structure. One alternative explanation would be that the extra trainable parameters added by RT improved the expressivity of WMG. But this possibility is undercut by the fact that hyperparameter tuning of the unmodified WMG agent (Loynd et al., 2019) converged to an intermediate model size, rather than a larger model for more expressivity.

## 6 Conclusion and future work

We propose the relational transformer (RT), a gentle generalization of the standard transformer to operate on graphs instead of sets, incorporating edge information through relational attention in a principled and computationally efficient way. Our experimental results demonstrate that RT is a competitive general-purpose model across a large number of graph-structured tasks. Specifically, RT outperforms baseline GNNs over the 30 CLRS Algorithmic Reasoning Benchmark tasks by wide margins, and also outperforms baseline models on end-to-end algorithmic reasoning. RT even boosts performance on the Sokoban task, where graph structure is entirely hidden and must be discovered by the RL agent.

Beyond establishing RT's state-of-the-art results on the CLRS Benchmark, we enhance performance of the CLRS baseline models, and contribute extensions to the CLRS framework, broadening the scope of models that can be evaluated on the benchmark tasks. All of these improvements make the CLRS Benchmark and baselines more appealing for evaluating future models.

In future work, we aim to fully leverage the richness of the CLRS Benchmark to more thoroughly investigate RT's capabilities. For example, we can test RT's end-to-end reasoning by disabling hints and only evaluating our model on the final algorithmic solution. Another promising research direction involves knowledge transfer between algorithms. (Veličković et al., 2019) find positive knowledge transfer between the shortest paths and reachability tasks, when MPNN is trained to solve both concurrently. The CLRS tasks can be used to explore knowledge transfer in depth: for example, by pretraining a model on some subset of the 30 tasks, then finetuning and evaluating the model on a disjoint subset of related tasks.

Beyond the CLRS Benchmark experiments, we plan to evaluate RT on a wider range of real-world graph settings, such as the molecular domain using the large-scale QM9 dataset (Wu et al., 2018). Finally, we hope to relax the locality bottleneck of RT's edge updates by allowing edges to attend to other edges directly in a computationally efficient manner.

## REFERENCES

Chongyang Bai, Srijan Kumar, Jure Leskovec, Miriam Metzger, Jay F. Nunamaker, and V. S. Subrahmanian. Predicting the visual focus of attention in multi-person discussion videos. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4504–4510. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/626. URL https://doi.org/10.24963/ijcai.2019/626.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. URL https://arxiv.org/abs/1806.01261.

Leon Bergen, Timothy J. O'Donnell, and Dzmitry Bahdanau. Systematic generalization with edge transformers, 2021. URL https://arxiv.org/abs/2112.00578.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 2787–2795, Red Hook, NY, USA, 2013. Curran Associates Inc.

KM Borgwardt, CS Ong, Stefan Schönauer, SVN Vishwanathan, Alexander Smola, and HP Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1:i47–56, 01 2005.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2021. URL https://arxiv.org/abs/2105.14491.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning, 2022. URL https://arxiv.org/abs/2202.03036.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021. URL https://arxiv.org/abs/2106.01345.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019. URL https://arxiv.org/abs/1901.02860.

Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991. doi: 10.1021/jm00106a046. URL https://doi.org/10.1021/jm00106a046.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. URL https://arxiv.org/abs/2010.11929.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2020. URL https://arxiv.org/abs/2012.09699.

C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: an automatic citation indexing system. In *Proceedings of the ACM International Conference on Digital Libraries*, pp. 89–98. ACM, 1998. Proceedings of the 1998 3rd ACM Conference on Digital Libraries ; Conference date: 23-06-1998 Through 26-06-1998.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

Liyu Gong and Qiang Cheng. Exploiting edge features in graph neural networks, 2018. URL https://arxiv.org/abs/1809.02709.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An investigation of model-free planning. In *ICML*, pp. 2464–2473, 2019.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer, 2019. URL https://arxiv.org/abs/1902.09113.

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs, 2021. URL https://arxiv.org/abs/2103.09430.

Md Shamim Hussain, Mohammed J. Zaki, and Dharmashankar Subramanian. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, aug 2022. doi: 10.1145/3534678.3539296. URL https://doi.org/10.1145%2F3534678.3539296.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem, 2021. URL https://arxiv.org/abs/2106.02039.

Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 2656–2662. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/369. URL https://doi.org/10.24963/ijcai.2019/369.

Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners, 2022. URL https://arxiv.org/abs/2207.02505.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016. URL https://arxiv.org/abs/1609.02907.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention, 2021. URL https://arxiv.org/abs/2106.03893.

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, 2008. URL https://arxiv.org/abs/0810.1355.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pp. 1361–1370, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589299. doi: 10.1145/1753326.1753532. URL https://doi.org/10.1145/1753326.1753532.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning, 2018. URL https://arxiv.org/abs/1801.07606.

Ricky Loynd, Roland Fernandez, Asli Celikyilmaz, Adith Swaminathan, and Matthew Hausknecht. Working memory graphs, 2019. URL https://arxiv.org/abs/1911.07141.

Andrew Kachites Mccallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning, 2019. URL https://arxiv.org/abs/1910.06764.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer, 2022. URL https://arxiv.org/abs/2205.12454.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018. URL https://arxiv.org/abs/1803.02155.

Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous graph neural networks, 2020. URL https://arxiv.org/abs/2010.13547.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Petar Veličković, Lars Buesing, Matthew Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 2232–2244. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/176bf6219855a6eb1f3a30903e34b6fb-Paper.pdf.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms, 2019. URL https://arxiv.org/abs/1910.10593.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark, 2022. URL https://arxiv.org/abs/2205.15659.

Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention, 2022. URL https://arxiv.org/abs/2201.08821.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530, 2018. doi: 10.1039/C7SC02664A. URL http://dx.doi.org/10.1039/C7SC02664A.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about?, 2019. URL https://arxiv.org/abs/1905.13211.

Naganand Yadati. Neural message passing for multi-relational ordered and recursive hypergraphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3275–3289. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/217eedd1ba8c592db97d0dbe54c7adfc-Paper.pdf.

Xuewen Yang, Yingru Liu, and Xin Wang. Reformer: The relational transformer for image captioning, 2021. URL https://arxiv.org/abs/2107.14178.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? 2021. doi: 10.48550/ARXIV.2106.05234. URL https://arxiv.org/abs/2106.05234.

Sanghyun Yoo, Young-Seok Kim, Kang Hyun Lee, Kuhwan Jeong, Junhwi Choi, Hoshik Lee, and Young Sang Choi. Graph-aware transformer: Is attention all graphs need?, 2020. URL https://arxiv.org/abs/2006.05213.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets, 2017. URL https://arxiv.org/abs/1703.06114.

Liang Zhang, Xudong Wang, Hongsheng Li, Guangming Zhu, Peiyi Shen, Ping Li, Xiaoyuan Lu, Syed Afaq Ali Shah, and Mohammed Bennamoun. Structure-feature based graph self-adaptive pooling. In *Proceedings of The Web Conference 2020*. ACM, apr 2020a. doi: 10.1145/3366423.3380083. URL https://doi.org/10.1145%2F3366423.3380083.

Shuo Zhang, Yang Liu, and Lei Xie. Molecular mechanics-driven graph neural network with multiplex graph for molecular structures, 2020b. URL https://arxiv.org/abs/2011.07457.

# A TRANSFORMERS

We describe the transformer architecture introduced by Vaswani et al. (2017). This description also applies to most transformer variants proposed over the years. Layer superscripts are employed to distinguish input vectors from output vectors, and are often omitted for vectors inside the same layer.

Although the transformer is a set-to-set model, it can be described using the graph-to-graph formalism as limited to computation over *nodes* only. Each transformer layer is a function passing updated node vectors to the next layer. A single transformer layer can therefore be expressed as a modified version of equation 1:

$$\mathbf{n}_i^{l+1} = \phi_n \left( \mathbf{n}_i^l, \bigoplus_{j \in \mathcal{N}_i} a\left(\mathbf{n}_i^l, \mathbf{n}_j^l\right) \psi^m\left(\mathbf{n}_j^l\right) \right) \tag{15}$$

where $a\left(\mathbf{n}_i^l, \mathbf{n}_j^l\right)$ computes the attentional coefficient $\alpha_{ij}^l$ applied by node $i$ to the value vector $\mathbf{v}_j^l$, which is computed by $\psi^m\left(\mathbf{n}_j^l\right)$, a linear transformation:

$$\alpha_{ij}^l = a\left(\mathbf{n}_i^l, \mathbf{n}_j^l\right) \qquad \mathbf{v}_j^l = \psi^m\left(\mathbf{n}_j^l\right) := \mathbf{n}_j^l W^V \tag{16}$$

