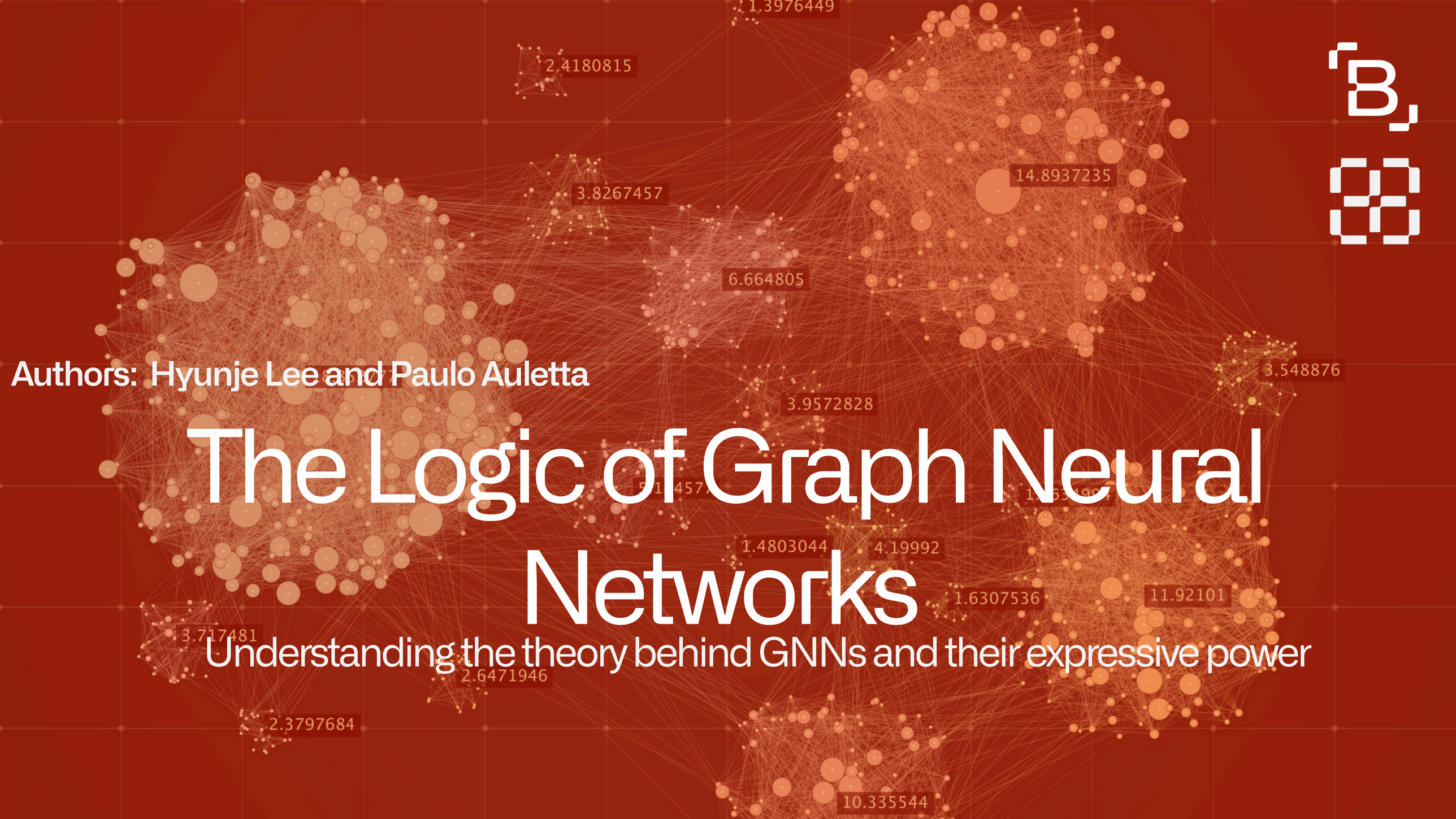


Authors: Hyunje Lee and Paulo Auletta

The Logic of Graph Neural Networks

Understanding the theory behind GNNs and their expressive power

Set, logics and notations

First-Order Logic (FO)

- Think of first-order logic as a formal language to describe properties of graphs.
  - Variables representing nodes.
  - Predicates like: E(x, y)
  - Logical connectives: $\wedge$, $\vee$, $\neg$
  - Quantifiers: $\exists x$, $\forall x$

$$\exists x \exists y \, E(x, y) \quad \Rightarrow \text{“there exists an edge”}$$

