Rethinking Message Passing for Algorithmic Alignment

Editors: List of editors' names

Abstract

Most Graph Neural Networks are based on the principle of message-passing, where all neighboring nodes exchange messages with each other simultaneously. We want to challenge this paradigm by introducing the Flood and Echo Net, a novel architecture that aligns neural computation with the principles of distributed algorithms. In our method, nodes sparsely activate upon receiving a message, leading to a wave-like activation pattern that traverses the graph. Through these sparse but parallel activations, the Net becomes more expressive than traditional MPNNs which are limited by the 1-WL test and also is provably more efficient in terms of message complexity. Moreover, the mechanism's ability to generalize across graphs of varying sizes positions it as a practical architecture for the task of algorithmic learning. We test the Flood and Echo Net on a variety of synthetic tasks and find that the algorithmic alignment of the execution improves generalization to larger graph sizes.

Keywords: Graph Neural Networks, Algorithmic Alignment, Message Passing

1. Introduction

The message-passing paradigm is central to graph learning as many proposed architectures are captured under the framework of Message Passing Neural Networks (MPNNs). In these networks, all nodes simultaneously update their states based on their neighbors' states through message exchange. While flexible, this approach requires considerable computation, as messages are sent over all edges in every round, even when many nodes may not actively participate in the computation.

We challenge this paradigm by proposing the Flood and Echo Net (FE Net), a new execution framework inspired by the flooding and echo pattern in distributed computing. The computation is initiated by an origin node, which starts a phase that consists of flooding and echo parts. During flooding, messages propagate away from the root, with nodes only sending messages farther from the origin. In the echo part, the flow reverses, with nodes sending messages closer to the origin. Crucially, the FE Net activates only a subset of nodes at each step, resulting in a sparse yet parallel activation pattern. As a result, this approach offers three key advantages over regular MPNNs: efficiency through message complexity, expressivity, and generalization to larger graph sizes.

While standard MPNNs exchange $\mathcal{O}(m)$ messages with their one-hop neighborhood in each round, a complete phase of a FE Net exchanges $\mathcal{O}(m)$ messages but can incorporate information beyond the immediate local neighborhood. Moreover, by implicitly leveraging distance information, the FE Net's expressiveness goes beyond the 1-WL test. In the context of algorithm learning, where GNNs should generalize across graph sizes, MPNNs typically need to scale the number of rounds with graph size. In contrast, the FE Net's execution naturally involves the entire graph, potentially allowing better generalization to larger graphs through direct alignment on the execution level. We hypothesize that this algorithmic alignment makes the FE Net particularly suitable for algorithm learning and provide evidence for this through empirical validation on a variety of algorithmic tasks.

2. Flood and Echo Net

Algorithm 1 Flood and Echo Net

```
1. $D \leftarrow \text{distances}(G, \text{origin})$
2. $x \leftarrow \text{Encoder}(x)$
3. For t = 1 to phases:
    (a) For d = 1 to max($D$): *flooding*
        i. $x[d] \leftarrow \text{FConv}^t(d-1 \to d)$
        ii. $x[d] \leftarrow \text{FCrossConv}^t(d \to d)$
    (b) For d = max($D$) to 1: *echo*
        i. $x[d] \leftarrow \text{ECrossConv}^t(d \to d)$
        ii. $x[d-1] \leftarrow \text{EConv}^t(d \to d-1)$
    (c) $x \leftarrow \text{Update(x)}$
4. $x \leftarrow \text{Decoder}(x)$
```

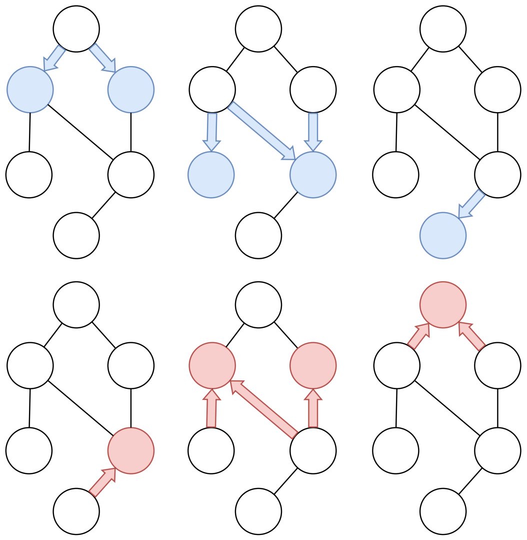

Figure 1: On the left, an algorithm describing the FE Net . First, the distances are pre-computed to activate and update the proper nodes. The convolutions $a \to b$ send messages from nodes at distance a to nodes at distance b, with only the nodes at distance b updating their state, indicated by $x[b]$. On the right, a single phase of a FE Net . At every update step, only a subset of nodes is active. The origin is the top node of the graph, and the blue arrows depict the information flow in the flooding, while the red arrows represent the echo part. Note that a single phase activates all nodes in the graph, regardless of the graph size.

