Asynchronous Algorithmic Alignment with Cocycles

Andrew Dudzik [1] **Tamara von Glehn** [1] **Razvan Pascanu** [1] **Petar Veličković** [1]

Abstract

State-of-the-art neural algorithmic reasoners make use of message passing in graph neural networks (GNNs). But typical GNNs blur the distinction between the definition and invocation of the message function, forcing a node to send messages to its neighbours at every layer, synchronously. When applying GNNs to learn to execute dynamic programming algorithms, however, on most steps only a handful of the nodes would have meaningful updates to send. One, hence, runs the risk of inefficiencies by sending too much irrelevant data across the graph—with many intermediate GNN steps having to learn identity functions. In this work, we explicitly separate the concepts of node state update and message function invocation. With this separation, we obtain a mathematical formulation that allows us to reason about asynchronous computation in both algorithms and neural networks.

1. Introduction

The message passing primitive—performing computation by aggregating information sent between neighbouring entities (Gilmer et al., 2017)—is known to be remarkably powerful. All the neural network architectures discussed within geometric deep learning (Bronstein et al., 2021), and especially graph neural networks (GNNs), can be elegantly expressed with message passing.

An active area of message passing research is *neural algorithmic reasoning* (Veličković & Blundell, 2021, NAR). NAR seeks to capture *classical computation* in neural nets, largely by learning to execute (Veličković et al., 2022). The use of GNNs in NAR is largely due to the theory of *algorithmic alignment* (Xu et al., 2019): as we increase the structural similarity between a neural network and an algorithm, it will be able to learn to execute this algorithm with improved sample complexity. Recently, multiple theoretical works (Xu et al., 2020; Dudzik & Veličković, 2022) demonstrated that message passing aligns with *dynamic programming* (Bellman, 1966, DP). This makes GNNs attractive in NAR, as DP offers a generic framework for *problem-solving*.

In this work, we make novel contributions to algorithmic alignment theory. We "zoom in" on the analysis of Dudzik & Veličković (2022), assuming a *node-centric* view: analysing the computations happening around individual nodes in the graph, in isolation. This perspective allows us to make an important observation: in all previously studied GNN architectures, it was implicitly assumed that all the messages in a given GNN layer were passed all-at-once and *synchronously* aggregated in each receiver node. This significantly influenced previous efforts to align GNNs and algorithms: most popular NAR benchmarks (Veličković et al., 2022) forcibly express target algorithms in a synchronous format.

But, while some algorithms may neatly fit within this paradigm, it is well-known that many algorithms tend to modify only *a small fraction* of the nodes at each atomic step. Clearly, for most forms of classical computation, it is unrealistic to expect us to be able to meaningfully update all of the nodes at once—in many cases, synchronised updates to individual nodes will mostly amount to identity functions.

Accordingly, in this work, *we explore the theoretical implications of making GNNs* ***asynchronous*** (Figure 1). We demonstrate how studying message passing under various synchronisation regimes can help us identify choices of message functions that better align with target algorithms, in a manner that was not possible previously. For example, our theory justifies the effectiveness of architectures such as PathGNNs (Tang et al., 2020) at executing the Bellman-Ford algorithm (Bellman, 1958). We refer to our framework as *asynchronous algorithmic alignment*, and formalise it using category theory, monoid actions, and cocycles.

2. Message passing

We define GNNs as in Bronstein et al. (2021). Let a graph be a tuple of *nodes* and *edges*, $G = (V, E)$, with one-hop neighbourhoods defined as $\mathcal{N}_u = \{v \in V \mid (v, u) \in E\}$. A

[1]Google DeepMind. Correspondence to: Andrew Dudzik <adudzik@deepmind.com>, Petar Veličković <petarv@deepmind.com>.

Presented at the 2^{nd} Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023.

