

Beyond WL: More Expressive GNNs

CPSC483: Deep Learning on Graph-Structured Data

Rex Ying

Readings

- Readings are updated on the website (syllabus page)
- **Lecture 9 readings:**
 - [Graph Isomorphism Network](#)
 - [Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks](#)
- **Lecture 10 readings:**
 - [Position-aware Graph Neural Networks](#)
 - [Identity-aware Graph Neural Networks](#)

Content

- Limitation of Message Passing GNN
- Expressive GNNs beyond WL test
 - Position-aware GNN
 - Identity-aware GNN

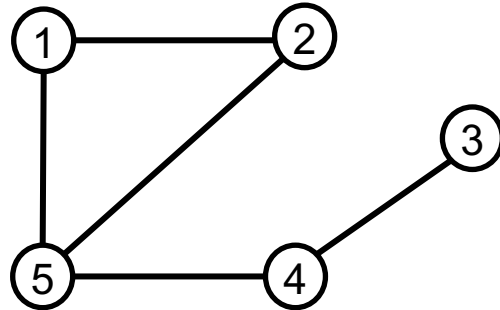
Content

- Limitation of Message Passing GNN
- Expressive GNNs beyond WL test
 - Position-aware GNN
 - Identity-aware GNN

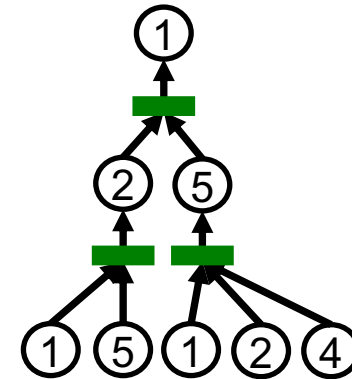
Recap: Computation Graph

- Consider how Message-passing GNN models work.
 - Key concept: **Computation Graphs**
- The **local neighborhoods** define the **computation graphs**
 - Example: computation graph of Node 1 with 2-layer GNN

Graph:



Computation Graph:



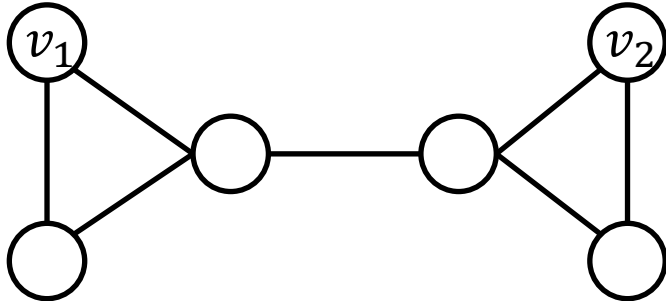
■ are GNN aggregators

- A “perfect” GNN builds an **injective** function between neighborhood structure and node embedding, **generating the same embedding for nodes with the same computation graph**

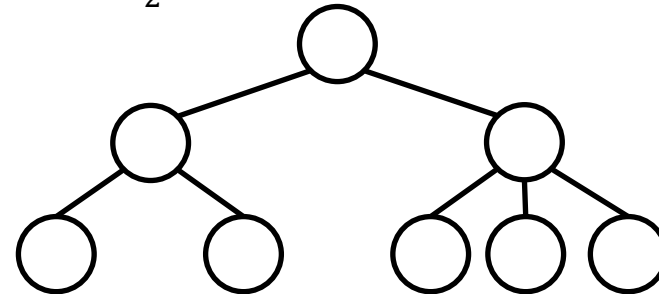
Limitations of GNNs (1)

- **Observation 1:** Message passing GNNs sometimes cannot capture **positional information of nodes**
 - 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 graph**

v_1 and v_2 have the same neighborhood structure but appear in different position



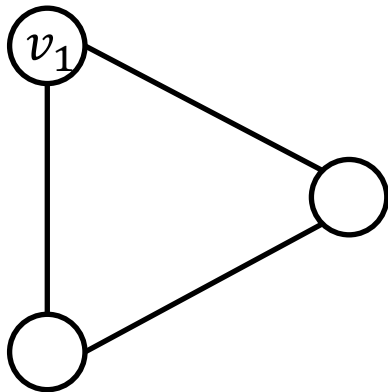
Computation graphs for nodes v_1 and v_2 are the same



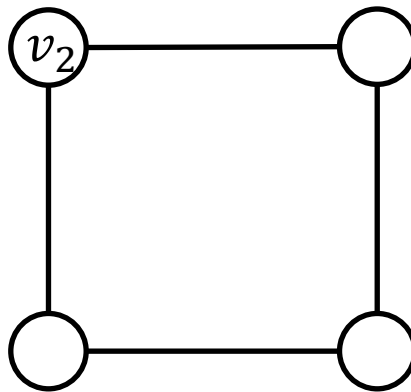
Limitations of GNNs (2)

- **Observation 2:** Different neighborhood structures sometimes create the same computation graph, thus leading to the same node embeddings.
- The GNNs we have introduced so far are **not perfect**
 - The expressive power of message-passing GNNs is upper bounded by WL test
 - Example: message passing GNNs cannot count cycle length

v_1 resides in a cycle with length 3



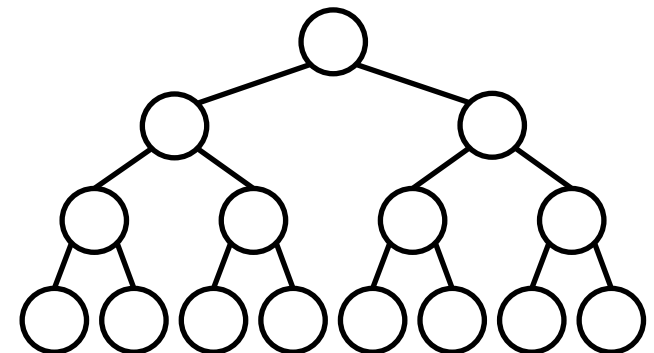
v_2 resides in a cycle with length 4



Computation graph

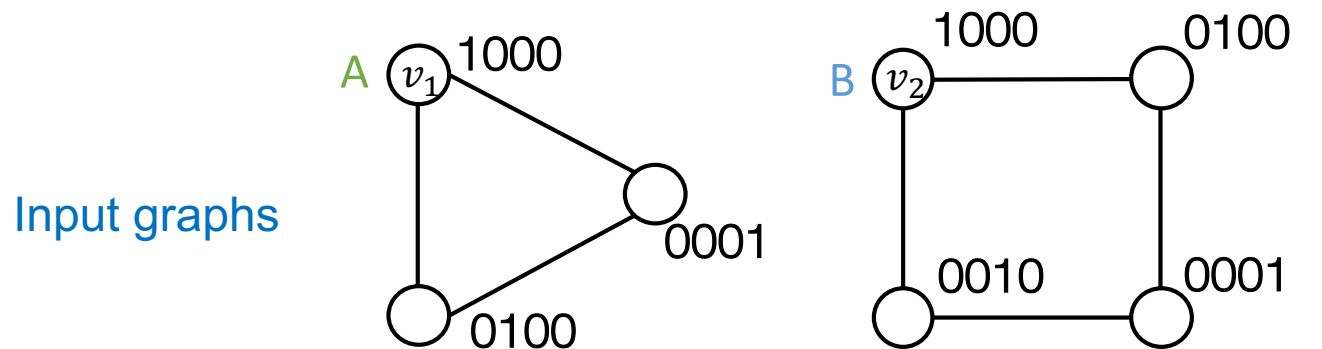


Computation graphs for nodes v_1 and v_2 are always the same

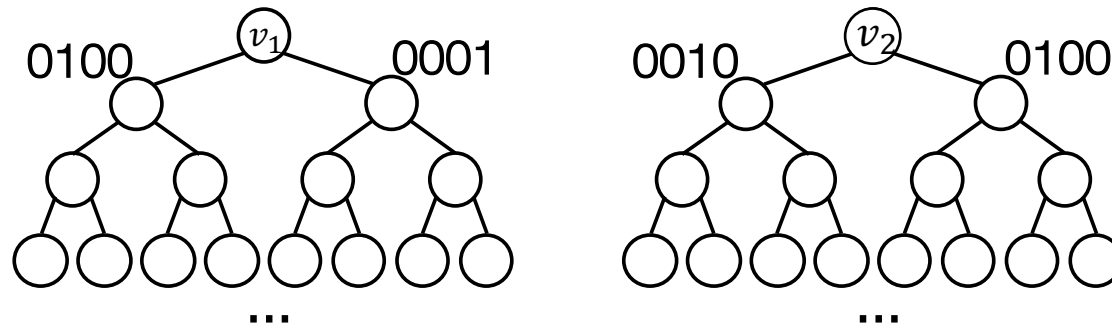


Naïve Solution is not Practical

- A naïve solution to assign different embeddings to nodes: **one-hot encoding**
 - Each node in the graph has a **different ID** (using one-hot encoding). Then we can always differentiate different nodes/edges/graphs.



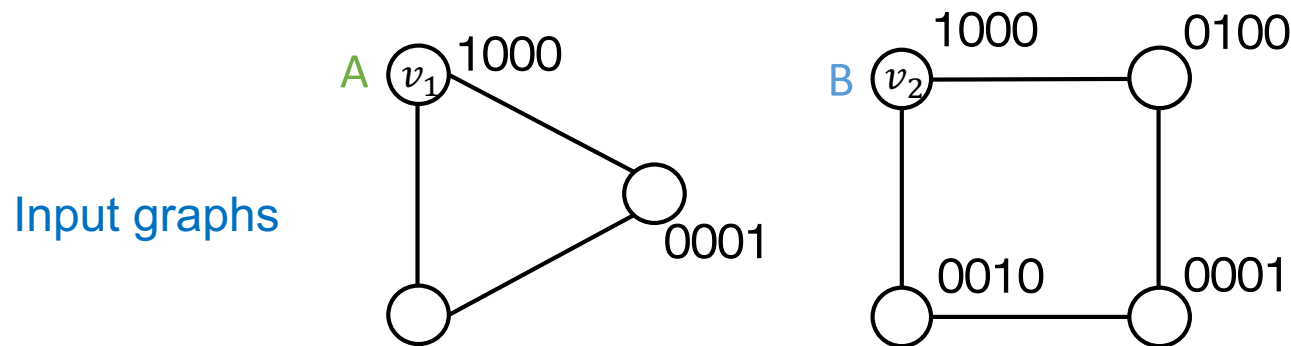
Computation graphs
of Node v_1 and v_2



Computation graphs
are clearly different
if each node has a
different ID

Naïve Solution is not Practical

- A naïve solution to assign different embeddings to nodes: **one-hot encoding**
 - Each node in the graph has a **different ID** (using one-hot encoding). Then we can always differentiate different nodes/edges/graphs.



