# DUAL ALGORITHMIC REASONING

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

## ABSTRACT

Neural Algorithmic Reasoning is an emerging area of machine learning which seeks to infuse algorithmic computation in neural networks, typically by training neural models to approximate steps of classical algorithms. In this context, much of the current work has focused on learning reachability and shortest path graph algorithms, showing that joint learning on similar algorithms is beneficial for generalisation. However, when targeting more complex algorithms, such "similar" algorithms become more difficult to find. Here, we propose to learn algorithms by exploiting *duality* of the underlying algorithmic problem. Many algorithms solve optimisation problems. We demonstrate that simultaneously learning the dual definition of these optimisation problems in algorithmic learning allows for better learning and qualitatively better solutions. Specifically, we exploit the max-flow min-cut theorem to simultaneously learn these two algorithms over synthetically generated graphs, demonstrating the effectiveness of the proposed approach. We then validate the real-world utility of our dual algorithmic reasoner by deploying it on a challenging brain vessel classification task, which likely depends on the vessels' flow properties. We demonstrate a clear performance gain when using our model within such a context, and empirically show that learning the max-flow and min-cut algorithms together is critical for achieving such a result.

## 1 INTRODUCTION

Learning to perform algorithmic-like computation is a core problem in machine learning that has been widely studied from different perspectives, such as learning to reason (Khardon & Roth, 1997) and automated theorem proving (Rocktäschel & Riedel, 2017). As a matter of fact, enabling reasoning capabilities of neural networks might drastically increase *generalisation*, i.e. the ability of neural networks to generalise beyond the support of the training data, which is usually a difficult challenge with current neural models (Neyshabur et al., 2017). Neural Algorithmic Reasoning (Velickovic & Blundell, 2021) is a recent response to this long-standing question, attempting to train neural networks to exhibit some degrees of *algorithmic reasoning* by learning to execute classical algorithms. Arguably, algorithms are designed to be general, being able to be executed and return "optimal" answers for any inputs that meet a set of strict pre-conditions. On the other hand, neural networks are more flexible, i.e. can adapt to virtually any input. Hence, the fundamental question is whether neural models may inherit some of the positive algorithmic properties and use them to solve potentially challenging real-world problems.

Historically, learning algorithms has been tackled as a simple supervised learning problem (Graves et al., 2014; Vinyals et al., 2015; Kool et al., 2019), i.e. by learning an input-output mapping. However, more recent works build upon the notion of *algorithmic alignment* (Xu et al., 2020) stating that there must be an "alignment" between the learning model structure and the target algorithm in order to ease optimisation. Much focus has been placed on Graph Neural Networks (GNNs) (Bacciu et al., 2020) learning graph algorithms, i.e Bellman-Ford (Bellman, 1958). Velickovic et al. (2020b) show that it is indeed possible to train GNNs to execute classical graph algorithms. Furthermore, they show that optimisation must occur on all the intermediate steps of a graph algorithm, letting the network learn to replicate step-wise transformations of the input rather than learning a map from graphs to desired outputs. Since then, algorithmic reasoning has been applied with success in reinforcement learning (Deac et al., 2021), physics simulation (Velickovic et al., 2021) and bipartite matching (Georgiev & Lió, 2020).

Furthermore, Xhonneux et al. (2021) verify the importance of training on multiple "similar" algorithms at once (*multi-task learning*). The rationale is that many classical algorithms share subroutines, i.e. Bellman-Ford and Breadth-First Search (BFS), which help the network learn more effectively and be able to transfer knowledge among the target algorithms. However, as the complexity of the target algorithm increases, it might become more and more difficult to find this set of "similar" algorithms. Furthermore, learning some algorithms might require learning of very specific properties of the input data, for which similarity between algorithms may not help. For instance, learning the *Ford-Fulkerson algorithm* (Ford & Fulkerson, 1956) for *maximum flow* entails learning to identify the set of *critical* edges of the flow network, i.e. edges for which a decrease in the *edge capacity* would decrease the maximum flow.

In this work, we explore different supervision signals and learning setups to be used both in isolation or together with established neural algorithmic reasoning techniques. We find a potentially good candidate in the *duality* information of the target algorithmic problem. The concept of duality fundamentally enables a target algorithmic problem, e.g. linear program, to be viewed from two perspectives, that of a *primal* and a *dual* problem. These two problems are usually complementary, i.e. the solution of one might lead to the solution of the other. Hence, we propose to incorporate duality information directly in the learning model both as an additional supervision signal and input feature (by letting the network reuse its dual prediction in subsequent steps of the algorithm), an approach we refer to as Dual Algorithmic Reasoning (DAR). We show that by training an *algorithmic reasoner* on both learning of an algorithm and optimisation of the dual problem we can relax the assumption of having multiple similar algorithms to train on while retaining all the benefits of multi-task learning. We demonstrate clear performance gain in both synthetically generated algorithmic tasks and real-world predictive graph learning problems.

## 2 Problem statement

We study the problem of neural algorithmic reasoning on graphs. Specifically, we target learning of graph algorithms $A : \mathbb{G} \rightarrow \mathbb{Y}$ that take in graph-structured inputs $G = (V, E) \in \mathbb{G}$, with $V$ being the set of nodes and $E$ the set of directed (or undirected) edges with node features $\boldsymbol{x}_i$ and edge features $\boldsymbol{e}_{ij}$, and compute a desired output $\boldsymbol{y} \in \mathbb{Y}$. Usually, the output space $\mathbb{Y}$ of an algorithm depends on the scope of the algorithm $A$. In the most general cases, it can either be $\mathbb{R}^{|V|}$ (node-level output), $\mathbb{R}^{|V|\times|V|}$ (edge-level output) or $\mathbb{R}$ (graph-level output). We mainly consider the class of algorithms outputting node-level and edge-level outputs, which includes many of the most well-known graph problems, e.g. reachability, shortest path and maximum flow. From a neural algorithmic reasoning perspective, we are particularly interested in learning a sequence of transformations (*steps* of the algorithm). Hence, we consider a sequence of graphs $\{G^{(0)}, \ldots, G^{(T-1)}\}$ where each element $G^{(t)} = (\boldsymbol{x}_i^{(t)}, \boldsymbol{e}_{ij}^{(t)})$ represents the intermediate state of the target algorithm we aim to learn. At each step $t$ we have access to intermediate node and edge features, i.e. $\boldsymbol{x}_i^{(t)}, \boldsymbol{e}_{ij}^{(t)}$, called *hints* as well as intermediate targets $\boldsymbol{y}^{(t)}$. Such hints and intermediate targets are incorporated in training as additional features/learning targets, effectively learning the whole sequence of steps (*algorithm trajectory*).