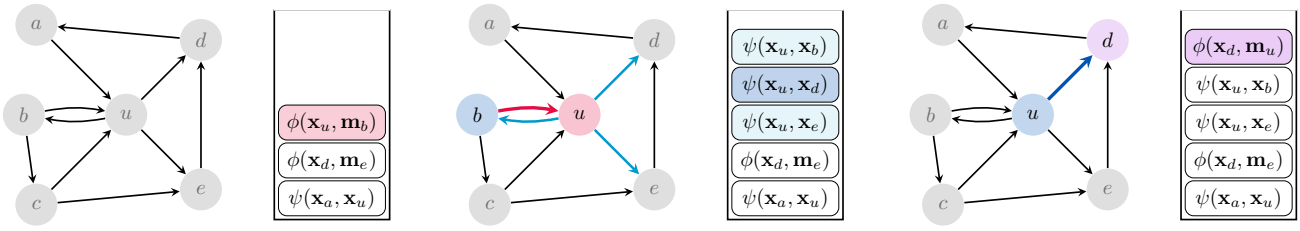
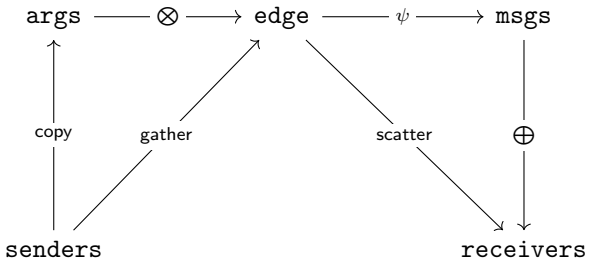


Figure 1. While traditional GNNs send and receive all messages synchronously at every step, under our model, at any step the GNN may choose to execute any number of possible operations (here these are depicted as organised in a collection, to the right of the graph). Here we demonstrate a specific asynchronous GNN execution trace. **Left-to-right:** We first choose to update the features of node u using the message sent from node b. This triggers a request for messages to be generated to all of u's neighbours (b, d, e). In the next step, we choose to compute the message sent from u to d. As a result, a new update for d can be performed, using the just-computed message.

node feature matrix $\mathbf{X} \in \mathbb{R}^{|V| \times k}$ gives the features of node u as $\mathbf{x}_u$; we omit edge- and graph-level features for clarity. A *(message passing)* GNN over this graph is executed as:

$$\mathbf{x}'_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right) \tag{1}$$

To put this equation in more abstract terms, we start by briefly reviewing the diagram of the message-passing framework of Dudzik & Veličković (2022), with the addition of the message function ψ to emphasise the symmetry:

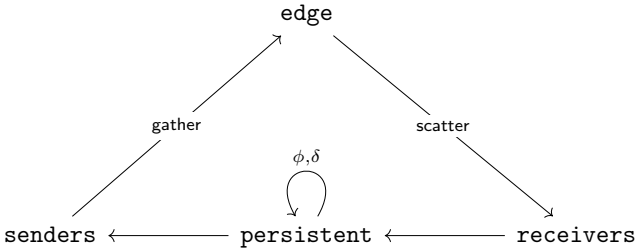


First, sender features (`senders`) are duplicated along outgoing edges to form the arguments (`args`) to a message function ψ. These arguments are collected into a list using the $\bigotimes$ operator—which is traditionally a concatenation, though it can be any operator with a monoidal structure. This list of arguments now lives on a new, transient, `edge` datatype. These two steps constitute a *gather* operation. In Equation 1, this corresponds to copying the node features in $\mathbf{X}$, considered as a V-indexed tensor, to obtain feature pairs $(\mathbf{x}_u, \mathbf{x}_v)$, considered as an E-indexed tensor.

Next, we perform a similar operation, a *scatter*, by first applying the message function ψ to the arguments, which computes the messages to be sent (`msgs`). Then, messages are copied to suitable receivers, which aggregate along their incoming edges to form the final set of receiver node features (`receivers`). In Equation 1, this refers to the application of the message function ψ, and the aggregation $\bigoplus$.

The gather-scatter paradigm is very common in message passing implementations. However, we still need to clarify the role of the *update function*, ϕ, which uses the aggregated receiver features $\bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)$ to produce the next step's senders. Hence, our main interest is now: How can a node convert received messages back into sendable arguments, so the overall computation steps can be chained?

As we make assume edge messages are *transient*, we do not need to store them across several iterations of Equation 1. Conversely, nodes make use of a *persistent* state which gets updated at each layer. It is the interaction with this state that allows nodes to perform nontrivial functions in interesting message passing schemes. We capture the stateful nature of this computation in the following complementary diagram:



We make two assumptions: (1) the sender features should have the structure of a *commutative monoid*. This implies that we can think of the message function arguments being sent in chunks, which may be assembled in any order; (2) the receiver features should have the structure of a *monoid*, not necessarily commutative. We can think of these received messages as being *instructions* which transform the state. The monoid operation *composes* these transformations.

3. Node-centric view on algorithmic alignment

We now focus our attention on a single node, and explore the relationship between the message monoid, $(M, \cdot, 1)$, and the argument commutative monoid, $(A, +, 0)$.