- **Limitations:**
 - **Not scalable:** Need $O(N)$ feature dimensions (N is the number of nodes)
 - **Not inductive:** Cannot generalize to new nodes/graphs
- **Goal:** Need to look for an **scalable, inductive** approach!

Outline of Today's Lecture

- We will resolve **two limitations of GNNs** 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**

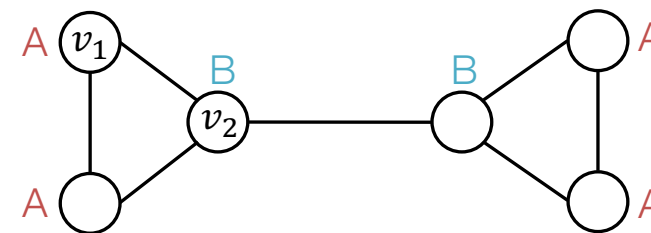
Content

- Limitation of Message Passing GNN
- Expressive GNNs beyond WL test
 - Position-aware GNN
 - Identity-aware GNN

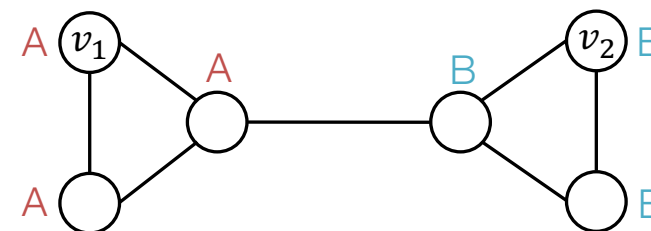
Two Types of Tasks on Graphs

- **Structure-aware Task:** Nodes are labeled by their **structural roles in the graph**
 - v_1 and v_2 have **different labels** because their **different neighborhood structures**
- **Position-aware Task:** Nodes are labeled by their **positions in the graph**
 - v_1 and v_2 have the same neighborhood structures but the different labels, because they **appear in different positions in graph**

Structure-aware task



Position-aware task

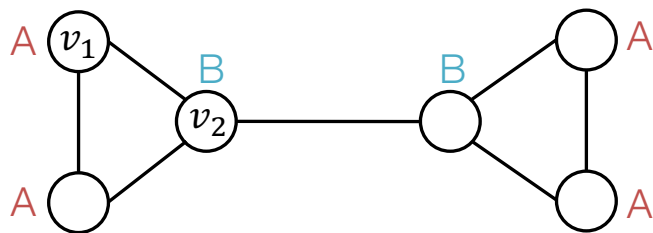


A and B denote different labels

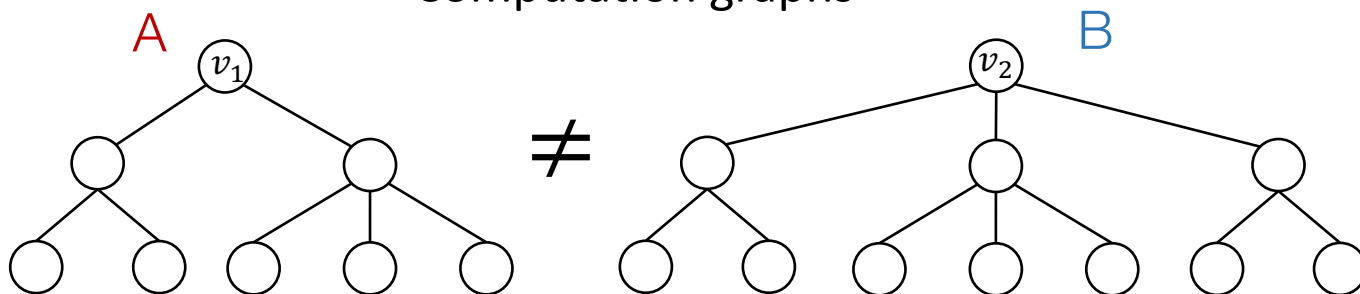
Structure-aware Tasks

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

Structure-aware task



Computation graphs

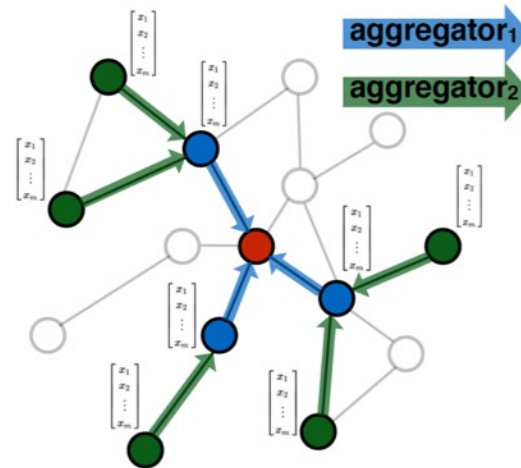
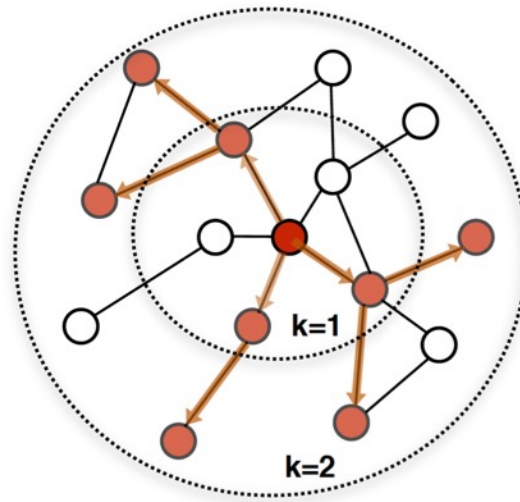


- Message passing GNNs can differentiate v_1 and v_2 by using **different computation graphs**

Structure-aware Embedding

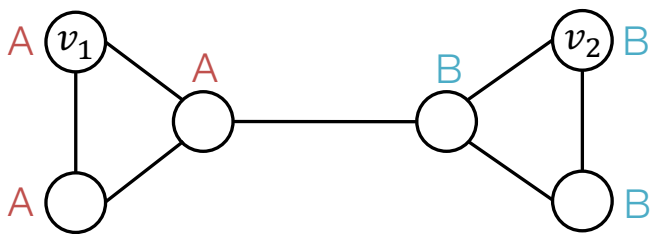
- **Structure-aware Embedding:**

- Embedding $z_i = f_{s_q}(v_i)$ is **structure-aware** if it is a function of up to q -hop neighbourhood of node v_i .
- GNNs that compute embedding by **aggregating q -hop neighborhood** information are **structure-aware**.

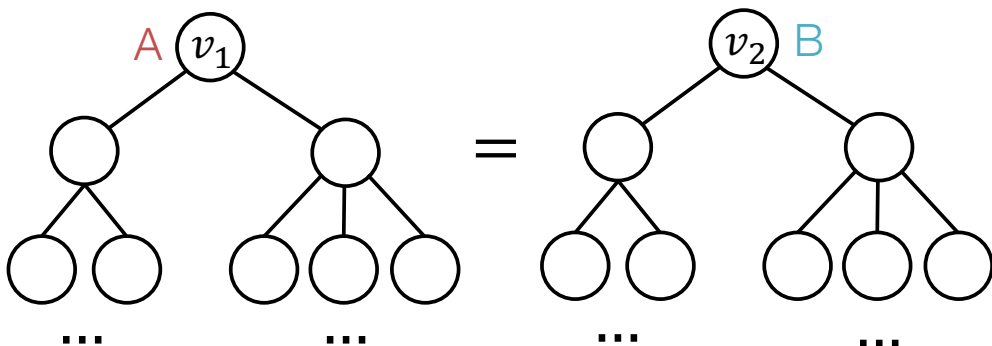


Position-aware Tasks

Position-aware task



Computation graphs



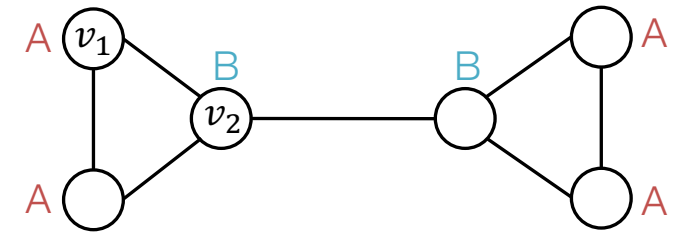
- Message-passing GNNs fail to distinguish v_1 and v_2 due to **the same computation graph**.
- **Observation:** v_1 and v_2 have a **symmetric** neighborhood structure in the graph
- **Key idea of Position-aware GNN:** break the symmetry by using **anchor-set to create difference**
 - Anchor set: a set of nodes that serve as reference points or “coordinates”

Position-aware Embedding

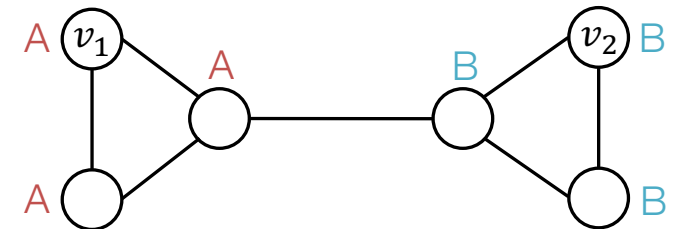
- **Position-aware Embedding:**

- Embedding $z_i = f_p(v_i)$ is **position-aware** if there exists a function $g_p(\cdot, \cdot)$ such that $d_{sp}(v_i, v_j) = g_p(z_i, z_j)$.
- $d_{sp}(\cdot, \cdot)$ is the **shortest path distance** in the graph
- Structure-aware embeddings **cannot** be mapped to position-aware embeddings.
- Structure-aware embeddings are not sufficient for tasks require node positional information
- **P-GNN learns position-aware embedding!**

Structure-aware task

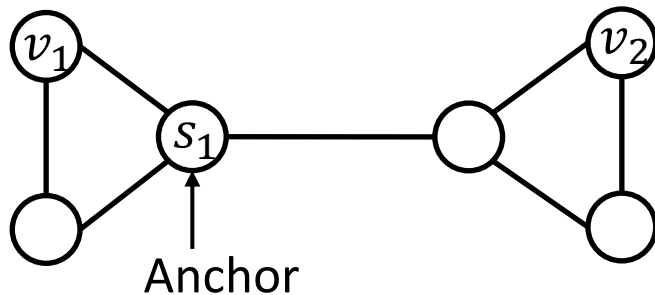


Position-aware task



Power of “Anchors”

