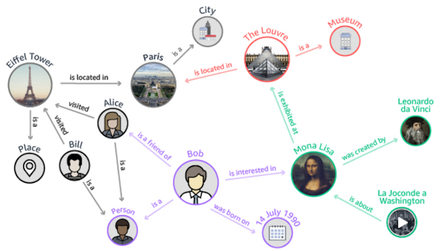
A Journey into Graph Representation Learning

İsmail İlkan Ceylan

AIMS, Guest Lecture, Oxford

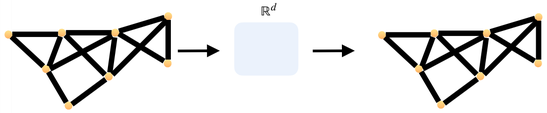
Michaelmas Term, 2023/2024

Overview of the Lecture



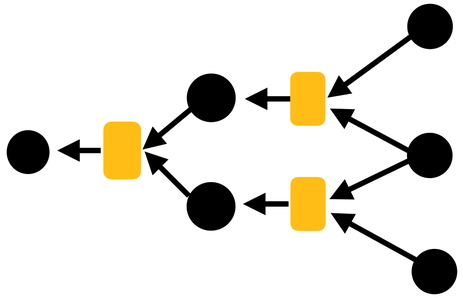
Relational data and inductive bias

Why is relational data prominent and where?



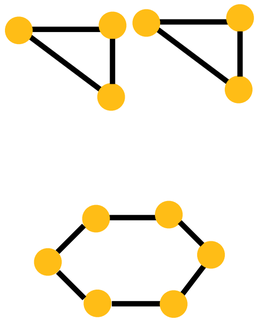
GRL: an encoder-decoder framework

What is GRL and what are the tasks of interest?