The attentional coefficients applied by node $i$ to the set of all nodes $j$ is a probability distribution $\mathbf{a}_i$ computed by the softmax function over a set of vector dot products:

$$\mathbf{a}_i = \text{softmax}_j \left( \frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d_n}} \right) \tag{17}$$

The *QKV* vectors introduced above are linear transformations of node vectors:

$$\mathbf{q}_i = \mathbf{n}_i W^Q \qquad \mathbf{k}_j = \mathbf{n}_j W^K \qquad \mathbf{v}_j = \mathbf{n}_j W^V \tag{18}$$

where $\mathbf{W}^Q \in \mathbb{R}^{d_n \times d_n}$, $\mathbf{W}^K \in \mathbb{R}^{d_n \times d_n}$, $\mathbf{W}^V \in \mathbb{R}^{d_n \times d_n}$, and $d_n$ is the node vector size. These $\mathbf{W}$ matrices (like all other trainable parameters in $\phi_n$) are not shared between transformer layers.

The aggregation function $\bigoplus$ sums the incoming messages from all nodes $\mathcal{N}_i$ in the completely connected graph:

$$\mathbf{m}_i^l = \bigoplus_{j \in \mathcal{N}_i} a\left(\mathbf{n}_i^l, \mathbf{n}_j^l\right) \psi^m\left(\mathbf{n}_j^l\right) := \sum_j \alpha_{ij}^l \mathbf{v}_j^l \tag{19}$$

This aggregated message $\mathbf{m}_i^l$ is then passed to the local update function $\phi_n$ (shared by all nodes), which is the following stack of linear layers, skip connections, layer normalization and a ReLU activation function:

$$\mathbf{u}_i^l = \text{LayerNorm}\left(\mathbf{m}_i^l \mathbf{W}_1 + \mathbf{n}_i^l\right) \tag{20}$$

$$\mathbf{n}_i^{l+1} = \text{LayerNorm}\left(\text{ReLU}(\mathbf{u}_i^l \mathbf{W}_2)\mathbf{W}_3 + \mathbf{u}_i^l\right) \tag{21}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_n \times d_n}$, $\mathbf{W}_2 \in \mathbb{R}^{d_n \times d_{nh}}$, $\mathbf{W}_3 \in \mathbb{R}^{d_{nh} \times d_n}$, and $d_{nh}$ is the hidden layer size of the feed-forward network.

This overview of the transformer architecture has focused on the fully connected case of self-attention. For brevity we have omitted the details of multi-head attention, bias vectors, and the stacking of vectors into matrices for maximal GPU utilization.

# B HYPERPARAMETERS

To tune the hyperparameters of RT and the CLRS baseline GNNs, we used Distributed Grid Descent (DGD) (Loynd et al., 2019), a self-guided form of random search. All searches were run to convergence, for the total numbers of runs listed in Table X. All tuning runs used the CLRS protocol described in C, except for the following details:

1. To prevent tuning on the canonical datasets or any fixed datasets at all, the dataset generation seeds were randomized at the start of each run.
2. To reduce variance, the minimum evaluation dataset size was raised from 32 to 100.
3. To mitigate the computational costs, all models were tuned on only the 8 core algorithms, and each training run was shortened to 32,000 examples.

Very similar procedures were used to tune RT hyperparameters for the other (non-CLRS) experiments.

### B.1 CLRS EXPERIMENTS

Table 2 lists the tuned hyperparameter values for CLRS experiments, and Table 3 reports the sets of values considered in those searches. The runtime sizes of the corresponding models are found in Table 4, along with their training speeds in Table 5.

Table 2: Tuned hyperparameter values for CLRS experiments.

| | **Deep Sets** | **GAT-v1** | **GAT-v2** | **MPNN** | **PGN-u** | **PGN-m** | **RT** |
|---|---|---|---|---|---|---|---|
| num_layers | 2 | 2 | 2 | 1 | 1 | 3 | 3 |
| learning_rate | 2.5e-4 | 6.3e-4 | 2.5e-4 | 1.6e-3 | 1.6e-3 | 1e-3 | 2.5e-4 |
| $d_n = d_e = d_g$ | 512 | - | - | 512 | 180 | 512 | - |
| nb_heads | - | 10 | 12 | - | - | - | 12 |
| head_size | - | 64 | 64 | - | - | - | 16 |
| $d_{nh}$ | - | - | - | - | - | - | 32 |
| $d_{eh1}$ | - | - | - | - | - | - | 16 |
| $d_{eh2}$ | - | - | - | - | - | - | 8 |
| ptr_from_edges | - | - | - | - | - | - | true |
| graph_vec | - | - | - | - | - | - | cat |