The fields of distributed computing and Graph Neural Networks are tightly connected (Papp and Wattenhofer, 2022; Sato et al., 2019; Loukas, 2020). Moreover, it was shown that aligning the architecture with the underlying learning objective (Xu et al., 2020; Dudzik and Veličković, 2022) can be beneficial in terms of performance and sample complexity. his raises the question of whether we could transfer other insights from distributed computing to the field of graph learning. Note in MPNNs all nodes exchange messages with all their neighbors in every round. We challenge this paradigm by taking inspiration from a design pattern called *flooding and echo* (Chang, 1982). This pattern is a common building block in distributed algorithms (Kuhn et al., 2007) to first broadcast (flooding) (Dalal and Metcalfe, 1978) messages throughout the entire graph and then gather back (echo) information from all nodes.

In the Flood and Echo Net, the computation is initiated from an origin node. Then, T phases, each consisting of a flooding and echo part, are executed. In Figure 1 we outline the pseudo code for the FE Net . At the beginning, nodes are partitioned according to their distance to the origin. Then, T phases are executed, in each phase a flooding followed by an echo is performed. During the flooding part, messages propagate outwards, away from the origin, by iterating through the distances in ascending order. We differentiate between two types of edges in the convolutions. First, FConv sends messages from nodes at distance $d-1$ towards nodes at distance d. However, only the nodes at distance d update their state, indicated by the notation $x[d]$. Then, FCrossConv sends messages between nodes that are at distance d. After the completion of the flooding part, the message flow reverses and is echoed back towards the origin. Again, we iterate over the distances but now in descending order. Similarly to above, we distinguish updates of nodes at the same distance using ECrossConv and updating nodes at distance $d-1$, which receive messages from nodes at distance d through EConv. Note, that only a subset of nodes, which are located at the same distance, are activated simultaneously. Therefore, FE Net can make use of a sparse but parallel activation pattern that propagates throughout the entire graph. Throughout this work, we assume that the origin is given by the task at hand, however, we outline different modes of operation, which we discuss in more detail in Appendix A, which also contains a comparison to the computation of regular MPNNs. For a visual illustration of an entire phase, we refer to Figure 1.

3. Theoretical Insights

First, we show that the FE Net not only matches the expressiveness of MPNNs but also surpasses them in terms of the 1-WL test.

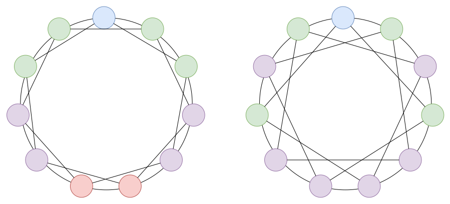
Figure 2: Two regular graphs that cannot be distinguished with standard MPNNs but the FE Net can.

Theorem 1 *On connected graphs, the Flood and Echo Net is at least as expressive as any MPNN . Furthermore, it exchanges at most as many messages.*

However, while MPNNs are limited by the 1-WL test, the FE Net is more expressive. Although it also exchanges messages solely on the original graph topology, the mechanism can implicitly leverage more information to distinguish nodes. This is achieved through the alignment of message propagation with the distance to the origin in the graph.

Theorem 2 *On connected graphs, Flood and Echo Net is strictly more expressive than 1-WL and, by extension, standard MPNNs.*

In regular MPNNs, if any information needs to be propagated over a distance of D hops, the total number of node updates is $\mathcal{O}(Dn)$ and exchanged messages is $\mathcal{O}(Dm)$. In a single phase of a FE Net , which consists of one flooding followed by one echo part, each node is activated a constant number of times, while there are also at most a constant number of messages passed along each edge. Therefore, a single phase performs $\mathcal{O}(n)$ node updates and exchanges $\mathcal{O}(m)$ messages. Crucially, if information needs to be exchanged over a distance of D hops, this can be achieved with a constant number of phases, as each phase exchanges information throughout the entire graph. Therefore, it is possible to exchange information over a distance of D hops using only $\mathcal{O}(m)$ messages compared to $\mathcal{O}(Dm)$ messages used by MPNNs. As a consequence, there exist tasks that can be solved much more efficiently using the FE Net . Moreover, by Theorem 1, it also uses at most the same number of messages. For a more detailed discussion on the runtime and message complexity, we refer to Appendix C.

Lemma 3 *There exist tasks that Flood Echo can solve using $\mathcal{O}(m)$ messages, whereas no MPNN can solve them using less than $\mathcal{O}(nm)$ messages.*

Table 1: Size generalization experiments on algorithmic tasks, all models were trained on graphs of size 10 and tested on graphs of size 100. We compare the FE Net against a regular GIN, which executes L rounds, PGN and RecGNN, which adapt the number of rounds. We report both the node accuracy with $n()$ and the graph accuracy with $g()$.

Model	Messages	PrefixSum			Distance			Path Finding		
		n(10)	n(100)	g(100)	n(10)	n(100)	g(100)	n(10)	n(100)	g(100)
GIN	$\mathcal{O}(Lm)$	0.78 ± 0.01	0.53 ± 0.00	0.00 ± 0.00	0.97 ± 0.01	0.91 ± 0.01	0.04 ± 0.06	0.99 ± 0.01	0.70 ± 0.05	0.00 ± 0.00
PGN	$\mathcal{O}(nm)$	0.94 ± 0.12	0.52 ± 0.01	0.00 ± 0.00	0.99 ± 0.01	0.89 ± 0.01	0.01 ± 0.02	1.00 ± 0.00	0.77 ± 0.03	0.00 ± 0.00
RecGNN	$\mathcal{O}(nm)$	1.00 ± 0.00	0.93 ± 0.07	0.66 ± 0.31	1.00 ± 0.00	0.99 ± 0.02	0.93 ± 0.15	1.00 ± 0.00	0.95 ± 0.04	0.45 ± 0.33
Flood and Echo Net	$\mathcal{O}(m)$	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.02	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00

