Beyond Message Passing: Expressive GNN Models

Jiaxuan You

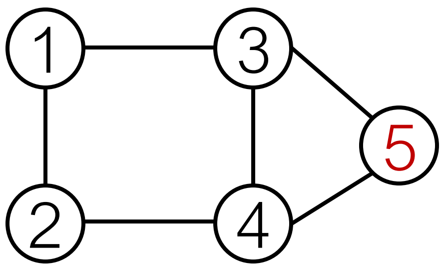
Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall

https://ulab-uiuc.github.io/CS598/

Recap: GraphRNN Tractability via BFS

- **Breadth-First Search node ordering**



BFS ordering

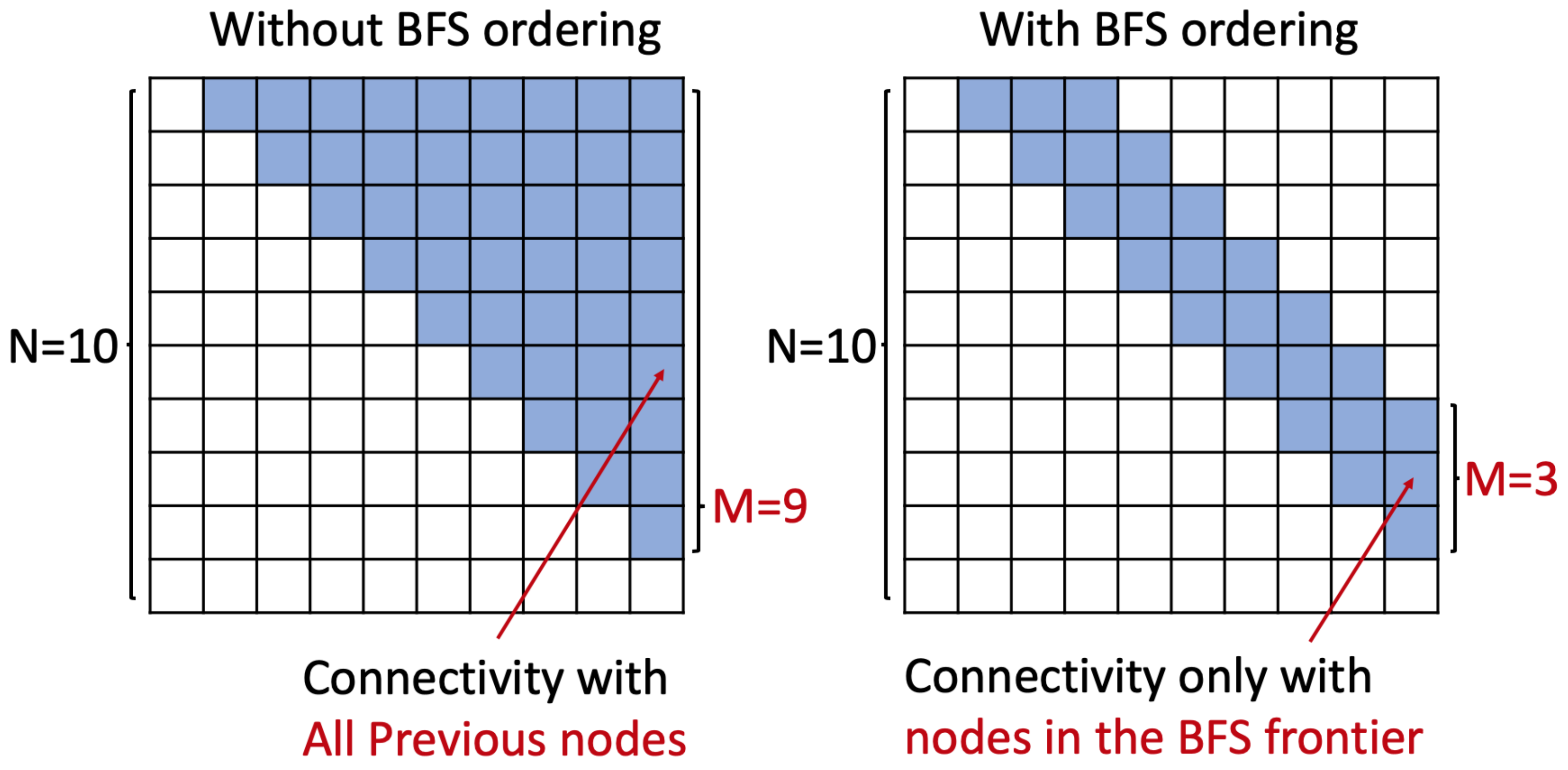
BFS node ordering: Node 5 will never connect to node 1 (only need memory of 2 "steps" rather than $n-1$ steps)

- **Benefits:**
  - Reduce possible node orderings
    - From $O(n!)$ to number of distinct BFS orderings
  - Reduce steps for edge generation
    - Reducing number of previous nodes to look at

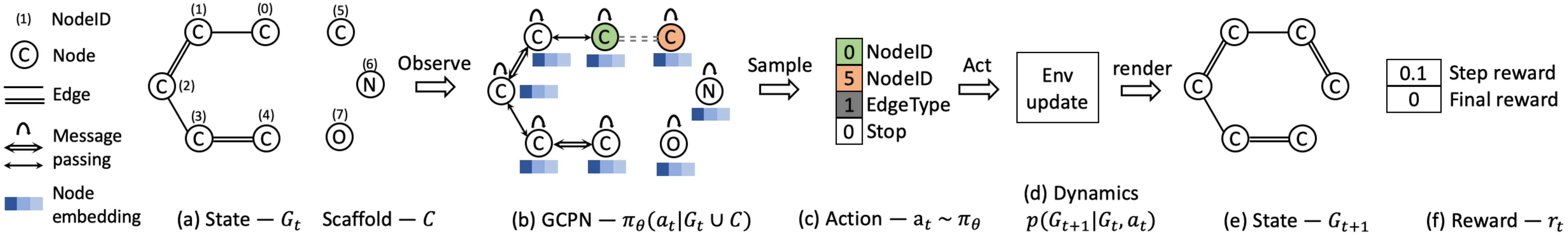
Recap: GraphRNN Tractability via BFS

- **BFS reduces the number of steps for edge generation**

Adjacency matrices

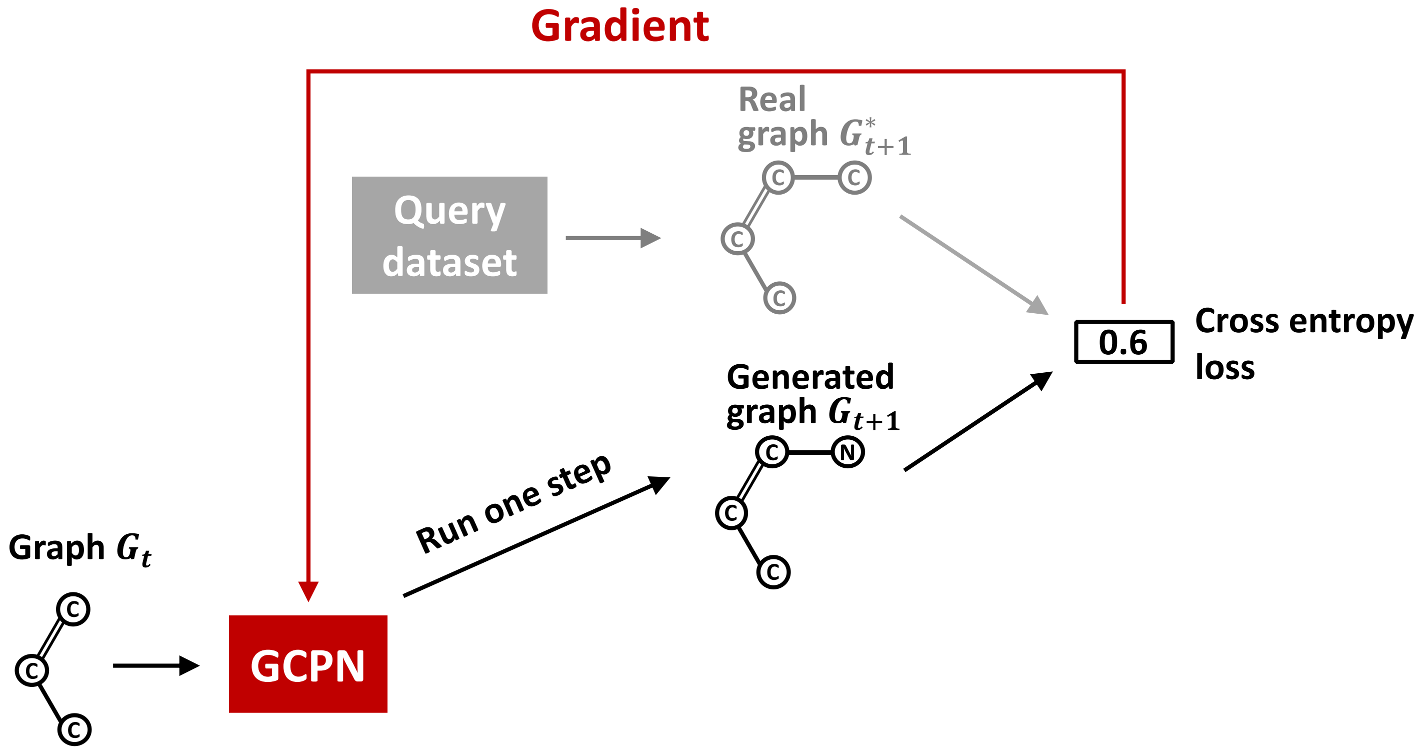


Recap: Overview of GCPN



- **(a)** Insert nodes
- **(b,c)** Use GNN to predict which nodes to connect
- **(d)** Take an action (check chemical validity)
- **(e, f)** Compute reward

Recap: Training Graph Conv. Policy Network



(1) Self-supervised training: Imitating the action given by real observed graphs with gradient

→ **Goal 1: imitation**

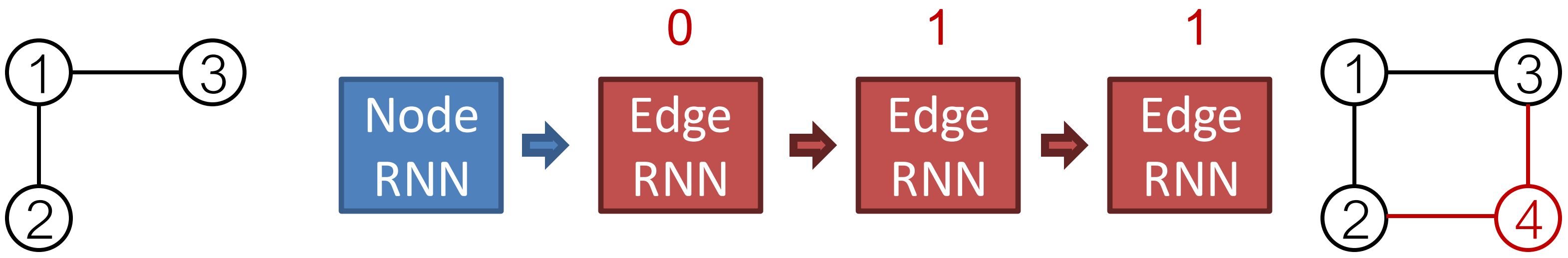
(2) RL training: Train policy to optimize rewards with policy gradient (PPO)