Table 3: Hyperparameter values considered for CLRS experiments.

| | |
|---|---|
| num_layers | 1, 2, 3, 4 |
| learning_rate | 4e-5, 6.3e-5, 1e-4, 1.6e-4, 2.5e-4, 4e-4, 6.3e-4, 1e-3, 1.6e-3, 2.5e-3, 4e-3, 6.3e-3, 1e-2 |
| $d_n = d_e = d_g$ | 45, 64, 90, 128, 180, 256, 360, 512 |
| nb_heads | 3, 4, 6, 8, 10, 12, 16 |
| head_size | 8, 12, 16, 24, 32, 45, 64 |
| $d_{nh}$ | 4, 6, 8, 12, 16, 24, 32, 45, 64, 90 |
| $d_{eh1}$ | 12, 16, 24, 32, 45, 64 |
| $d_{eh2}$ | 4, 6, 8, 12, 16, 24, 32, 45 |
| ptr_from_edges | false, true |
| graph_vec | core, cat |

Table 4: Number of trainable parameters in each model tested on CLRS, including the framework's encoding and decoding layers, on the reference algorithm Bellman Ford.

| | **Deep Sets** | **GAT-v1** | **GAT-v2** | **MPNN** | **PGN-u** | **PGN-m** | **RT** |
|---|---|---|---|---|---|---|---|
| tuned | 8,675,332 | 11,168,732 | 17,730,076 | 6,573,060 | 816,844 | 10,777,604 | 1,103,404 |
| untuned | 414,468 | 418,960 | 397,957 | 414,468 | 414,468 | 414,468 | - |

### B.2 LOBSTER GRAPH HYPERPARAMETERS

Table 6 lists the tuned hyperparameter values for RT on the shortest-path task over lobster graphs, along with the sets of values considered in that search.

### B.3 SOKOBAN HYPERPARAMETERS

Table 7 lists the tuned hyperparameter values for RT on the Sokoban task, along with the sets of values considered in that search.

Table 5: Training speed in examples per second on a T4 GPU, on the reference algorithm Bellman Ford.

| | **Deep Sets** | **GAT-v1** | **GAT-v2** | **MPNN** | **PGN-u** | **PGN-m** | **RT** |
|---|---|---|---|---|---|---|---|
| tuned | 13.4 | 14.8 | 10.1 | 15.7 | 18.7 | 11.6 | 8.7 |
| untuned | 128.9 | 140.7 | 134.8 | 128.8 | 128.6 | 128.6 | - |

Table 6: Tuned hyperparameter values for RT on the lobster graph experiment.

| | **Tuned values** | **Values considered** |
|---|---|---|
| learning_rate | 6.3e-5 | 4e-5, 6.3e-5, 1e-4, 1.6e-4, 2.5e-4, 4e-4 |
| output_layer_size | 180 | 64, 90, 128, 180, 256, 360, 512 |
| nb_heads | 8 | 3, 4, 6, 8, 10 |
| head_size | 28 | 12, 16, 20, 24, 28, 32 |
| $d_{nh}$ | 12 | 3, 4, 6, 8, 12, 16, 24 |
| $d_e$ | 128 | 16, 24, 32, 45, 64, 90, 128 |
| $d_{eh1}$ | 32 | 12, 16, 24, 32, 45, 64, 90 |
| $d_{eh2}$ | 8 | 8, 12, 16, 24, 32, 45, 64 |
| dropout_rate | 0.0 | 0.0, 0.01, 0.02, 0.04, 0.08 |
| grad_clip | 128.0 | 16.0, 32.0, 64.0, 128.0, 256.0, 512.0 |

## C TRAIN/TEST PROTOCOL

Except where specifically noted, our experiments follow the exact train/test protocol defined by the CLRS Benchmark. CLRS provides canonical datasets (training, validation, and test) which can also be generated from specific random seeds: 1, 2, 3. The graphs in the training and validation datasets contain 16 nodes, while the test graphs are of size 64 to evaluate the out-of-distribution (OOD) generalization of models. Training is performed on a random sequence of 320,000 examples drawn with replacement from the train set, where each example contains a variable number of reasoning steps. During training, the model is evaluated on the validation set after every 320 examples. At the end of training, the model with the highest validation score is evaluated on the test set. The average test micro-F1 score is report for all results. The published CLRS benchmark results are reported as averages over 3 random run seeds, but we use 20 seeds in all of our experiments.