In particular, we focus on learning *maximum flow* via the neural execution of the *Ford-Fulkerson algorithm*. Differently from Georgiev & Lió (2020), who learn Ford-Fulkerson to find the independent set of edges in bipartite graphs, we aim to learn Ford-Fulkerson for general capacitated graphs. We report the pseudo-code of Ford-Fulkerson in the appendix. Ford-Fulkerson poses two key challenges: (i) it comprises two sub-routines, i.e. finding augmenting paths from $s$ to $t$, and updating the flow assignment $\boldsymbol{F}^{(t)} \in \mathbb{R}^{|V|\times|V|}$ at each step $t$; (ii) $\boldsymbol{F}$ must obey a set of strict constraints, namely the *edge-capacity constraint* and *conservation of flows*. The former states that a scalar value $c_{ij}$ (capacity) is assigned to every $(i, j) \in E$ and $\boldsymbol{F}$ must satisfy:

$$\forall (i, j) \in E \; . \; \boldsymbol{F}_{ij} \leq c_{ij}, \tag{1}$$

i.e. flow assignment to an edge must not exceed its capacity. The latter states that the assignment needs to satisfy:

$$\forall i \in V \setminus \{s, t\} : \sum_{(i,j)\in E} \boldsymbol{F}_{ij} + \sum_{(j,i)\in E} \boldsymbol{F}_{ji} = 0 \quad \wedge \sum_{(s,j)\in E} \boldsymbol{F}_{sj} = - \sum_{(j,t)\in E} \boldsymbol{F}_{jt} \tag{2}$$

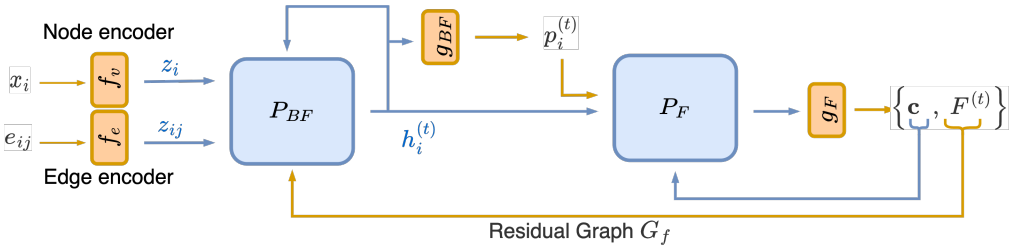



Figure 1: High-level architecture of the Dual Algorithmic Reasoner (DAR) for the Ford-Fulkerson algorithm. Refer to the text for a comprehensive explanation.

i.e. the flow sent out from the source is not lost nor created by intermediate nodes. This also leads to $\boldsymbol{F} = -\boldsymbol{F}^T$, i.e. antisymmetry. An optimal solution $\boldsymbol{F}^*$ to the max flow problem is the one maximising the total flow in the network, i.e. $\sum_{(s,j)\in E} \boldsymbol{F}_{sj}$. We show how we address both challenge (i) and (ii) directly in the model architecture, through the concept of *algorithmic alignment* and by carefully adjusting and rescaling the model predictions.

# 3 Methodology

## 3.1 Leveraging duality in algorithmic reasoning

We leverage the concept of *duality* when learning to neurally execute classical graph algorithms. In particular, most of the problems solved by classical algorithms (including maximum flow) can be expressed in the form of constrained optimisation problems such as *linear programming* or *integer linear programming*. In mathematical optimisation, the duality principle ensures that any optimisation problem may be viewed from two perspectives: the "direct" interpretation (called the *primal* problem) and the *dual* problem, which is usually derived from the *Lagrangian* of the primal problem. The duality principle ensures that the solutions of the two problems are either linked by an upper-bound/lower-bound relation (*weak duality*) or equal (*strong duality*) (Boyd et al., 2004). Hence, the two problems are interconnected.

In the context of neural algorithmic reasoning, we identify several reasons why primal-dual information might be useful to consider. First, by incorporating *primal-dual* objectives, we let the network reason on the task from two different and complementary perspectives. This can substantially simplify learning of algorithms which require to identify and reason on properties which are not explicitly encoded in input data. For instance, to effectively solve max-flow problems, the network needs the ability to identify and reason on *critical edges*. By the *max-flow min-cut theorem* (Ford Jr & Fulkerson, 2015), this set of edges correspond to the *minimum cut*, i.e. dual problem, that separates the source node $s$ from the sink $t$.

Second, being able to output a better intermediate solution at each step of the algorithm means that there is less chance for error propagation throughout the trajectory of the neurally executed algorithm. This is especially true for more complex algorithms, such as Ford-Fulkerson, consisting of multiple interlocked sub-routines. There, an imprecise approximation of one sub-routine can negatively cascade on the results of the following ones. Finally, learning jointly on the primal-dual can be seen as an instance of *multi-task learning*, but relaxing the assumption of having multiple algorithms to train on.

In the following, we study dual algorithmic reasoning on the max-flow primal complemented with min-cut dual information. Note that graph neural networks have been formally proven to be able to learn minimum cut (Fereydounian et al., 2022).

## 3.2 Architecture

We rely on the neural algorithmic reasoning blueprint (Velickovic & Blundell, 2021), building on the encode-process-decode framework (Hamrick et al., 2018). The abstract architecture of the Dual

Algorithmic Reasoner (DAR) is depicted in Figure 1 for the Ford-Fulkerson algorithm. Since the latter is composed of two sub-routines, we introduce two processors to align neural execution with the dynamics of the algorithm. The first processor $P_{BF}$ learns to retrieve augmenting paths, while $P_F$ learns to perform flow-update operations $\boldsymbol{F}^{(t)}$.

Intuitively, the encode-process-decode architecture allows *decoupling* learning of the algorithmic steps from the use of specific input features. By this means, the algorithm can be neurally executed on a latent-space which is a learnt representation of the input features required by the original algorithm. We will show how we can exploit this property to perform steps of the Ford-Fulkerson algorithm even with missing input features.

