A. Examples

We present two worked examples of classical algorithms (Bellman-Ford and addition with carry) that naturally and provably lend themselves to corresponding cocycles.

A.1. Semilattices and Bellman-Ford

Let P be any join-semilattice, i.e. a poset with all finite joins, including the empty join, which we denote 0 as it is a lower bound for P. A standard result says that P is equivalently a commutative, idempotent monoid $(P, \vee, 0)$. When P is totally ordered, we usually write $\vee = \sup$ or $\max$.

In Bellman-Ford, P will generally be either the set of all path lengths, (taken with a negative sign) or the set of all relevant paths ending at the current vertex, equipped with a total order that disambiguates between paths of equal length.

We set $M = S = A = P$, with action of M on S given by $m \cdot s := m \vee s$.

Since we want to inform our neighbors when our state improves, i.e. gets smaller, we could likewise define our argument function δ by $\delta_m(s) := m \vee s$, as this in fact satisfies the cocycle equation due to Proposition 4.1. However, this will lead to an algorithm that never terminates, as new redundant arguments will continue to be generated at each step.

So we define the argument function a bit more delicately:[2]

$$\delta_m(s) := \begin{cases} 0 & \text{if } m \leq s \\ m \vee s & \text{otherwise} \end{cases}$$

Proposition A.1. *δ is a 1-cocycle $M \to [S, A]$.*

Proof. Pick $m, n \in M$, $s \in S$. We wish to show that $\delta_{m \vee n}(s) = \delta_m(n \vee s) \vee \delta_n(s)$.

If $m, n \leq s$ then both sides equal 0. If $m \leq s$ but $n \not\leq s$ then the LHS is $m \vee n \vee s = n \vee s$ while the RHS is $0 \vee (n \vee s)$. If $n \leq s$ but $m \not\leq s$ then the LHS is $m \vee n \vee s = m \vee s$ while the RHS is $(m \vee n \vee s) \vee 0 = m \vee s$. □

Now, we can see that the definition of δ prevents new arguments from being generated once the optimal value has been obtained, if we take the convention that zero arguments are ignored. In particular, if P is well-ordered, only finitely many arguments can be generated regardless of the input.

Lastly, note that Bellman-Ford is also a perfect example of message generation asynchrony; its ψ_e, for each edge, simply adds the edge length of e to the sender node distance. $+$ is easily seen to distribute over $\max$, making ψ_e a monoid homomorphism. All conditions combined, Bellman-Ford can be indeed made fully asynchronous, without sacrificing the fidelity of the final output. This solidifies the algorithmic alignment of our PathGNN variant with Bellman-Ford.

A.2. The natural numbers and addition with carry

If $M = (\mathbb{N}, +, 0)$ is the monoid of natural numbers under addition,[3] then an action of M on a set S is equivalently an endofunction $\pi : S \to S$, with $m \cdot s := \pi^m(s)$.

The following proposition gives a characterisation of all possibly cocycles $M \to [S, A]$.

Proposition A.2. *Given any function $\omega : S \to A$, there is a unique 1-cocycle δ with $\delta_1 = \omega$.*

Proof. If δ is a cocycle and $\omega = \delta_1$, the cocycle condition gives us an inductive definition of δ:

$$\delta_{n+1}(s) = \delta_n(s) + \delta_1(s + n)$$

On the other hand, given ω we can define:

[2] If P has more structure, say if it is a lattice with relative complements, then we can improve on this equation further by sending the complement argument $m \setminus s$.

[3] Note that we are using additive notation here, so the identity element is denoted 0 instead of 1.

$$\delta_n(s) := \sum_{i=0}^{n-1} \omega(s+i)$$

We note that $\delta_{m+n} = \sum_{i=0}^{m-1} \omega(s+i) + \sum_{j=m}^{m+n-1} \omega(s+j) = \sum_{i=0}^{m-1} \omega(s+i) + \sum_{j=0}^{n-1} \omega(m+s+j) = \delta_m(s) + \delta_n(m+s)$ so δ is a 1-cocycle. □

Now, we consider the example of digit arithmetic with carries. Suppose that $S = \{0, \cdots, 9\}$ is the set of digits in base 10, and our action is given by the permutation $\pi(s) = \overline{s+1}$, where the bar denotes reduction modulo 10.

If $M = (\mathbb{N}, +)$, we can define an argument function based on the number of carries produced when 1 is added to s m times:

$$\delta_m(s) := \left\lfloor \frac{m+s}{10} \right\rfloor$$

Proposition A.3. *δ is a 1-cocycle $M \to [S, A]$.*

Proof. We directly verify that, for all $m, n \in \mathbb{N}$, $s \in \{0, \ldots, 9\}$:[4]

$$\left\lfloor \frac{m+n+s}{10} \right\rfloor = \left\lfloor \frac{m+\overline{n+s}}{10} \right\rfloor + \left\lfloor \frac{n+s}{10} \right\rfloor$$

This follows quickly by induction on m: for $m = 0$ the two sides are equal, and as m increments by 1, the LHS and RHS are both incremented if and only if $m + n + s + 1 \equiv 0 \pmod{10}$. □

B. Argument functions as cocycles

First, consider the set $F = [S, A]$ of *readout functions*; functions mapping states to corresponding arguments. F inherits structure from both S and A. First, F is a commutative monoid because A is; we define a zero function $0(s) := 0$ and function addition $(f+g)(s) := f(s) + g(s)$.