## D TEST RESULTS ON THE CLRS BENCHMARK

Test performance of all tuned models evaluated on the CLRS Benchmark may be found in Tables 18 (mean accuracy) and 17 (standard deviation). Following Veličković et al. (2022), we report the best performance between PGN-u and PGN-m on every task in the column titled PGN-c (for PGN-combination). Note therefore that PGN-c does not represent a single model.

We bold the results of the best-performing single model for each specific task. RT was the best-performing on **15 out of 30** of the tasks. Overall, MPNN was the top-scoring baseline model, winning on 9 tasks. PGN-u won on 1 task, and PGN-m won on 5 tasks.

Table 1 shows test performances of all tuned models across 8 algorithm classes. As mentioned in Section 5.1.3, RT performed best in **6 out of 8** of the classes, only losing on greedy algorithms to MPNN.

Table 7: Tuned hyperparameter values for RT on Sokoban.

| | **Tuned values** | **Values considered** |
|---|---|---|
| $d_e$ | 128 | 8, 12, 16, 24, 32, 45, 64, 90, 128, 180, 256, 360, 512, 720, 1024 |
| $d_{eh1}$ | 64 | 6, 8, 12, 16, 24, 32, 45, 64, 90, 128, 180, 256, 360, 512 |
| $d_{eh2}$ | 12 | 6, 8, 12, 16, 24, 32, 45, 64, 90, 128, 180, 256, 360, 512 |

Table 8 shows which algorithms belong to which classes.

Table 9 shows test performance of the top-scoring baseline models trained on datasets of varying size, without hyperparameter tuning. Specifically, we compare test scores between the models trained on datasets of size 1000, and datasets of size 10000. We note that the results in the first row agree closely with those in the CLRS paper (Table 11).

## E CLRS BENCHMARK ABLATIONS

We perform our ablation studies on the 8 core CLRS Benchmark tasks. Table 13 shows the drop in test performance that resulted from restricting RT to one layer. The original, tuned RT had 3 layers, and is labeled RT-3. Table 14 shows the marginal improvement that resulted from decoding node vectors using only edge vectors. Finally, Table 15 shows the effects of handling global input vectors through a core node (vs. concatenation with the input node vectors) has on RT test performance. RT with core node only won on 3 out of 7 tasks, but had the higher average test performance by 0.08%.

Table 8: Algorithms and their respective classes

| Algorithm | D & C | DP | Geometry | Graphs | Greedy | Search | Sorting | Strings |
|---|---|---|---|---|---|---|---|---|
| Activity Selector | | | | | ■ | | | |
| Articulation Points | | | | ■ | | | | |
| Bellman-Ford | | | | ■ | | | | |
| BFS | | | | ■ | | | | |
| Binary Search | | | | | | ■ | | |
| Bridges | | | | ■ | | | | |
| Bubble Sort | | | | | | | ■ | |
| DAG Shortest Paths | | | | ■ | | | | |
| DFS | | | | ■ | | | | |
| Dijkstra | | | | ■ | | | | |
| Find Max. Subarray | ■ | | | | | | | |
| Floyd-Warshall | | | | ■ | | | | |
| Graham Scan | | | ■ | | | | | |
| Heapsort | | | | | | | ■ | |
| Insertion Sort | | | | | | | ■ | |
| Jarvis' March | | | ■ | | | | | |
| KMP Matcher | | | | | | | | ■ |
| LCS Length | | ■ | | | | | | |
| Matrix Chain Order | | ■ | | | | | | |
| Minimum | | | | | | ■ | | |
| MST-Kruskal | | | | ■ | | | | |
| MST-Prim | | | | ■ | | | | |
| Naïve String Match | | | | | | | | ■ |
| Optimal BST | | ■ | | | | | | |
| Quickselect | | | | | | ■ | | |
| Quicksort | | | | | | | ■ | |
| Segments Intersect | | | ■ | | | | | |
| SCC | | | | ■ | | | | |
| Task Scheduling | | | | | ■ | | | |
| Topological Sort | | | | ■ | | | | |
| Total | 1 | 3 | 3 | 12 | 2 | 3 | 4 | 2 |

Table 9: Average test scores of untuned baseline models trained on either the original small datasets, versus trained on the expanded training datasets, for the 8 core tasks.