More in detail, the DAR architecture comprises two encoder networks, $f_v$ and $f_e$, which are applied respectively to node features $\boldsymbol{x}_i^{(t)}$ and edge features $\boldsymbol{e}_{ij}^{(t)}$ to produce encoded node-level and edge-level features:

$$\boldsymbol{Z}_V^{(t)} = \{\boldsymbol{z}_i^{(t)} = f_v(\boldsymbol{x}_i^{(t-1)}) \mid \forall i \in V\}$$
$$\boldsymbol{Z}_E^{(t)} = \{\boldsymbol{z}_{ij}^{(t)} = f_e(\boldsymbol{e}_{ij}^{(t-1)}) \mid \forall (i,j) \in E\}.$$

These encoded representations are used as inputs for the processor network $P_{BF}$ which computes the latent node representations $\boldsymbol{H}^{(t)}$ as:

$$\boldsymbol{H}^{(t)} = P(\boldsymbol{Z}_V^{(t)}, \boldsymbol{Z}_E^{(t)}, \boldsymbol{H}^{(t-1)})$$

with $\boldsymbol{H}^{(0)} = \{\boldsymbol{0} \mid \forall i \in V\}$. In our DAR instance, this processor performs Bellman-Ford steps to retrieve the shortest augmenting path from $s$ to $t$, following (Georgiev & Lió, 2020). $\boldsymbol{H}^{(t)}$ is then passed to a decoder network $g$ producing the augmenting path $\boldsymbol{p}^{(t)}$:

$$p_i^{(t)} = g_{BF}(\boldsymbol{z}_i^{(t)}, \boldsymbol{h}_i^{(t)}).$$

The augmenting path is represented as a vector of predecessors for all nodes in the graph, i.e. each entry $p_i^{(t)}$ is a pointer to another node $j$ in the graph. This way, we are able to reconstruct a path from any node (included $t$) back to the source node $s$. The augmenting path $\boldsymbol{p}^{(t)}$ is then passed to $P_F$ as an input feature. The target quantities of the algorithm, i.e. flow assignment $\boldsymbol{F}$ and minimum cut $\boldsymbol{c}$, are finally predicted as:

$$\{\boldsymbol{F}^{(t)}, \boldsymbol{c}\} = g_F\big(P_F(\boldsymbol{Z}_V^{(t)} \cup \{\boldsymbol{p}^{(t)}\}, \boldsymbol{Z}_E^{(t)}, \boldsymbol{H}^{(t)})\big).$$

We choose to represent the minimum *s-t* cut $\boldsymbol{c}$ as node-level features w.l.o.g, where $c_i = 0$ indicates that $i$ is in the cluster of nodes of $s$, and $c_i = 1$ otherwise. Note that the minimum *s-t* cut includes all edges $(i,j)$ for which $c_i = 0$ and $c_j = 1$. Furthermore, $\boldsymbol{F}^{(t)}$ is reused as an input feature in the next step of the algorithm ($\boldsymbol{F}^{(0)} = \boldsymbol{0}$).

We pay additional attention to the prediction of the flow assignment matrix $\boldsymbol{F}^{(t)}$, in order to be compliant with the maximum flow problem constraints described in subsection 3.1. In particular, we transform $\boldsymbol{F}$ to ensure compliance with anti-symmetry, i.e. $\boldsymbol{F}' = \boldsymbol{F} - \boldsymbol{F}^T$. To satisfy edge-capacity constraints we further rescale the matrix according to the hyperbolic tangent and the actual value of the capacity $c_{ij}$ for each $(i,j) \in E$, as such:

$$\boldsymbol{F} = \tanh(\boldsymbol{F}) \odot C,$$

where $C_{i,j} = c_{ij}$ for all edges in the graph. We note that this only satisfies the box constraint on the edge capacities, however the conservation of flows might still be violated, i.e. nodes in the path from the source to the target may either retain some amount of in-flow (sending out less than what is received) or vice versa. To address this last constraint, we simulate the entire algorithm until termination and apply a corrective procedure in order to correct all the flow conservation error. We refer to the appendix for further details and pseudo-code of the corrective procedure.

Finally, we describe how all the neural modules in the architecture are implemented. To keep most of the model expressiveness in the processors, we implement $f_v$ and $f_e$ through simple linear transformations, following Velickovic et al. (2020b). Note that different choices may be possible, as highlighted in Xhonneux et al. (2021), which used non-linear encoders. However, for our task linear

Table 1: Mean Absolute Error (MAE) and accuracy of predicting the network flow $\boldsymbol{F}$ and min-cut $\boldsymbol{c}$ (if applicable) on *2-community* and *bipartite* graphs. *(primal)* corresponds to training on max-flow only. *(dual)* corresponds to training with both primal-dual heads. *(pipeline)* corresponds to learning min-cut first.

| | 2-Community | | Bipartite | |
|---|---|---|---|---|
| **Model** | $\boldsymbol{F}$ | $\boldsymbol{c}$ | $\boldsymbol{F}$ | $\boldsymbol{c}$ |
| PGN-max *(primal)* | $0.266 \pm 0.001$ | - | $0.559 \pm 0.23$ | - |
| PGN-mean *(primal)* | $0.274 \pm 0.001$ | - | $1.089 \pm 0.47$ | - |
| MPNN-max *(primal)* | $0.263 \pm 0.008$ | - | $0.752 \pm 0.47$ | - |
| MPNN-mean *(primal)* | $0.278 \pm 0.008$ | - | $0.746 \pm 0.47$ | - |
| PGN-max *(dual)* | $\mathbf{0.234 \pm 0.002}$ | $100\% \pm 0.0\%$ | $0.485 \pm 0.22$ | $100\% \pm 0.0\%$ |
| PGN-mean *(dual)* | $0.240 \pm 0.004$ | $100\% \pm 0.0\%$ | $1.098 \pm 0.30$ | $99.2\% \pm 0.7\%$ |
| MPNN-max *(dual)* | $0.236 \pm 0.002$ | $100\% \pm 0.0\%$ | $0.716 \pm 0.32$ | $100\% \pm 0.0\%$ |
| MPNN-mean *(dual)* | $0.258 \pm 0.008$ | $100\% \pm 0.0\%$ | $0.815 \pm 0.09$ | $100\% \pm 0.0\%$ |
| PGN-max *(pipeline)* | $0.256 \pm 0.001$ | $61.6\% \pm 0.1\%$ | $\mathbf{0.450 \pm 0.18}$ | $95.3\% \pm 0.1\%$ |
| PGN-mean *(pipeline)* | $0.244 \pm 0.001$ | $100\% \pm 0.0\%$ | $0.984 \pm 0.44$ | $99.4\% \pm 0.8\%$ |
| MPNN-max *(pipeline)* | $0.261 \pm 0.002$ | $61.7\% \pm 0.3\%$ | $\mathbf{0.465 \pm 0.23}$ | $90.0\% \pm 11\%$ |
| MPNN-mean *(pipeline)* | $0.255 \pm 0.002$ | $100\% \pm 0.0\%$ | $0.637 \pm 0.35$ | $100\% \pm 0.0\%$ |
| Random | $0.74 \pm 0.002$ | $50\% \pm 0.0\%$ | $1.00 \pm 0.00$ | $50\% \pm 0.0\%$ |

layers showed to be sufficient. The two processors $P_{BF}$ and $P_F$ are implemented as Graph Neural Networks (GNNs) with Message-Passing Neural Network (MPNN) convolution (Gilmer et al., 2017):

$$\boldsymbol{h}_i^{(t)} = \psi_\theta \left( \boldsymbol{h}_i^{(t)}, \bigoplus_{(j,i) \in E} \phi_\theta(\boldsymbol{h}_i^{(t)}, \boldsymbol{h}_j^{(t)}, \boldsymbol{e}_{ij}^{(t)}) \right), \tag{3}$$

where $\psi_\theta$ and $\phi_\theta$ are neural networks with ReLU activations and $\bigoplus$ is a permutation-invariant function, i.e. summation, mean or max.

