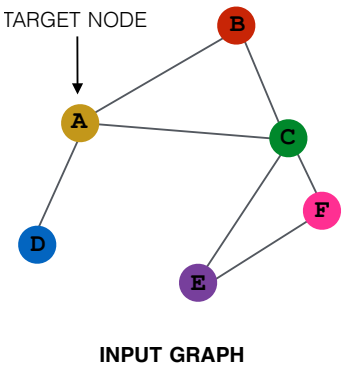
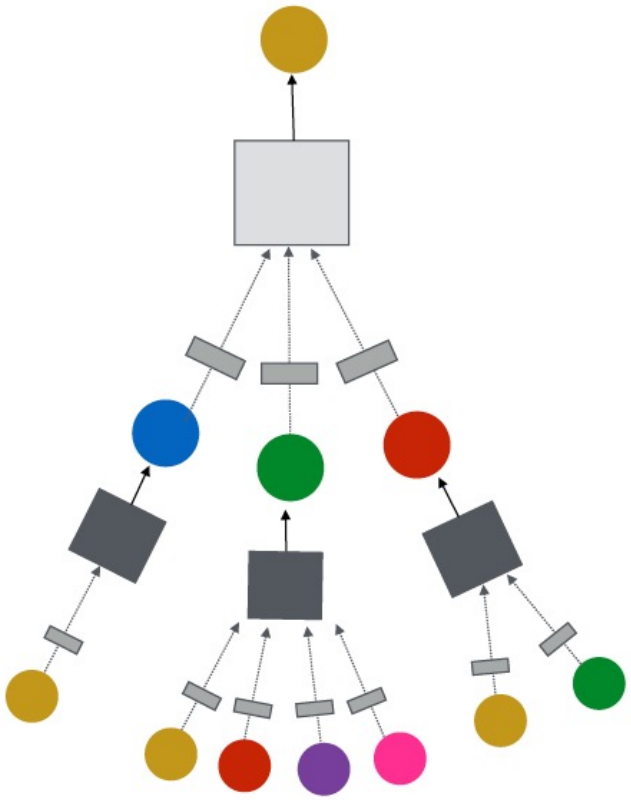
Graph Augmentation for GNNs

General GNN Framework



Idea: Raw input graph ≠ computational graph

- **Graph feature augmentation**
- **Graph structure augmentation**



(4) Graph augmentation

Why Augment Graphs

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Features:**
  - The input graph **lacks features**
- **Graph structure:**
  - The graph is **too sparse** → inefficient message passing
  - The graph is **too dense** → message passing is too costly
  - The graph is **too large** → cannot fit the computational graph into a GPU
- It's **unlikely that the input graph happens to be the optimal computation graph** for embeddings

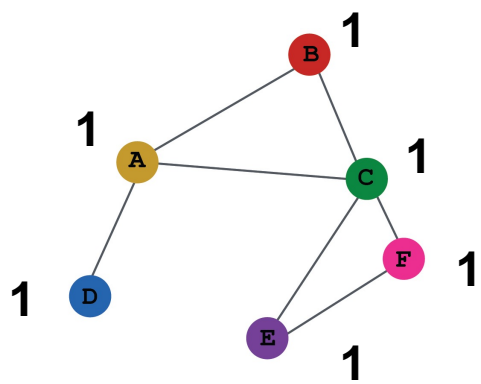
Graph Augmentation Approaches

- **Graph Feature augmentation**
  - The input graph **lacks features** → **feature augmentation**
- **Graph Structure augmentation**
  - The graph is **too sparse** → **Add virtual nodes / edges**
  - The graph is **too dense** → **Sample neighbors when doing message passing**
  - The graph is **too large** → **Sample subgraphs to compute embeddings**
    - Will cover later in lecture: Scaling up GNNs

Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **a) Assign constant values to nodes**

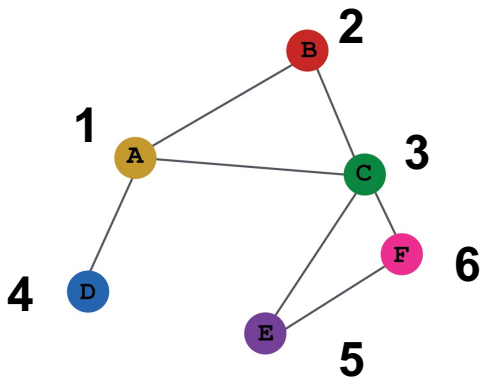


INPUT GRAPH

Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **b) Assign unique IDs to nodes**
  - These IDs are converted into **one-hot vectors**