→ **Goal 2: optimization**

ChatGPT [OpenAI, 2022] uses the similar idea: **self-supervised** + **RL training**

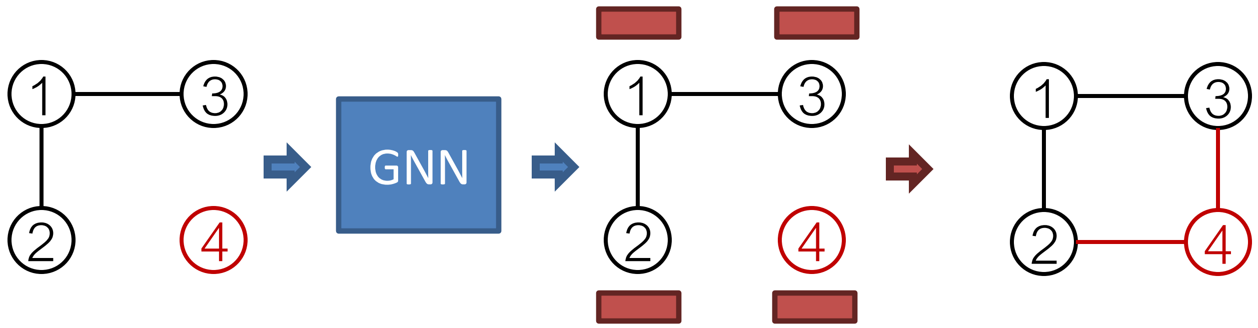
Recap: GCPN vs. GraphRNN

- **Sequential graph generation**
- **GraphRNN**: predict action based on **RNN hidden states**



RNN hidden state captures the generated graph so far

- **GCPN:** predict action based on **GNN node embeddings**



Recall the link prediction head:

$\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}) :=$
$\text{Linear}(\text{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$

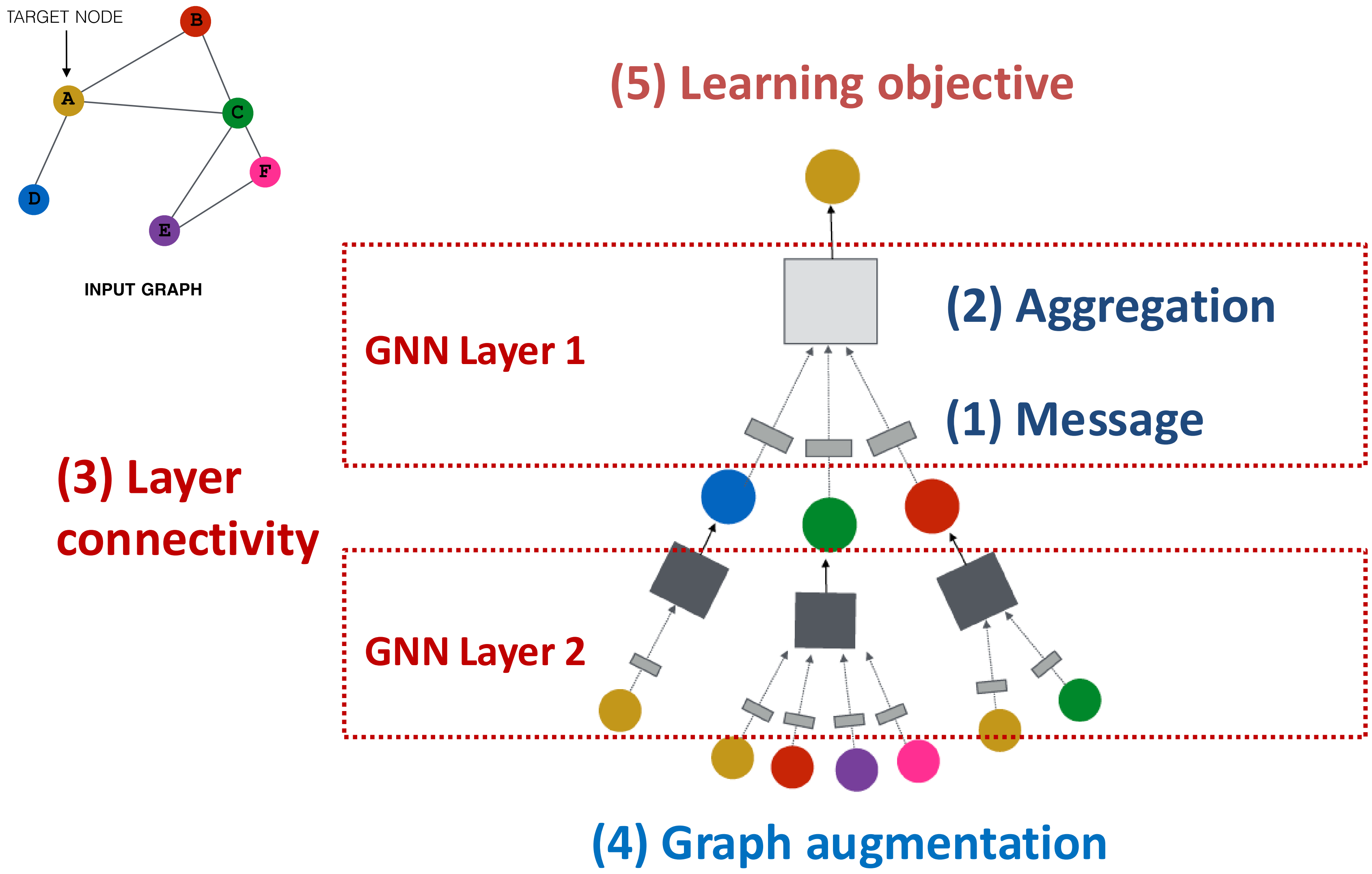
Node embeddings

Predict potential links using node embeddings

Beyond Message Passing: Expressive GNN Models

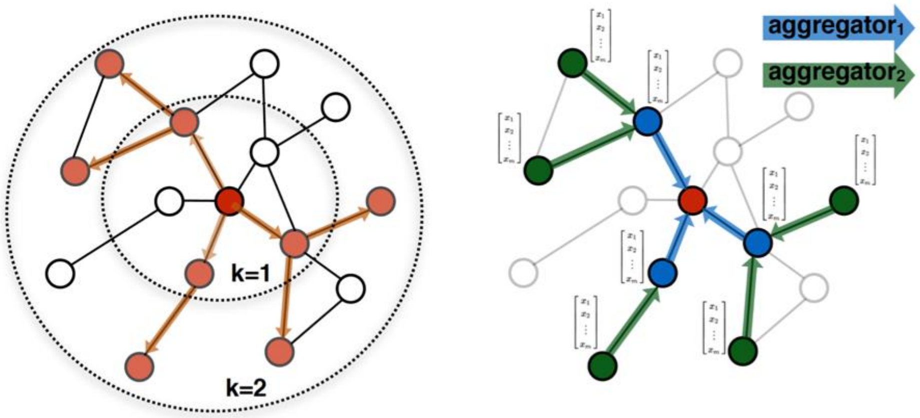
Limitations of Graph Neural Networks

Recap: A General GNN Framework



A "Perfect" GNN Model

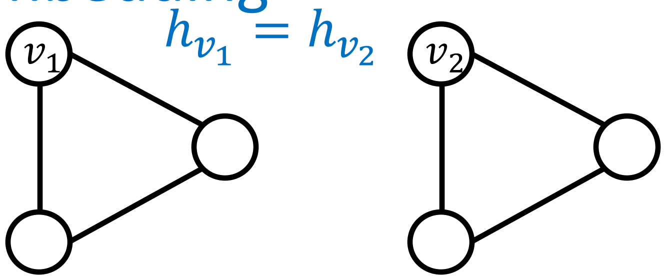
- **A thought experiment:** What should a perfect GNN do?
  - A k-layer GNN embeds a node based on the K-hop neighborhood structure



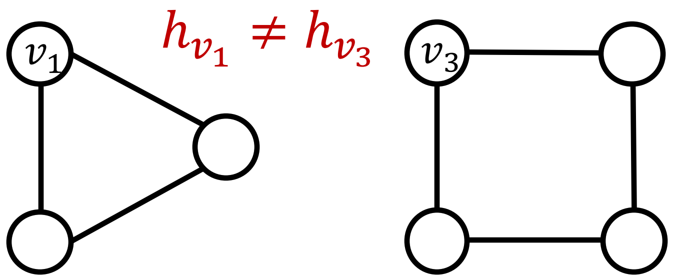
  - A perfect GNN should build **an injective function** between neighborhood structure (regardless of hops) and node embeddings

A "Perfect" GNN Model

- **Therefore, for a perfect GNN:**
  - **Observation 1:** If two nodes have the same neighborhood structure, they must have the same embedding

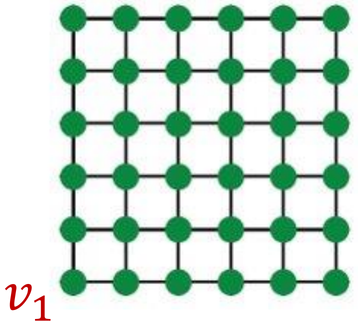


  - **Observation 2:** If two nodes have different neighborhood structure, they must have different embeddings



Imperfections of Existing GNNs

- **However, Observations 1 & 2 are imperfect**
- **Observation 1 could have issues:**
  - Even though two nodes may have the same neighborhood structure, we may want to assign different embeddings to them
  - Because these nodes appear in **different positions in the graph**
  - We call these tasks **Position-aware tasks**
  - **Even a perfect GNN will fail for these tasks:**



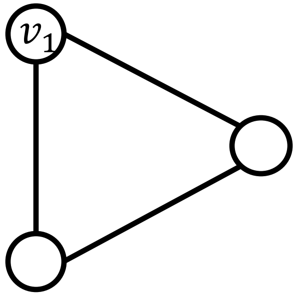
A grid graph

NYC road network

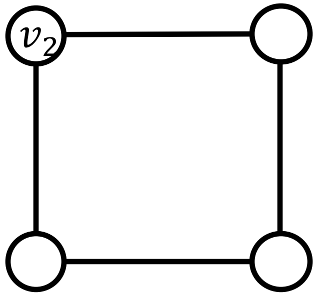
Imperfections of Existing GNNs

- **Observation 2 often cannot be satisfied:**
  - **The GNNs we have introduced so far are not perfect**
  - In Lecture 9, we discussed that their expressive power is **upper bounded by the WL test**
  - For example, message passing GNNs **cannot count the cycle length:**