4. Empirical Results

One key challenge for generalization to larger graph sizes is how to adapt the architecture. If it does not adjust at all, the information might be not present in the same receptive field, but located farther away. Therefore, a common strategy is to adjust the number of rounds, according to the increase of the problem size. However, the FE Net generalizes to larger sizes in a different way compared to regular MPNNs. In fact, during a single phase, messages already propagate throughout the entire graph and can, therefore, be updated using information beyond the immediate neighborhood. Previous work has already indicated that changes in the architecture or so-called "algorithmic alignment" (Engelmayer et al., 2023; Dudzik and Veličković, 2022; Xu et al., 2020) can be beneficial for learning and generalization. Based on these insights, we hypothesize that the FE Net can significantly improve size generalization for algorithm learning. In the following, we empirically validate our hypothesis on a variety of algorithmic tasks: PrefixSum, Distance and Path Finding. For a detailed description we refer to the Appendix, all tasks can be defined on graphs of various sizes and require information exchange beyond the immediate neighborhood. All models are trained on small graphs of size 10 and tested on graphs of size 100. From the results in Table 1, we observe that the baseline using a fixed number of layers already struggles to fit the training data and deteriorates on larger instances. The other models exhibit better generalization when considering the node accuracy. Further, we report graph accuracy, measuring instances where all nodes are correctly labeled. This metric is crucial, as a single incorrect node can cause the entire algorithm to fail. There, the overall model performance of the baselines drop significantly compared to the FE Net.

5. Conclusion

We challenge the standard message-passing paradigm commonly used in graph learning and introduce the Flood and Echo Net. Our approach aligns its execution with a design pattern from distributed algorithms, propagating messages in a wave-like activation throughout the entire graph. The sparse node activations of the FE Net result in greater expressiveness and improved efficiency in terms of message complexity, facilitating message propagation across the entire graph more effectively. A key advantage of the FE Net is its natural ability to generalize to graphs of varying sizes, which proves beneficial in algorithm learning tasks.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2021a.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=i80OPhOCVH2.

E.J.H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, SE-8(4):391–401, 1982. doi: 10.1109/TSE.1982.235573.

Yogen K. Dalal and Robert M. Metcalfe. Reverse path forwarding of broadcast packets. *Commun. ACM*, 21:1040–1048, 1978. URL https://api.semanticscholar.org/CorpusID:5638057.

Andrew Dudzik and Petar Veličković. Graph neural networks are dynamic programmers, 2022.

Andrew Dudzik, Tamara von Glehn, Razvan Pascanu, and Petar Veličković. Asynchronous algorithmic alignment with cocycles, 2023.

Valerie Engelmayer, Dobrik Georgiev, and Petar Veličković. Parallel algorithms align with neural execution, 2023.

Lukas Faber and Roger Wattenhofer. Asynchronous message passing: A new framework for learning in graphs, 2023. URL https://openreview.net/forum?id=2_I3JQ70U2.

Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio', and Michael Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology, 2023.

Florian Grötschla, Joël Mathys, and Roger Wattenhofer. Learning graph algorithms with recurrent graph neural networks, 2022.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner, 2022.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Fabian Kuhn, Thomas Locher, and Roger Wattenhofer. Tight Bounds for Distributed Selection. In *19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), San Diego, CA, USA*, June 2007.

Andreas Loukas. What graph neural networks cannot learn: depth vs width, 2020.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks, 2020.

Karolis Martinkus, Pál András Papp, Benedikt Schesch, and Roger Wattenhofer. Agent-based graph neural networks, 2023.

Julian Minder, Florian Grötschla, Joël Mathys, and Roger Wattenhofer. Salsa-clrs: A sparse and scalable benchmark for algorithmic reasoning, 2023.

Moritz Neun, Christian Eichenberger, Henry Martin, Markus Spanring, Rahul Siripurapu, Daniel Springer, Leyan Deng, Chenwang Wu, Defu Lian, Min Zhou, et al. Traffic4cast at neurips 2022–predict dynamics along graph edges from sparse node data: Whole city traffic and eta from stationary vehicle detectors. In *NeurIPS 2022 Competition Track*, pages 251–278. PMLR, 2022.

Pál András Papp and Roger Wattenhofer. An introduction to graph neural networks from a distributed computing perspective. In Raju Bapi, Sandeep Kulkarni, Swarup Mohalik, and Sathya Peri, editors, *Distributed Computing and Intelligent Technology*, pages 26–44, Cham, 2022. Springer International Publishing. ISBN 978-3-030-94876-4.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks, 2021.

Ladislav Rampasek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=lMMaNf6oxKM.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems, 2019.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks, 2021.

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

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR, 2022.

Petar Veličković, Lars Buesing, Matthew C. Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

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

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks, 2021.

Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks, 2021.

Bohang Zhang, Guhao Feng, Yiheng Du, Di He, and Liwei Wang. A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests, 2023a.

Bohang Zhang, Shengjie Luo, Liwei Wang, and Di He. Rethinking the expressive power of gnns via graph biconnectivity, 2023b.

Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification, 2021.