## 4 Experiments

To assess the benefits of the dual algorithmic reasoning approach, we test the learning model in two specific scenarios. First, we train and test the dual algorithmic reasoning pipeline on synthetic-generated graphs, to evaluate the benefits in the key task of algorithmic learning (section 4.1). Then, to evaluate the generality of the model we test on a classical graph learning predictive task. Specifically, we compare the model with several graph learning baselines on a biologically relevant *vessel* classification task (Paetzold et al., 2021), comprehending large-scale *vessel* graphs (section 4.2). We stress that all the training is done on synthetic graphs and there is no further re-training for real-world ones.

### 4.1 Synthetic graphs

**Data generation** We consider two different families of graphs: (i) *2-community* graphs, in which communities are sampled from the *Erdős–Rényi* distributions with probability 0.75 and their nodes are interconnected with probability 0.05; (ii) *bipartite* graphs. We follow the standard CLRS-30 benchmark (Veličković et al., 2022) setup for algorithmic reasoning. Specifically, we sample 1000 training graphs with 16 nodes each. The validation set is used to assess in-distribution performance, thus comprehending 128 graphs with still 16 nodes. To assess out-of-distribution capabilities we provide 128 test samples comprehending graphs with 64 nodes (4x larger). Furthermore, we generate data of all intermediate steps of the Ford-Fulkerson algorithm to be used as *hints* and additional training targets, in order to train the network on all intermediate data manipulations. Furthermore, to assess generalisation *out-of-family*, bipartite graphs are solely used for testing purposes. Finally, features of the algorithm are once again generated following the CLRS-30 standard and it comprises:

(i) *inputs*: source node $s$, sink node $t$, edge-capacity matrix $C \in \mathbb{N}^{|V|\times|V|}$ and additional weights $W \in [0,1]^{|V|\times|V|}$ for the Bellman-Ford processor; (ii) *hints* (algorithm steps): augmenting paths $\boldsymbol{p}^{(t)}$ and intermediate flow assignments $\boldsymbol{F}^{(t)}$; (iii) *outputs* (learning targets): final flow matrix $\boldsymbol{F}$ and minimum cut $\boldsymbol{c}$. Lastly, capacities are sampled as integers from $U(1, 10)$ and then rescaled via a min-max normalisation for *2-community* graphs, while they are sampled as either 0 or 1 for *bipartite* graphs.

**Ablation & neural architectures** To check if the dual information is really needed, we perform an ablation study by training the same DAR architecture without the additional *min-cut head* (consequently the dual information does not flow back in $P_F$ in Figure 1). As an additional check, we also train a neural architecture where the minimum cut is learnt prior the Ford-Fulkerson algorithm. Specifically, we introduce a third processor that is trained solely on minimum cut, whose output is then used as an additional feature for the architecture presented in Figure 1. Furthermore, we compare two different types of processors: (i) a fully-connected Message-Passing Neural Network (MPNN) (Gilmer et al., 2017), which implements equation 3 and exchanges messages between all pairs of nodes; (ii) Pointer-Graph Network (PGN) (Velickovic et al., 2020a), which instead exchanges messages only between a node and its neighbourhood defined by the *inputs* and *hints* of the algorithm. For all processors, we try different aggregate operations of equation 3, namely $\bigoplus = \{\max, \text{mean}, \text{sum}\}$. We train all models for 20,000 epochs with SGD optimiser and average the results across 5 runs. We also use *teacher forcing* with a decaying factor of 0.999. This has the effect of providing the network with ground-truth *hints* for early stage of the training phase, while letting the network predictions flow in for the majority of training. To choose optimal hyperparameters, e.g. learning rate, hidden dimension, we employ a bi-level random search scheme, where the first level samples values of hyperparameters in a large range of values, while the second one "refines" the search based on the first level results. We choose the best hyperparameters based on the validation error on $\boldsymbol{F}$. We additionally show validation aggregate loss curves in Figure 2. For further details on the model selection we refer to the appendix.

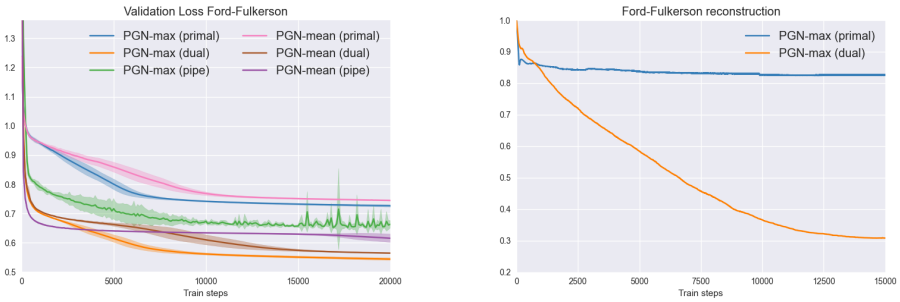
**Results analysis** We report results on Ford-Fulkerson simulation in Table 1. Specifically, we use the Mean Absolute Error (MAE) as a metric for assessing the predictions of the final flow assignment $\boldsymbol{F}$. Where applicable, we report accuracy on the minimum cut as well, i.e. for *dual* and *pipeline* models. First, we notice how models incorporating the prediction of the dual problem consistently outperform the *primal* baselines on both *2-community* and *bipartite* graphs. This suggests that the duality min-cut information helps the network achieving a lower prediction error. This finding is also strengthen by the observation that whenever the min-cut prediction is imprecise, e.g. {PGN, MPNN}-max pipeline for *2-community*, the prediction of $\boldsymbol{F}$ becomes consequently worse. From our experiments, the *dual* PGN architecture with max aggregator emerges as the best-performing model, at least for what concerns *2-community* graphs, being able to perfectly predict also the minimum cuts of all the graphs in the test set. Contrastingly, learning min-cut first is less stable (while still outperforming the primal baseline) confirming prior work findings on the effectiveness of multi-task learning.