Suppose that the internal state takes values in a set S. The

process by which a received message updates the state and produces an argument is described by a function $M \times S \to S \times A$. This is equivalent, by currying, to a Kleisli arrow $M \to [S, S \times A] = \texttt{state}_S(A)$ for the state monad.

Given a pair (m, s), we denote the image under this function by $(m \bullet s, \delta_m(s))$, where $\bullet : M \times S \to S$ is written as a binary operation and $\delta : M \times S \to A$ is given by some argument function. First, we look into the properties of $\bullet$.

Each incoming message (an element of M) transforms the state (an element of S) in some way. We assume that the multiplication of M corresponds to a composition of these transformations. Specifically, we ask that $\bullet$ satisfies the unit and associativity axioms:

$$1 \bullet s = s \qquad (n \cdot m) \bullet s = n \bullet (m \bullet s) \tag{2}$$

Next, we interpret Equation 2 in terms of the argument function δ. In the first equation, the action $1 \bullet s$ generates an argument $\delta_1(s)$. But on the right-hand side there is no action, so no argument is produced. In order to process both sides consistently, $\delta_1(s)$ must be the zero argument.

Similarly, in the second equation, the left-hand side produces one argument $\delta_{n \cdot m}(s)$, while the right-hand side produces two, $\delta_m(s)$ and $\delta_n(m \bullet s)$. Setting these equal, we have the following two argument axioms:

$$\delta_1(s) = 0 \qquad \delta_{n \cdot m}(s) = \delta_m(s) + \delta_n(m \bullet s) \tag{3}$$

Equation 3 can be related to a rich mathematical concept: *derivations*, also known as *1-cocycles*. Please see Appendix B for details.

4. (A)synchrony in GNNs

With the preliminaries ironed out, we can now leverage the "cocycle conditions" of Equation 3 to more rigorously discuss the synchronisation of GNN operations (such as gathers and scatters). First, we can explicitly formalise the residual map ϕ in Equation 1: it is just another description of what we have called an "action". That is, we have $\phi(s, m) = m \bullet s$ for all node features s and (aggregated) non-null messages m.

Message aggregation asynchrony. $\bigoplus$ is usually taken, axiomatically, to be the operation of a commutative monoid (Ong & Veličković, 2022). This already allows us to support a certain form of asynchrony: we can aggregate messages online as we receive them, rather than waiting for all of them before triggering $\bigoplus$, due to permutation invariance.

Node update asynchrony. Similarly, the axiom that ϕ defines an associative operation, as in Equation 2, corresponds to another type of asynchrony. When ϕ satisfies:

$$\phi(s, m \bigoplus n) = \phi(\phi(s, m), n) \tag{4}$$

this tells us that ϕ itself can be applied asynchronously. Put differently, after each message arrives into the receiver node, we can use it to update the node features by triggering ϕ, without waiting for the messages to be fully aggregated. One common way to enforce associativity is to take $\phi = \bigoplus$, in which case ϕ inherits the associativity properties of $\bigoplus$.

Argument generation asynchrony. Now we focus on the 1-cocycle condition (Equations 3). This concerns the argument function δ, which determines which arguments are produced after a node update. Specifically, the cocycle condition allows us to express the arguments produced by receiving two messages together ($\delta_{n \cdot m}$) as a combination of the arguments produced by receiving them in isolation (δ_n and δ_m). Therefore, it gives us a mechanism that allows for each sender node to prepare their arguments to the message function, ψ, asynchronously, rather than waiting for all the relevant node updates to complete first.

Note that Equation 1 does not distinguish between node features and sent arguments. In other words, it implicitly defines the argument function $\delta_m(s) = m \bullet s = \phi(s, m)$. Accordingly, we explore some conditions under which the update function ϕ will satisfy the cocycle condition. For this, we prove the following useful statement:

We say that A is *idempotent* if $a + a = a$ for all $a \in A$.

Proposition 4.1. *Suppose that $S = A$. Define $\delta_m(s) = m \bullet s$ if $m \neq 1$, and 0 otherwise. If δ is a 1-cocycle, then A is idempotent. If $M = S = A$ and $\cdot = +$, the converse holds.*

Proof. In this case, the second equation of Equation 3 becomes $(n \cdot m) \bullet s = m \cdot s + n \bullet (m \bullet s)$. Setting $n = m = 1$ gives $s = s + s$. If $\cdot = +$ and A is idempotent, then the equation is $n+m+s = m+s+n+m+s$, which holds since $m+s+n+m+s = (m+s)+n+(m+s) = m+s+n$. □

As we already set $\phi = \bigoplus$ previously, we can use Proposition 4.1 to conclude it is sufficient to show $\bigoplus$ is idempotent, to satisfy the cocycle condition. Not all commutative monoids are idempotent; $\bigoplus = \max$ is idempotent, while other aggregators, like sum, are not. Note that this aligns with the utility of the $\max$ aggregation in algorithmic tasks, as observed by prior works (Xu et al., 2020).