Lingxiao Zhao, Louis Härtel, Neil Shah, and Leman Akoglu. A practical, progressively-expressive gnn, 2022.

Appendix A. Flood and Echo Net Definition

First, recall the standard execution of a message-passing-based GNN. Whenever we refer to an MPNN throughout this paper, we will refer to a GNN that operates on the original graph topology and exchanges messages in the following way:

$$\begin{aligned} a_v^t &= \text{AGGREGATE}^k(\{\{x_u^t \mid u \in N(v)\}\}) \\ x_v^{t+1} &= \text{UPDATE}(x_v^t, a_v^t) \end{aligned}$$

Now, the mechanism for the Flood and Echo Net. Let r be the origin of the computation phase and let $d(v)$ denote the shortest path distance from v to r. Then, the update rule for of the Flood and Echo Net looks is defined as follows, assume T phases are executed. At the beginning of each phase t, the flooding is performed, where the nodes are sequentially activated one after another depending on their distance towards the root. Each convolution is either from nodes at distance d to $d+1$ (flood), from $d+1$ to d (echo) or between nodes at the same distance (floodcross, echocross). The term $x[d]$ denotes that only nodes at distance d update their state. For each distance d from 1 to the max distance in the graph the following update is performed:

$$\begin{aligned} f_v^t &= \text{AGGREGATE}_{Flood}(\{\{x_u^t \mid d(u) = d-1, u \in N(v)\}\}) \\ x_v^{t+1}[d] &= \text{UPDATE}_{Flood}(x_v^t, f_v^t) \\ fc_v^t &= \text{AGGREGATE}_{FloodCross}(\{\{x_u^{t+1} \mid d(u) = d, u \in N(v)\}\}) \\ x_v^{t+1}[d] &= \text{UPDATE}_{FloodCross}(x_v^{t+1}, fc_v^t) \end{aligned}$$

And similarly for each distance d from max distance -1 to 0 the Echo phase

$$\begin{aligned} ec_v^t &= \text{AGGREGATE}_{EchoCross}(\{\{x_u^t \mid d(u) = d+1, u \in N(v)\}\}) \\ x_v^{t+1}[d] &= \text{UPDATE}_{EchoCross}(x_v^t, ec_v^t) \\ e_v^t &= \text{AGGREGATE}_{Echo}(\{\{x_u^{t+1} \mid d(u) = d, u \in N(v)\}\}) \\ x_v^{t+1}[d] &= \text{UPDATE}_{Echo}(x_v^{t+1}, e_v^t) \end{aligned}$$

The phase is completed after another update for all nodes.

$$x_v^{t+1} = \text{UPDATE}(x_v^{t+1})$$

Note that the node activations are done in a sparse way, therefore, for all updates that take an empty neighborhood set as the second argument no update is performed and the state is maintained. Furthermore, in practise we did not find a significant difference in performing the last update step, which is why in the implementation we do not include it. In Figure 4 we outline the differences between the computation of an MPNN and a Flood and Echo Net.

Modes of Operation The computation of the Flood and Echo Net starts from an origin node. This allows for different usages of the proposed method. In the following, we outline three different strategies, which we will refer to as different modes of operations: *fixed, random* and *all*. Across all modes of operation, once the origin is chosen, the same flooding and echo parts are executed to compute node embeddings. These are directly used for node classification tasks; for graph classification, we sum up the final predicted class probabilities of the individual nodes.

In the *fixed* mode, the origin is given or defined by the problem instance, i.e. by a marked source node specific to the task. Alternatively, in the *random* mode, an origin is chosen amongst all nodes uniformly at random. In the *all* mode, we execute the Flood and Echo Net once for every node. In every run, we keep only the node embedding for the chosen origin. This can be seen as a form of ego graph prediction Zhao et al. (2021) for each node. Although computationally more expensive, it could also be used for efficient inference on tasks where only a subset of nodes is of interest.

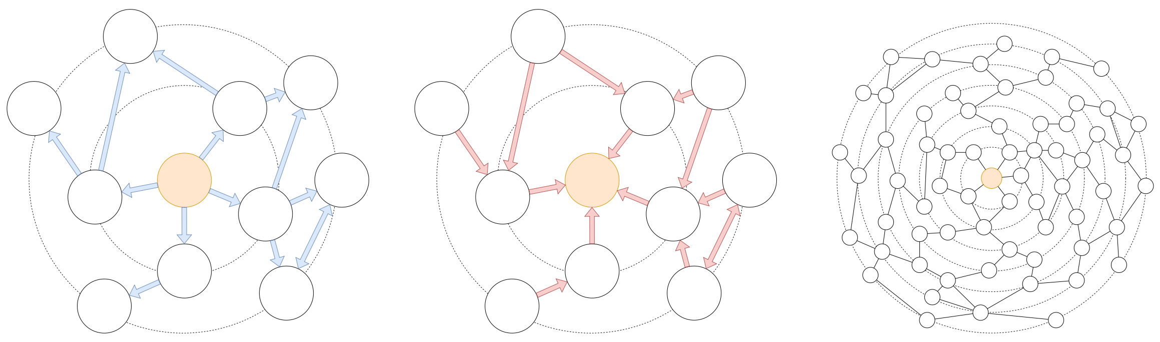

Figure 3: The Flood and Echo Net propagates messages in a wave-like pattern throughout the entire graph. Starting from an origin (orange), messages are sent toward the origin's neighbors and then continuously sent or "flooded" farther away outwards (blue). Afterward, the flow reverses, and messages are "echoed" back (red) toward the origin. Throughout the computation, only a small subset of nodes is active at any given time, passing messages efficiently throughout the entire graph. Moreover, the mechanism naturally generalizes to graphs of larger sizes.