Message passing neural networks

Message passing paradigm and graph neural network architectures

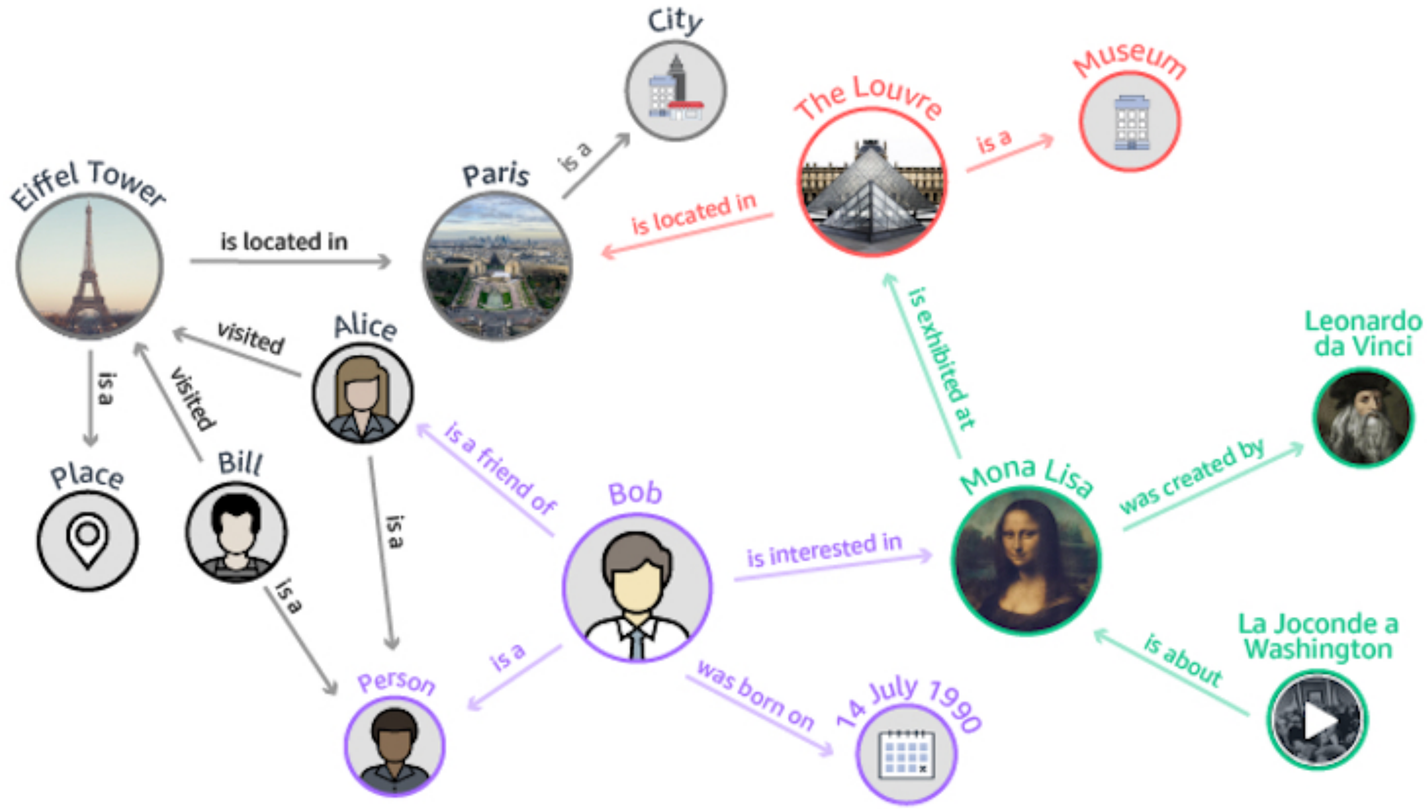


Expressive power of MPNNs

1-WL algorithm for graph isomorphism testing

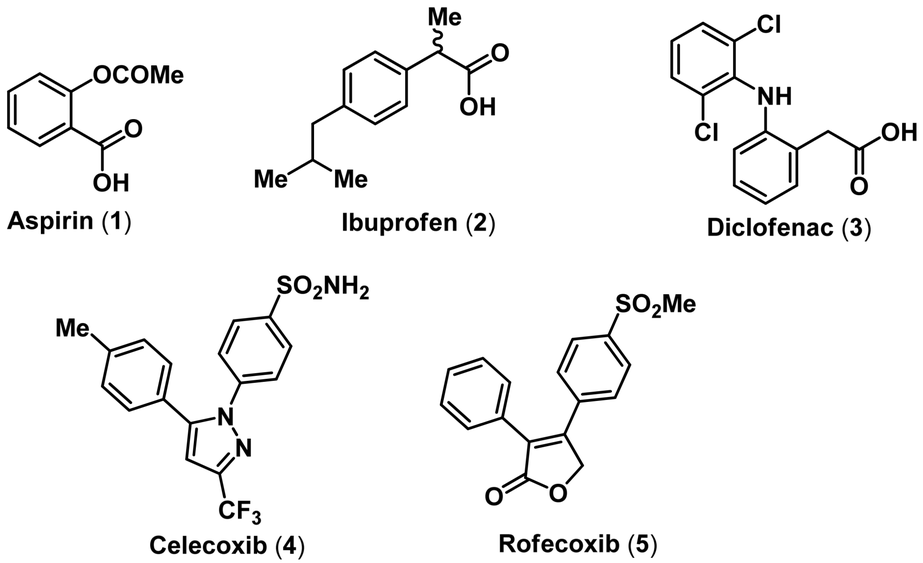
Relational Data

Knowledge Graphs



Knowledge graphs: Graph-structured data models, storing relations (e.g., isFriendOf) between entities (e.g., Alice, Bob) and thereby capture structured knowledge.