INPUT GRAPH

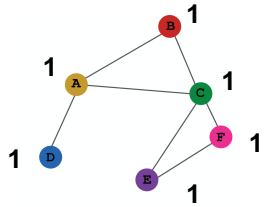
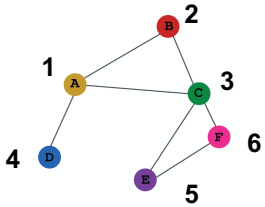
One-hot vector for node with ID=5

ID = 5

$$\underbrace{[0, 0, 0, 0, 1, 0]}_{\text{Total number of IDs = 6}}$$

Feature Augmentation on Graphs

- Feature augmentation: **constant** vs. **one-hot**

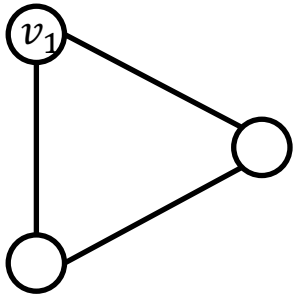
	Constant node feature	**One-hot node feature**
		
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**Low**. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	**Low**. Only 1 dimensional feature	**High**. $O(\|V\|)$ dimensional feature, cannot apply to large graphs
Use cases	**Any graph, inductive settings** (generalize to new nodes)	**Small graph, transductive settings** (no new nodes)

Feature Augmentation on Graphs

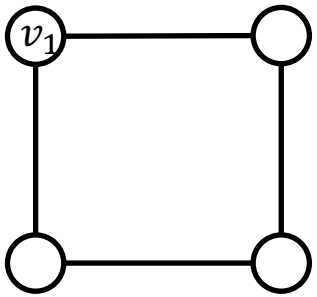
Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- **Example:** Cycle count feature:
  - Can GNN learn the length of a cycle that v_1 resides in?
  - **Unfortunately, no**

v_1 resides in a cycle with length 3



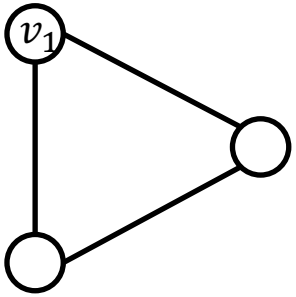
v_1 resides in a cycle with length 4



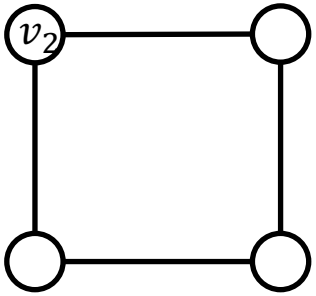
Feature Augmentation on Graphs

- v_1 cannot differentiate which graph it resides in
  - Because all the nodes in the graph have degree of 2
  - The computational graphs will be the same binary tree

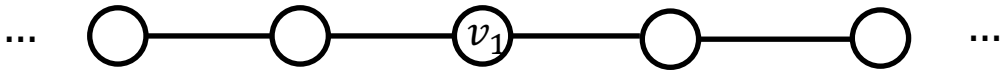
v_1 resides in a cycle with length 3



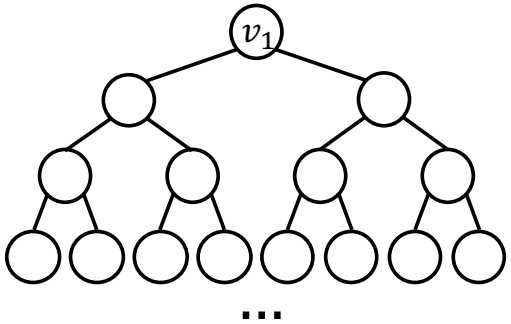
v_1 resides in a cycle with length 4



v_1 resides in a cycle with infinite length



The computational graphs for node v_1 are always the same



More about this topic later!

Feature Augmentation on Graphs

Why do we need feature augmentation?

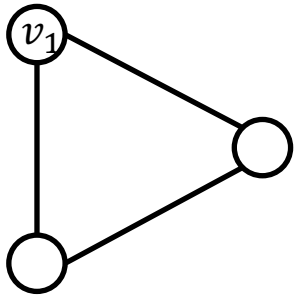
- **(2) Certain structures are hard to learn by GNN**
- **Solution:**
  - We can use cycle count as augmented node features

We start from cycle with length 0

Augmented node feature for v_1

[0, 0, 0, 1, 0, 0]