- Randomly pick a node s_1 as an anchor node
- Represent v_1 and v_2 via their **relative distance w.r.t. the anchor s_1**
 - Different relative distances create different representations
- Anchor node serves as a **coordinate axis** to locate the target node
 - Example: v_1 and v_2 have **the symmetric position in graph** but have **different relative distance to the anchor s_1**

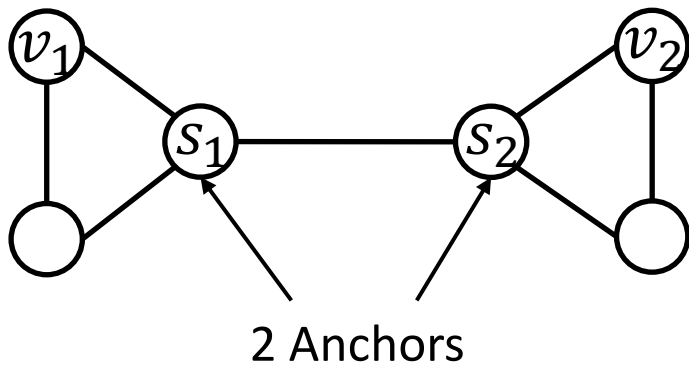


Relative Distances

	s_1
v_1	1
v_2	2

Power of “Anchors”

- Pick more nodes s_1 and s_2 as anchor nodes
- **Observation:** More anchor nodes locate the target node better
- Understand more anchors as more coordinate axes



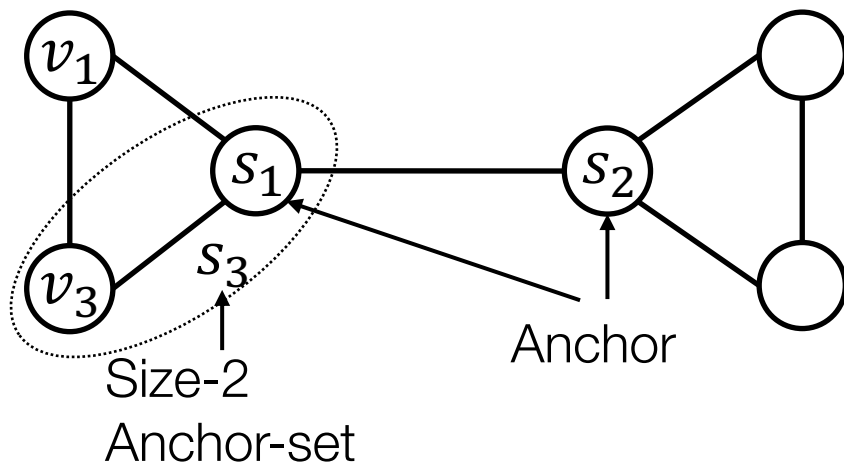
Relative Distances

	s_1	s_2
v_1	1	2
v_2	2	1

Intuitively, more anchor nodes characterize node position better.

Power of “Anchor-sets”

- Generalize anchor from a single node to a **set of nodes** (we call **Anchor-sets**)
 - The distance to an anchor-set is defined as the minimum distance to all the nodes in this anchor-set.
- **Observation:** Large anchor-sets can sometimes provide more precise position estimate



Relative distances to anchor s_1 and s_2 are not sufficient to distinguish node v_1, v_3 .

Relative Distances

	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

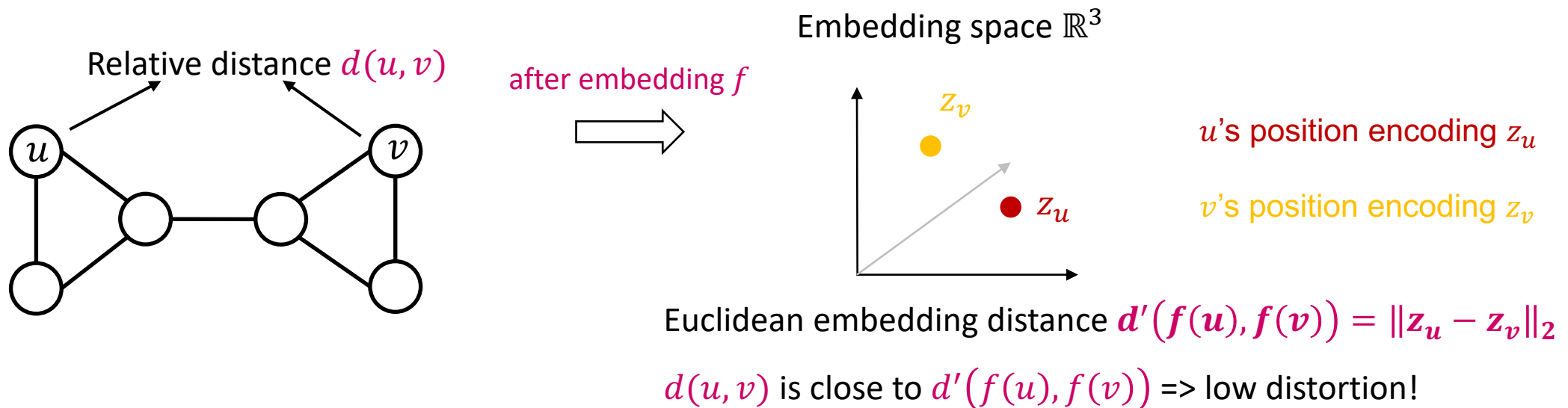
v_1 's Position encoding [1,2,1]

v_3 's Position encoding [1,2,0]

anchor-set s_3 can distinguish node v_1, v_3 .

Anchor Set: Theory

- **Goal:** Embed **the metric space** (V, d) into the Euclidian space \mathbb{R}^k and try to preserve the original distance metric.
- Node embedding should have **low distortion**.
 - **Distortion** measures the faithfulness of the embeddings in **preserving distances when mapping from the original metric space to another metric space** (\mathbb{R}^k)



Anchor Set: Theory

- **Definition:**

The embedding f has distortion α if : $\frac{1}{\alpha} d(u, v) \leq d'(f(u), f(v)) \leq d(u, v)$, for any u, v in the graph.

- **Bourgain Theorem:**

- Given any finite metric space (V, d) with $|V| = n$, there **exists** an embedding of (V, d) into \mathbb{R}^k under any l_p metric, where $k = \mathcal{O}(\log^2 n)$, and the distortion α of the embedding is $\mathcal{O}(\log n)$.
- k is the dimension of embedding space and n is the number of nodes.
- We use **constructive proof** to prove Bourgain Theorem

Bourgain Theorem

- Consider the following **embedding function** f of node $v \in V$

$$f(v) = \left(\frac{d(v, S_{1,1})}{k}, \frac{d(v, S_{1,2})}{k}, \dots, \frac{d(v, S_{\log n, c \log n})}{k} \right)$$

- Where
 - c is a constant,
 - $S_{i,j} \subset V$ is chosen by including each node in V **independently with probability** $\frac{1}{2^i}$,
 - $d(v, S_{i,j}) = \min_{u \in S_{i,j}} d(v, u)$.
- Then, f is an embedding method satisfies Bourgain Theorem.
- The embedding distance produced by f is provably close to the original distance metric (V, d) .

Position Information

- P-GNN follows the Bourgain theorem

- Sample anchor sets $S_{i,j}$ for each graph
- Embed each node v via

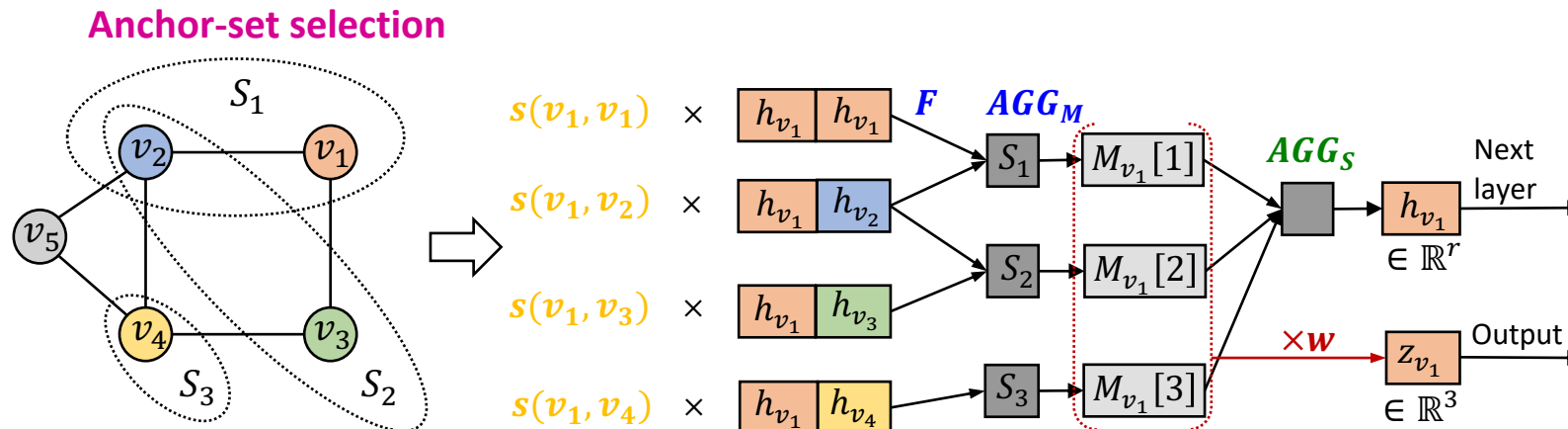
$$f(v) = \left(\frac{d(v, S_{1,1})}{k}, \frac{d(v, S_{1,2})}{k}, \dots, \frac{d(v, S_{\log n, c \log n})}{k} \right) \in \mathbb{R}^{c \log^2 n}$$

- Each dimension of the position encoding is tied to an anchor-set
- P-GNN maintains the **inductive capability**
 - During training, new anchor sets are re-sampled every time.
 - At test time, given a new unseen graph, new anchor sets are sampled

Overview of Position-aware GNNs (P-GNNs)

- (a) Randomly select anchor-sets (Anchor-set selection)
- (b) Compute pairwise node distances ($s(v, u)$)
- (c) Compute messages from anchor-sets (F, AGG_M)
- (d) Transform messages to embeddings (AGG_S, w)