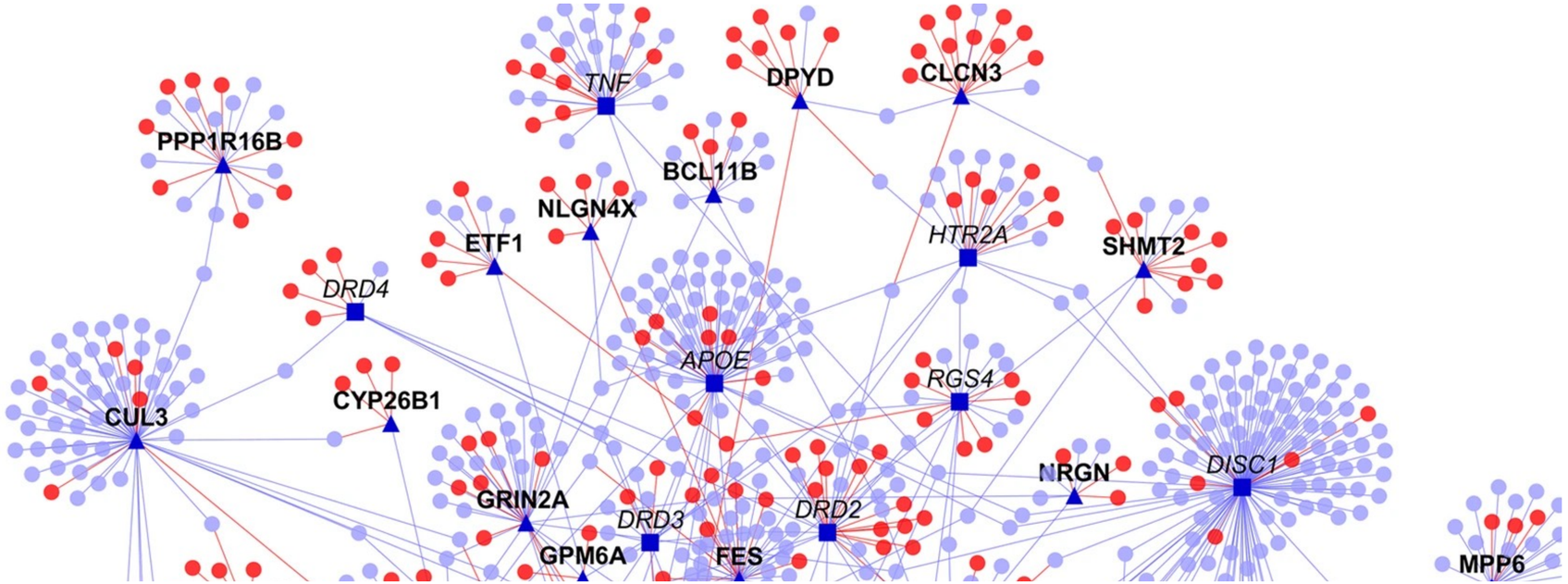
Biomedical Data: Molecular Scale



Molecules (Rao et al, 2013): Figure shows the molecule structure of NSAID drugs. “Me" is an abbreviation for "methyl" (CH3).

Molecular scale: Small molecule drugs can be represented as graphs relating their constituent atoms and chemical bonding structure. Complex molecules, such as proteins can be represented as graphs capturing spatial and structural relationships between their amino acid residues.