Second, F has an action of M, given by $(f \bullet m)(s) := f(m \bullet s)$. Importantly, this is a right action rather than a left action; the associativity axiom $f \bullet (m \cdot n) = (f \bullet m) \bullet n$ holds but the reversed axiom fails unless M is commutative.

With these definitions, if we write δ in its curried form $D : M \to [S, A]$, we can rewrite Equation 3 as follows:

$$D(1) = 0 \qquad\qquad D(n \cdot m) = D(m) + D(n) \bullet m \tag{8}$$

Equation 8 specify that D is a *1-cocycle*, otherwise known as a *derivation*. To understand the latter term, consider the more general situation where F also has a left action of M. Then we may write $D(n \cdot m) = n \bullet D(m) + D(n) \bullet m$, which is just the Leibniz rule for the derivative. Our equation describes the special case where this left action is trivial.

Note that group cocycles are most commonly defined in terms of left actions. To convert between the two descriptions, we can replace M by its opposite M^{op}, defined to be the monoid with the same underlying set as M, but reversed multiplication. Since inversion gives an isomorphism of any group with its opposite, it's a standard technique to let a group act on the left on functions by $(gf)(s) := f(g^{-1}s)$. This technique is heavily leveraged in group convolutional neural networks (Cohen & Welling, 2016).

C. Asynchronous message functions

In general, we think of the message function ψ as being a point of synchronisation, which blocks until it has the final values for each of its arguments.

However, under certain conditions ψ may return a partial result. Consider the special case of *isotropic* message passing, where the message function for each edge, ψ_e, is a function of a single variable, the sender argument, and produces a single value, a receiver message. That is, we have a function $\psi_e : (A_{s(e)}, +, 0) \to (M_{t(e)}, \cdot, 1)$.

[4]We could instead verify that δ can be put in the form of the previous proposition, but this amounts to the same proof.

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

$$\begin{aligned} \psi(0) &= 1 \\ \psi(a+b) &= \psi(a) \cdot \psi(b) \end{aligned} \tag{9}$$

We observe that Equation 9 says exactly that, given two arguments, aggregating and then applying ψ is the same as applying ψ first, then aggregating.

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

Note that Bellman-Ford is a perfect example of this type of asynchrony; ψ_e simply adds the edge weight of e. $+$ is easily seen to distribute over $\max$, making ψ_e a monoid homomorphism.

In the non-isotropic case, where ψ may take multiple arguments, different properties of ψ may correspond to different forms of potential asynchrony. Since various DP algorithms can be parallelised in many different ways, this seems like a promising avenue for exploration.

D. Attention as synchronised message passing

Consider message passing along edges whose weights are determined by a Transformer-style self-attention mechanism. For example, suppose that nodes a, b, c send to node u, where the values $\mathbf{v}_a, \mathbf{v}_b, \mathbf{v}_c$ are multiplied by attention coefficients $(\alpha_a, \alpha_b, \alpha_c) = \mathrm{softmax}(\mathbf{q}_u^\top \mathbf{k}_a, \mathbf{q}_u^\top \mathbf{k}_b, \mathbf{q}_u^\top \mathbf{k}_c)$, to give final messages of $\alpha_a \mathbf{v}_a, \alpha_b \mathbf{v}_b, \alpha_c \mathbf{v}_c$.

At first glance, this looks incompatible with the notion of asynchronous message aggregation, since the normalisation step of the softmax means we have to compute all the messages simultaneously.

To address this, we may introduce a synchronisation step via a multivariate message function:

$$\begin{gathered} \psi_u(\mathbf{q}_u, \mathbf{v}_a, \mathbf{k}_a, \mathbf{v}_b, \mathbf{k}_b, \mathbf{v}_c, \mathbf{k}_c) := \\ \mathrm{softmax}(\mathbf{q}_u^\top \mathbf{k}_a, \mathbf{q}_u^\top \mathbf{k}_b, \mathbf{q}_u^\top \mathbf{k}_c) \odot (\mathbf{v}_a, \mathbf{v}_b, \mathbf{v}_c) \end{gathered} \tag{10}$$

where $\odot$ is an elementwise product.

Hence, we have a single message function whose arguments come from all four nodes, and whose output maps to the three incoming edges. Since ψ blocks on waiting for the entire computation to finish, once the messages are produced we are free to aggregate them asynchronously as usual.