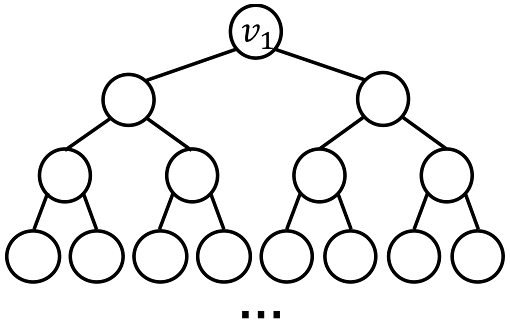
v_1 resides in a cycle with length 3



v_2 resides in a cycle with length 4



The computational graphs for nodes v_1 and v_2 are always the same

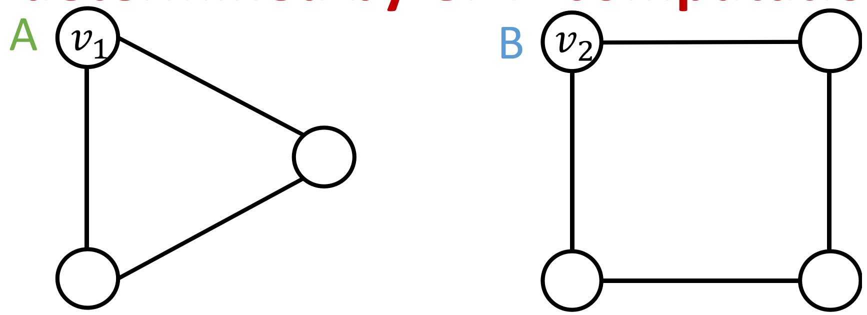


Plan for the Lecture

- We will resolve both issues by **building more expressive GNNs**
- **Fix issues in Observation 1:**
  - Create node embeddings based on their positions in the graph
  - Example method: **Position-aware GNNs**
- **Fix issues in Observation 2:**
  - Build message passing GNNs that are more expressive than WL test
  - Example method: **Identity-aware GNNs**

Our Approach

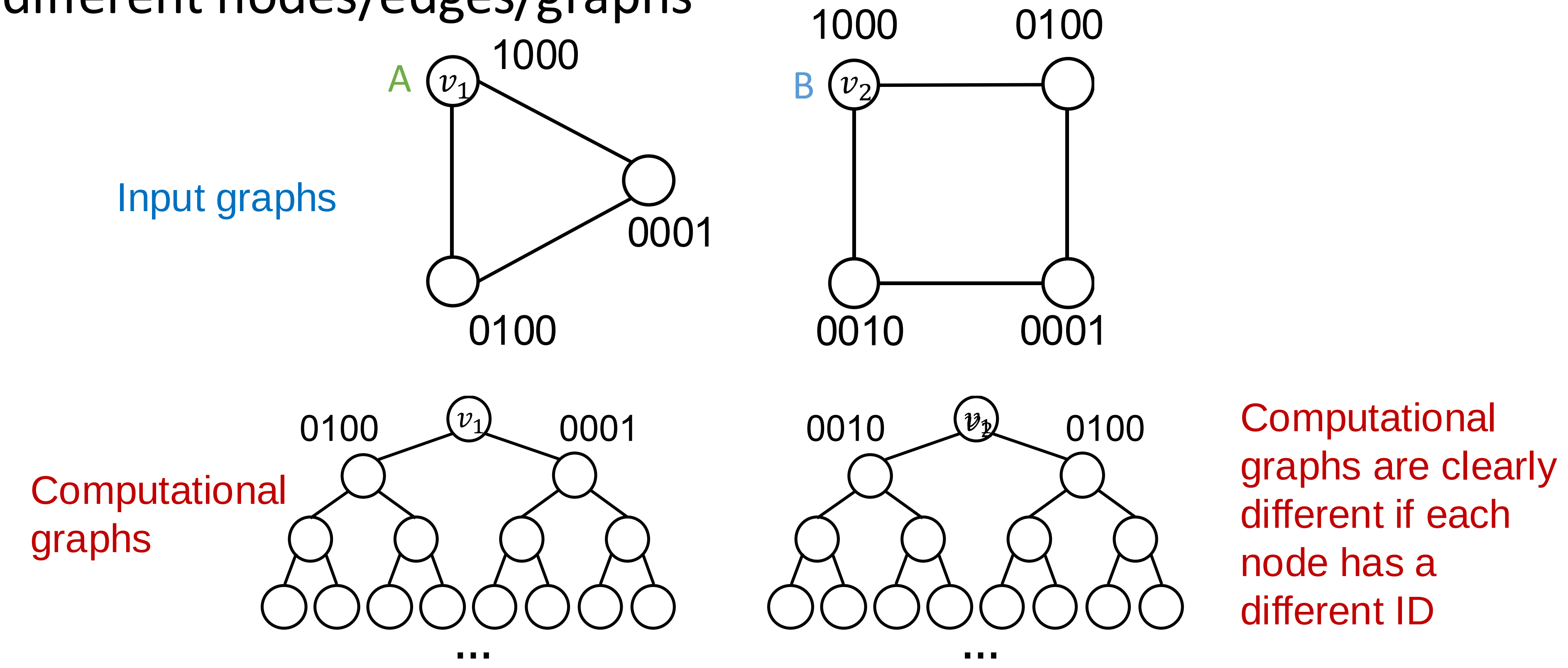
- **We use the following thinking:**
  - Two different inputs (nodes, edges, graphs) are labeled differently
  - A "failed" model will always assign the same embedding to them
  - A "successful" model will assign different embeddings to them
  - **Embeddings are determined by GNN computational graphs:**



Two inputs: nodes v_1 and v_2
Different labels: A and B
Goal: assign different embeddings to v_1 and v_2

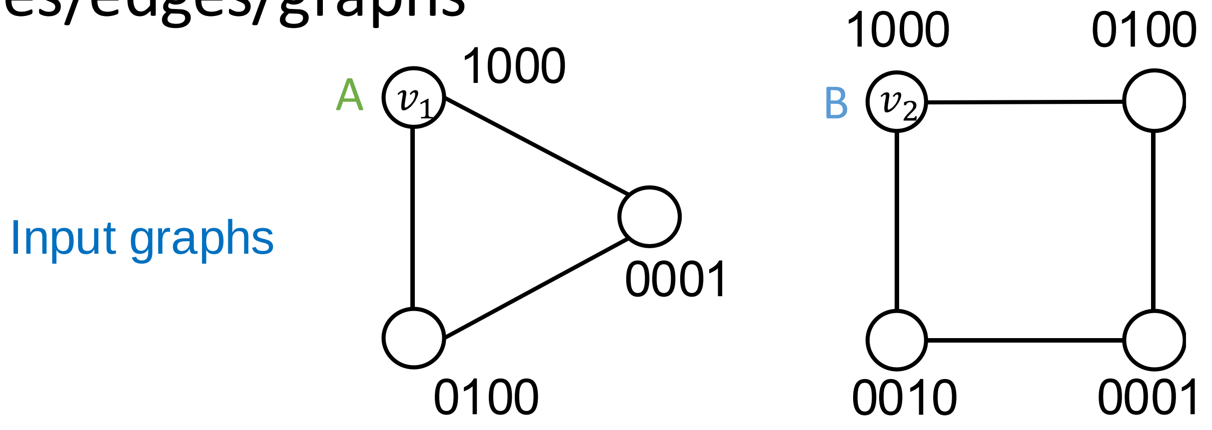
Naïve Solution is not Desirable

- **A naïve solution:** One-hot encoding
  - Encode each node with a different ID, then we can always differentiate different nodes/edges/graphs



Naïve Solution is not Desirable

- **A naïve solution:** One-hot encoding
  - Encode each node with a different ID, then we can always differentiate different nodes/edges/graphs



  - **Issues:**
    - **Not scalable**: Need $O(N)$ feature dimensions (N is the number of nodes)
    - **Not inductive:** Cannot generalize to new nodes/graphs

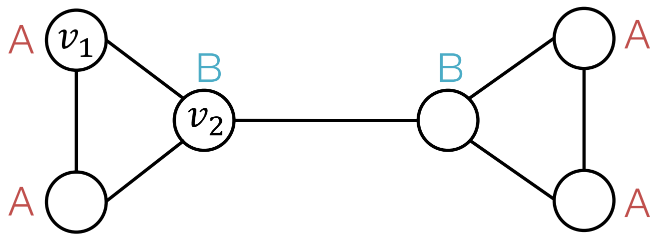
Beyond Message Passing: Expressive GNN Models

Position-aware Graph Neural Networks

Two Types of Tasks on Graphs

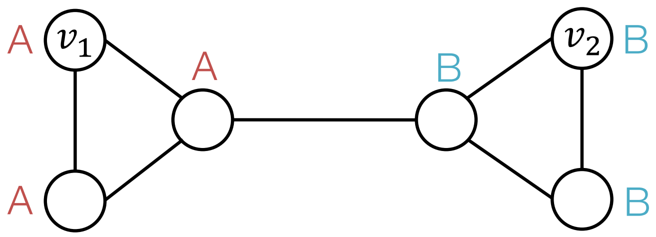
- **There are two types of tasks on graphs**