The performance gap also increases when testing *out-of-family* on bipartite graphs, where *dual* and *pipeline* with max aggregator are both very competitive. We note that for bipartite graphs we record higher mean and standard deviations. While this behaviour is emphasised by the fact that capacities are sampled as either 0 or 1, i.e. leaving more chances for prediction errors, this testifies that generalise for arbitrary graph distributions is still a challenging task.

**Qualitative analysis** To further evaluate the performance of DAR, we perform a qualitative study, whose results are presented in Table 2. For *2-community* graphs we assess how close the predicted flow matrix $\boldsymbol{F}$ is to the optimal max-flow solution without considering errors for intermediate nodes. This gives a measure of how well the network can predict the maximal flow value in the graphs and use it in the predicted solution. To achieve that, we ignore intermediate errors and only measures flow signal exiting the source node $s$ and entering the sink node $t$, i.e. $\sum_{(s,j)\in E} \boldsymbol{F}_{sj}$ and $\sum_{(j,t)\in E} \boldsymbol{F}_{jt}$. Thus, we take the maximum (absolute value) between the two and compare this value to the ground truth maximum flow value $\boldsymbol{F}^*$. From Table 2 we observe that all the dual architectures exhibit a solution which reflects the true maximum flow quantity in the input graphs, i.e. $\approx 0.30$ of MAE from the optimum on average. This analysis further solidifies our claim that the network can

Table 2: Qualitative analysis on the prediction of $\boldsymbol{F}$. Mean Absolute Error (MAE) is used as the regression error from the ground truth maximum flow value. For simplicity, we only reports result of the best-performing models (PGNs).

| **Model** | $\lvert\boldsymbol{F} - \boldsymbol{F}^*\rvert$ |
|---|---|
| PGN-max *(primal)* | $7.86 \pm 0.47$ |
| PGN-mean *(primal)* | $8.68 \pm 0.21$ |
| PGN-max *(dual)* | $\mathbf{0.34} \pm \mathbf{0.04}$ |
| PGN-mean *(dual)* | $0.41 \pm 0.01$ |
| PGN-max *(pipeline)* | $7.58 \pm 0.10$ |
| PGN-mean *(pipeline)* | $0.38 \pm 0.08$ |




(a) Ford-Fulkerson validation loss

(b) Ford-Fulkerson reconstruction loss

Figure 2: **(a)** validation Ford-Fulkerson loss for different models. **(b)** normalised loss curve of reconstructing Ford-Fulkerson with new encoders for $l_{ij}, d_{ij}, \rho_{ij}$, with both the primal and dual PGN-max.

positively transfer knowledge across the primal and dual problems in order to output more precise and qualitatively better solutions. This claim is also supported by the fact that both primal architectures and dual architectures for which min-cut results are worse, i.e. pipeline versions of {PGN, MPNN}-max, miss the optimal solution by a large margin.

### 4.2 REAL-WORLD GRAPHS

**Benchmark description** As a general assessment of the generality of the dual algorithmic reasoning pipeline, we take on a more classical edge classification task, for which prior knowledge of the concept of *max-flow* might be helpful. We test both the primal and the dual architectures on the Brain Vessel Graphs (BVG) benchmark (Paetzold et al., 2021). This benchmark contains 9 large-scale graphs, where edges represent vessels and nodes represent bifurcation of the vessel branches in a brain network. The task is to classify each edge in three categories: *capillaries*, *veins* and *arteries* based on the following features: vessel length $l_{ij}$; shortest distance between bifurcation points $d_{ij}$; and curvature $\rho_{ij}$. Note that the three classes can be distinguished by the radius of the vessel, or equivalently, by the amount of blood flow that can traverse the vessel. Hence, being able to simulate the blood flow in the entire brain network might be important information. Furthermore, the classification task is highly imbalanced, i.e. 95% of samples are capillaries, 4% veins and only 1% arteries.

We test the model on three graphs of the benchmark, namely CD1-E-1, CD1-E-2 and CD1-E-3. CD1-E-1 is the largest graph of all nine with 5,791,309 edges, whereas CD1-E-2 and CD1-E-3 have respectively 2,150,326 and 3,130,650 edges. They also provide 5 synthetic graphs with approximately 3k edges each for testing purposes.

**Algorithm reconstruction** The main difference with our synthetic tasks is that here we need to estimate the vessel diameter/radius which is a quantity that can be related with the vessel *capacity*, i.e. how much blood (flow) can traverse the edge. Therefore, here the *capacity* is a learning target rather than one of the feature to fed our algorithmic reasoner with. Here, we exploit the generality of the encode-process-decode architecture and learn to *reconstruct* the Ford-Fulkerson neural execution. Specifically, we reuse PGN-max pre-trained networks on the synthetic Ford-Fulkerson task (section 4.1).

As the *capacity* is no longer an input feature, we drop the *capacity* encoder from $f_v$ and introduce three new encoder layers in $f_v$, one for each feature of the vessel graph benchmark, i.e. $l_{ij}, d_{ij}, \rho_{ij}$. Thus, we freeze all the parameters in the pre-trained models apart from the introduced encoder layers. Hence, we only learn weights for $l_{ij}, d_{ij}, \rho_{ij}$ in order to perform Ford-Fulkerson steps, even in case of missing input features, i.e. capacity. In other words, the model learns to use $l_{ij}, d_{ij}, \rho_{ij}$ to estimate the edge flows in the network, thus potentially giving information on the edge capacities, i.e. our primary objective. We perform these learning steps of algorithm reconstruction on a synthetic vessel graph provided by the BVG benchmark, with exactly 3159 nodes and 3234 edges. Source and sink nodes $s, t$ are chosen as two random nodes whose shortest distance is equal to the diameter of the graph. We train algorithm reconstruction for 15000 epochs, with Adam optimiser (Kingma & Ba, 2015) and learning rate 1e-5. We show a comparison of the two loss curves, i.e. primal and dual models, of algorithm reconstruction in Figure 2.

Finally, we simulate one single step of Ford-Fulkerson and extract the learnt node embeddings which are summed together to get edge embeddings. These edge embeddings will be used as additional input features, measuring how much information acquired from the synthetic tasks we can transfer to real-world data.