Counting Logic (C)

- C is like FO, but it adds counting quantifier, written

$$\exists_{\geq p} x\, \varphi(x) \quad \Rightarrow \quad \text{“there exist at least } p \text{ nodes satisfying } \varphi(x)$$

- Example

$$\exists_{\geq 3} y\, E(x, y) \quad \Rightarrow \text{“x has at least 3 neighbors”}$$

Finite Variable Logic

- C_k means you can only use k variables total:
  - C_2 → Use only x and y
  - C_3 → Use only x, y and z
- GC_2, is like C_2 but slightly restricted: You can only quantify over neighbours of the current node.
- $C_k^{(\varphi)}$ → formulas with ≤ k variables and quantifier depth ≤ φ

  This depth directly corresponds to φ rounds of WL/GNN message passing, so the logic mirrors how GNNs iteratively aggregate neighborhood information.

Graph theory

Col(G,v) and isomorphism

- An **isomorphism** from a graph G to a graph G′ is a bijective mapping $f : V(G) \to V(G')$ that preserves edges as well as labels

- We sometimes refer to the vector

$$\mathrm{col}(G, v) := (c_1, \ldots, c_\ell) \in \{0, 1\}^\ell$$

with ci = 1 if $v \in$ Pi(G) and ci = 0 otherwise as the colour of vertex $v \in V(G)$.

- $ww \in E(G) \iff f(v)f(w) \in E(G')$ for all $v, w \in V(G)$ and $v \in \mathrm{Pi}(G) \iff f(v) \in \mathrm{Pi}(G')$ for all $i \in [\ell]$ and $v \in V(G)$.

k-ary Invariants

- A graph invariant (or 0-ary graph invariant) is a function ξ defined on graphs such that ξ(G) = ξ(G′) for isomorphic graphs G,G′.

- For k ≥ 1, a **k-ary invariant** is a function ξ that associates with each graph G a function ξ(G) defined on $V(G)^k$ in such a way that for all graphs G,G′, all isomorphisms f from G to G′, and all tuples v $\in V(G)^k$ it holds that

$$\xi(G)(v) = \xi\big(f(G)\big)\big(f(v)\big)$$

Formally, such a mapping ξ is called **equivariant**.

- eg) the number of vertices is a graph(0-ary) invariant, the degree of a vertex is a vertex invariant(1-ary)

Distinguishing and Completeness

$$If\ \xi(G,v) \neq \xi(G',v'),$$

$$then\ \xi\ distinguishes\ (G,v)\ and\ (G',v').$$

- equivariance condition implies that if ξ distinguishes (G,v) and (G′,v′) then there is no isomorphism from G to G′ that maps v to v′.

- If the converse also holds,(that is xi(G,v)=xi(G',v') implies that there exists an isomorphism) then ξ is a **complete invariant**.

- $$An\ invariant\ \xi\ is\ complete$$
$$if\ the\ reverse\ direction\ also\ holds:$$
$$\xi(G,v) = \xi(G',v') \iff \exists\ \text{isomorphism}\ f: G \to G'\ \text{with}\ f(v) = v$$
$$\xi\ \text{captures}\ exactly\ \text{all isomorphism information.}$$

Multiset $\hat{\xi}(G)$

- From a k-ary invariant ξ we can derive a graph invariant ξ^ by mapping each graph G to the multiset

$$\hat{\xi}(G) := \{\, \xi(G, v) \mid v \in V(G)^k \,\}$$

- it is a multiset of all ξ-values over all k-tuples. It depends only on the isomorphism class of G because it is an equivariant.

$$\xi \text{ distinguishes two graphs } G, G' \text{ if } \hat{\xi}(G) \neq \hat{\xi}(G').$$

- If ξ is a complete invariant and ξ^(G)=ξ^(G') then ξ^ is complete as well.

Atomic Type $\mathrm{atp}_k(G, v)$

- atp_k is a k-ary invariant such that for a graph G with ℓ labels and a tuple v = (v1,...,vk) ∈ V(G),

$$\mathrm{atp}_k(G, v) \in \{0, 1\}^{2\binom{k}{2} + k\ell}$$

- the vector which for 1 ≤ i < j ≤ k has two entries indicating whether vi = vj and whether vivj ∈ E(G) and for 1 ≤ i ≤ k has ℓ entries indicating whether vi is in P1(G),...,Pℓ(G).