Structure-aware task



- Nodes are labeled by their **structural roles** in the graph

Position-aware task

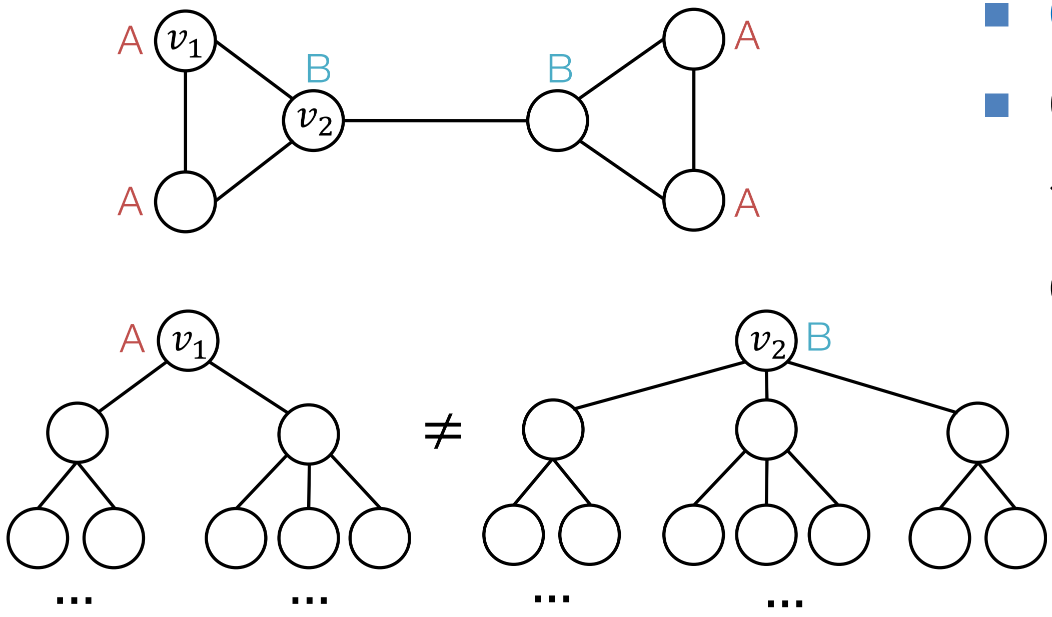


- Nodes are labeled by their **positions** in the graph

Structure-aware Tasks

- **GNNs often work well for structure-aware tasks**

Structure-aware task

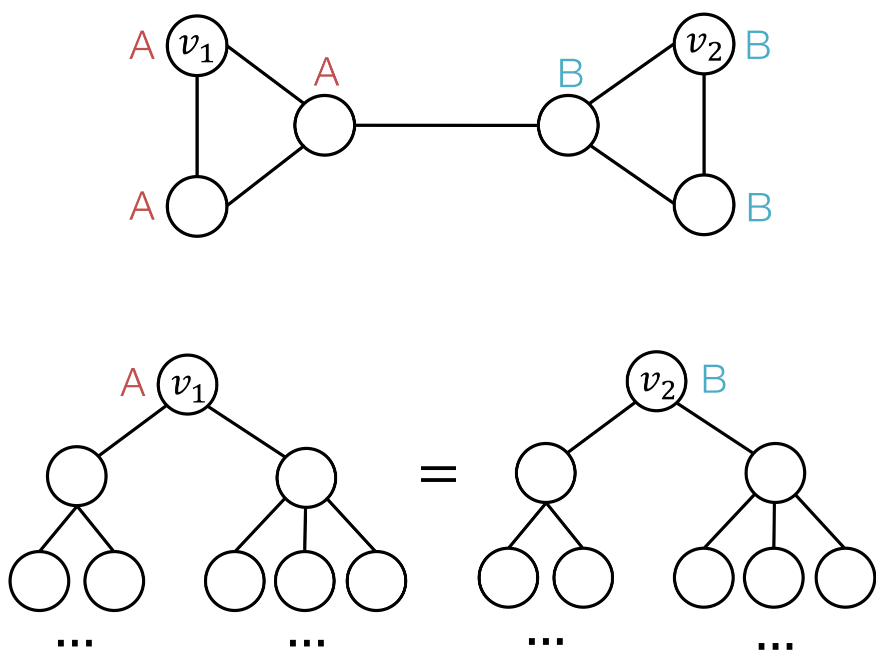


- GNNs work ☺
- Can differentiate v_1 and v_2 by using different computational graphs

Position-aware Tasks

- **GNNs will always fail for position-aware tasks**

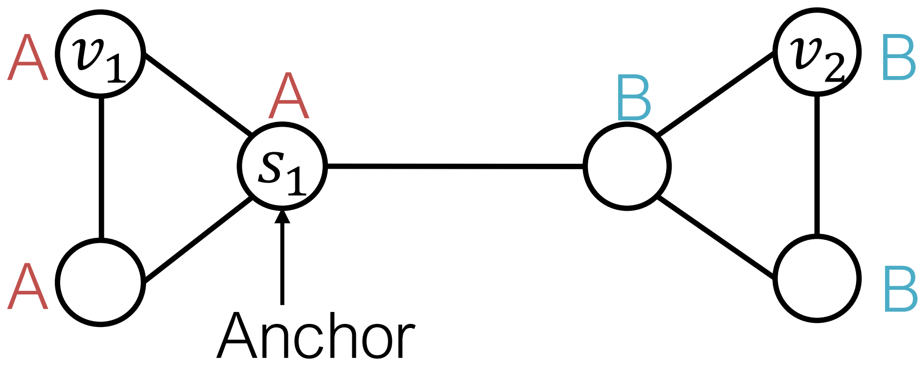
Position-aware task



- GNNs fail ☹
- v_1 and v_2 will always have the same computational graph, **due to structure symmetry**
- **Can we define deep learning methods that are position-aware?**

Power of "Anchor"

- Randomly pick a node s_1 as an **anchor node**
- Represent v_1 and v_2 via their relative distances w.r.t. the anchor s_1, **which are different**
- An anchor node serves as **a coordinate axis**
  - Which can be used to **locate nodes in the graph**

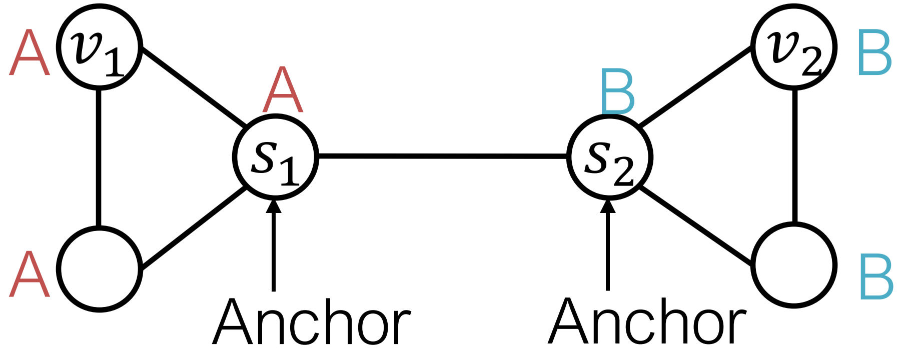


Relative Distances

	s_1
v_1	1
v_2	2

Power of "Anchors"

- Pick more nodes s_1, s_2 as **anchor nodes**
- **Observation:** More anchors can better characterize node position in different regions of the graph
- Many anchors –> Many coordinate axes

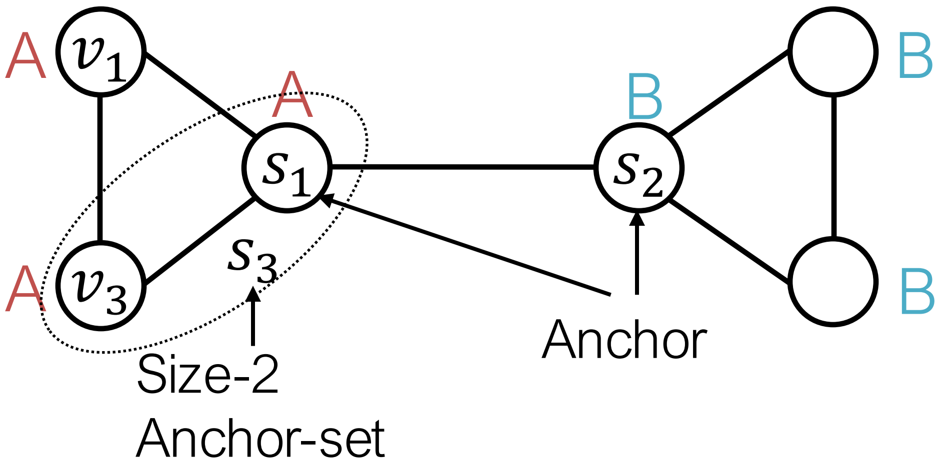


Relative Distances

	s_1	s_2
v_1	1	2
v_2	2	1

Power of "Anchor-sets"

- Generalize anchor from a single node to **a set of nodes**
  - We define distance to an anchor-set as the minimum distance to all the nodes in the ancho-set
- **Observation:** Large anchor-sets can sometimes provide more precise position estimate
  - We can save the total number of anchors



Relative Distances

	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

Anchor s_1, s_2 cannot differentiate node v_1, v_3, but anchor-set s_3 can

Anchor Set: Theory

- **Goal:** Embed the metric space (V, d) into the Euclidian space $\mathbb{R}^k$ such that the original distance metric is preserved.
  - For every node pairs $u, v \in V$, the Euclidian embedding distance $||\boldsymbol{z}_u - \boldsymbol{z}_v||_2$ is close to the original distance metric $d(u, v)$.

Anchor Set: Theory

- Bourgain Theorem [Informal] [Bourgain 1985]
  - Consider the following embedding function of node $v \in V$.

    $$f(v) = \left(d_{\min}(v, S_{1,1}), d_{\min}(v, S_{1,2}), \dots, d_{\min}(v, S_{\log n, c\log n})\right) \in \mathbb{R}^{c \log^2 n}$$

  - where
    - c is a constant.
    - $S_{i,j} \subset V$ is chosen by including each node in V independently with probability $\frac{1}{2^i}$.
    - $d_{\min}(v, S_{i,j}) \equiv \min_{u \in S_{i,j}} d(v, u)$.
  - **The embedding distance produced by f is provably close to the original distance metric (V, d).**

Anchor Set: Theory

P-GNN follows the theory of Bourgain theorem

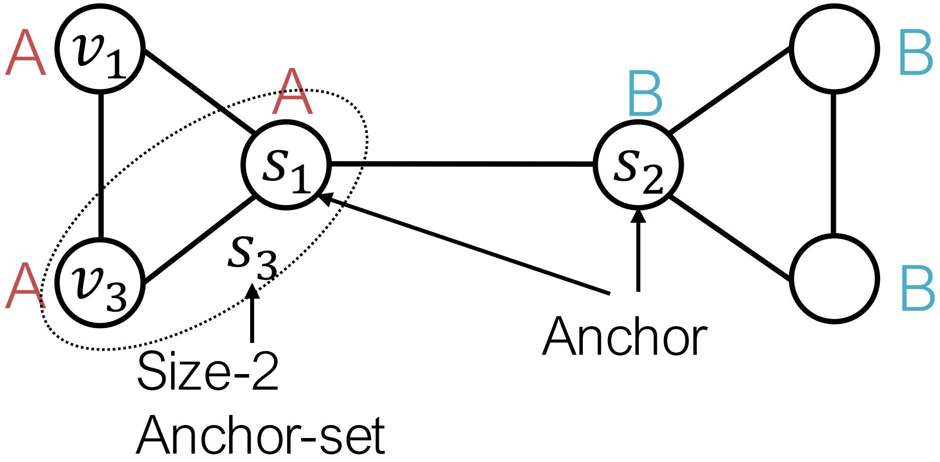
- First samples $O(\log^2 n)$ anchor sets $S_{i,j}$.
- Embed each node v via
  $\left(d_{\min}(v, S_{1,1}), d_{\min}(v, S_{1,2}), \ldots, d_{\min}(v, S_{\log n, c\log n})\right) \in \mathbb{R}^{c \log^2 n}$.

P-GNN maintains the inductive capability

- During training, new anchor sets are *re-sampled* every time.
- P-GNN is learned to operate over the new anchor sets.
- At test time, given a new unseen graph, new anchor sets are sampled.