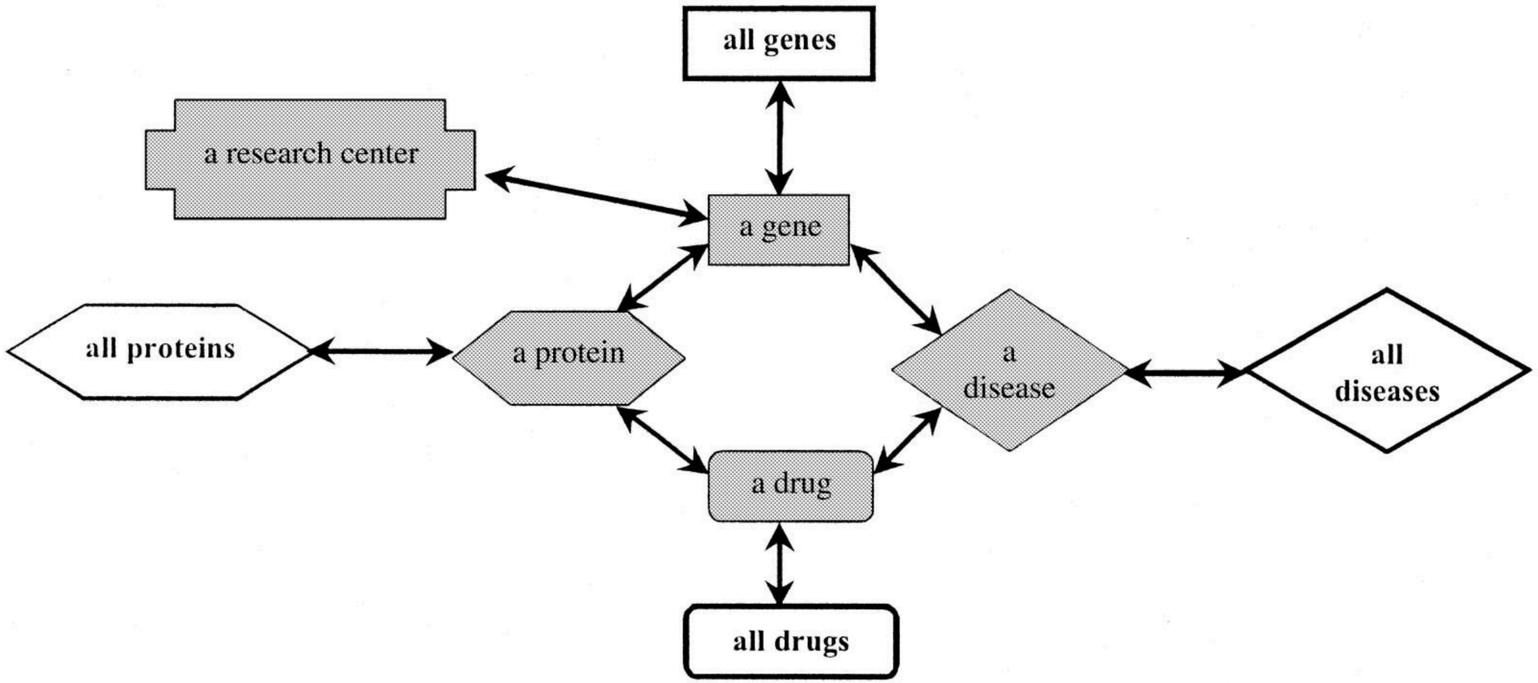
Biomedical Data: Intermediary Scale



Excerpt from Schizophrenia interactome (Ganapathiraju et al, 2016): Genes are shown as nodes and PPIs as edges connecting the nodes. Schizophrenia-associated genes are shown as dark blue nodes, novel interactors as red color nodes and known interactors as blue color nodes. Red edges are the novel interactions, whereas blue edges are known interactions.

Intermediary scale: An interactome defines a set of molecular interactions in a particular cell — They can be represented as graphs, e.g., protein–protein interaction graphs.

Biomedical Data: Abstract Scale



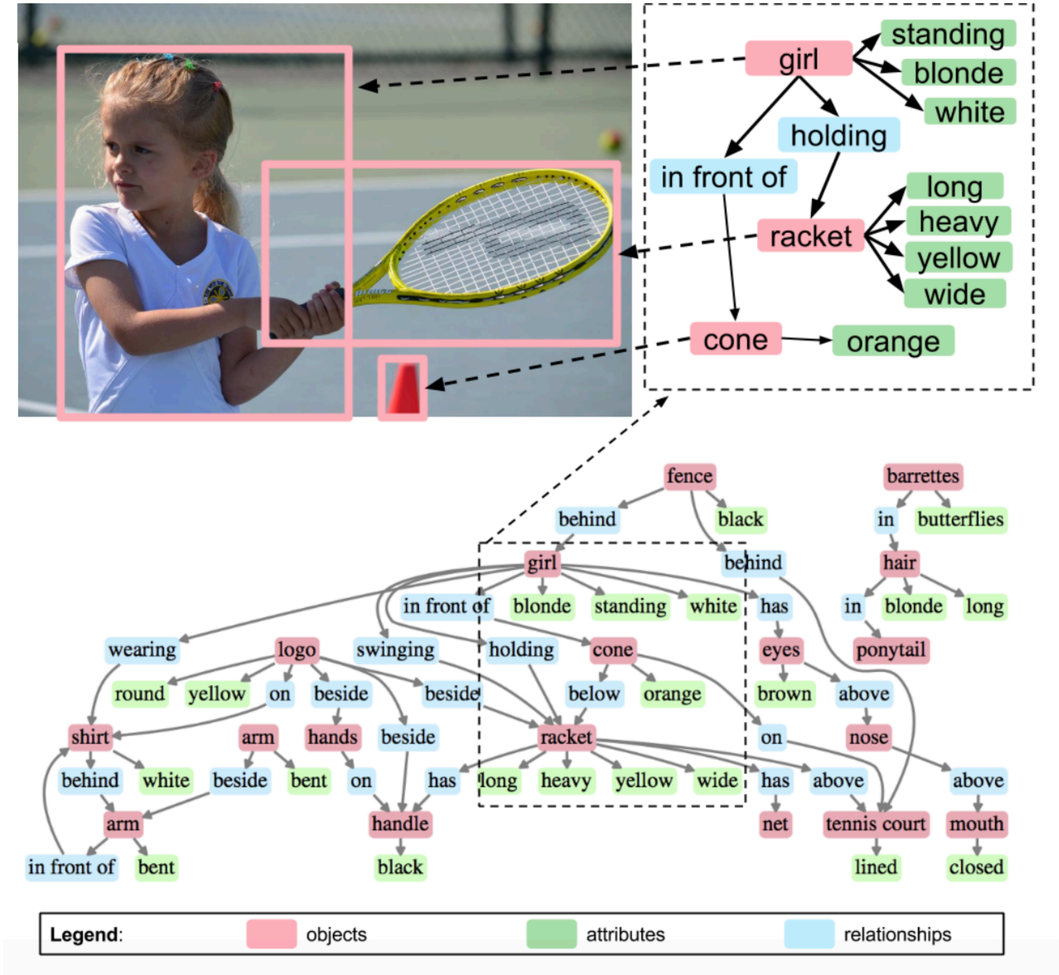
PharmGKB (Hewett et al., 2002): Abstract, complex relationships among the objects, including 'expresses', as in 'a gene expresses a protein': 600+ different relationships.