**Neural architectures** We consider different graph neural networks used in the BVG benchmark paper as our baselines, namely Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017) and ClusterGCN (Chiang et al., 2019) with GraphSAGE convolution (ClusterSAGE). In general, the architecture consists of several graph convolutional layers with ReLU activations followed by a linear module. Furthermore, we utilise the extracted embeddings from PGN-max *(primal)* and PGN-max *(dual)* to train a simple linear classifier (LC) as an assessment of how much information we add over the simple features $l_{ij}, d_{ij}, \rho_{ij}$. We also use those representations in combination with GraphSAGE and ClusterSAGE. Specifically, in this variant our embeddings are concatenated together with the GNN's learnt embeddings prior the final linear layer. As an additional sanity check, we also train Node2Vec (Grover & Leskovec, 2016) on each of the three datasets and concatenate its learnt embeddings the same way. We train all the models with optimal hyperparameters taken from the BVG paper, which we report in the appendix for completeness. Finally, we average the results across 3 trials of 1500 epochs each.

**Results analysis** Results on the BVG benchmark are reported in Table 3. As the learning problem is highly imbalanced, we use the default BVG metric for assessing models performance, namely the balanced accuracy score (average of recall for each class). We also collect different metrics, namely ROC AUC and F1 score, whose results are reported in the supplementary material.

First, by considering the linear classifiers (LC) performance we see that the embeddings extracted from the algorithmic reasoner (both primal and dual) are informative, i.e. +16.6% on average balanced accuracy across the three datasets compared to the base input features, with dual embeddings that still show superior performance compared to primal embeddings. This might be caused by a better algorithm reconstruction performed by the dual architecture (see Figure 2), resulting in more informative representations. Performance of linear classifier also gives a measure of how well the algorithmic reasoner is able to positively transfer knowledge acquired on synthetic algorithmic tasks to unseen real-world predictive graph learning ones.

Other significant results emerge when the learnt embeddings are used in combination with GNN architectures, with significant performance improvements over single GNNs. In particular, the ClusterSAGE model with *dual* embeddings achieves the best performance on all three datasets with consistent performance gap for CD1-E-3 and CD1-E-2. Interestingly, dual embeddings consistently outperform Node2Vec embeddings. This is remarkable, considering that Node2Vec is trained directly on the real data, whereas DAR only performs inference on them. A reason to this performance

Table 3: Balanced accuracy on large-scale brain vessel graphs. LC refers to a linear classifier. In addition to the standard architectures, we consider variants where the final linear classification layer takes in additional Node2Vec (Grover & Leskovec, 2016) learnt embeddings and embeddings extracted from PGN-max primal and dual architectures.

| **Model** | CD1-E-3 | CD1-E-2 | CD1-E-1 |
|---|---|---|---|
| LC | $39.41\% \pm 0.36$ | $34.69\% \pm 0.28$ | $45.60\% \pm 0.15$ |
| LC & Node2Vec | $44.18\% \pm 0.03$ | $72.63\% \pm 0.03$ | $46.95\% \pm 0.08$ |
| LC & PGN-max *(primal)* | $50.60\% \pm 0.10$ | $58.73\% \pm 0.05$ | $47.98\% \pm 0.68$ |
| LC & PGN-max *(dual)* | $53.81\% \pm 0.27$ | $67.33\% \pm 0.07$ | $48.15\% \pm 0.71$ |
| GCN | $58.06\% \pm 0.45$ | $74.58\% \pm 1.67$ | $58.96\% \pm 0.24$ |
| SAGE | $63.51\% \pm 0.15$ | $73.92\% \pm 0.63$ | $64.72\% \pm 0.66$ |
| SAGE & Node2Vec | $64.98\% \pm 0.11$ | $84.16\% \pm 1.86$ | $65.88\% \pm 0.06$ |
| SAGE & PGN-max *(primal)* | $64.46\% \pm 0.16$ | $83.76\% \pm 0.40$ | $66.20\% \pm 0.48$ |
| SAGE & PGN-max *(dual)* | $66.67\% \pm 0.35$ | $85.22\% \pm 0.10$ | $66.35\% \pm 0.33$ |
| ClusterSAGE | $68.59\% \pm 0.75$ | $81.77\% \pm 0.49$ | $59.31\% \pm 0.89$ |
| ClusterSAGE & Node2Vec | $68.63\% \pm 0.22$ | $84.79\% \pm 0.20$ | $67.42\% \pm 0.57$ |
| ClusterSAGE & PGN-max *(primal)* | $67.28\% \pm 0.19$ | $82.54\% \pm 1.89$ | $67.71\% \pm 0.12$ |
| ClusterSAGE & PGN-max *(dual)* | $\mathbf{70.16\% \pm 0.19}$ | $\mathbf{85.56\% \pm 0.25}$ | $\mathbf{68.13\% \pm 0.19}$ |

gap might be that Node2Vec essentially widens the local perceptive field of graph neural networks with random walks as an attempt to capture global graph features. On the contrary, DAR utilises a more principled approach based on the simulation of graph flow. This means that the learnt latent space encodes the information necessary to reconstruct the flow assignment and consequently edge capacities, being then more informative for the specific task. Our learning models also exhibit very good generalisation capabilities. In fact, we recall that the networks are only trained on graphs with 16 nodes and extract meaningful representations for graphs with millions of nodes, being able to provide clear performance advantage over the provided baselines. This might also indicate a way worth pursuing of *data-efficient* learning models.

## 5 Conclusion

We have presented **dual algorithmic reasoning** (DAR), a neural algorithmic reasoning approach that leverages duality information when learning classical algorithms. Unlike other approaches, we relax the assumption of having multiple algorithms to be learnt jointly and show that incorporating the dual of the problem targeted by algorithms represents a valuable source of information for learnt algorithmic reasoners. We show that multi-task learning on primal-dual can lead to both quantitatively and qualitatively better solutions than learning the algorithm in isolation. Furthermore, for the first time we demonstrate how a more classical graph learning task can be tackled through exploitation of algorithmic reasoning, via *algorithm reconstruction*. We believe that results showed in this manuscript should strongly motivate further work in this direction, extending the analysis to other pairs of primal-dual problems and algorithms.

## References

Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

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

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, pp. 257–266. ACM, 2019.

Andreea Deac, Petar Velickovic, Ognjen Milinkovic, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolic. Neural algorithmic reasoners are implicit planners. In *NeurIPS*, pp. 15529–15542, 2021.

Mohammad Fereydounian, Hamed Hassani, Javid Dadashkarimi, and Amin Karbasi. The exact class of graph functions generated by graph neural networks. *CoRR*, abs/2202.08833, 2022.

Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.

Dobrik Georgiev and Pietro Lió. Neural bipartite matching. *Graph Representation learning and Beyond Workshop, ICML.*, 2020.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 2017.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pp. 855–864. ACM, 2016.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pp. 1024–1034, 2017.