| Training Set Size | Deep Sets | GAT-v1 | GAT-v2 | MPNN | PGN-u | PGN-m | PGN-c |
|---|---|---|---|---|---|---|---|
| 1000 | 45.95% | 42.24% | 42.53% | 54.66% | 51.07% | 56.18% | 61.29% |
| 10000 | 46.70% | 42.97% | 45.37% | 56.93% | 56.77% | 58.20% | 67.11% |

Table 10: Test score improvement from hyperparameter tuning on the 8 core tasks

| Algorithm | Deep Sets | GAT-v1 | GAT-v2 | MPNN | PGN-u | PGN-m | PGN-c |
|---|---|---|---|---|---|---|---|
| No Tuning | 46.70% | 42.97% | 45.37% | 56.93% | 56.77% | 58.20% | 67.11% |
| Tuning | 51.55% | 50.47% | 48.97% | 59.75% | 59.97% | 57.53% | 67.86% |

Table 11: Reproduction of baseline model results on the 8 core algorithms.

| Results | # Seeds | Deep Sets | MPNN | PGN-c |
|---|---|---|---|---|
| Published (Veličković et al., 2022) | 3 | 45.50% | 53.05% | 61.15% |
| Reproduced (Ours) | 20 | 45.95% | 54.66% | 61.29% |

Table 12: Plain transformer ablation evaluated on the 8 core algorithms

| Algorithm | Transformer | RT (Ours) |
|---|---|---|
| Activity Selector | 82.25% | **87.72%** |
| Bellman-Ford | 40.16% | **94.24%** |
| Binary Search | 3.43% | **81.48%** |
| Find Max. Subarray | 30.19% | **66.52%** |
| Graham Scan | 70.25% | **74.15%** |
| Insertion Sort | 23.31% | **89.43%** |
| Matrix Chain Order | 84.06% | **91.89%** |
| Naïve String Match | 1.39% | **65.01%** |
| Average | 41.88% | **81.30%** |

Table 13: Single-layer RT ablation evaluated on the 8 core algorithms.

| Algorithm | RT-1 | RT-3 (Ours) |
|---|---|---|
| Activity Selector | 85.10% | **87.72%** |
| Bellman-Ford | 94.09% | **94.24%** |
| Binary Search | 68.22% | **81.48%** |
| Find Max. Subarray | 63.61% | **66.52%** |
| Graham Scan | 68.77% | **74.15%** |
| Insertion Sort | 80.36% | **89.43%** |
| Matrix Chain Order | 91.20% | **91.89%** |
| Naïve String Match | 1.77% | **65.01%** |
| Average | 69.14% | **81.30%** |

Table 14: Ablation of the node pointer decoding procedure evaluated on the 8 core algorithms.

| Algorithm | RT with original decoding | RT with edge-only decoding (Ours) |
|---|---|---|
| Activity Selector | **88.09%** | 87.72% |
| Bellman-Ford | **94.95%** | 94.24% |
| Binary Search | 80.06% | **81.48%** |
| Find Max. Subarray | **67.44%** | 66.52% |
| Graham Scan | **75.71%** | 74.15% |
| Insertion Sort | 71.32% | **89.43%** |
| Matrix Chain Order | **92.12%** | 91.89% |
| Naïve String Match | **78.43%** | 65.01% |
| Average | 81.00% | **81.30%** |

Table 15: Ablation of RT core node (vs. concatenation of the global vector) evaluated on 7 representative algorithms that use global input features or hints.

| Algorithm | RT with core node | RT without core node (Ours) |
|---|---|---|
| Binary Search | 75.40% | **81.48%** |
| Find Max. Subarray | **66.96%** | 66.52% |
| Graham Scan | 71.83% | **74.15%** |
| Heapsort | 30.67% | **32.96%** |
| KMP Matcher | 0.02% | **0.03%** |
| MST-Kruskal | **75.59%** | 64.91% |
| Task Scheduling | **83.09%** | 82.93% |
| Average | **57.65%** | 57.57% |

Table 16: Ablation of RT's edge update procedure evaluated on the 8 core algorithms

| **Algorithm** | **RT without edge updates** | **RT with edge updates (Ours)** |
|---|---|---|
| Activity Selector | 82.13% | **87.72%** |
| Bellman-Ford | 91.03% | **94.24%** |
| Binary Search | 60.29% | **81.48%** |
| Find Max. Subarray | 19.09% | **66.52%** |
| Graham Scan | 69.19% | **74.15%** |
| Insertion Sort | 22.30% | **89.43%** |
| Matrix Chain Order | 85.68% | **91.89%** |
| Naïve String Match | 2.22% | **65.01%** |
| Average | 53.99% | **81.30%** |