Abstract scale: KGs can represent the complex relationships between drugs, side effects, diagnosis, associated treatments, and test results etc.

Social Networks

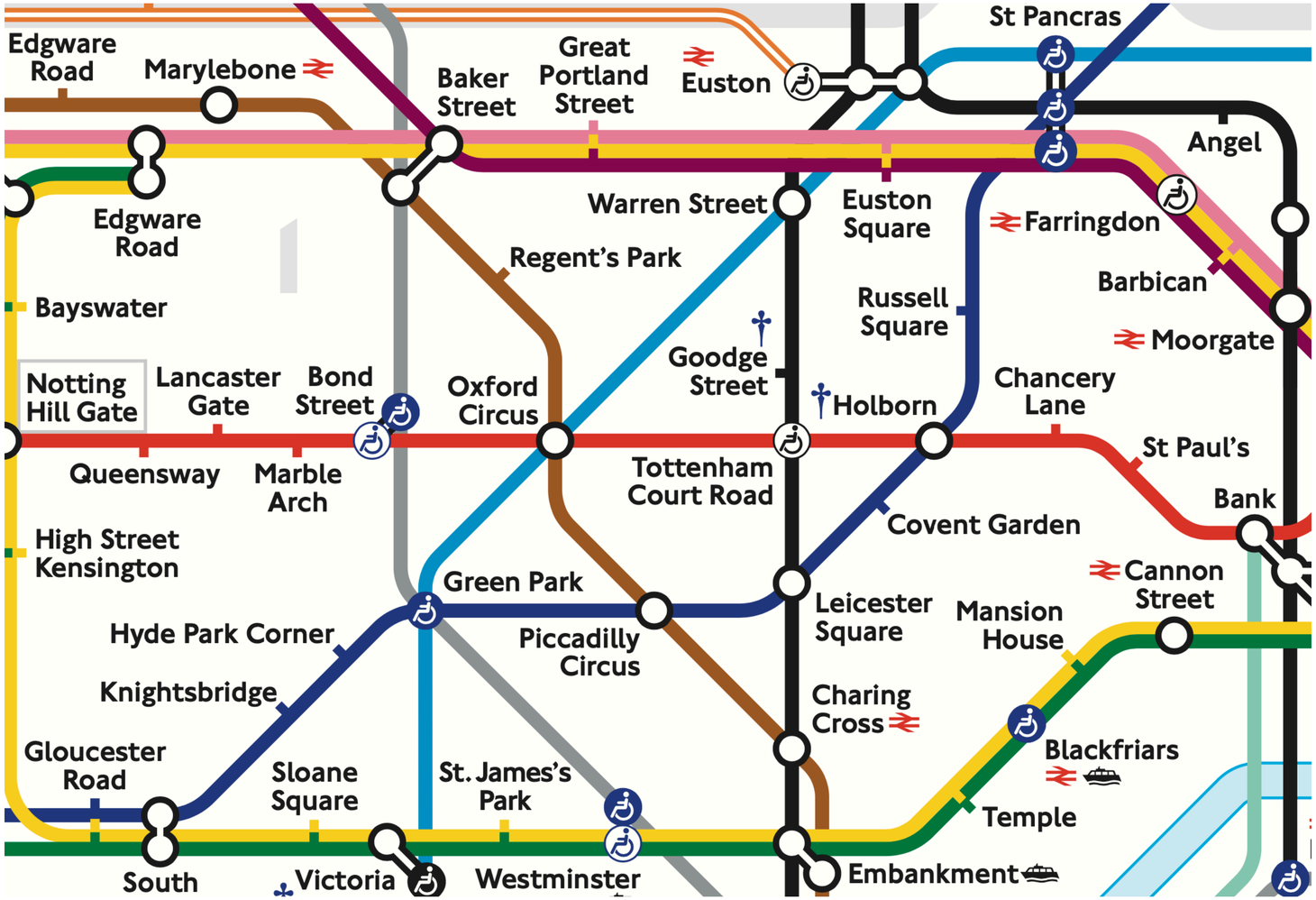
Social networks: Entities (e.g., individuals, groups, organizations) interacting with other entities on social platforms.