We can observe that Equation 1 now looks as follows:

$$\mathbf{x}'_u = \max\left(\mathbf{x}_u, \max_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right) \tag{5}$$

Such a $\max$-$\max$ GNN variant allows for a high level of *asynchrony*: messages can be sent, received and processed in an arbitrary order, arguments can be prepared in an online fashion, and it is mathematically guaranteed that the outcome will be identical as if we fully synchronise all of these steps, as is the case in typical GNN implementations.

The only remaining point of synchronisation is the invocation of the message function, ψ; messages can only be generated once all of the arguments for the message function are fully prepared (i.e., no invocations of ϕ are left to perform for the sender nodes). Under additional constraints on ψ, even this can be made asynchronous, as we now show.

We can remark that Equation 5 is almost exactly equal to the hard PathGNN model from Tang et al. (2020). The only missing aspect is to remove the dependence of ψ on the receiver node (i.e., to remove u from `senders`), as follows[1]:

$$\mathbf{x}'_u = \max\left(\mathbf{x}_u, \max_{v \in \mathcal{N}_u} \psi(\mathbf{x}_v)\right) \tag{6}$$

This modelling choice is very useful—in fact, it will allow us to rigorously discuss when can such a model reach *full asynchrony*; that is, where we do not even need to wait on the argument to ψ to be fully prepared by previous steps.

Message generation asynchrony. PathGNNs are an example of *isotropic* message passing, where the message function, $\psi : A \to M$, is a function of a single variable, the sender argument, and produces a single receiver message.

We say that an isotropic ψ is a *monoid homomorphism* if it satisfies the following two properties:

$$\psi(0) = 1 \qquad \psi(a + b) = \psi(a) \cdot \psi(b) \tag{7}$$

The first property of Equations 7 merely states that no message is produced (the "null message", 1) if no argument is prepared (the "null argument", 0). The second says exactly that, given two arguments, aggregating them and then applying ψ is the same as applying ψ on each of them first, then aggregating the corresponding produced messages.

This is exactly the condition needed for argument asynchrony and message asynchrony to be compatible. It means that ψ can be called—and messages generated—even before its arguments are fully ready, so long as it is called again each time the arguments are updated.

Note that PathGNNs, in their default formulation, do not always satisfy this constraint. We have, therefore, used our analysis to find a way to extend PathGNNs to a *fully asynchronous* model. One way to obtain such a GNN is to make ψ be a *tropical linear* transformation. That is, ψ would be parametrised by a $k \times k$ matrix, which is multiplied with $\mathbf{x}_v$, but replacing "$+$" with $\max$ and "$\cdot$" with $+$.

Originally, PathGNN was designed to align with the Bellman-Ford algorithm (Bellman, 1958), due to its claimed structural similarity to the algorithm's operation—though this claim was not rigorously established. Now, using our mathematical framework, we can rigorously conclude where this alignment comes from: the choice of aggregator ($\max$) and making ψ only dependent on one sender node is fully aligned with the Bellman-Ford algorithm (as in Xu et al. (2020)), *and* both PathGNNs and Bellman-Ford can be implemented fully asynchronously without any errors in the final output. We have already showed that PathGNNs satisfy the cocycle condition; in Appendix A, we prove the same statement about Bellman-Ford, completing our argument.

[1] PathGNNs may have access to *edge features*. They are assumed fixed, so we can consider them "embedded" within ψ.

References

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

Bellman, R. Dynamic programming. *Science*, 153(3731): 34–37, 1966.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999. PMLR, 2016.

Dudzik, A. and Veličković, P. Graph neural networks are dynamic programmers, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Ong, E. and Veličković, P. Learnable commutative monoids for graph neural networks. *arXiv preprint arXiv:2212.08541*, 2022.

Tang, H., Huang, Z., Gu, J., Lu, B.-L., and Su, H. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020.

Veličković, P. and Blundell, C. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pp. 22084–22102. PMLR, 2022.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020.