Table 17: Standard deviations over 20 seeds for all tuned models on all algorithms

| **Algorithm** | **Deep Sets** | **GAT-v1** | **GAT-v2** | **MPNN** | **PGN-u** | **PGN-m** | **RT (Ours)** |
|---|---|---|---|---|---|---|---|
| Activity Selector | 1.7% | 1.4% | 2.1% | 1.3% | 2.6% | 1.9% | 2.7% |
| Articulation Points | 6.0% | 8.0% | 6.3% | 6.1% | 3.5% | 7.5% | 14.6% |
| Bellman-Ford | 2.4% | 1.5% | 1.6% | 1.9% | 0.9% | 1.3% | 1.5% |
| BFS | 1.0% | 0.4% | 0.4% | 0.2% | 0.3% | 0.5% | 0.7% |
| Binary Search | 3.8% | 5.8% | 7.9% | 5.0% | 10.4% | 3.7% | 6.7% |
| Bridges | 4.8% | 6.4% | 8.0% | 17.8% | 11.0% | 7.8% | 11.8% |
| Bubble Sort | 3.1% | 2.5% | 1.3% | 5.0% | 1.9% | 0.2% | 13.0% |
| DAG Shortest Paths | 3.8% | 2.3% | 2.2% | 1.6% | 3.2% | 0.9% | 1.6% |
| DFS | 1.9% | 2.1% | 1.7% | 2.7% | 1.5% | 1.5% | 10.5% |
| Dijkstra | 3.6% | 9.5% | 7.1% | 4.3% | 11.0% | 3.4% | 5.8% |
| Find Max. Subarray | 1.5% | 2.4% | 2.1% | 2.9% | 1.6% | 15% | 3.7% |
| Floyd-Warshall | 4.2% | 6.0% | 3.7% | 4.7% | 4.8% | 2.5% | 7.6% |
| Graham Scan | 3.3% | 6.6% | 5.3% | 1.4% | 5.7% | 6.0% | 7.4% |
| Heapsort | 8.8% | 3.8% | 4.1% | 7.1% | 8.9% | 0.4% | 14.8% |
| Insertion Sort | 1.5% | 14.8% | 11.6% | 9.1% | 11.2% | 0.2% | 9.0% |
| Jarvis' March | 8.6% | 2.6% | 3.9% | 29.3% | 2.5% | 5.8% | 2.2% |
| KMP Matcher | 1.0% | 0.6% | 0.6% | 1.1% | 0.7% | 0.4% | 0.1% |
| LCS Length | 6.5% | 7.2% | 4.6% | 2.7% | 7.8% | 3.5% | 4.1% |
| Matrix Chain Order | 2.8% | 5.4% | 5.1% | 2.3% | 0.9% | 1.1% | 1.2% |
| Minimum | 5.8% | 2.6% | 10.4% | 3.4% | 3.4% | 27.3% | 2.0% |
| MST-Kruskal | 4.9% | 2.9% | 4.2% | 5.9% | 6.1% | 4.3% | 11.8% |
| MST-Prim | 11.44% | 9.2% | 10.7% | 5.1% | 4.6% | 7.2% | 7.9% |
| Naïve String Match | 0.3% | 0.7% | 0.5% | 1.7% | 0.8% | 0.7% | 32.30% |
| Optimal BST | 6.0% | 5.7% | 5.1% | 3.5% | 2.0% | 2.0% | 2.6% |
| Quickselect | 2.4% | 0.5% | 0.4% | 1.6% | 0.9% | 1.0% | 17.3% |
| Quicksort | 4.3% | 1.3% | 0.8% | 3.9% | 1.9% | 0.2% | 13.2% |
| Segments Intersect | 0.7% | 0.5% | 0.7% | 2.5% | 0.8% | 0.8% | 2.6% |
| SCC | 5.9% | 8.8% | 9.9% | 5.2% | 6.0% | 6.9% | 15.2% |
| Task Scheduling | 0.5% | 0.4% | 0.8% | 0.9% | 0.5% | 0.5% | 1.8% |
| Topological Sort | 10.3% | 12.7% | 17.8% | 5.8% | 6.5% | 10.1% | 17.5% |

Table 18: Average test scores of all tuned models on all algorithms.