Jessica B. Hamrick, Kelsey R. Allen, Victor Bapst, Tina Zhu, Kevin R. McKee, Josh Tenenbaum, and Peter W. Battaglia. Relational inductive bias for physical construction in humans and machines. In *CogSci*. cognitivesciencesociety.org, 2018.

Roni Khardon and Dan Roth. Learning to reason. *Journal of the ACM (JACM)*, 44(5):697–725, 1997.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2017.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR (Poster)*. OpenReview.net, 2019.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. *Advances in neural information processing systems*, 30, 2017.

Johannes C. Paetzold, Julian McGinnis, Suprosanna Shit, Ivan Ezhov, Paul Büschl, Chinmay Prabhakar, Anjany Sekuboyina, Mihail I. Todorov, Georgios Kaissis, Ali Ertürk, Stephan Günnemann, and Bjoern H. Menze. Whole brain vessel graphs: A dataset and benchmark for graph learning and neuroscience. In *NeurIPS Datasets and Benchmarks*, 2021.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *Advances in neural information processing systems*, 30, 2017.

Petar Velickovic and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

Petar Velickovic, Lars Buesing, Matthew C. Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. In *NeurIPS*, 2020a.

Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *ICLR*. OpenReview.net, 2020b.

Petar Velickovic, Matko Bosnjak, Thomas Kipf, Alexander Lerchner, Raia Hadsell, Razvan Pascanu, and Charles Blundell. Reasoning-modulated representations. *CoRR*, abs/2107.08881, 2021.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. *arXiv preprint arXiv:2205.15659*, 2022.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, pp. 2692–2700, 2015.

Louis-Pascal A. C. Xhonneux, Andreea Deac, Petar Velickovic, and Jian Tang. How to transfer algorithmic reasoning knowledge to learn new algorithms? In *NeurIPS*, pp. 19500–19512, 2021.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *ICLR*. OpenReview.net, 2020.

# A PSEUDO-CODE

## A.1 FORD-FULKERSON PSEUDO-CODE

**Algorithm 1** Ford-Fulkerson

**Input:** $G = (V, E, c)$**,** $s$**,** $t$;
$f_{uv} = 0 \ : \ \forall (u, v) \in E$
**while** $\exists$ augmenting path $p = s, \ldots, t$ in $G_f$ **do**;
  $df = \min\{c_{uv} : (u, v) \in p\}$;
  **for** each $(u, v) \in p$ **do**
    $f_{uv} = f_{uv} - df$
    $f_{vu} = f_{vu} + df$
  **end for**
**end while**
**return** $f$

## A.2 CORRECTIVE FLOW PROCEDURE PSEUDO-CODE

(Figure 3).

**Algorithm 2** Flow correction algorithm

*Input*: $G = (V, E)$, $\boldsymbol{F}$: flow matrix, $C$: capacity matrix, *s*: source, *t*: sink
$V^- = \{v \mid v \in V \wedge \sum_{(i,v)\in E} \boldsymbol{F}_{iv} + \sum_{(v,i)\in E} \boldsymbol{F}_{vi} < 0\}$
$V^+ = \{v \mid v \in V \wedge \sum_{(i,v)\in E} \boldsymbol{F}_{iv} + \sum_{(v,i)\in E} \boldsymbol{F}_{vi} > 0\}$
**for** each $v- \in V^-$ **do**
  $\varepsilon = |\sum_{(i,v)\in E} \boldsymbol{F}_{iv} + \sum_{(v,i)\in E} \boldsymbol{F}_{vi}|$
  find a path from $v^-$ to $s$ and send back $\varepsilon$ amount of flow to $s$
**end for**
**for** each $v+ \in V^+$ **do**
  $\varepsilon = |\sum_{(i,v)\in E} \boldsymbol{F}_{iv} + \sum_{(v,i)\in E} \boldsymbol{F}_{vi}|$
  find a path from $v^+$ to $t$ and reduce the out-flow by $\varepsilon$
**end for**
$\varepsilon_s = |\sum_{(s,i)\in E} \boldsymbol{F}_{si} - \sum_{(s,i)\in E} C_{si}|$
**while** $\varepsilon_s > 0$ **do**
  find a path from $s$ to $t$ and reduce the out-flow by enforcing capacity constraints
  recompute $\varepsilon_s$
**end while**
**return** $\boldsymbol{F}$

# B HYPERPARAMETERS OPTIMISATION

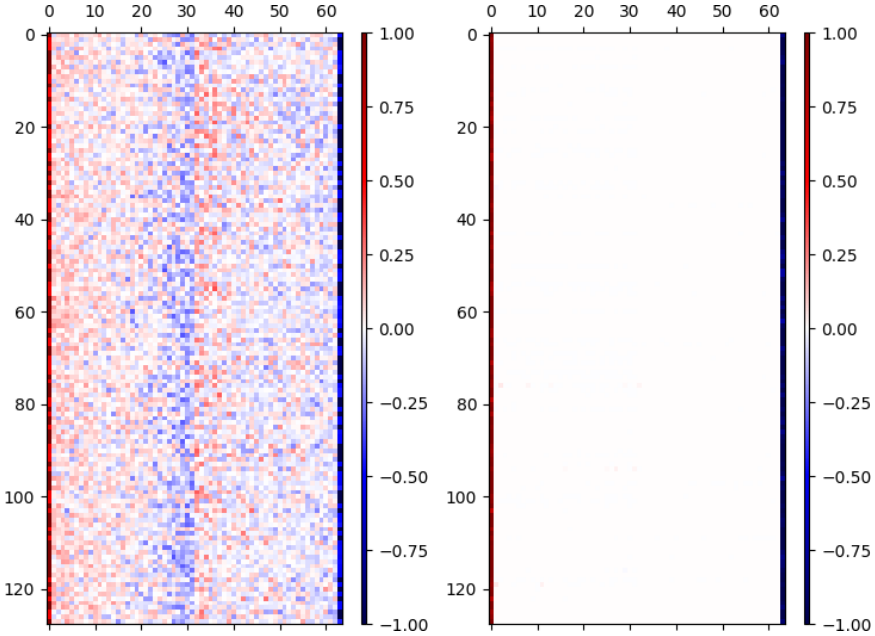
We sample $n = 50$ configurations for each level of the bi-level random search. In Table 4 we report details for the first and second level random searches, where the second refines the first based on validation results of the primal model. Winner hyperparameters for *primal* model are: hidden dimension = 68, learning rate = 0.009341, weight decay = 0.003420. Winner hyperparameters for *dual* model are: hidden dimension = 65, learning rate = 0.009868, weight decay = 0.001734. *Pipeline* shares hyperpameters with *dual*.