- atp_k(G,v) = atp_k(G′,v′) if and only if the mapping vi → v′i is an isomorphism from the induced subgraph G[{v1,...,vk}] to the induced subgraph G′[{v′1,...,v′k}].

Colourings

atp1 = col (vertex colouring)

- the range of χ is not necessarily{red,blue,green,etc...} it can be R or even R^n in which case we call it **n dimensional feature map**.

- We often think of mappings χ defined on V(G) or $V(G)^k$ as colourings of the vertices or k-tuples of vertices of a graph G

- elements of the image of χ: colors pre-images $\chi^{-1}(c)$:color classes

Refinement

- χ, χ' are equivalent (we write $\chi \equiv \chi'$) $\chi \preceq \chi'$ and $\chi' \preceq \chi$.

- For colourings χ,χ′ : V (G)^k → C, we say that χ refines χ′ for all v,w ∈ V(G)^->C if χ(v) = χ(w) then χ′(v) = χ′(w).

- For k-ary invariants ξ,ξ′ we say that ξ refines ξ′ if ξ(G) refines ξ′(G) for every graph G, and similarly for equivalence

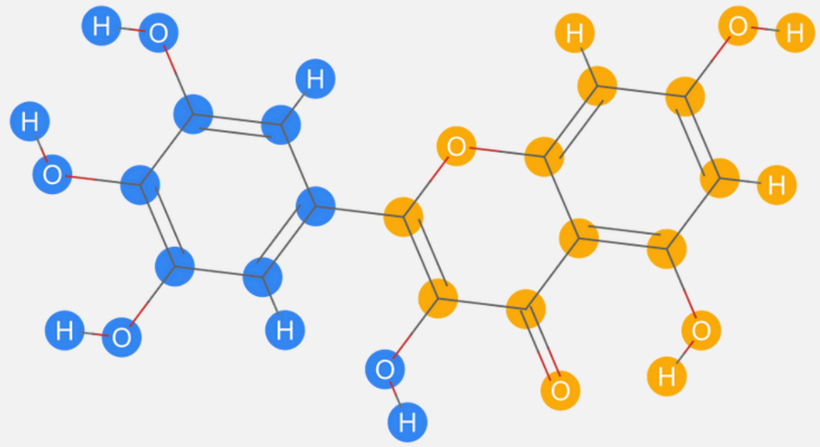
Graph Neural Networks (GNNs)

Why Graphs Matter?

- Graphs show up everywhere:
  - Social networks (users + friendships)
  - Molecules (atoms + bonds)
  - Web pages (links)
  - Transportation systems (stations + routes)

- Unlike images or text, graphs have no fixed shape or order

  We need models that can handle arbitrary structures and permutation invariance



What are GNNs?