| Algorithm | Deep Sets | GAT-v1 | GAT-v2 | MPNN | PGN-u | PGN-m | PGN-c | RT (Ours) |
|---|---|---|---|---|---|---|---|---|
| Activity Selector | 72.22% | 68.89% | 67.45% | **95.45%** | 67.05% | 69.83% | 69.83% | 87.72% |
| Articulation Points | 38.50% | 31.46% | 31.96% | 46.21% | 46.87% | **49.73%** | 49.73% | 34.15% |
| Bellman-Ford | 51.00% | 93.10% | 93.75% | 95.42% | **95.83%** | 95.43% | 95.83% | 94.24% |
| BFS | 98.14% | 99.76% | 99.49% | **99.78%** | 99.71% | 99.27% | 99.71% | 99.14% |
| Binary Search | 55.77% | 17.21% | 31.11% | 38.00% | 61.71% | **87.03%** | 87.03% | 81.48% |
| Bridges | 36.20% | 22.90% | 24.07% | **61.29%** | 44.72% | 57.15% | 57.15% | 37.88% |
| Bubble Sort | **65.03%** | 9.69% | 8.46% | 13.14% | 7.54% | 2.05% | 7.54% | 38.22% |
| DAG Shortest Paths | 77.98% | 86.44% | 89.35% | 97.33% | 96.23% | **98.16%** | 98.16% | 96.61% |
| DFS | 7.62% | 11.71% | 12.08% | 13.85% | 10.78% | 8.88% | 10.78% | **39.23%** |
| Dijkstra | 44.51% | 61.42% | 68.10% | **92.18%** | 78.45% | 90.02% | 90.02% | 91.20% |
| Find Max. Subarray | 12.29% | 15.19% | 14.80% | 16.14% | 16.71% | 51.30% | 51.30% | **66.52%** |
| Floyd-Warshall | 6.24% | 30.53% | 37.58% | 28.81% | 27.60% | **39.24%** | 39.24% | 31.59% |
| Graham Scan | 64.48% | 65.05% | 67.07% | **95.31%** | 67.89% | 64.66% | 67.89% | 74.15% |
| Heapsort | **72.39%** | 9.90% | 7.65% | 27.08% | 17.88% | 1.60% | 17.88% | 32.96% |
| Insertion Sort | 72.88% | 61.00% | 50.19% | 50.30% | 82.42% | 3.64% | 82.42% | **89.43%** |
| Jarvis' March | 46.01% | 55.40% | 50.23% | 59.31% | 49.54% | 49.57% | 49.57% | **94.57%** |
| KMP Matcher | **3.71%** | 1.03% | 1.32% | 2.35% | 1.75% | 0.44% | 1.75% | 0.03% |
| LCS Length | 55.85% | 48.14% | 48.30% | 54.42% | 53.85% | 56.00% | 56.00% | **83.32%** |
| Matrix Chain Order | 81.65% | 81.59% | 65.58% | 85.53% | 86.65% | 86.47% | 86.65% | **91.89%** |
| Minimum | 91.69% | 86.87% | 80.60% | 90.73% | 89.22% | 70.69% | 89.22% | **95.28%** |
| MST-Kruskal | 69.07% | 67.26% | 68.60% | **70.84%** | 63.47% | 69.35% | 69.35% | 64.91% |
| MST-Prim | 35.70% | 61.83% | 68.72% | **86.43%** | 81.59% | 83.61% | 83.61% | 85.77% |
| Naïve String Match | 2.12% | 1.70% | 1.81% | 1.83% | 1.46% | 1.90% | 1.90% | **65.01%** |
| Optimal BST | 67.37% | 61.91% | 63.78% | 66.48% | 65.19% | 70.74% | 70.74% | **74.40%** |
| Quickselect | 5.51% | 2.03% | 2.40% | 3.07% | 2.00% | 4.92% | 4.92% | **19.18%** |
| Quicksort | **65.26%** | 4.41% | 5.00% | 17.95% | 7.90% | 2.63% | 7.90% | 39.42% |
| Segments Intersect | 85.94% | 86.36% | 86.10% | **94.46%** | 85.89% | 85.66% | 85.89% | 84.94% |
| SCC | 21.13% | 23.66% | 28.90% | 12.58% | 12.97% | 17.20% | 17.20% | **28.59%** |
| Task Scheduling | 83.45% | 82.61% | 82.61% | 83.36% | 83.54% | **83.62%** | 83.62% | 82.93% |
| Topological Sort | 19.02% | 43.47% | 43.23% | 54.93% | 51.71% | 64.28% | 64.28% | **80.62%** |
| Average | 50.29% | 46.42% | 46.70% | 55.15% | 51.94% | 52.17% | 56.57% | **66.18%** |