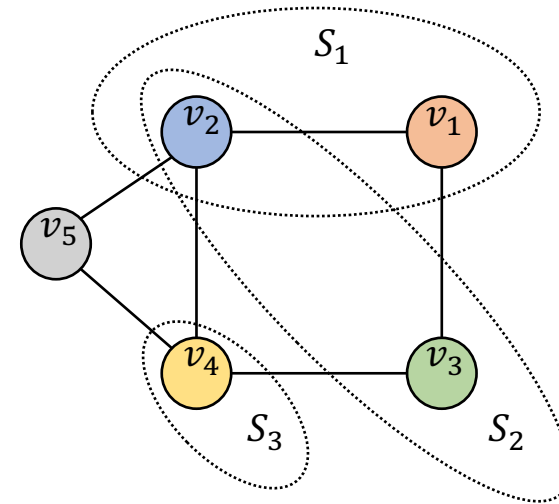
Steps of Embedding computation for node v_1 :



Selection of Anchor-set

Step(a):

- Randomly choose anchor-sets with sizes from 1, 2, 4, ..., $n/2$ (**$\log n$** number of sizes)
- For each size of anchor-set, repeat **$c \log n$** times
- In total, **$k = c \log^2 n$** anchors

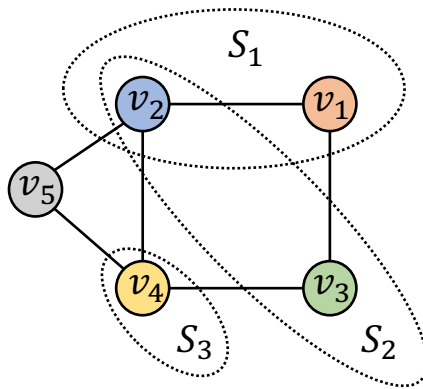


Pairwise Node Distance

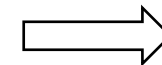
Step(b):

- Compute pairwise node distances
- Position-based similarities: shortest path, personalized PageRank, etc.
- Use **k-hop shortest path distance** $d_{sp}^k(v_i, v_j)$ for fast computation

$$d_{sp}^k(v, u) = \begin{cases} d_{sp}(v, u), & \text{if } d_{sp}(v, u) \leq k \\ \infty, & \text{otherwise} \end{cases}$$



$$s(v_i, v_j) = \frac{1}{d_{sp}^k(v_i, v_j) + 1}$$



pairwise node distances $s(v_i, v_j)$

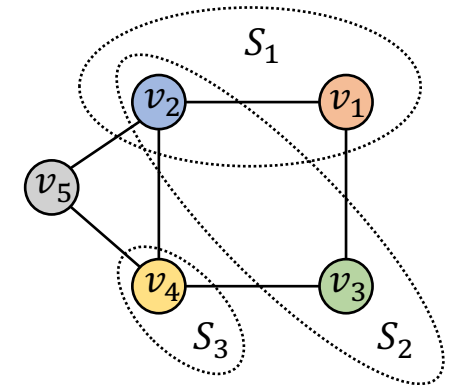
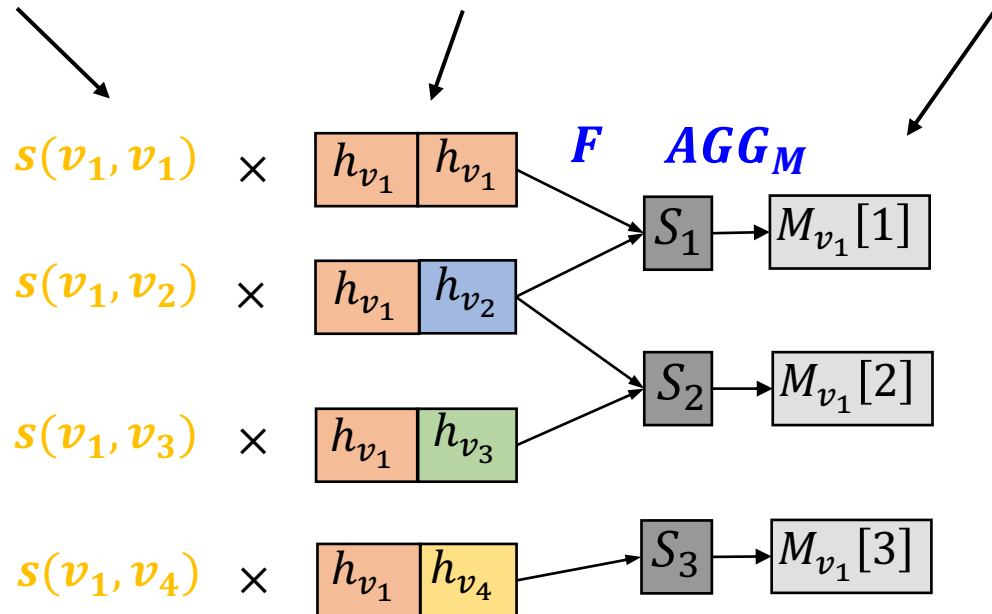
	v_1	v_2	v_3	v_4	v_5
v_1	1	0.5	0.5	0.3	0.3
v_2	0.5	1	0.3	0.5	0.5
v_3	0.5	0.3	1	0.5	0.3
v_4	0.3	0.5	0.5	1	0.5
v_5	0.3	0.5	0.3	0.5	1

Messages from Anchor-set

Step(c):

- Compute messages from anchor-sets

Position info + Feature info \rightarrow Messages from anchor-sets



(1) Combine position and feature

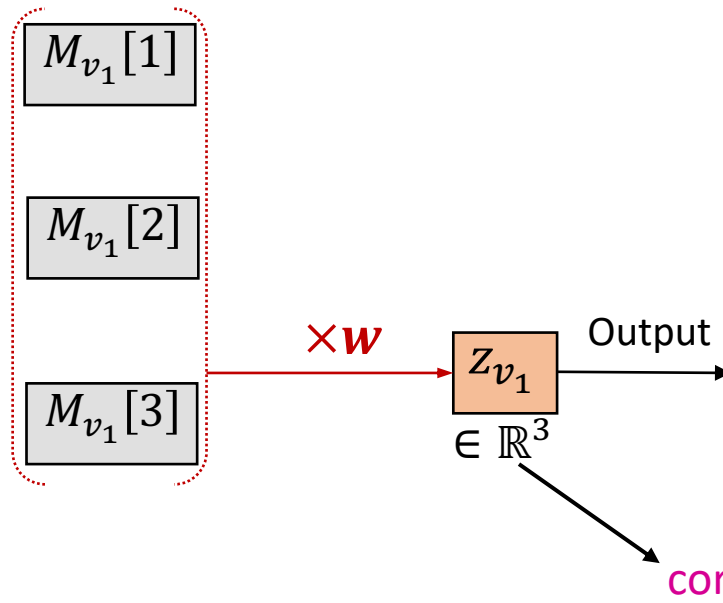
$$F(v, u, \mathbf{h}_v, \mathbf{h}_u) = s(v, u) \text{CONCAT}(\mathbf{h}_v, \mathbf{h}_u)$$

(2) GNN style aggregation
(over nodes in an anchor-set)

$$\text{AGG}_M(\{\mathbf{x}_1, \dots, \mathbf{x}_n\}) = \mathbf{W}_2 \cdot \text{MEAN}(\{\sigma(\mathbf{W}_1 \cdot \mathbf{x}_i + \mathbf{b}_1)\}) + \mathbf{b}_2$$

Transform to Embeddings

- **Step(d):**
- Transform messages to embeddings (2 parts)
 - **Output component (position-aware embedding):**

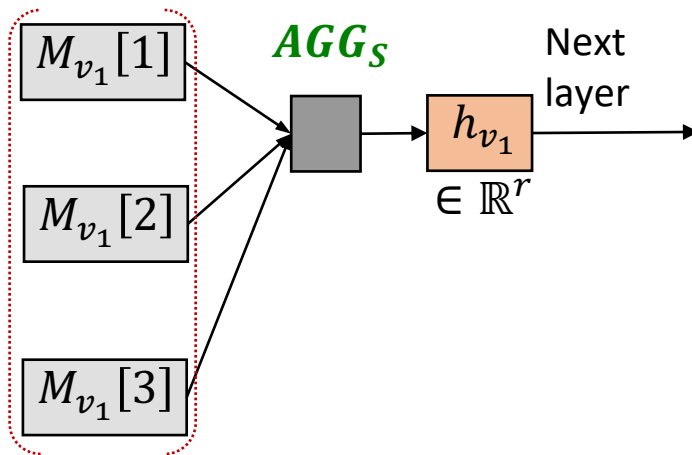


(1) Position-aware Embedding z_{v_1} :

- **Model's output for prediction**
- Transform each message to scalar
- **Each dimension is tied with an anchor-set, thus is position-aware**

Transform to Embeddings

- **Step(d):**
- Transform messages to embeddings (2 parts)
 - Input to the next layer (structure-aware embedding):

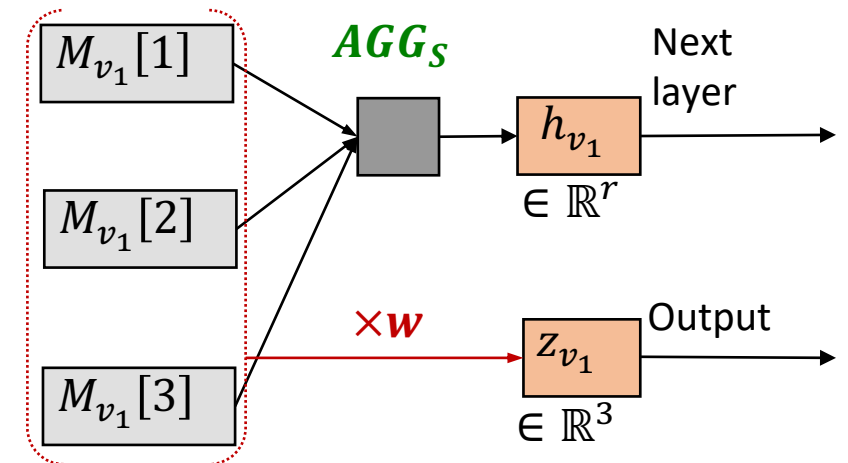


(2) Structure-aware Embedding h_{v_1} :

- **Fed into next layer of P-GNN**
- **Order-invariant message aggregation AGG_S** aggregates messages in GNN manner
- **Each dimension is independent of anchor-set selection.**