# C ADDITIONAL RESULTS

We report in Table 6 additional performance metrics for brain vessel classification other than the balanced accuracy presented in the main paper. Furthermore, we also provide a visual examples (Figure 3) of how the flow conservation is corrected by the corrective procedure of Algorithm 2.

Table 4: First-level and second-level random searches details.

| Level | Hyperparameter | Min. value | Max. value | Distribution |
|---|---|---|---|---|
| Level 1 | Learning rate | 1e-5 | 1e-1 | log-uniform |
| | Weight decay | 1e-5 | 1e-1 | log-uniform |
| | Hidden dimension | 16 | 512 | uniform |
| Level 2 | Learning rate | 1e-3 | 1e-2 | log-uniform |
| | Weight decay | 1e-3 | 4e-3 | log-uniform |
| | Hidden dimension | 60 | 100 | uniform |




(a) $\boldsymbol{F}$ conservation error prior correction (b) $\boldsymbol{F}$ conservation error after correction

Figure 3: This image represents the (normalised) flow conservation error for all nodes (y axis) of each test graph (x axis). Sources and sinks are sorted to be always first and last node in this plot. Nodes which create or lose flow have positive or negative error, represented by red and blue in the plot. Through Algorithm 2 we are able to zero all the inner error (white represents zero error), with $s$ and $t$ being the only nodes which correctly do not conserve flow.

Table 5: Error of predicting the intermediate $\boldsymbol{F}^{(t)}$, averaged on all steps.

| **Model** | 2-Community | Bipartite |
|---|---|---|
| PGN-max *(primal)* | $0.293 \pm 0.003$ | $0.821 \pm 0.17$ |
| PGN-mean *(primal)* | $0.314 \pm 0.006$ | $1.129 \pm 0.18$ |
| MPNN-max *(primal)* | $0.294 \pm 0.003$ | $0.783 \pm 0.11$ |
| MPNN-mean *(primal)* | $0.314 \pm 0.002$ | $0.921 \pm 0.22$ |
| PGN-max *(dual)* | $\mathbf{0.271 \pm 0.002}$ | $\mathbf{0.781 \pm 0.29}$ |
| PGN-mean *(dual)* | $0.278 \pm 0.004$ | $1.052 \pm 0.12$ |
| MPNN-max *(dual)* | $0.294 \pm 0.002$ | $0.988 \pm 0.22$ |
| MPNN-mean *(dual)* | $\mathbf{0.270 \pm 0.003}$ | $1.064 \pm 0.35$ |
| PGN-max *(pipeline)* | $0.287 \pm 0.003$ | $\mathbf{0.774 \pm 0.26}$ |
| PGN-mean *(pipeline)* | $0.297 \pm 0.002$ | $1.033 \pm 0.32$ |
| MPNN-max *(pipeline)* | $0.261 \pm 0.002$ | $0.951 \pm 0.34$ |
| MPNN-mean *(pipeline)* | $0.290 \pm 0.003$ | $0.922 \pm 0.20$ |

Table 6: Additional performance metrics for BVG edge classification.

| | **Model** | F1 score | ROC AUC |
|---|---|---|---|
| CD1-E-3 | GCN | $78.04\% \pm 2.91$ | $67.91\% \pm 0.18$ |
| | SAGE | $81.72\% \pm 0.32$ | $70.94\% \pm 0.26$ |
| | SAGE & Node2Vec | $81.73\% \pm 0.11$ | $71.88\% \pm 0.09$ |
| | SAGE & PGN-max *(primal)* | $82.03\% \pm 1.45$ | $71.97\% \pm 0.18$ |
| | SAGE & PGN-max *(dual)* | $\mathbf{83.04\% \pm 0.73}$ | $74.96\% \pm 0.16$ |
| | Cluster-SAGE | $\mathbf{83.07\% \pm 0.75}$ | $74.21\% \pm 0.49$ |
| | Cluster-SAGE & Node2Vec | $82.33\% \pm 0.34$ | $74.14\% \pm 0.13$ |
| | Cluster-SAGE & PGN-max *(primal)* | $81.16\% \pm 0.34$ | $73.57\% \pm 1.89$ |
| | Cluster-SAGE & PGN-max *(dual)* | $\mathbf{83.61\% \pm 1.26}$ | $\mathbf{76.25\% \pm 0.05}$ |
| CD1-E-2 | GCN | $88.07\% \pm 1.40$ | $78.67\% \pm 0.13$ |
| | SAGE | $88.20\% \pm 0.68$ | $82.49\% \pm 0.22$ |
| | SAGE & Node2Vec | $89.37\% \pm 0.29$ | $82.45\% \pm 0.44$ |
| | SAGE & PGN-max *(primal)* | $90.74\% \pm 1.60$ | $83.65\% \pm 0.40$ |
| | SAGE & PGN-max *(dual)* | $91.92\% \pm 0.84$ | $85.50\% \pm 0.15$ |
| | Cluster-SAGE | $\mathbf{92.32\% \pm 0.65}$ | $85.55\% \pm 0.24$ |
| | Cluster-SAGE & Node2Vec | $89.14\% \pm 0.22$ | $84.82\% \pm 0.48$ |
| | Cluster-SAGE & PGN-max *(primal)* | $89.22\% \pm 3.72$ | $83.93\% \pm 1.53$ |
| | Cluster-SAGE & PGN-max *(dual)* | $\mathbf{92.38\% \pm 0.19}$ | $\mathbf{86.66\% \pm 0.28}$ |
| CD1-E-1 | GCN | $80.58\% \pm 0.53$ | $67.86\% \pm 0.16$ |
| | SAGE | $85.05\% \pm 0.89$ | $74.21\% \pm 0.22$ |
| | SAGE & Node2Vec | $87.05\% \pm 0.51$ | $74.78\% \pm 0.11$ |
| | SAGE & PGN-max *(primal)* | $87.11\% \pm 1.87$ | $74.80\% \pm 0.32$ |
| | SAGE & PGN-max *(dual)* | $87.22\% \pm 0.70$ | $74.80\% \pm 0.10$ |
| | Cluster-SAGE | $\mathbf{87.98\% \pm 1.10}$ | $74.79\% \pm 0.50$ |
| | Cluster-SAGE & Node2Vec | $\mathbf{87.34\% \pm 1.59}$ | $\mathbf{75.94\% \pm 0.16}$ |
| | Cluster-SAGE & PGN-max *(primal)* | $\mathbf{87.66\% \pm 0.48}$ | $\mathbf{75.82\% \pm 0.17}$ |
| | Cluster-SAGE & PGN-max *(dual)* | $\mathbf{87.57\% \pm 0.05}$ | $\mathbf{75.77\% \pm 0.14}$ |