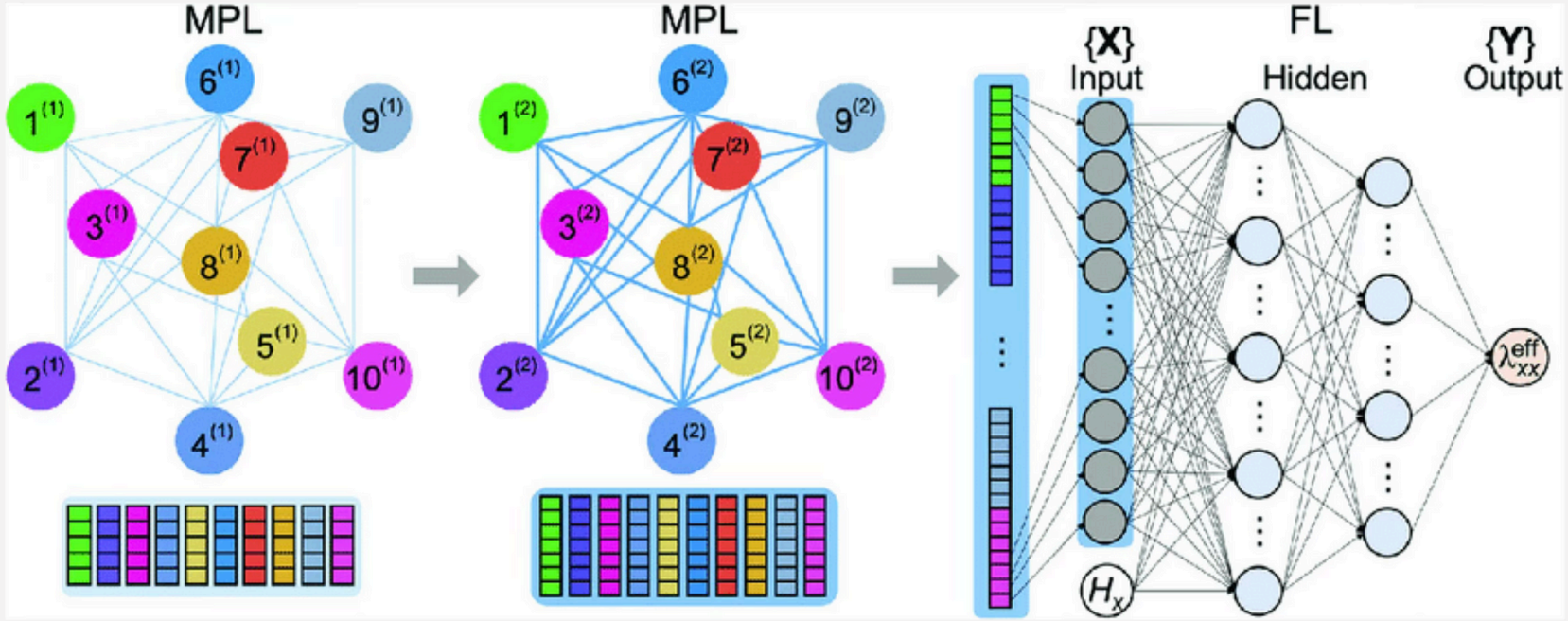
- Graphs don’t have a natural order. A GNN can process any of them because it looks only at how nodes are connected, not their order or layout.

- So a GNN is built to:
  - Handle arbitrary graph structures
  - Treat nodes that play the same structural role in a graph equivalently
  - Produce representations (vectors) for nodes or the whole graph

How GNNs Work

1. Each node starts with some initial feature vector
   - The initial feature map represents the coloring of the input graph

$$h^0 = \zeta^{(0)} := \mathrm{col}(G)$$



Feature Map

- A feature map is simply a function that assigns a feature vector to each node in the graph.

$$\zeta : V(G) \rightarrow \mathbb{R}^p$$

  - $V(G)$ The set of vertices (nodes) in graph G
  - $\mathbb{R}^p$ The space of p-dimensional real vectors
  - So for every node v, $\zeta(v)$ is a p-dimensional feature vector

Feature Map

Isomorphism invariance

- If two graphs G and G' are isomorphic via a mapping f, and their initial feature maps satisfy ζ(v) = ζ'(f(v)), then after applying any GNN layer L, their new features remain matched:

$$\eta(v) = \eta'(f(v))$$

How GNNs Work

2. Then, in each layer, the node updates its state by
   - Aggregating information from its neighbours
   - Combining that information with its own current state

Intuitively

$$h_v^{(t+1)} = \mathrm{COMBINE}\Big(h_v^{(t)}, \mathrm{AGGREGATE}\big(\{h_u^{(t)} : u \in N(v)\}\big)\Big)$$

Formally

$$h_v^{(t+1)} = \mathrm{ReLU}\Big(W_{\text{self}} h_v^{(t)} + \sum_{u \in N(v)} W_{\text{msg}} h_u^{(t)}\Big)$$

How GNNs Work

- 3. After several layers, nodes have gathered information from farther and farther away in the graph.

- 4. In the end, this feature map, we have obtained gets used as the input for a MLP, (“readout” or “decoder”) which will:
  - outputs node-level predictions,
  - or graph-level predictions,
  - or edge-level predictions,

Recurrent GNNs

- Instead of stacking multiple distinct layers, a recurrent GNN reuses the same layer repeatedly