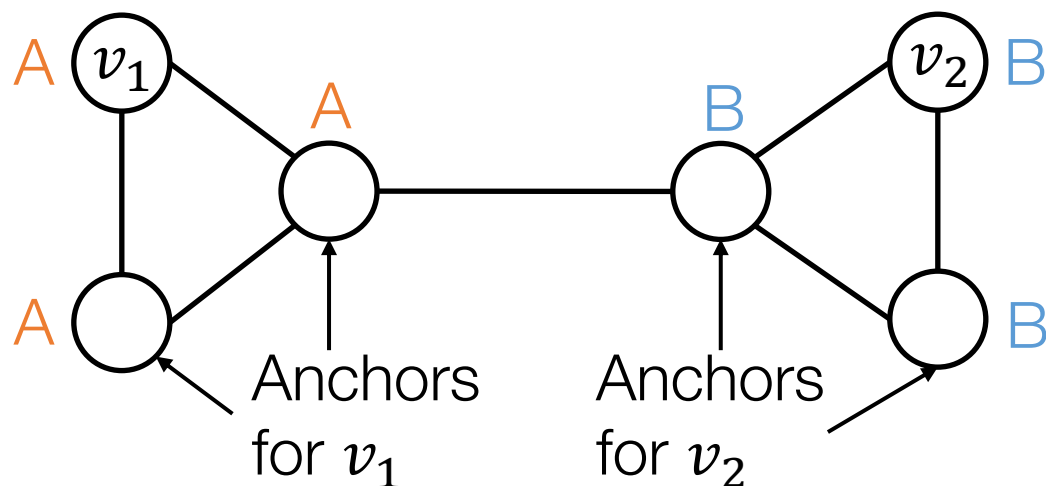
Transform to Embeddings

- The resulting model can potentially **capture both structure and position information**
- Model outputs **position-aware embedding z_{v_i}** for downstream tasks
- **Structure-aware embedding h_{v_i}** is fed to the next layer of model



Connection to GNNs

- Anchor: What you refer to when you compute (position-aware) embeddings
- For normal **GNNs**, each node **independently** selects **its own anchors** to aggregate information
 - Example: GNNs look at the **k-hop neighbor nodes** when compute an embedding
 - $s(v_i, v_j) = 1$ iff $d_{sp}^k(v_i, v_j) = k$ for k-hop GNNs.

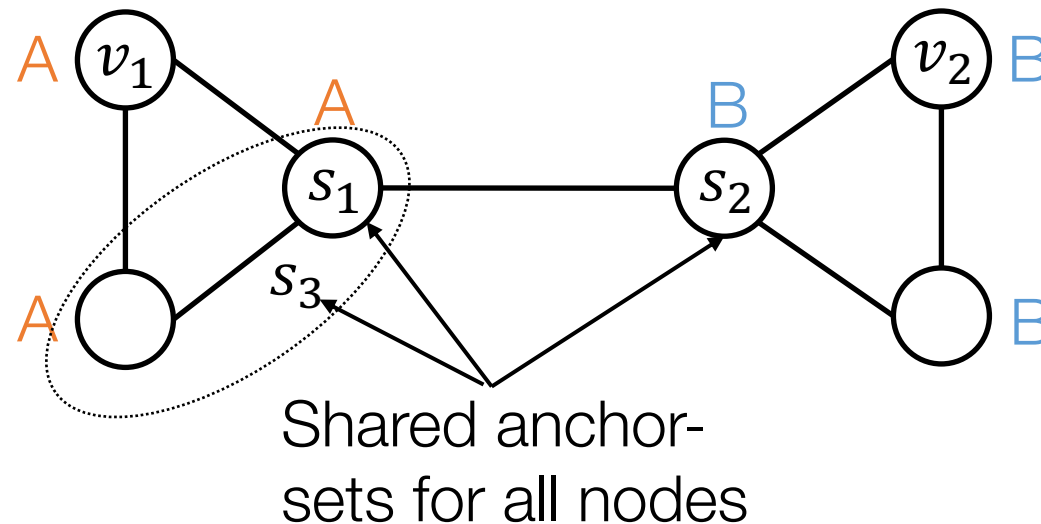


$$s(v_i, v_j) = \frac{1}{d_{sp}^k(v_i, v_j) + 1}$$

	v_1	v_2	v_3	v_4	v_5
v_1	1	0.5	0.5	0.3	0.3
v_2	0.5	1	0.3	0.5	0.5
v_3	0.5	0.3	1	0.5	0.3
v_4	0.3	0.5	0.5	1	0.5
v_5	0.3	0.5	0.3	0.5	1

Connection to GNNs

- While for P-GNNs, each node aggregates information from **shared** anchor-sets
- That's why **anchor-sets** can serve as **coordinate axes**
 - Example: P-GNNs look at shared anchor sets to aggregate neighborhood information



Ways 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!

Ways to Use Position Information

- **The rigorous 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
- Refer to the [Position-aware GNN](#) paper for more details

P-GNN vs. GNN

- Two families of models

	P-GNNs	GNNs
Message aggregation	Mean, Sum, Attention, ...	
Neighborhood selection	Shared anchor selection	Independent anchor selection
Embedding computation	Each dimension is tied to an anchor	Each dimension is aggregated across anchors
Embedding property	Position -aware	Structure -aware

Position-aware GNNs: Summary

- **Position-aware GNNs:** A new class of GNNs that incorporate node positional information
- **Key idea:** select shared anchor-sets as coordinate axes for all the nodes
- Position-aware GNNs work well for position-aware tasks

Content

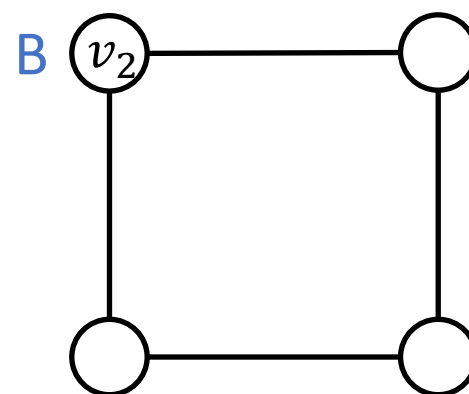
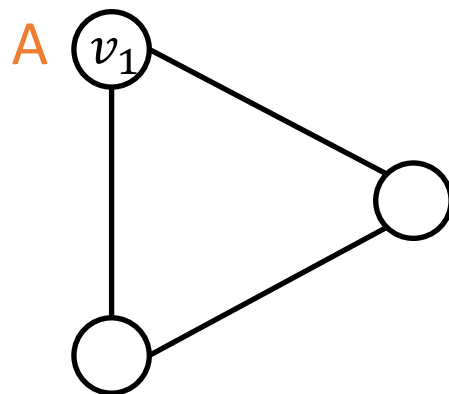
- Limitation of Message Passing GNN
- Expressive GNNs beyond WL test
 - Position-aware GNN
 - Identity-aware GNN

Limitation of GNNs

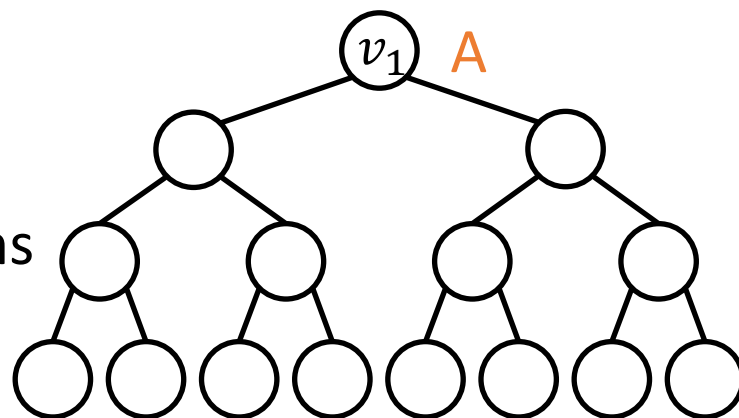
- We learned that GNNs would fail for position-aware tasks
- **Question:** Can GNN always perform perfectly in structure-aware tasks?
 - **No!** The expressive power of GNNs we have introduced so far is **upper bounded by WL test.** (lecture 9)
- GNNs exhibit (at least) three levels of failure cases in structure-aware tasks
 - Node level
 - Edge level
 - Graph level
- Main cause of failure cases: **the same computation graph**

GNN Failure 1: Node-level Tasks

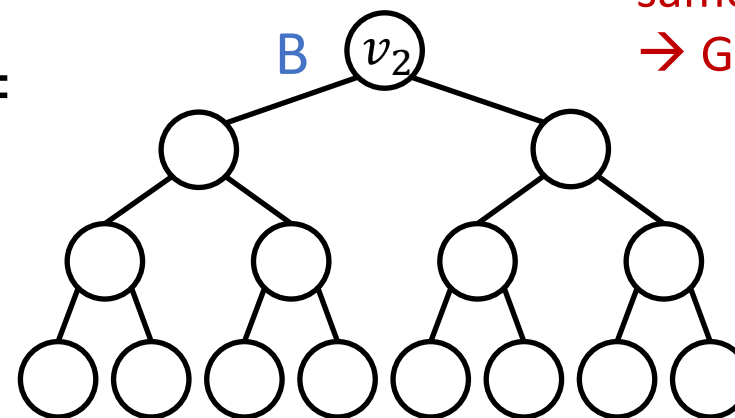
Example input graphs



Existing GNNs' computation graphs



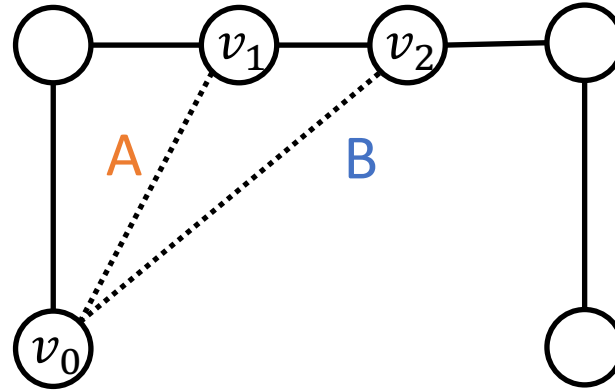
=



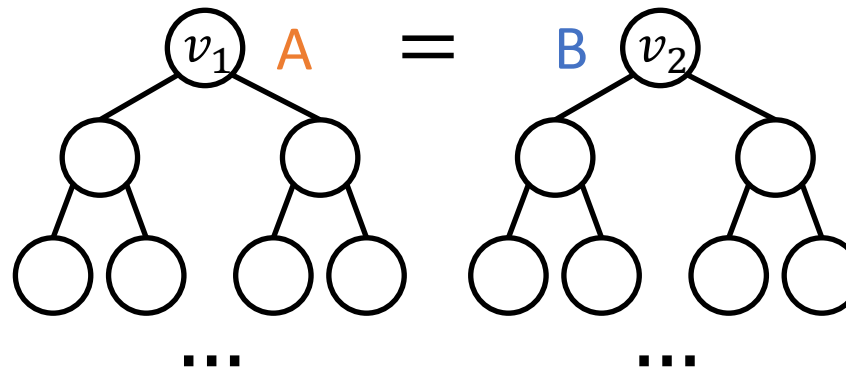
Different Inputs but the same computation graph
→ GNN fails