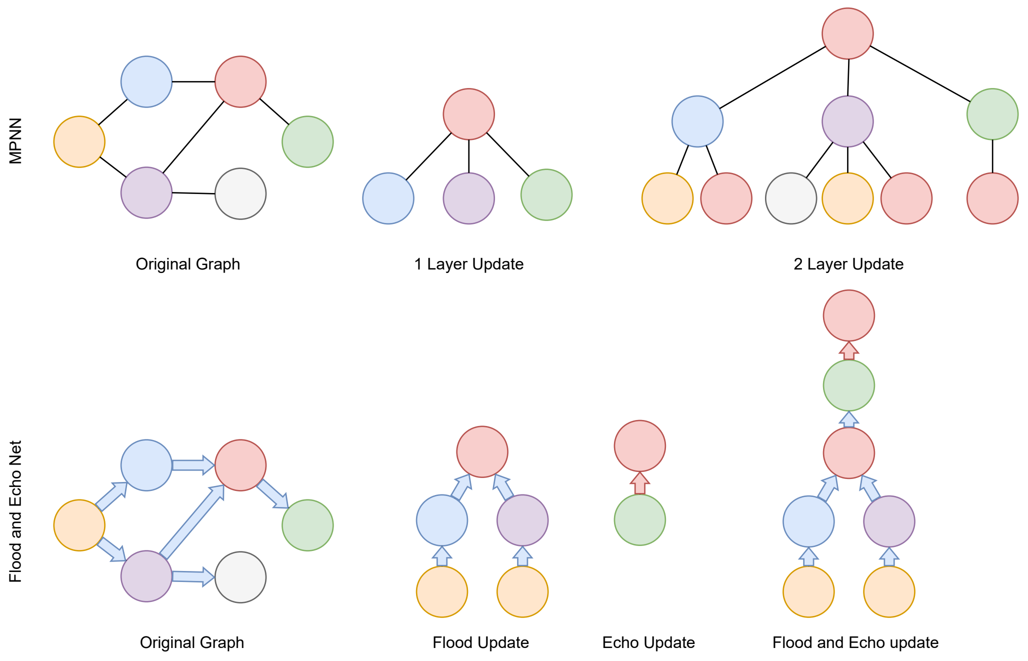

Figure 4: Visualization of the computation executed on the same graph for a regular MPNN and a Flood and Echo Net from the perspective of the red node. The top row shows the computation for regular MPNN both for 1 and 2 layers of message-passing. Note that executing l layers takes into account the l-Hop neighborhood. On the bottom row, the computation from the perspective of the red node in a Flood and Echo net is shown. Note that the origin of the Flood and Echo Net is the orange node. The two middle figures illustrate the updates in the flood and the echo part respectively. Furthermore, the figure on the right shows the combined computation for an entire phase.

Appendix B. Related Work

Originally proposed by Scarselli et al. (2008), Graph Neural Networks have seen a resurgence with applications across multiple domains (Veličković et al., 2017; Kipf and Welling, 2016; Neun et al., 2022). Notably, this line of research has gained theoretical insights through its connection to message-passing models from distributed computing (Sato et al., 2019; Loukas, 2020; Papp and Wattenhofer, 2022). This includes strengthening existing architectures to achieve maximum expressiveness (Xu et al., 2018; Sato et al., 2021) or going beyond traditional models by changing the graph topology (Papp et al., 2021; Alon and Yahav, 2021b). In this context, multiple architectures have been investigated to combat information bottlenecks in the graph (Alon and Yahav, 2021a), i.e. using graph transformers (Rampasek et al., 2022). Note that our work is orthogonal to this, as we focus on message-passing on the original graph topology. Moreover, we investigate how specific information can be exchanged throughout the entire graph, which might be challenging even if no bottleneck is present. Similarly, higher order propagation mechanisms (Zhang et al., 2023b; Maron et al., 2020; Zhao et al., 2022) have been proposed to tackle this issue or gain more expressiveness. While some of these approaches also incorporate distance information, this usually comes at the cost of higher-order message-passing. Whereas our work emphasizes a simple execution mechanism on the original graph topology. In recent work, even the synchronous message-passing among all nodes has been questioned (Martinkus et al., 2023; Faber and Wattenhofer, 2023), giving rise to alternative neural graph execution models.

How GNNs can generalize across graph sizes (Yehudai et al., 2021) and especially their generalization capabilities for algorithmic tasks, attributed to their structurally aligned computation (Xu et al., 2020) has been of much interest. This has led to investigations into the proper alignment of parts of the architecture (Dudzik and Veličković, 2022; Engelmayer et al., 2023; Dudzik et al., 2023). A central focus has been on how these networks learn to solve algorithms (Veličković et al., 2022; Ibarz et al., 2022; Minder et al., 2023). Moreover, the ability to extrapolate (Xu et al., 2021) and dynamically adjust the computation in order to reason for longer when confronted with more challenging instances remains a key aspect (Schwarzschild et al., 2021; Grötschla et al., 2022; Tang et al., 2020).