Computer Vision: Scene Graphs



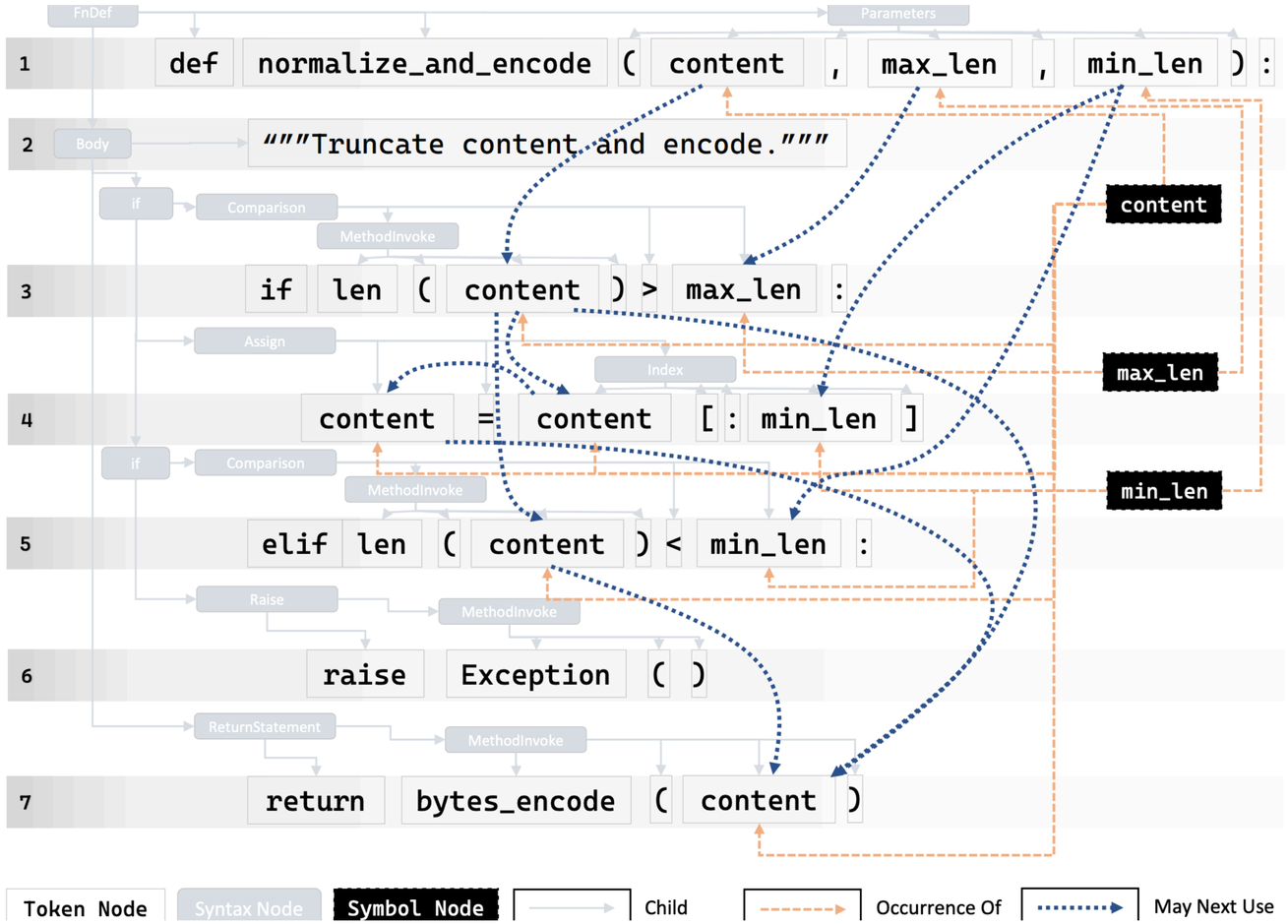
Scene graphs (Johnson et al., 2015): A scene as a graph.