GNN Failure 2: Edge-level Tasks

Example input graph



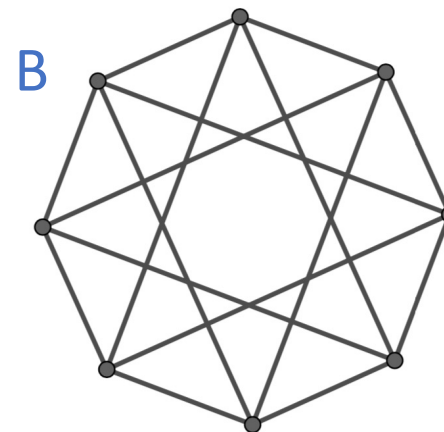
Existing GNNs' computation graphs



Different Inputs but the same computation graph
→ GNN fails

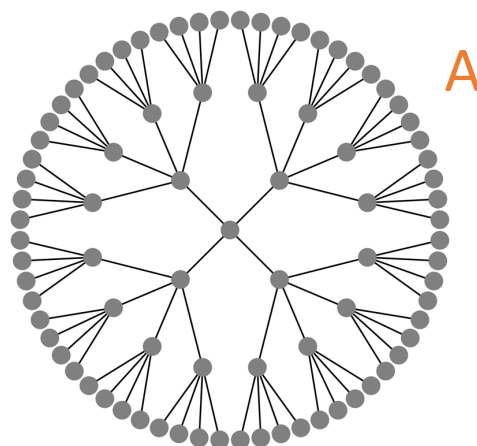
GNN Failure 3: Graph-level Tasks

Example input graph

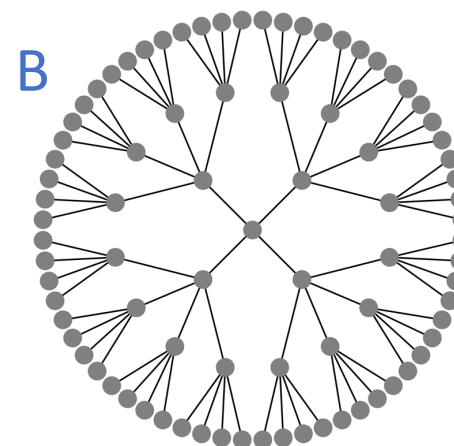


Existing GNNs' computation graphs

For each node:



For each node:

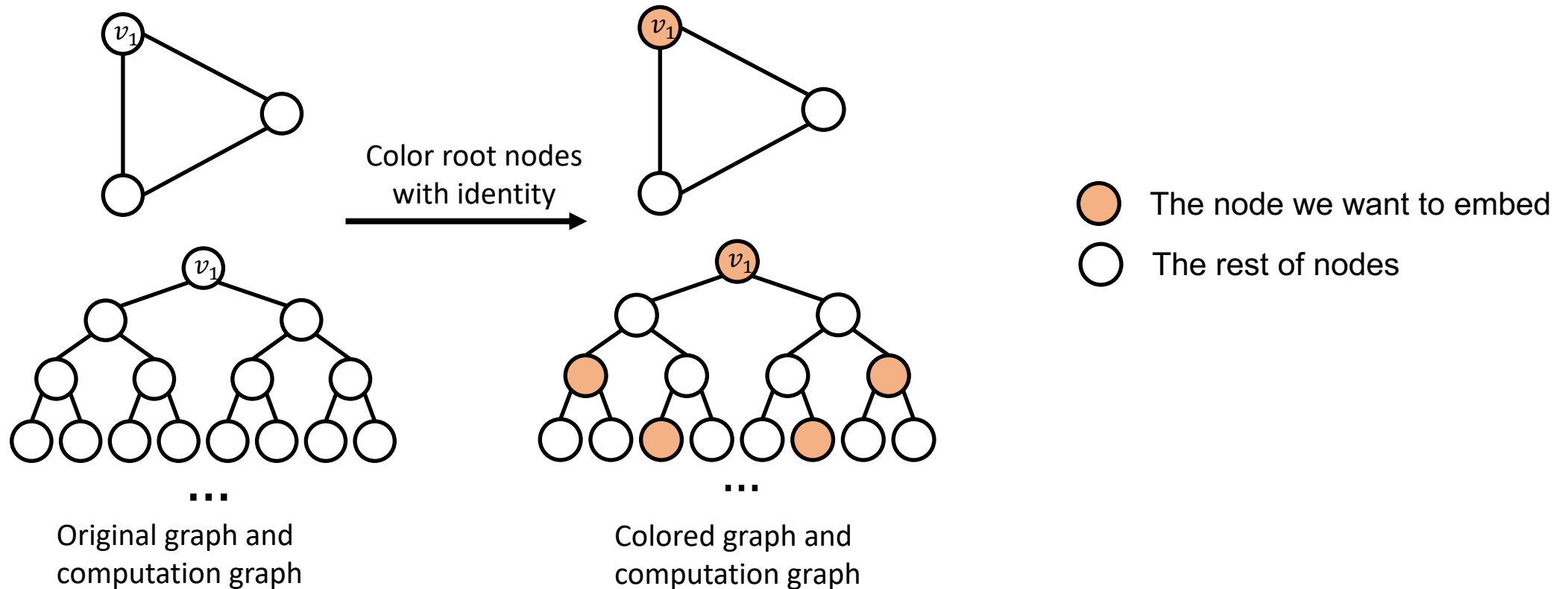


=

Different Inputs but the same computation graph
→ GNN fails

Idea of Identity-aware GNN

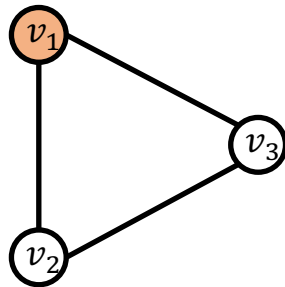
- Idea: assign a color to the node we want to embed



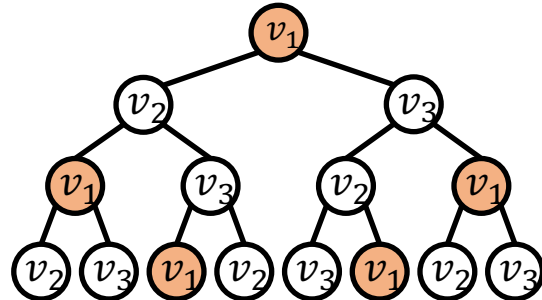
Inductive Node Coloring

- This coloring is **inductive**:
 - It is **invariant** to node ordering/identities
 - **Easily generalize** to unseen graphs

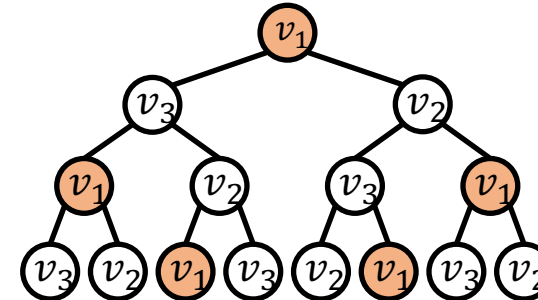
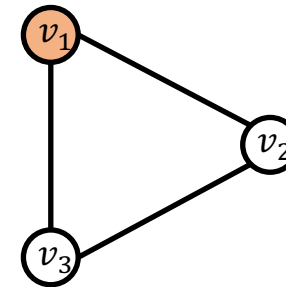
Example inputs graphs



Computation graph of colored graph



Permute the node ordering
between v_2 and v_3

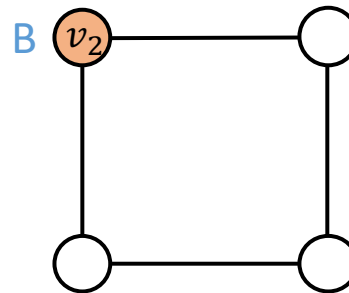
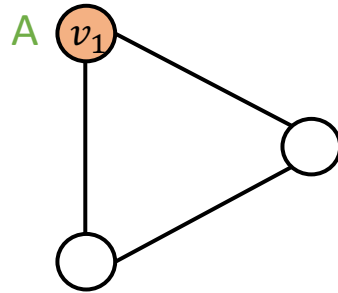


The computation graph stays the same

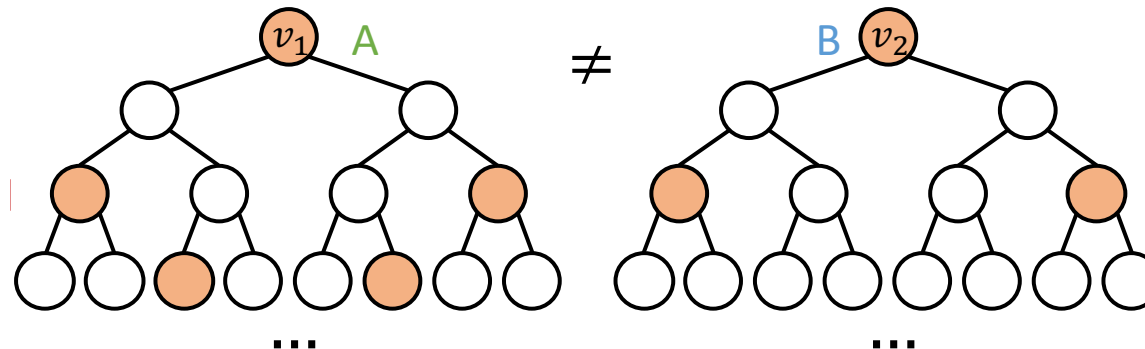
Inductive Node Coloring – Node level

- Inductive node coloring can help **node classification**
 - We color root nodes with identity

Example inputs graphs



Computation graph
of colored graph

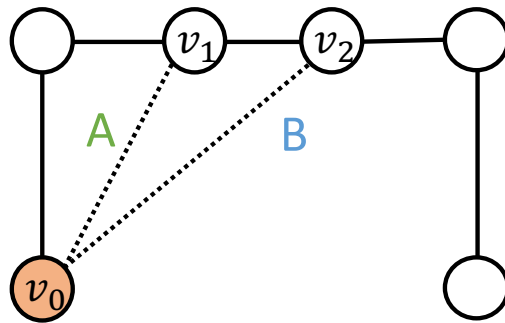


Different computation graphs
Successfully differentiate nodes

Inductive Node Coloring – Edge level

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

Example inputs graphs

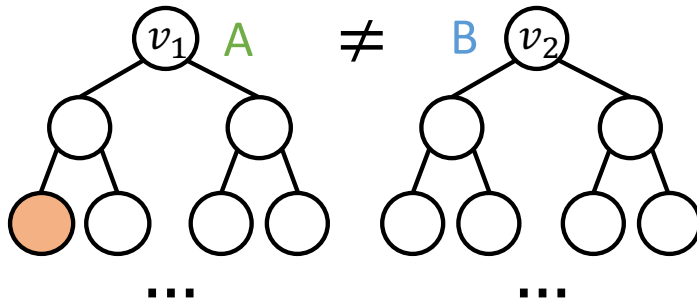


Edge **A** and **B** share node v_0 .
We look at embeddings for v_1 and v_2 .