A variety of GNNs that do not follow the 1 hop neighborhood aggregation scheme have been unified under the view of so-called Subgraph GNNs. The work of Zhang et al. (2023a) analyses these models in terms of their expressiveness and gives the following general definition:

Definition 4 *A general subgraph GNN layer has the form*

$$h_G^{(l+1)}(u,v) = \sigma^{(l+1)}(\mathsf{op}_1(u,v,G,h_G^{(l)}), \cdots, \mathsf{op}_r(u,v,G,h_G^{(l)})),$$

where $\sigma^{(l+1)}$ is an arbitrary (parameterized) continuous function, and each atomic operation $\mathsf{op}_i(u,v,G,h)$ can take any of the following expressions:

- *Single-point: $h(u,v)$, $h(v,u)$, $h(u,u)$, or $h(v,v)$;*
- *Global: $\sum_{w\in\mathcal{V}_G} h(u,w)$ or $\sum_{w\in\mathcal{V}_G} h(w,v)$;*
- *Local: $\sum_{w\in\mathcal{N}_{G^u}(v)} h(u,w)$ or $\sum_{w\in\mathcal{N}_{G^v}(u)} h(w,v)$.*

We assume that $h(u,v)$ is always present in some op_i.

This allows us to capture a more general class of Graph Neural Networks, i.e., the work of Zhang et al. (2023b), which can incorporate distance information into the aggregation mechanism this way. Note that the proposed mechanism of the Flood and Echo Net differs from that of this particular notion of subgraph GNNs. At each update step, only a subset of nodes is active. This allows nodes to take into account nodes that are activated earlier, which is not directly comparable to subgraph GNNs where the node updates still happen simultaneously for the nodes in question.

Another important issue that GNNs often struggle with is the so-called phenomenon of oversquashing (Alon and Yahav, 2021a). In simple terms, if too much information has to be propagated through the graph using a few edges, a bottleneck occurs, squashing the relevant information together, leading to information loss and subsequent problems for learning. Recent work of (Giovanni et al., 2023) theoretically analyses the reasons leading to the oversquashing phenomena and identifies the width and depth of the network but also the graph topology as key contributors. Note that the proposed Flood and Echo Net is not designed to tackle the problem of oversquashing. Rather, it tries to facilitate information throughout the graph, assuming that there is no inherent (topological) bottleneck. It only affects the aforementioned depth aspect of the network. However, as outlined by (Giovanni et al., 2023), the depth is likely to have a marginal effect compared to the graph topology.

The works of Martinkus et al. (2023), namely AgentNet, and Faber and Wattenhofer (2023), who proposes AMP (Asynchronous Message Passing), also draw inspiration from the field of distributed computing. Although they share some aspects in their mechanisms, their respective settings differ quite a bit. In AgentNet, there exist agents which traverse the graph which gives them the possibility to solve problems on the graph in sublinear time. In contrast, our approach tries to enable communication throughout the whole graph, especially in the context of different graph sizes. On the other hand, AMP activates nodes one at a time, benefiting from a similar computational sparsity as our method. However, note that the Flood and Echo Net's execution is more structured. On one side, this leads to less flexible activation patterns, however, on the other hand, it translates naturally across graph sizes. Whereas AMP has to additionally learn a termination criteria which must generalize.

Appendix C. Runtime

C.1. Runtime Complexity

We denote n the number of nodes, m the number of edges and D the diameter of the graph. Furthermore, let T be the number of phases for a Flood and Echo Net and L be the number of layers for an MPNN.

A single round of regular message-passing exchanges $\mathcal{O}(m)$ messages. Therefore, executing L such rounds results in $\mathcal{O}(L)$ steps and $\mathcal{O}(Lm)$ messages. Note that in order for communication between any two nodes L has to be in the order of $\mathcal{O}(D)$.

A single phase of a Flood and Echo Net, consisting of one starting node, exchanges $\mathcal{O}(m)$ messages and does so in $\mathcal{O}(D)$ steps. Therefore, executing T phases of a Flood and Echo Net results in $\mathcal{O}(Tm)$ messages exchanged in $\mathcal{O}(TD)$ steps. Note, that it is sufficient for T to be constant $\mathcal{O}(1)$ in order to communicate throughout the whole graph and does not necessarily have to be scaled according to the size of the graph.

The variations *fixed* and *random* perform their executions only for a specific single node. Contrary, the *all* variation performs such an execution for each of its nodes individually. Therefore, both the number of messages and the number of steps is increased by a factor of n.

Appendix D. Proofs and Derivations

Proof of Theorem 1 It has been shown by the work of Xu et al. (2018) that the *Graph Isomorphism Network* (GIN) achieves maximum expressiveness amongst MPNN. In the following, we will show that a Flood and Echo Net can simulate the execution of a GIN on connected graphs, therefore matching it in its expressive power. Let G_I be a GIN using a node state vector h_v^k of dimension d_i.

$$h_v^{(k)} = \text{MLP}^{(k)}((1+\epsilon)h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)})$$