Road Networks



Traffic networks: An excerpt of the London Tube of Zone 1, showing different lines.

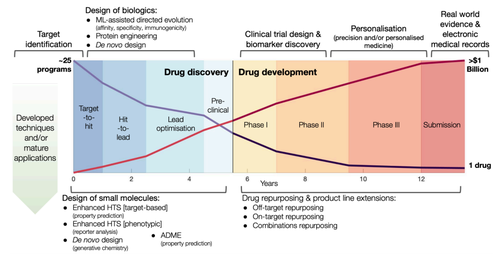
Programs: Dependency Graphs



Program dependency graphs (Allamanis, 2021): Figure shows a Python program and its dependencies represented as a graph.

Plethora of Applications

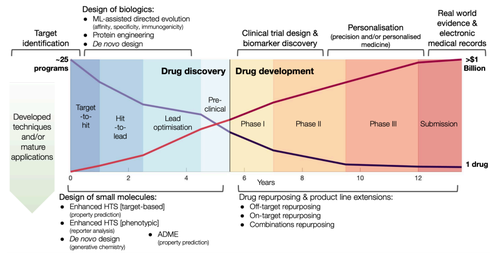
Plethora of Applications



Drug Discovery

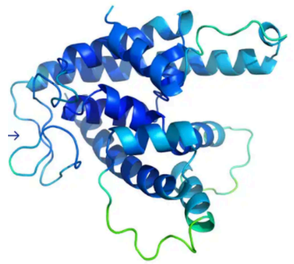
Target identification, property prediction, drug repurposing, protein engineering

Plethora of Applications



Drug Discovery

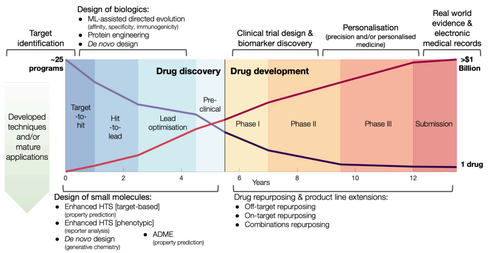
Target identification, property prediction, drug repurposing, protein engineering



Protein Folding

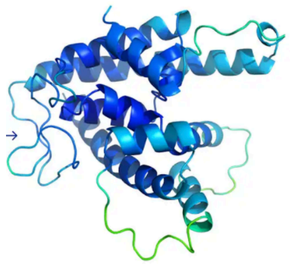
How do amino acids fold to form proteins?

Plethora of Applications



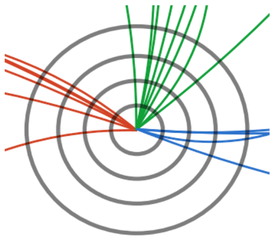
Drug Discovery

Target identification, property prediction, drug repurposing, protein engineering



Protein Folding

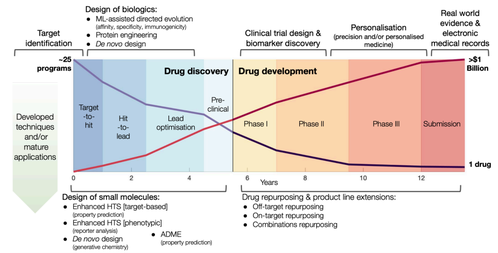
How do amino acids fold to form proteins?