Without node coloring:

- the computation graphs of v_1 and v_2 are the same.

Computation graph
of colored graph



With node coloring (v_0):

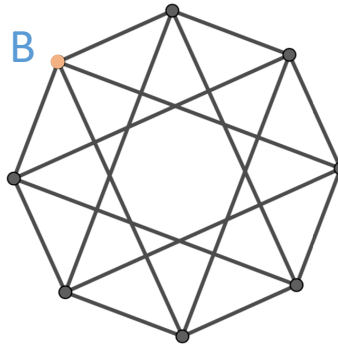
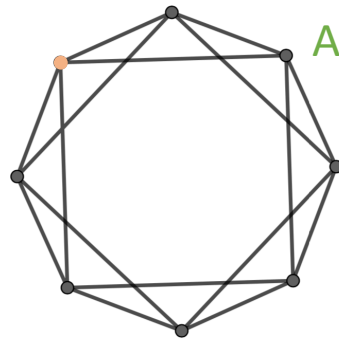
- We **embed the other node in the node pair (v_1 or v_2)**
- We use the **node embedding for v_1 or v_2 conditioned on v_0 being colored or not** to make edge-level prediction

Different computation graphs successfully differentiate edges

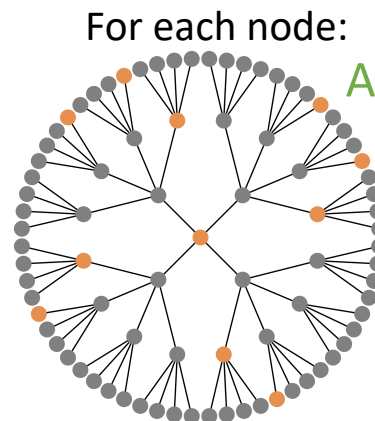
Inductive Node Coloring – Graph Level

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

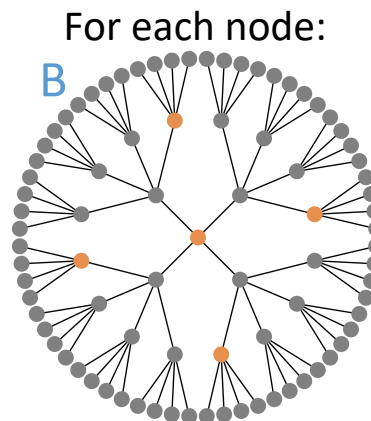
Example inputs graphs



Computation graph
of colored graph



\neq

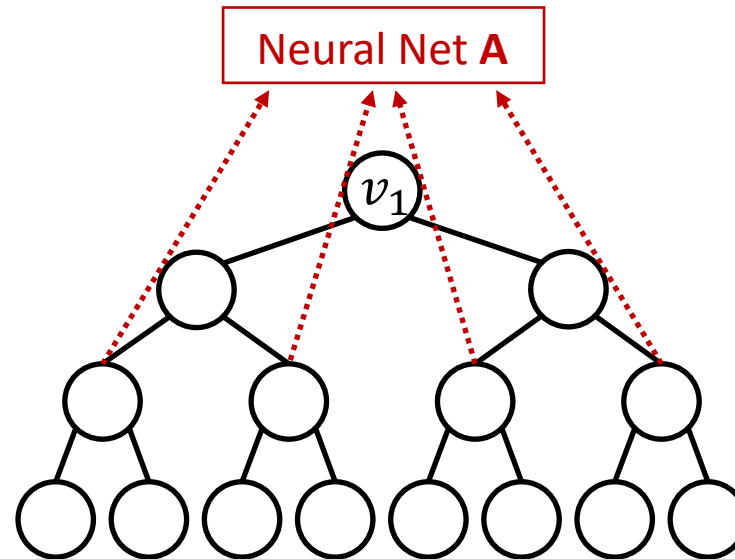


Different computation graphs
Successful differentiate graphs

Heterogenous Message Passing

- How to build a GNN using node coloring?
- **Idea: Heterogenous message passing**
 - Normally, a GNN applies the same message/aggregation computation to all the nodes

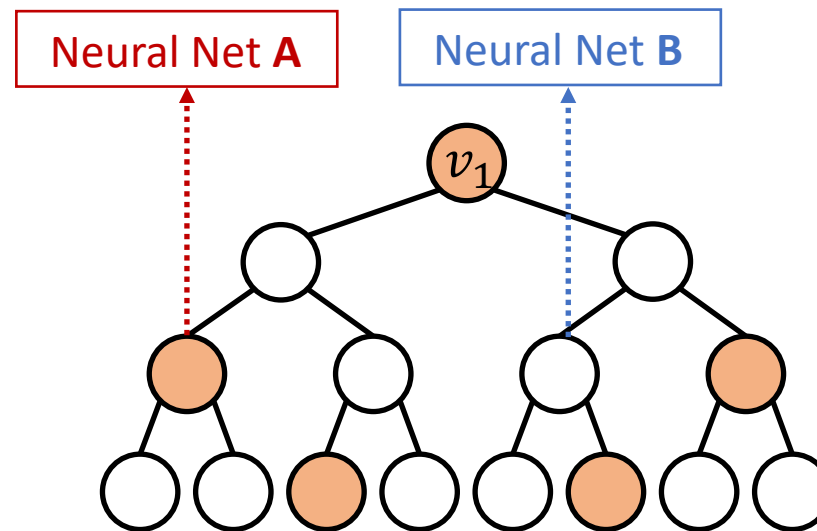
GNN:



Heterogenous Message Passing

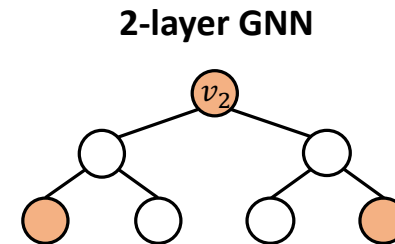
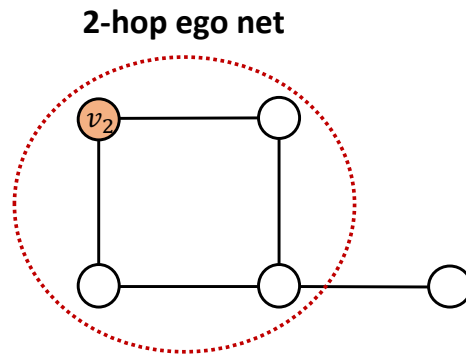
- **Idea:** Heterogenous message passing
 - **Heterogenous:** different types of message passing is applied to different nodes
 - An ID-GNN applies different neural networks to nodes with different colorings

ID-GNN



Identity-aware GNN Algorithm

- **Computation of Node v embedding:**
- **Step 1: extract the ego-network**
 - $\mathcal{G}_v^{(K)}$: K -hop neighborhood graph around v
 - Set the initial node feature to each node in $\mathcal{G}_v^{(K)}$
 - Example:



- We extract the ego net for each node, and initialize node embedding by raw node features.
- Ego net can be used to determine the GNN computation graph for a node

Identity-aware GNN Algorithm

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

$$h_u^{(k)} = \text{AGG}^{(k)} \left(\left\{ \text{MSG}_{\mathbb{I}[s=v]}^{(k)} \left(h_s^{(k-1)} \right), s \in \mathcal{N}(u) \right\}, h_u^{(k-1)} \right)$$

- $h_u^{(k)}$ is the **embedding** of node u at the k -th layer.
- $\text{AGG}^{(k)}$ is the **aggregator** of the k -th layer.
- $\mathbb{I}[s = v] = 0$ if $s = v$ else 0.
- **Message passing equation** $\text{MSG}_0^{(k)}$ for **nodes with identity coloring** and $\text{MSG}_1^{(k)}$ for the rest of nodes.
- $\mathcal{N}(u)$ is the set of **neighborhood nodes** of u .

Identity-aware GNN Algorithm

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

Depending on whether $s = v$ (s is the center node v) or not, we use different neural network functions to transform $h_s^{(k-1)}$.
 $MSG_1^{(k)}$ for central node v and $MSG_0^{(k)}$ for others.

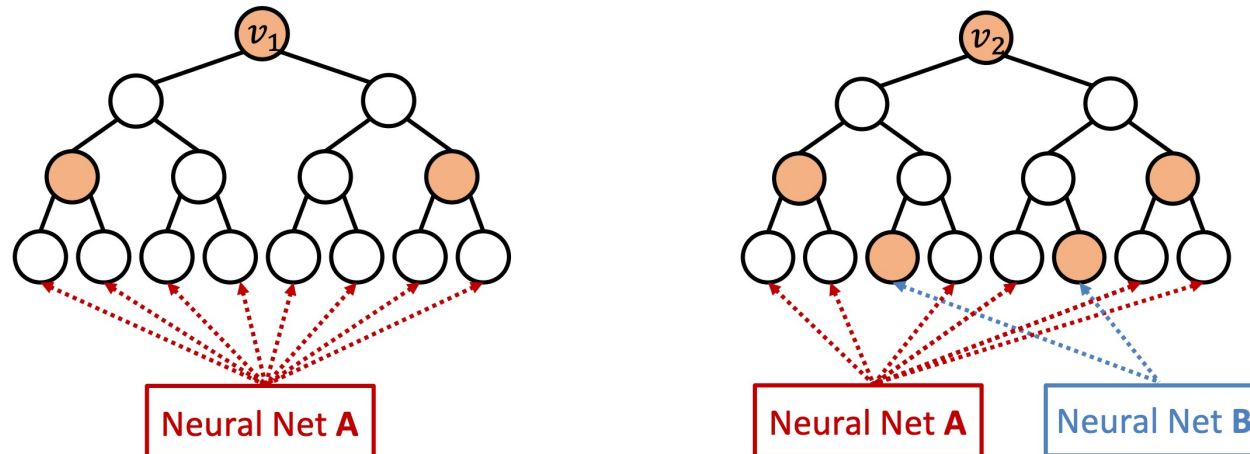
$$h_u^{(k)} = \text{AGG}^{(k)} \left(\left\{ \text{MSG}_{\mathbb{I}[s=v]}^{(k)} \left(h_s^{(k-1)} \right), s \in \mathcal{N}(u) \right\}, h_u^{(k-1)} \right)$$

- $h_u^{(k)}$ is the **embedding** of node u at the k -th layer.
- $\text{AGG}^{(k)}$ is the **aggregator** of the k -th layer.
- $\mathbb{I}[s = v] = 0$ if $s = v$ else 0.
- **Message passing equation** $MSG_0^{(k)}$ for **nodes with identity coloring** and $MSG_1^{(k)}$ for the rest of nodes.
- $\mathcal{N}(u)$ is the set of **neighborhood nodes** of u .