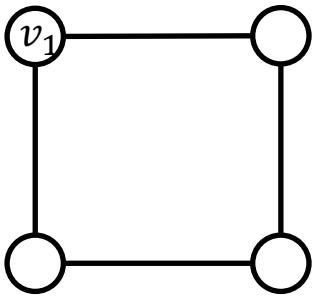
v_1 resides in a cycle with length 3



Augmented node feature for v_1

[0, 0, 0, 0, 1, 0]

v_1 resides in a cycle with length 4



Feature Augmentation on Graphs

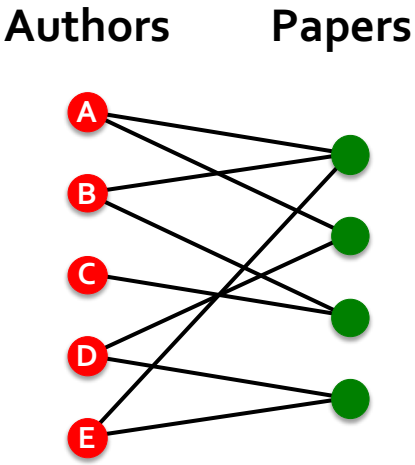
Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- Other commonly used augmented features:
  - **Node degree**
  - **Clustering coefficient**
  - **PageRank**
  - **Centrality**
  - **…**
- Any **feature we have introduced in Lecture 2 can be used!**

Add Virtual Nodes / Edges

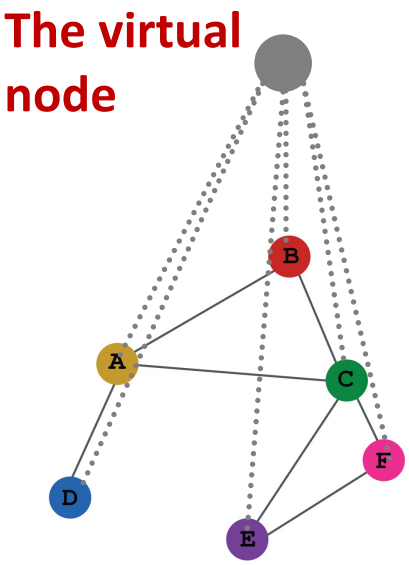
- **Motivation:** Augment sparse graphs
- **(1) Add virtual edges**
  - **Common approach:** Connect 2-hop neighbors via virtual edges
  - **Intuition:** Instead of using adj. matrix A for GNN computation, use $A + A^2$

- **Use cases:** Bipartite graphs
  - Author-to-papers (they authored)
  - 2-hop virtual edges make an author-author collaboration graph



Add Virtual Nodes / Edges

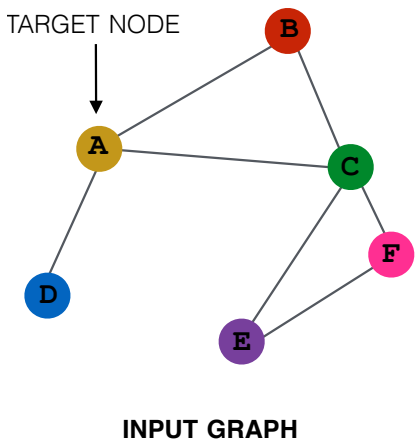
- **Motivation:** Augment sparse graphs
- **(2) Add virtual nodes**
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of two**
      - **Node A – Virtual node – Node B**
  - **Benefits:** Greatly **improves message passing in sparse graphs**