Position Information: Summary

- **Position encoding for graphs:** Represent a node's position by its distance to randomly selected anchor-sets
  - Each dimension of the position encoding is tied to an anchor-set



	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

v_1's Position encoding

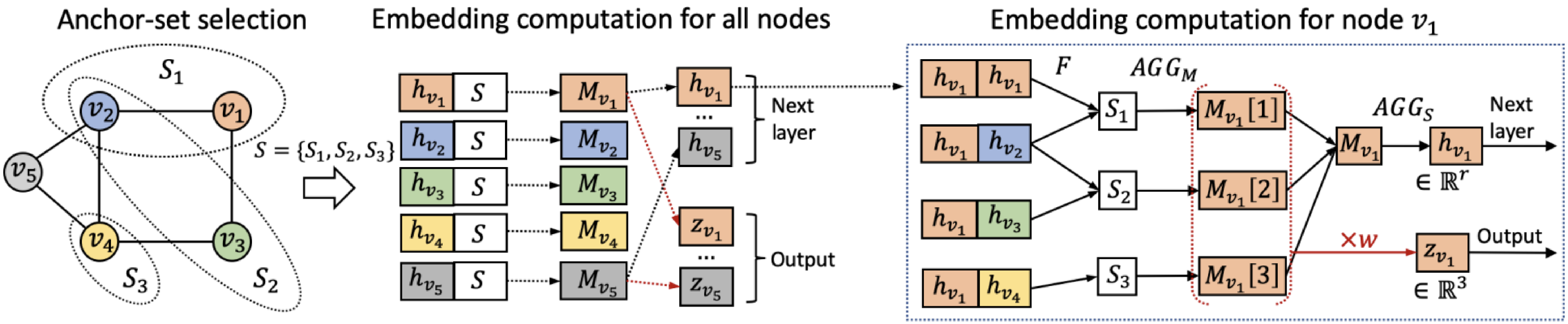
v_3's Position encoding

How to Use Position Information

- **The simple way:** Use position encoding as an augmented node feature (works well in practice)
  - **Issue:** Since each dimension of position encoding is tied to a random anchor set, dimensions of positional encoding can be randomly permuted, without changing its meaning
  - Imagine you permute the input dimensions of a normal NN, the output will surely change

How to Use Position Information

- **A more expressive solution:** Requires a special NN that can maintain the **permutation invariant property of position encoding**
  - Permuting the input feature dimension will only result in the permutation of the output dimension, the value in each dimension won't change
  - Position-aware GNN paper has more details



Beyond Message Passing: Expressive GNN Models

Identity-Aware Graph Neural Networks

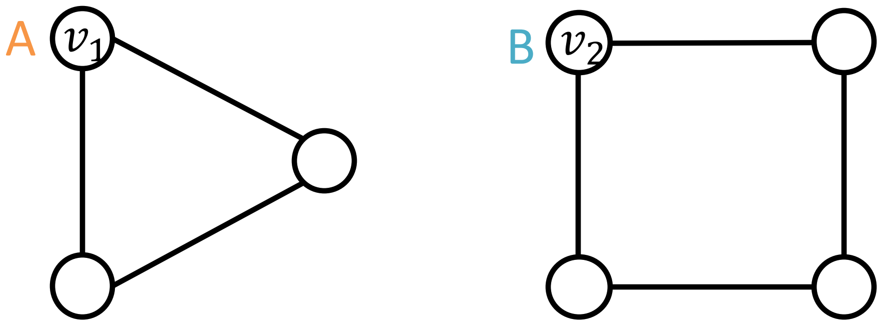
More Failure Cases for GNNs

- We learned that **GNNs would fail for position-aware tasks**
- **But can GNN perform perfectly in structure-aware tasks?**
  - Unfortunately, **NO.**
- GNNs exhibit three levels of failure cases in structure-aware tasks:
  - Node level
  - Edge level
  - Graph level

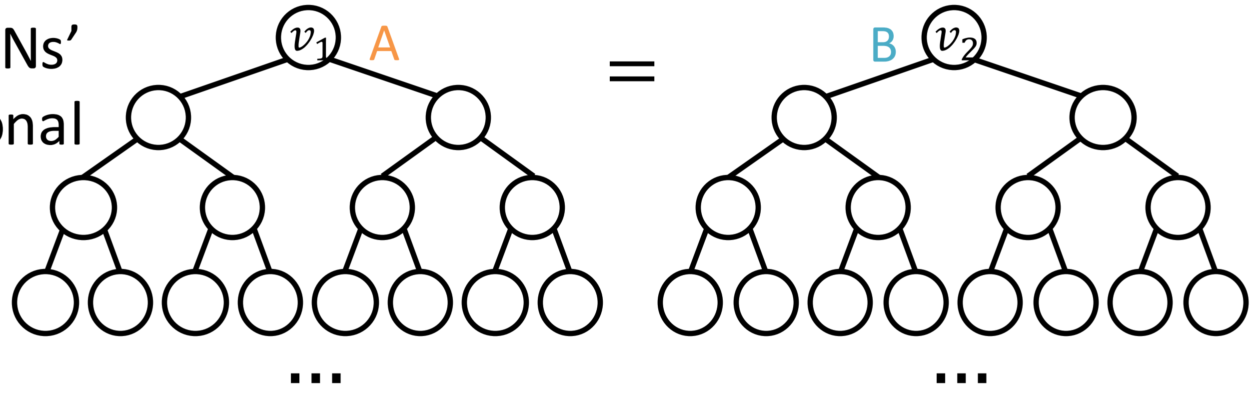
GNN Failure 1: Node-level Tasks

Different Inputs but the same computational graph → GNN fails

Example input graphs



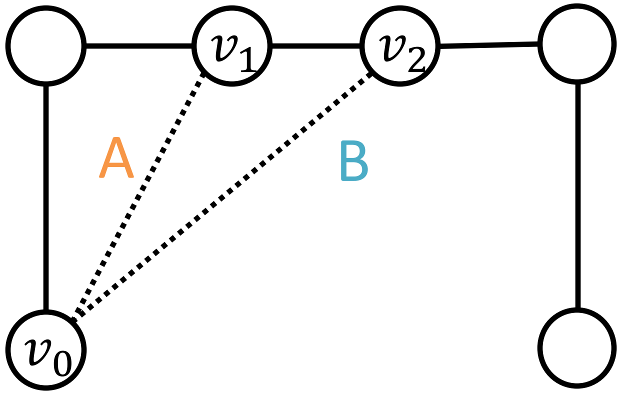
Existing GNNs' computational graphs



GNN Failure 2: Edge-level Tasks

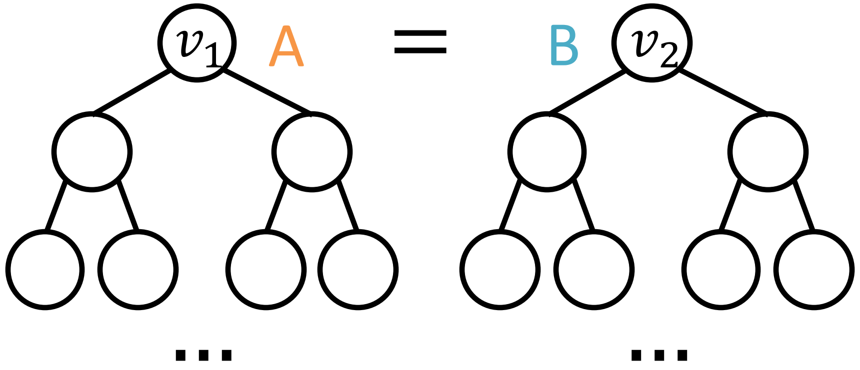
Different Inputs but the same computational graph → GNN fails

Example input graphs



Edge A and B share node v_0
We look at embeddings for v_1 and v_2

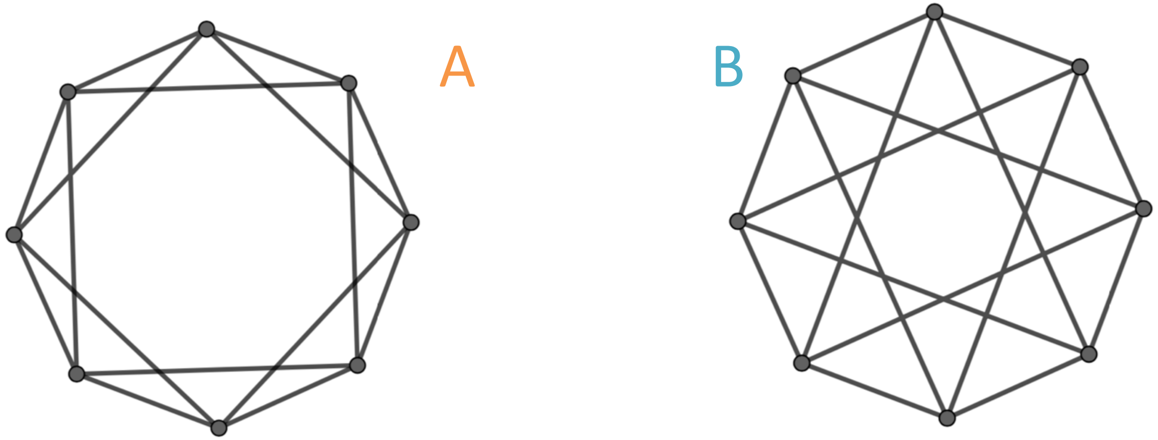
Existing GNNs' computational graphs



GNN Failure 3: Graph-level Tasks

Different Inputs but the same computational graph → GNN fails

Example input graphs

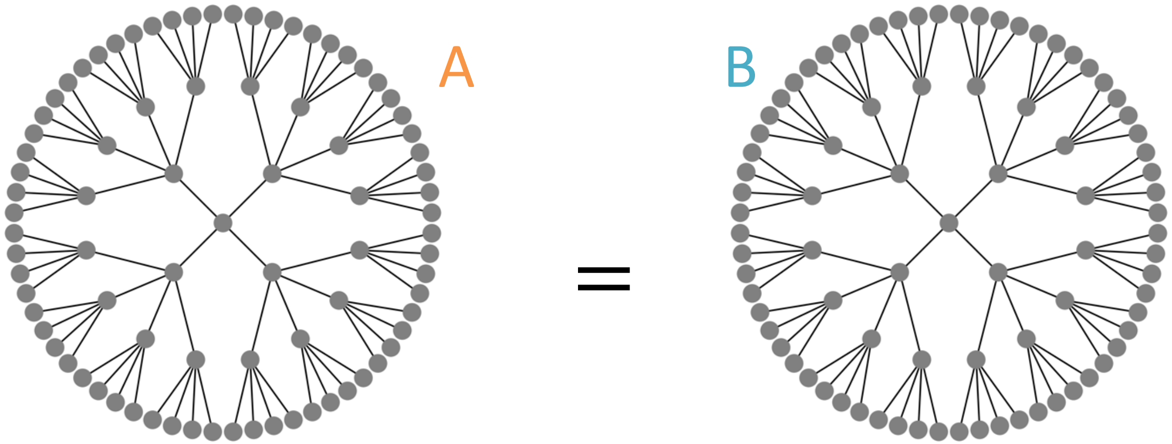


We look at embeddings for each node

For each node:

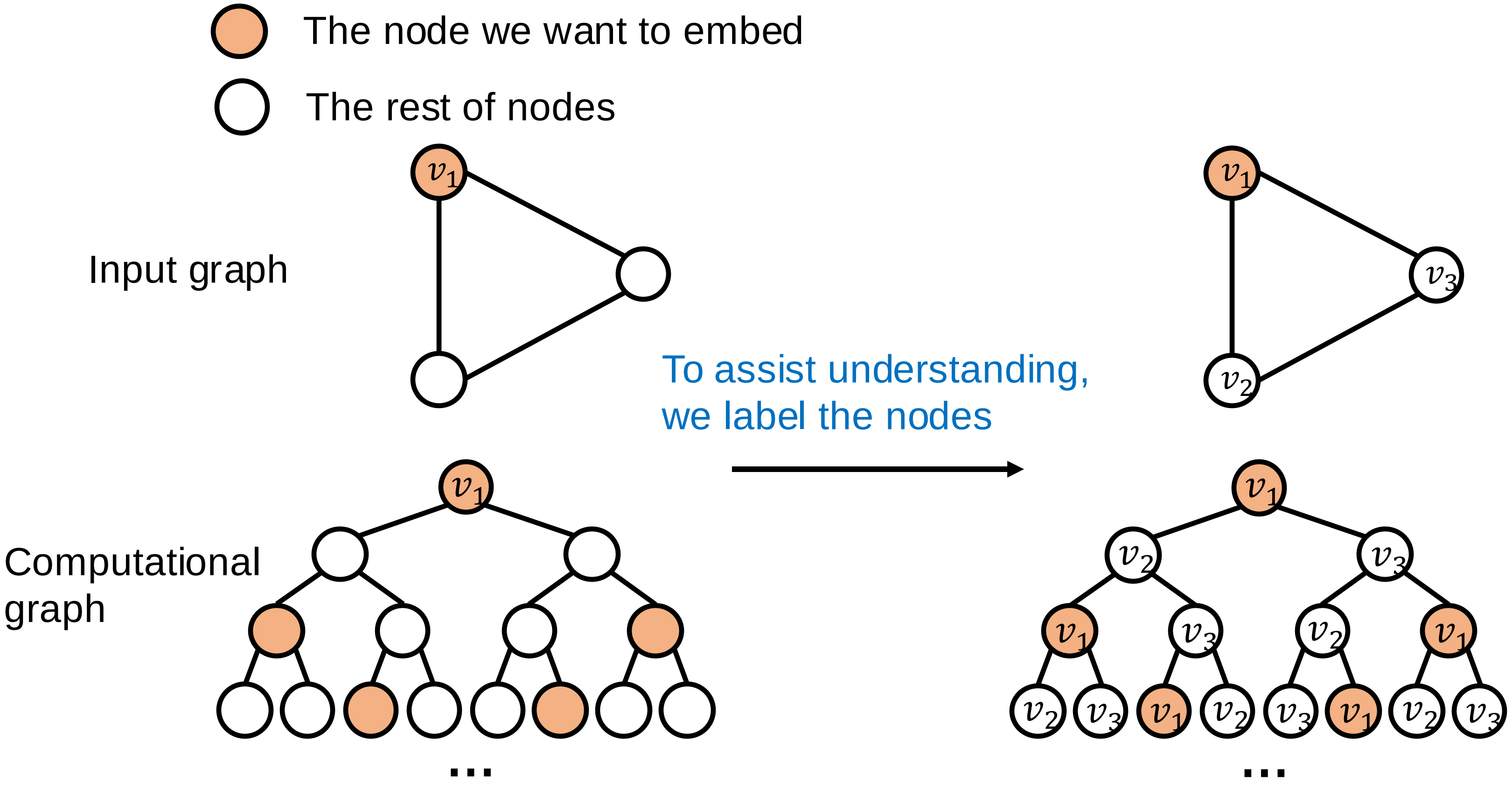
For each node:

Existing GNNs' computational graphs



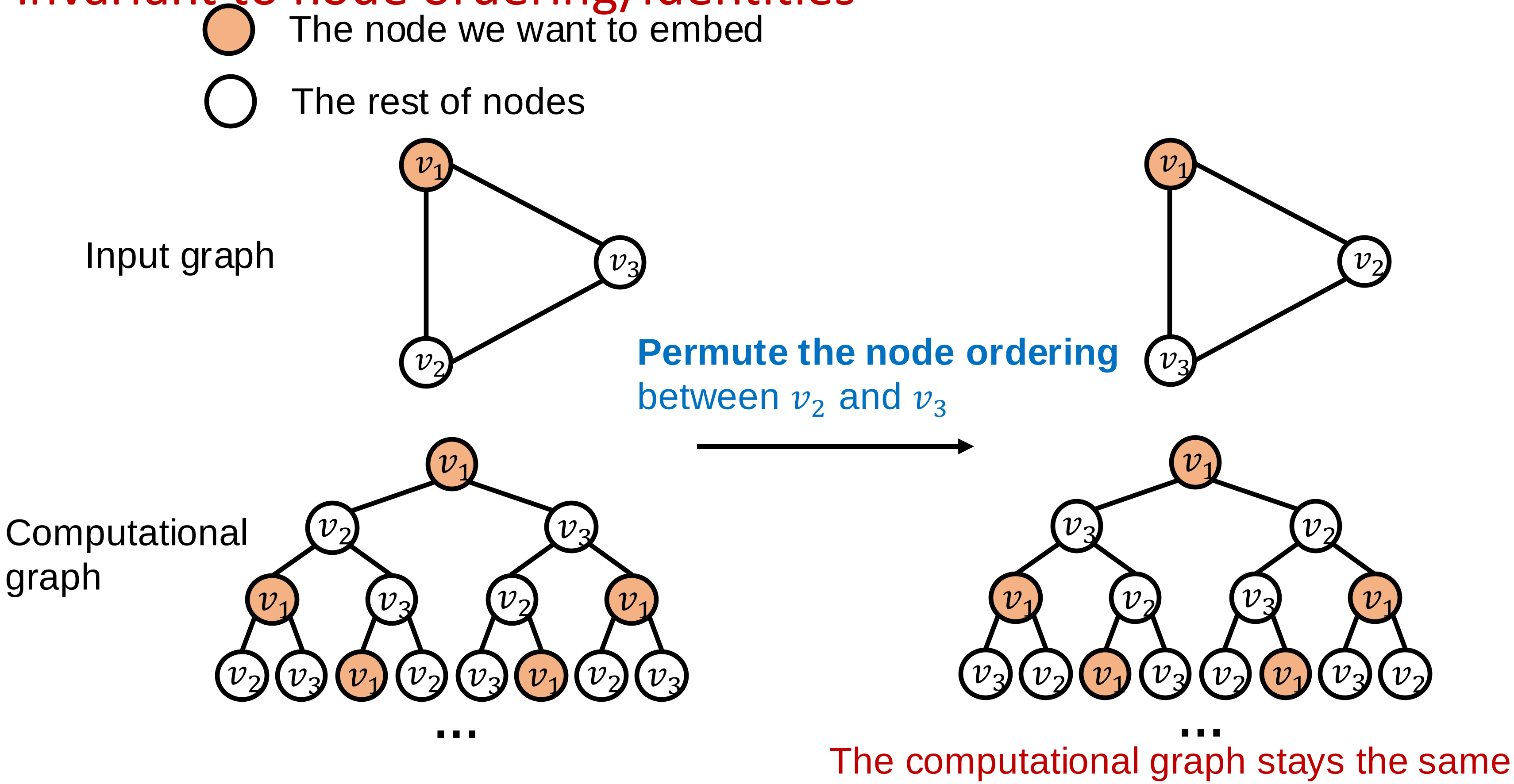
Idea: Inductive Node Coloring

- **Idea:** We can assign a color to the node we want to embed



Idea: Inductive Node Coloring

- This coloring is **inductive**:
  - It is invariant to node ordering/identities

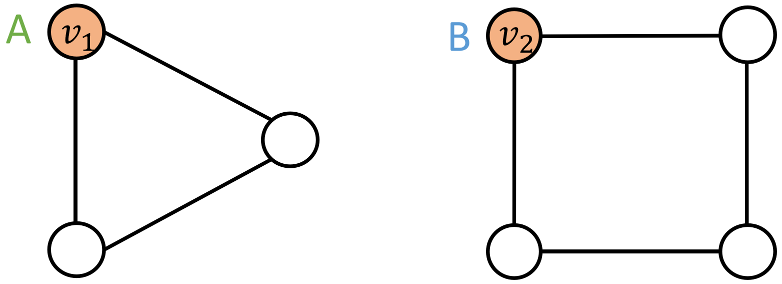


Inductive Node Coloring – Node level

- Inductive node coloring can help **node classification**

Node classification

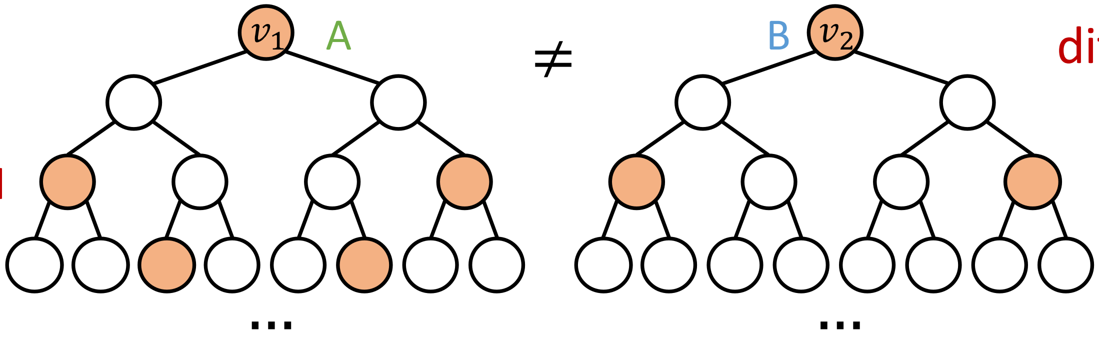
Example input graphs



Different computational graphs → Successfully differentiate nodes

We color root nodes with identity

ID-GNNs' computational graphs



Two types of nodes:

node with augmented identity

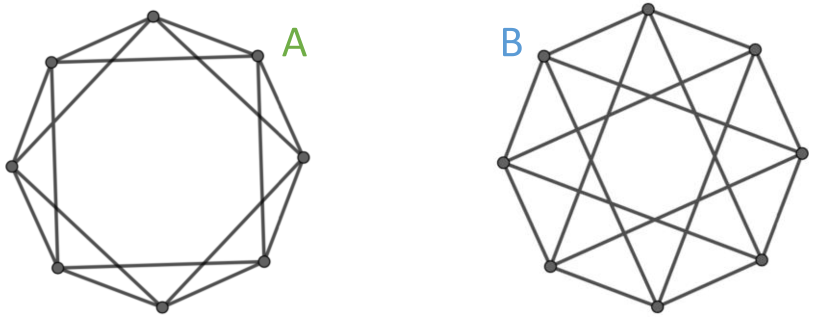
node without augmented identity

Inductive Node Coloring – Graph Level

- Inductive node coloring can help **graph classification**

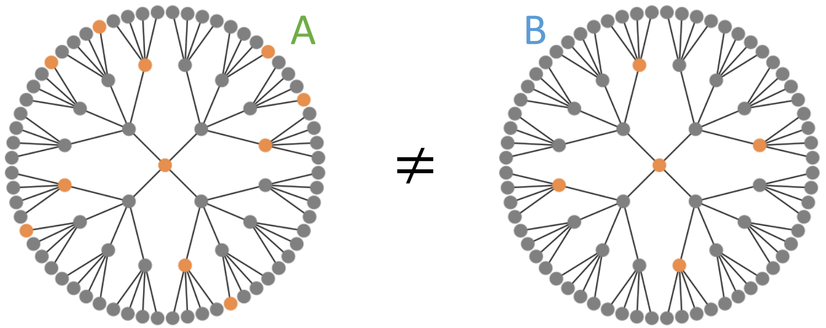
Graph classification

Example input graphs



We color root nodes with identity

ID-GNNs' computational graphs



Different computational graphs → Successful differentiate graphs

Two types of nodes: node with augmented identity; node without augmented identity

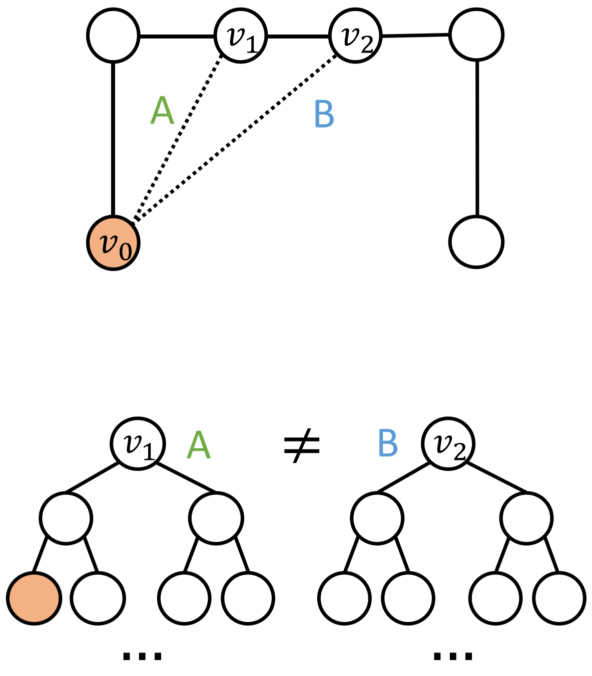
Inductive Node Coloring – Edge Level

- Inductive node coloring can help **link prediction**

Link prediction

Example input graphs

ID-GNNs' computational graphs



An edge-level task involves classifying a pair of nodes:

1. We color one of the node (v_0)
2. We then embed the other node in the node pair (v_1 or v_2)
3. We use the node embedding for v_1 or v_2 conditioned on v_0 being colored or not to make edge-level prediction

Two types of nodes:

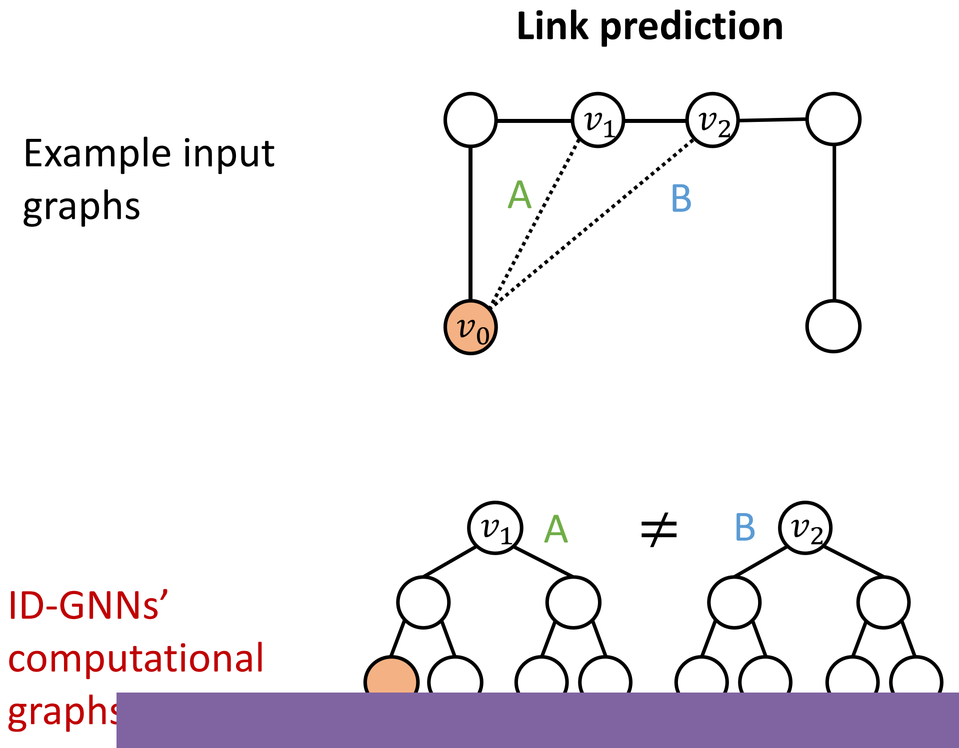
node with augmented identity

node without augmented identity

Different computational graphs → Successfully differentiate edges

Inductive Node Coloring – Edge Level

- Inductive node coloring can help **link prediction**



An edge-level task involves classifying a pair of nodes:

1. We color one of the node (v_0)
2. We then embed the other node in the node pair (v_1 or v_2)
3. We use the node embedding for v_1 or v_2 conditioned on v_0 being colored or not to make edge-level prediction

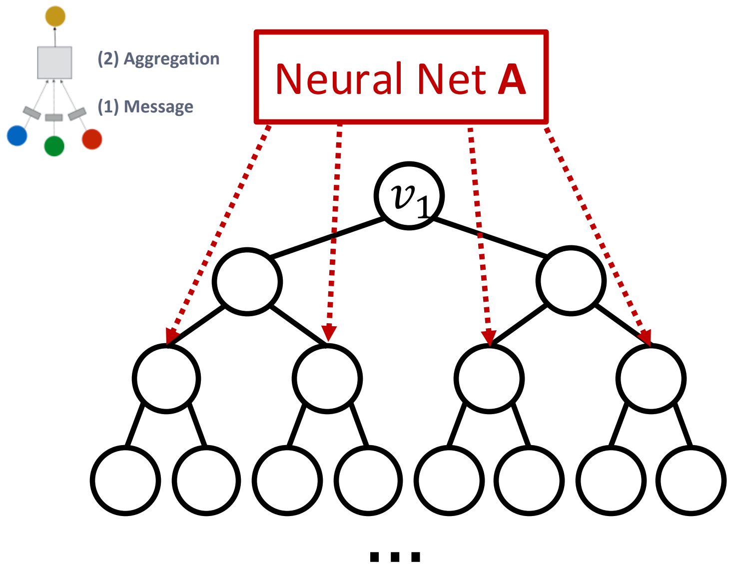
...hs

Two t

How to build a GNN using node coloring?

Identity-aware GNN

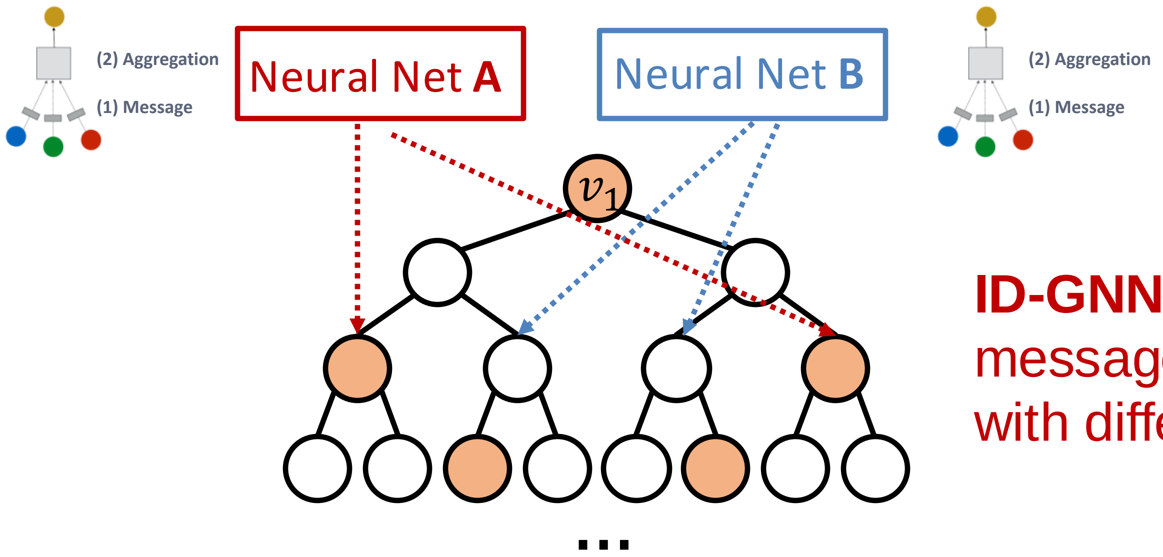
- **Utilize inductive node coloring** in embedding computation
  - **Idea: Heterogenous message passing**
    - Normally, a GNN applies **the same message/aggregation computation to all the nodes**



GNN: At a given layer, we apply the same message/aggregation to each node

Identity-aware GNN

- **Idea:** **Heterogenous message passing**
  - **Heterogenous:** different types of message passing is applied to different nodes
  - **An ID-GNN** applies **different message/aggregation to nodes with different colorings**



ID-GNN: At a given layer, different message/aggregation to nodes with different colorings

Identity-aware GNN

- **Output**: Node embedding $\boldsymbol{h}_v^{(K)}$ for $v \in \mathcal{V}$.
- **Step 1**: Extract the ego-network
  - $\mathcal{G}_v^{(K)}$: K-hop neighborhood graph around v
  - Set the initial node feature
    - For $u \in \mathcal{G}_v^{(K)}$, $\boldsymbol{h}_u^{(0)} \leftarrow \boldsymbol{x}_u$ (input node feature)