Identity-aware GNN

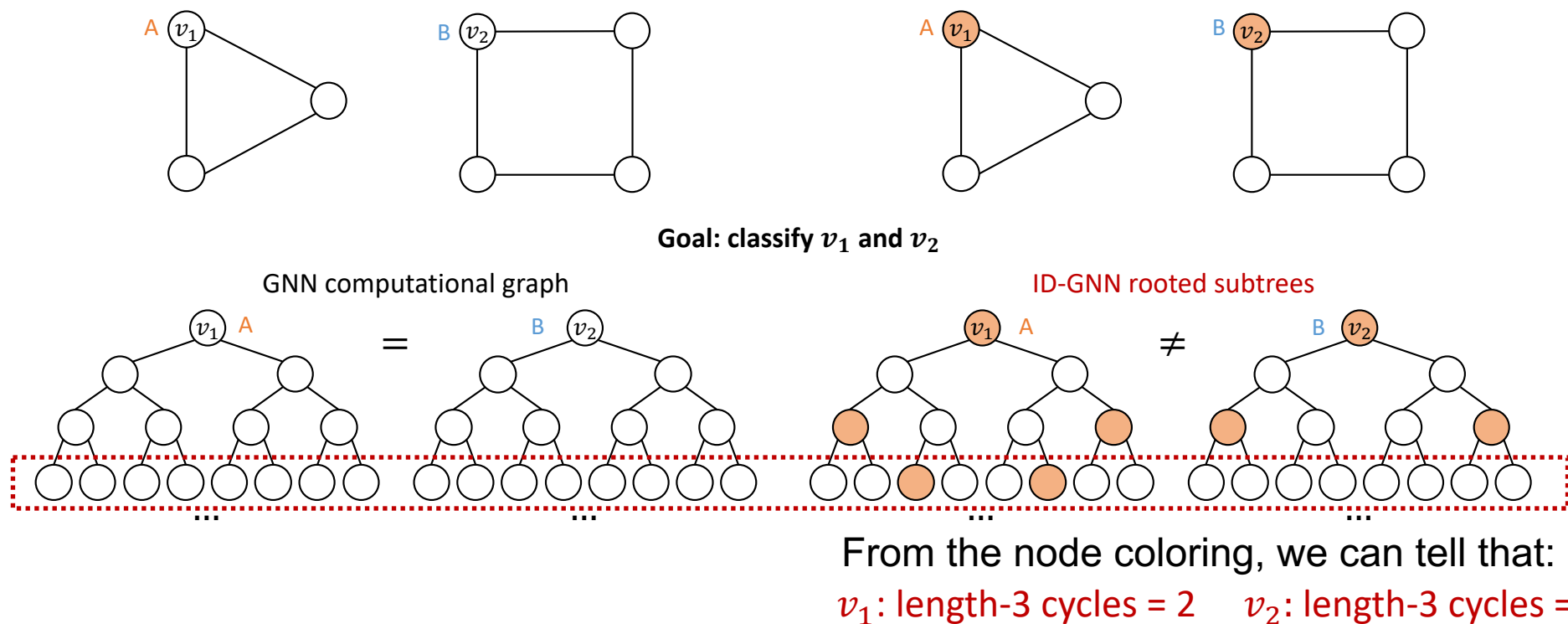
- **Why does heterogenous message passing work?**

- Suppose two nodes v_1, v_2 have the same computation graph structure, but have different node colorings
- Since we will **apply different neural network for embedding computation**, their embeddings will be different (implement via $MSG_{\mathbb{I}[s=v]}^{(k)}$)
- Thus can distinguish nodes even with the same computation graphs



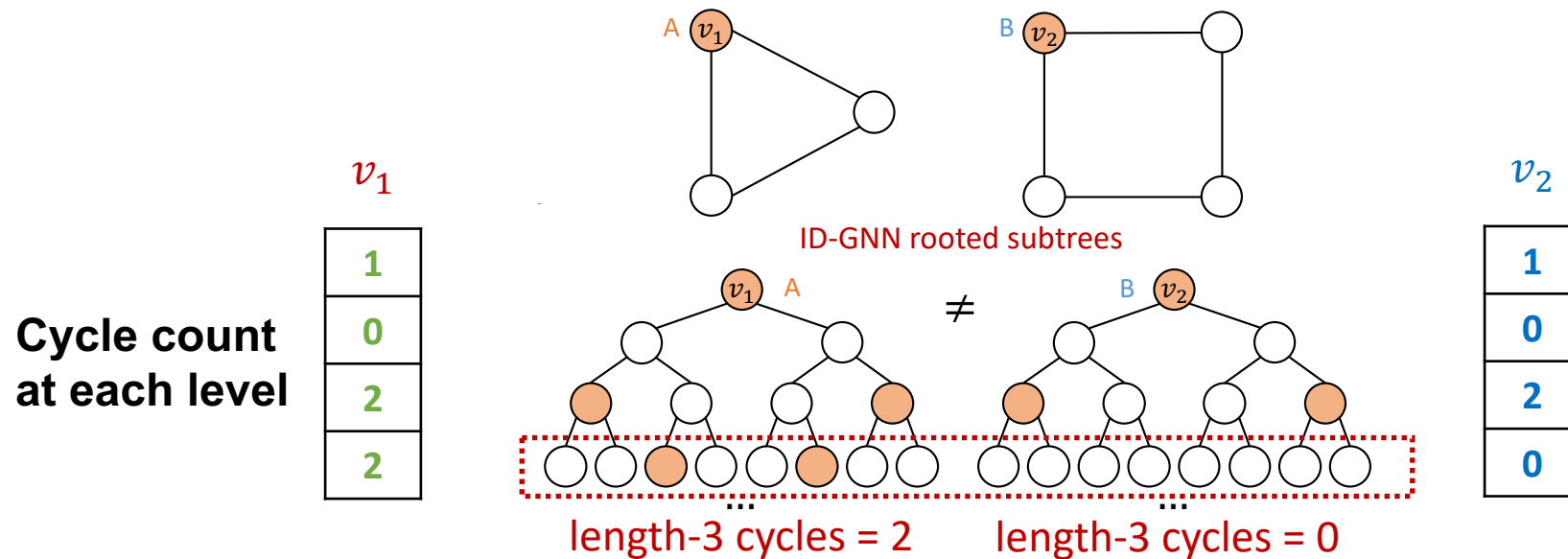
GNN vs Identity-aware GNN

- 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

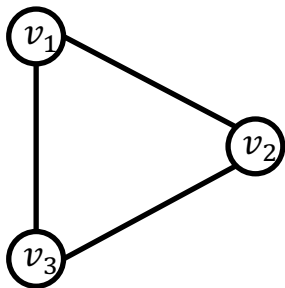
- Based on the intuition, we propose a simplified version **ID-GNN-Fast**
 - Include identity information as an **augmented node feature** (no need to do heterogenous message passing)
 - Use cycle counts in each layer as an augmented node feature.** Also can be used together with **any GNN**



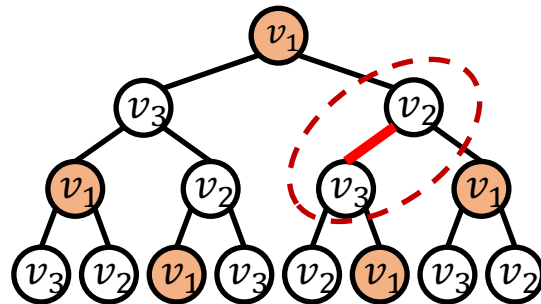
Limitation of Identity-aware GNN

- ID-GNN usually must be implemented with **minibatch training**.
 - The root of one computation tree is no longer the root of another computation tree
 - ID-GNN uses heterogeneous message passing
 - Message type of the same edge is different for different computation trees
 - Less efficient compared to full batch matrix multiplication

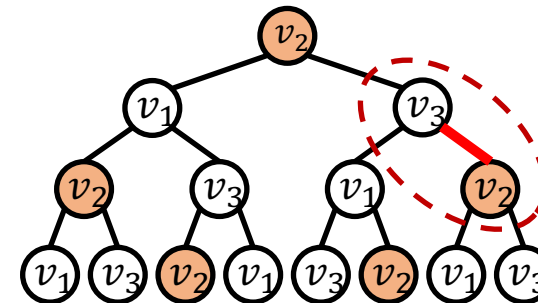
Original graph



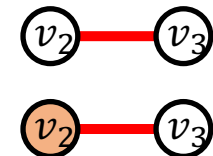
Computation graph of Node v_1



Computation graph of Node v_2



Same edge, but different message types in different computation graphs



Identity-aware GNN: Summary

- **ID-GNN:** A general and powerful extension to GNN framework
 - We can apply ID-GNN on **any message passing GNNs** (GCN, GraphSAGE, GIN, ...)
 - We can **easily implement** ID-GNN using popular GNN tools (PyG, DGL, ...)
- **ID-GNN:** A general solution to the limitations of expressive power in existing message passing GNNs
 - ID-GNN has **expressive power beyond 1-WL test**.
 - ID-GNN can **count cycles originating from a given node**, but GNN cannot
- Key idea of ID-GNN: **assign a color to the node we want to embed and apply heterogenous message passing to nodes with different colorings**

More Expressive GNNs

- [Provably expressive graph neural networks](#)
- [Improving graph neural network expressivity via subgraph isomorphism counting](#)
- [Building powerful and equivariant graph neural networks with structural message-passing](#)
- [Relational pooling for graph representations](#)
- [Distance encoding: Design provably more powerful neural networks for graph representation learning](#)
- [Reconstruction for powerful graph representations](#)
- [Ego-GNNs: Exploiting ego structures in graph neural networks](#)
- [Weisfeiler and Lehman Go Cellular: CW Networks](#)
- [Weisfeiler and Lehman Go Topological: Message Passing Simplicial Networks](#)