Let G_F be a Flood and Echo Net using node state vector $q_v^{(k)}$ of dimension $d_f = 2 \cdot d_i$. We partition the vector $q_v^{(k)} = o_v^{(k)} \mid\mid n_v^{(k)}$ into two vectors of dimension d_i. Initially, we assume that the encoder gives us $o_v^{(0)} = h_v^{(0)}$ and $n_v = 0^{d_i}$ the zero vector. We now define the updates of flood, floodcross, echo, and echocross in a special way, that after the flood and echo part $o_v^{(k)}$ is equal to $h_v^{(k)}$ and $n_v^{(k)}$ is equal to $\sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}$. If this is ensured, the final update in a flood and echo phase can update $q_v^{(k)} = \text{MLP}^{(k)}((1+\epsilon)o_v^{(k-1)} + n_v^{(k-1)}) \mid\mid 0^{d_i}$, which exactly mimics the GIN update. It is easy to verify that if we set the echo and flood updates to add the full sum of the $o_v^{(k)}$ part of the incoming messages (and similarly half of the sum of the incoming messages during the cross updates) to $n_v^{(k-1)}$ the desired property is fulfilled. Moreover, there are at most four messages exchanged over each edge of the graph. Specifically, four is for cross edges and two is for all other edges. Therefore, a total of $\mathcal{O}(m)$ messages are exchanged, which is asymptotically the same number of messages GIN exchanges in a single update step. This enables a single phase of the Flood and Echo Net to mimic the execution of a single GIN round. Repeating this process the whole GIN computation can be simulated by the Flood and Echo Net.

Therefore, given a GIN network G_I of width d_i, we can construct a Flood and Echo Net G_F of width $\mathcal{O}(d)$ that can simulate one round of G_I in a single flood and echo phase using $\mathcal{O}(m)$ messages.

■

Proof of Theorem 2

To show that the Flood and Echo Net goes beyond 1-WL, it suffices to find two different graphs that are equivalent under the 1-WL test but can be distinguished by a Flood and Echo Net. Observe that a Flood and Echo Net can calculate its distance, in number of hops, to the root for each node. See the graphs illustrated in Figure 2 for a comparison. On the left is a cycle with 11 nodes, which have additional connections to the nodes that are at distance two away. Similarly, the graph on the right has additional connections at a distance of three. Both graphs are four regular and can, therefore, not be distinguished using the 1-WL test. However, no matter where the starting node for Flood and Echo is placed, it can distinguish that there are nodes which have distance four to the starting root in one graph, which is not the case in the other graph. Therefore, Flood and Echo Net can distinguish the two graphs and is more expressive than the 1-WL test. Moreover, due to the Theorem 1 it matches the expressiveness of the 1-WL test on connected graphs by a reduction to the graph isomorphism network. ■

Proof of Lemma 3 Consider either one of the Distance, or Path Finding tasks presented in Appendix F.1. All of them require information that is $\mathcal{O}(D)$ apart and must be exchanged according to Corollary 5. It follows that all MPNNs must execute at least $\mathcal{O}(D)$ rounds of message-passing to facilitate this information. Moreover, in these graphs, the graph diameter can be $\mathcal{O}(n)$. As in each round, there are $\mathcal{O}(m)$ messages exchanged, MPNNs must use at least $\mathcal{O}(nm)$ messages to solve these tasks. Furthermore, from Lemma **??**, it follows that Flood and Echo Net can solve the task in a single phase using $\mathcal{O}(m)$ messages. ■

Corollary 5 *Let D be the diameter of the graph. In order to correctly solve the Distance or Path-finding tasks, nodes require information that is $\mathcal{O}(D)$ hops away.*

Proof of Lemma 3 For the Distance and Path Finding tasks we outline the proof as follows: Assume for the sake of contradiction that this is not the case and only information has to be exchanged, which is $d' = o(D)$ hops away to solve the task. Therefore, as both tasks are node prediction tasks, the output of each node is defined by its d'-hop neighborhood. For both tasks, we construct a star-like graph G, which consists of a center node c and k paths of length $\frac{n}{k}$, which are connected to c for a constant k. For the Path Finding task, let the center c be one marked node, and the end of path j be the other marked node. Consider the nodes x_i, $i = 1, 2, ..., k$ which lie on the i-th path at distance $\frac{n}{2k}$ from c. Note that all x_i are $\frac{n}{2k}$ away from both their end of the path and c the root. Moreover, the diameter of the graph is $\frac{2n}{k}$. This means that neither the end of the i-th path nor the center c will ever be part of the d'hop neighborhood. Therefore, if we can only consider the d'-hop neighborhood for each x_i, they are all the same and as a consequence will predict the same solution. However, x_j lies on the path between the marked nodes while the other x_i's do not. So they should have different solutions, a contradiction. A similar argument holds for the Distance task. Again let c be the marked node in the graph and x_i for $i = 1, 2, ..., k$ be the nodes which lie on the i-th path at distance $\frac{n}{2k}$ for even i and $\frac{n}{2k} + 1$ for odd i. Again, note that the d'-hop neighborhood of all x_i is identical and therefore must compute the same solution. However, the solution of even x_i should be different from the odd x_i, a contradiction. ■

Appendix E. Algorithmic Tasks

We test performance on the different Flood and Echo Net modes: *fixed, random* and *all.* All modes execute two phases, which results in $\mathcal{O}(m)$ messages exchanged per chosen origin. Moreover, we choose the marked nodes in the tasks for the origin in the *fixed* mode. Note that the *all* mode, requires n executions, one for each node, therefore, we only consider it