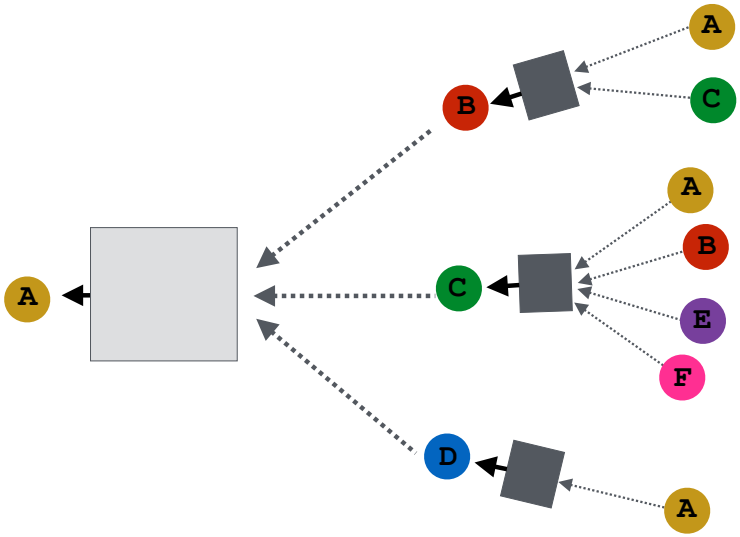


INPUT GRAPH

Node Neighborhood Sampling

- **Previously:**
  - All the nodes are used for message passing

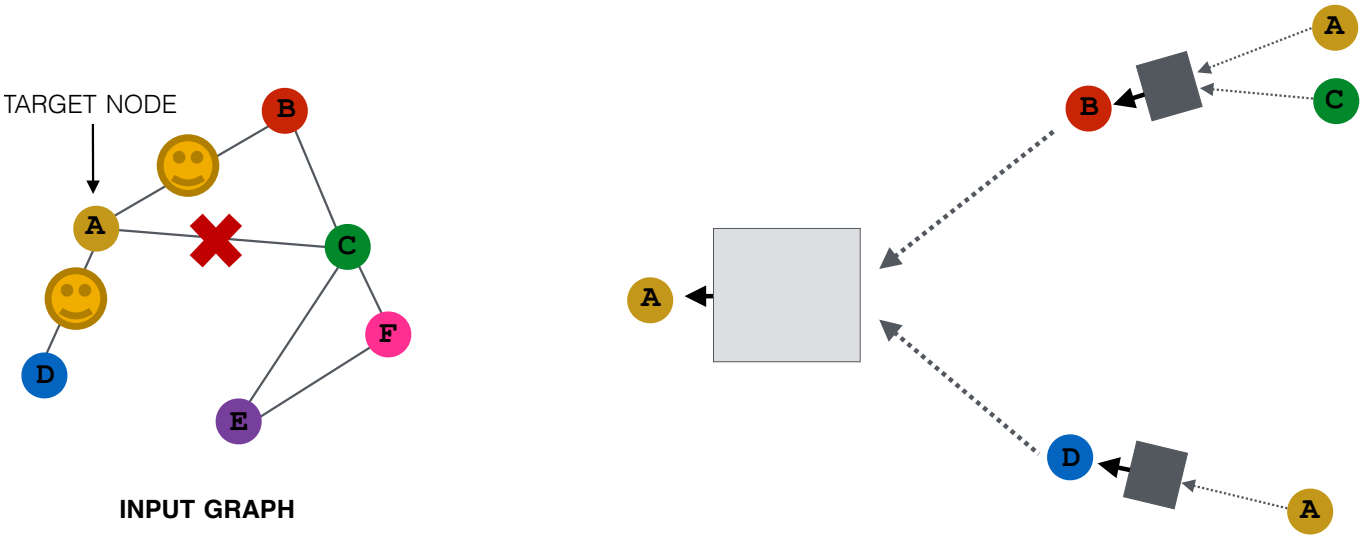




- **New idea:** (Randomly) sample a node's neighborhood for message passing

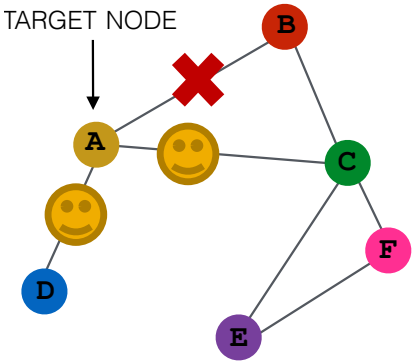
Neighborhood Sampling Example

- **For example, we can randomly choose 2 neighbors to pass messages in a given layer**
  - Only nodes B and D will pass messages to A

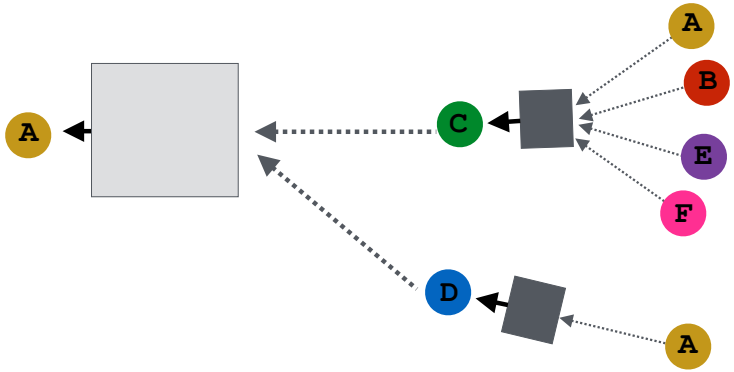


Neighborhood Sampling Example

- **In the next layer when we compute the embeddings, we can sample different neighbors**
  - Only nodes C and D will pass messages to A



INPUT GRAPH



Neighborhood Sampling Example

- **In expectation, we get embeddings similar to the case where all the neighbors are used**
  - **Benefits:** Greatly reduces computational cost
    - **Allows for scaling to large graphs (more about this later)**
  - And in practice it works great!