Identity-aware GNN

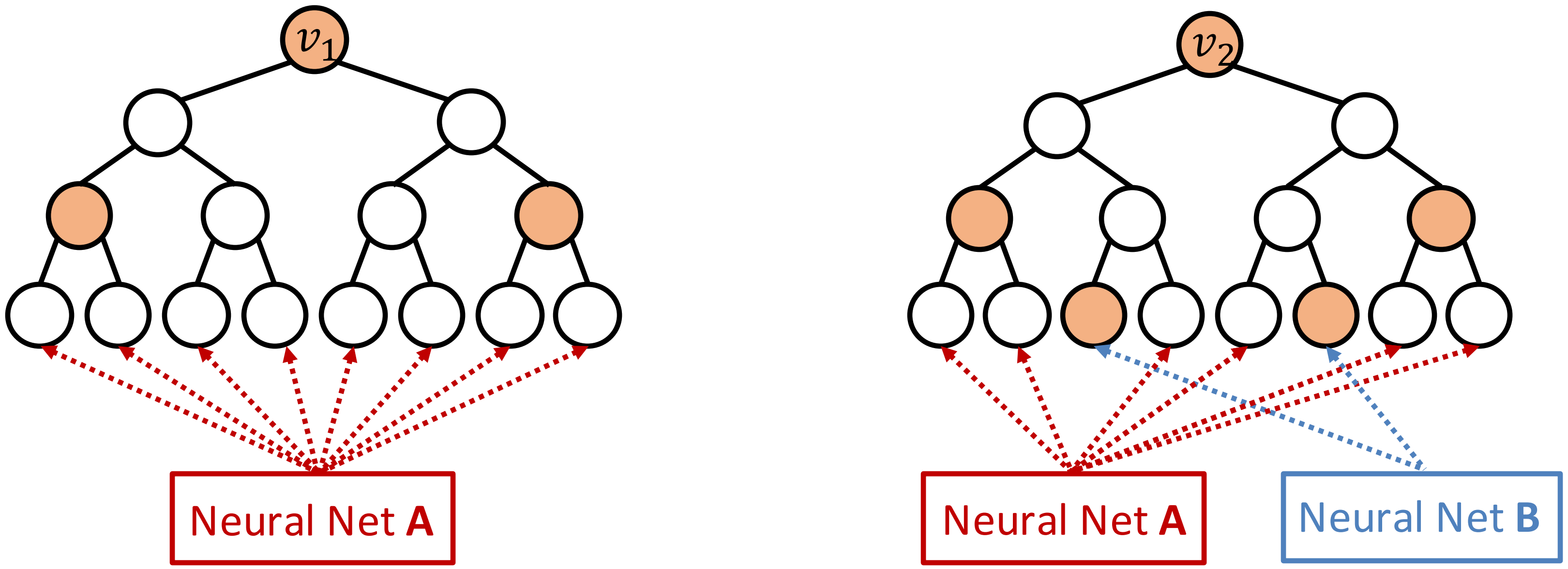
- **Step 2**: Heterogeneous message passing
  - For $k = 1, \dots, K$ do
    - For $u \in \mathcal{G}_v^{(K)}$ do

$$\boldsymbol{h}_{\boldsymbol{u}}^{(\boldsymbol{k})} \leftarrow AGG^{(k)}\left(\left\{\mathrm{MSG}_{\mathbf{1}[s=v]}^{(k)}\left(\boldsymbol{h}_s^{(k-1)}\right), s \in N(u)\right\}, \boldsymbol{h}_u^{(k-1)}\right)$$

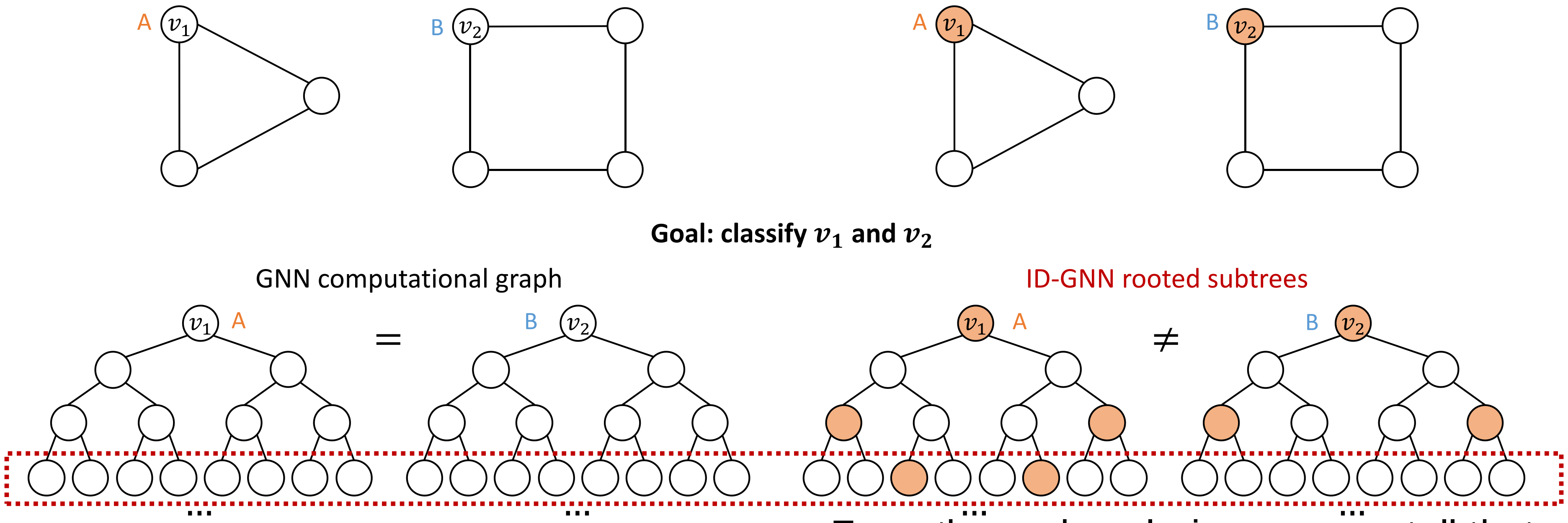
Depending on whether $s = v$ (s is the center node v) or not, we use different neural network functions to transform $\boldsymbol{h}_s^{(k-1)}$.

Identity-aware GNN

- **Why does heterogenous message passing work:**
  - Suppose two nodes v_1, v_2 have the same computational graph structure, but have different node colorings
  - Since we will apply different neural network for embedding computation, their embeddings will be different



GNN vs. ID-GNN

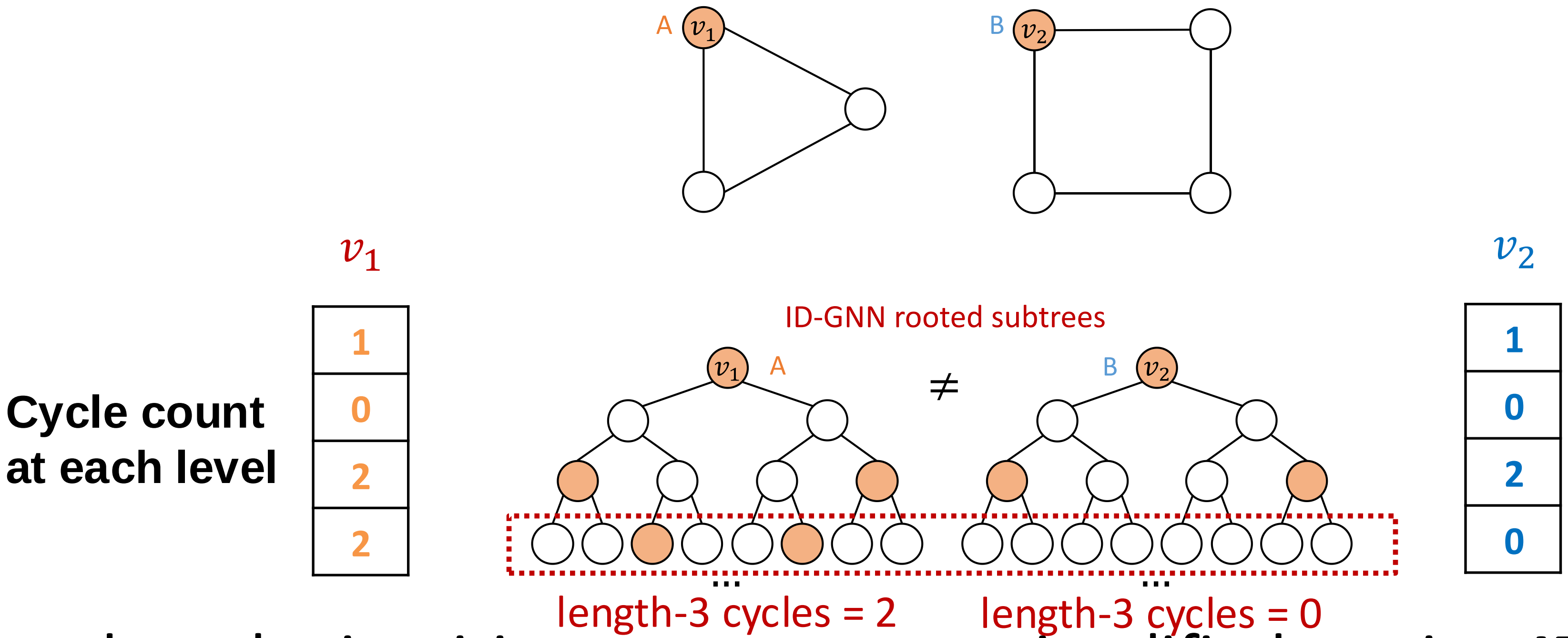


From the node coloring, we can tell that:

v_1: length-3 cycles = 2 v_2: length-3 cycles = 0

- **Why does ID-GNN work better than GNN?**
- **Intuition:** ID-GNN can count cycles originating from a given node, but GNN cannot

Simplified Version: ID-GNN-Fast



- Based on the intuition, we present a simplified version **ID-GNN-Fast**
  - Include identity information as an **augmented node feature** (no need to do heterogenous message passing)
  - **Use cycle counts in each layer as an augmented node feature**. Also can be used together with **any GNN**

Identity-aware GNN

- **Summary of ID-GNN: A general and powerful extension to GNN framework**
  - We can apply ID-GNN on **any** message passing GNNs (GCN, GraphSAGE, GIN, …)
    - ID-GNN provides **consistent performance gain** in node/edge/graph level tasks
  - ID-GNN is more expressive than their GNN counterparts. ID-GNN is the first message passing GNN that is more expressive than 1-WL test
  - We can easily implement ID-GNN using popular GNN tools (PyG, DGL, …)