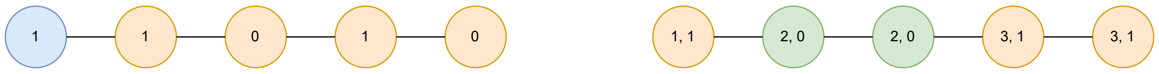

Figure 5: Example graph from the PrefixSum task. The left graph represents the input graph with a binary value associated with each node and the blue node being the starting node. The right graph represents the ground truth solution, each node contains two values the cumulative sum and the desired result which is the cumulative sum modulo 2.

for graphs of size at most one hundred. Nevertheless, the other modes can scale more easily and we believe them to be better suited for the study of algorithm learning. We choose GIN as a representative of a maximal expressive MPNN baseline which executes a fixed number of rounds. More precisely, five rounds are executed as the model begins to destabilize for more rounds. We also consider two recurrent baselines, which adapt the number of rounds according to the graph size. Therefore, we consider RecGNN Grötschla et al. (2022) and PGN (Veličković et al., 2020). We scale the number of rounds by $1.2n$, where n denotes the number of nodes in the graph.

Appendix F. Datasets

F.1. Algorithmic Datasets

For all the below tasks, we use train set, validation set, and test set sizes of 1024, 100, and 1000, respectively. The sizes of the respective graphs in the train, validation, and test sets are 10, 20, and 100. Performance on this test set demonstrates the model's ability to extrapolate to larger graph sizes. Note that many of the tasks only require the output modulo 2. We reduce the problem to this specific setting so that all numbers involved in the computation stay within the same range, as otherwise, the values have to be interpreted almost in a symbolic way, which is very challenging for learning-based models.

PrefixSum Task(Grötschla et al., 2022) Each graph in this dataset is a path graph where each node has a random binary label with one marked vertex at one end, which indicates the starting point. The objective of this task is to predict whether the PrefixSum from the marked node to the node in consideration is divisible by 2.

Distance Task (Grötschla et al., 2022) In this task every graph is a random graph of n nodes with a source node being distinctly marked. The objective of this task is to predict for each node whether its distance to the source node is divisible by 2.

Path Finding TaskGrötschla et al. (2022) In this task the dataset consists of random trees of n nodes with two distinct vertices being marked separately. The objective of this task is to predict for each node whether it belongs to the shortest path between the 2 marked nodes.

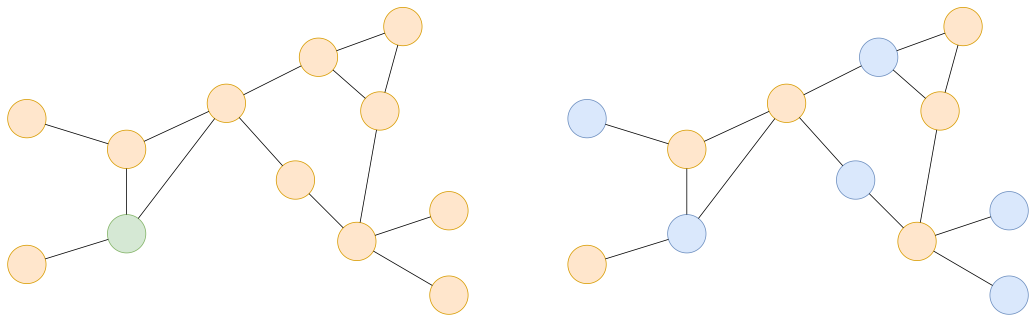

Figure 6: Example graph from the distance task. The green node in the left graph (input graph) represents the source node, and the remaining nodes are unmarked. On the right graph (ground truth) all orange nodes are at an odd distance away from the source while the blue nodes are at an even distance away from the source.

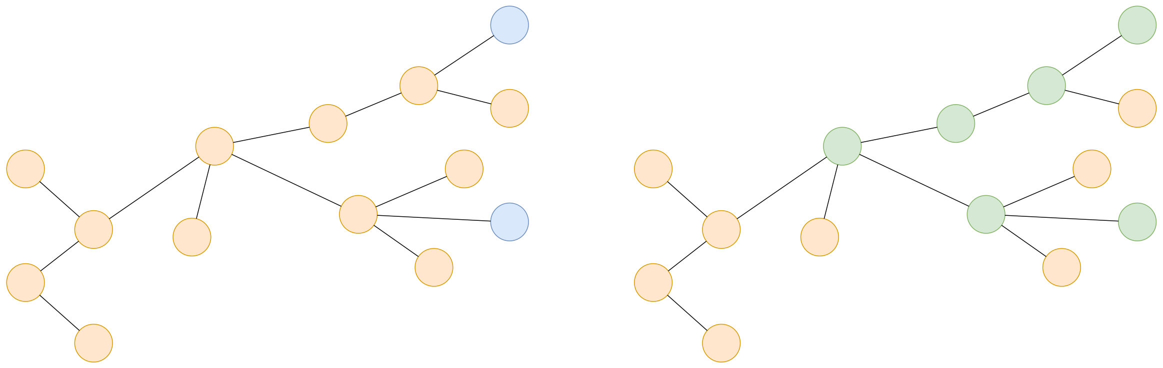

Figure 7: Example graph from the pathfinding task. The left graph represents the input graph, where the blue nodes are the marked nodes. The right is the corresponding solution, where the path between the marked nodes is highlighted in green.