$$\zeta^{(t+1)} = L(G, \zeta^{(t)})$$

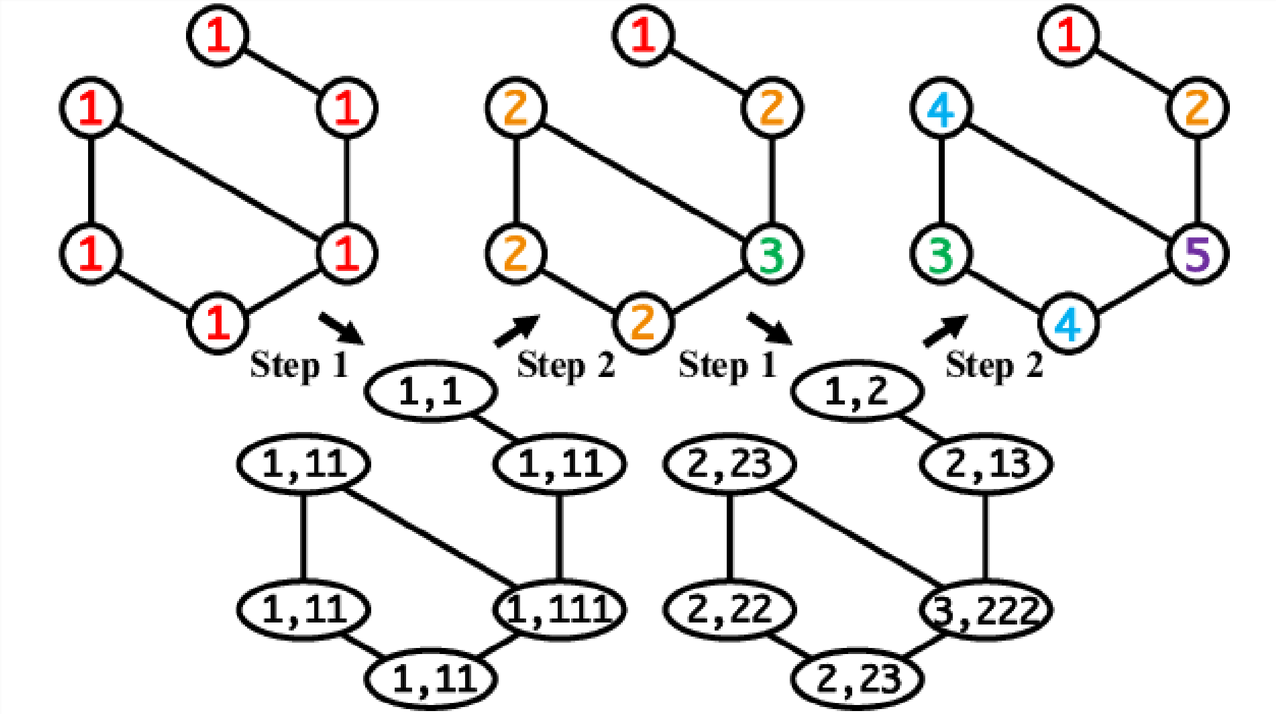
- This defines an infinite sequence of feature maps $\zeta^{(0)}, \zeta^{(1)}, \zeta^{(2)}, \ldots$, where $\zeta(t+1)$ is obtained by applying the GNN layer to $(G,\zeta(t))$.

The Weisfeiler-Leman Algorithm (WL)

Basic idea

- we change the colours of each vertex each step based on the number of its colour, number of neighbours and neighbours' colours. And there is some number that after this number the vertex colours don't change anymore.

- The initial colouring given by the vertex labels in a graph is repeatedly refined by counting, for each colour, the number of neighbours of that colour.



Step 1: generate signature strings. Step 2: sort signature strings and recolor.

4 steps of WL

$$\mathrm{cr}^{(0)}(G, v) \;:=\; \mathrm{col}(G, v).$$

- 1. Start: cr(0) is a vertex colouring. cr(0):V(G)->C defined by col(G,v)

$$\mathrm{cr}^{(t+1)}(G) \;\preccurlyeq\; \mathrm{cr}^{(t)}(G)$$

2. Iterate: in each round t+1, each node v updates its colour based on: - its current colour, and - the **multiset** of neighbor colours

$$\mathrm{cr}^{(t+1)}(G, v) := \left(\mathrm{cr}^{(t)}(G, v),\ \{\!\{\, \mathrm{cr}^{(t)}(G, w) \mid w \in N(v) \,\}\!\}\right)$$

- 3. Stop: There is some number the coloring doesn't become strictly finer anymore

$$\mathrm{cr}^{(t_\infty+1)}(G) \;\equiv\; \mathrm{cr}^{(t_\infty)}(G).$$

- 4. Compare graphs: If two graphs have different multisets of final colours, they are not isomorphic.

K-dimensional WL

- $\mathrm{wl}_k^{(0)}(G,v) := \mathrm{atp}_k(G,v)$ and
  $\mathrm{wl}_k^{(t+1)}(G,v) := \left(\mathrm{wl}_k^{(t)}(G,v),\ M\right)$
  with

- The essential idea is the same as 1dimensional case.
  each vertex is replaced with k tuple of vertices and the neigbours are replaced with v[w/i] = (v1,...,vi−1,w,vi+1,...vk)

- $$M = \left\{\!\!\left\{ \left(\mathrm{atp}_{k+1}(G, \boldsymbol{v}w), \mathrm{wl}_k^{(t)}(G, \boldsymbol{v}[w/1]), \mathrm{wl}_k^{(t)}(G, \boldsymbol{v}[w/2]), \vdots \mathrm{wl}_k^{(t)}(G, \boldsymbol{v}[w/k])\right) \;\middle|\; w \in V(G) \right\}\!\!\right\}.$$

Expressiveness of GNN and 1WL

1-WL is as expressive as GNN

- **Theorem VIII.1 ([43],[58]).** *Let $d \geq 1$, and let ξ be a vertex invariant computed by an d-layer GNN. Then* $\mathsf{cr}^{(d)}$ *refines ξ, that is, for all graphs G, G' and vertices $v \in V(G), v' \in V(G')$,*

$$\mathsf{cr}^{(d)}(G, v) = \mathsf{cr}^{(d)}(G', v') \implies \xi(G, v) = \xi(G', v').$$

This theorem holds regardless of the choice of the aggregation function agg and the combination function comb.

- ξ is the vertex invariant computed by GNN, in other words the final feature vector GNN assigns to node v and cr^(d) is the colouring at step d of the 1 dimensional WL algorithm.

- The theorem states that if WL assigns the same colour to two nodes after d steps then a d-layer GNN assigns the same feature vector to those two nodes.

GNN is as expressive as 1-WL node wise

Theorem VIII.4 ([43, 58]). *Let $n \in \mathbb{N}$. Then there is a recurrent GNN such that for all $t \leq n$, the vertex invariant $\xi^{(t)}$ computed in the t-th iteration of the GNN refines $\mathsf{cr}^{(t)}$ on all graphs of order at most n.*

That is, for all graph G, G' of order at most n and all vertices $v \in V(G), v' \in V(G')$:

$$\xi^{(t)}(G, v) = \xi^{(t)}(G', v') \implies \mathsf{cr}^{(t)}(G, v) = \mathsf{cr}^{(t)}(G', v').$$

- ξ(t) is the vertex feature map produced by GNN after t steps and cr(t) is the colouring produced by 1-WL algorithm after t steps. ξ(t) refines cr(t) means ξ(t) can distinguish at least as much as cr(t)

GNN is as expressive as 1-WL graph wise

Corollary VIII.7. *Let $n \geq 1$. Then there is a recurrent GNN such that for all graphs G, G' of order at most n, if* $\mathsf{cr}^{(\infty)}$ *distinguishes G and G' then $\xi(G) \neq \xi(G')$, where ξ is the graph invariant computed by the GNN in n iterations.*

- This corollary is slightly different from the previous theorem, xi was vertex invariant before and it's now graph invariant and it compares GNN in n iterations to cr^(∞), the stable colouring
- It states that if cr(∞) distinguishes G and G', so does xi. In other words, a recurrent GNN can match the full distinguishing power of 1-WL

Conclusions

1. Why we care about “expressive power”

- When you design a GNN you rely on the network’s ability to distinguish different graph structures.

- So the key question became:
  > “How well can a GNN tell apart different graphs?”

2. WL gives us a known benchmark

- The Weisfeiler–Leman test is a classic graph isomorphism algorithm. It's been studied for decades, and we know exactly where it succeeds and where it fails.

- By showing:
  > GNNs $\equiv$ 1-WL $\rightarrow$ we immediately inherit all that knowledge about WL

  $\rightarrow$ We now know precisely the limitations of standard message-passing GNNs.

3. Practical Implications

Guidance for new architectures

- Once researchers saw this limit, they started designing more expressive GNN variants:
  - k-GNNs / k-WL GNNs → correspond to higher-dimensional WL tests
  - Graph Transformers, subgraph GNNs, positional encodings → break WL limitations by adding more structure or global information

References

- Grohe, M. (2021). The Logic of Graph Neural Networks. arXiv:2104.14624.

Any Questions?