Jet Classification

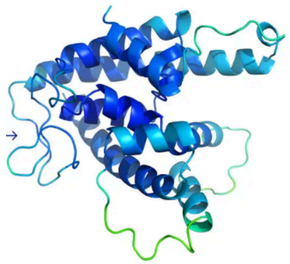
What is the original object that gave rise to the jet?

Plethora of Applications



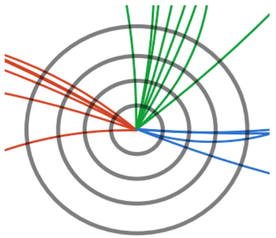
Drug Discovery

Target identification, property prediction, drug repurposing, protein engineering



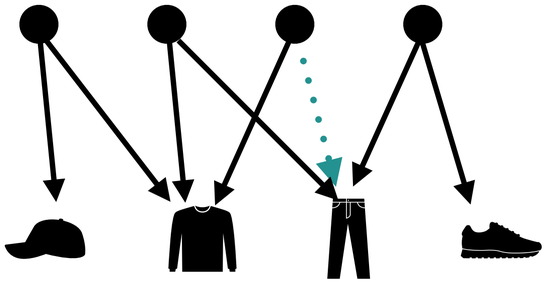
Protein Folding

How do amino acids fold to form proteins?



Jet Classification

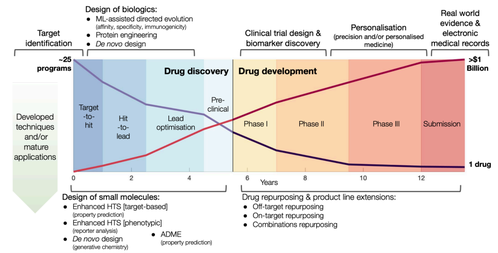
What is the original object that gave rise to the jet?



Recommender Systems

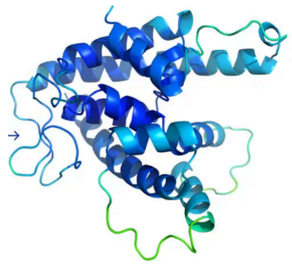
Realistic recommendations for users

Plethora of Applications



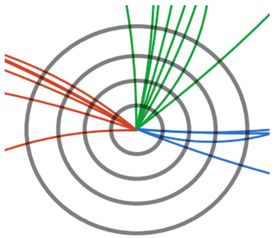
Drug Discovery

Target identification, property prediction, drug repurposing, protein engineering



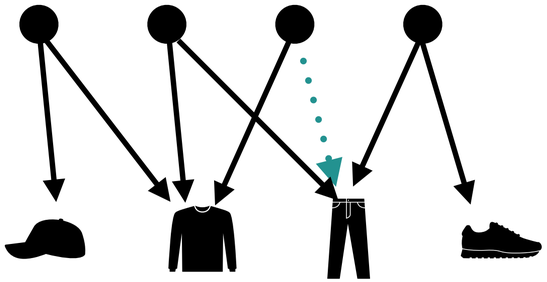
Protein Folding

How do amino acids fold to form proteins?



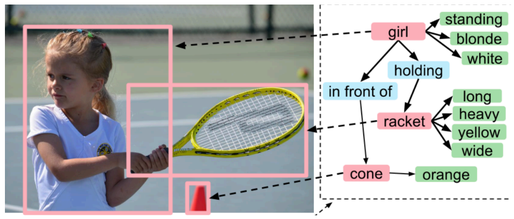
Jet Classification

What is the original object that gave rise to the jet?



Recommender Systems

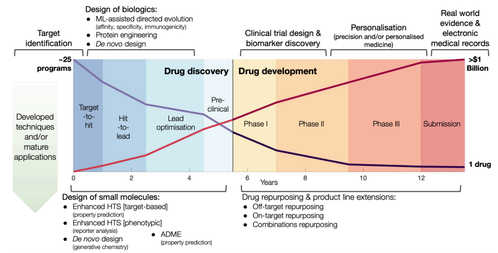
Realistic recommendations for users



Visual Question Answering

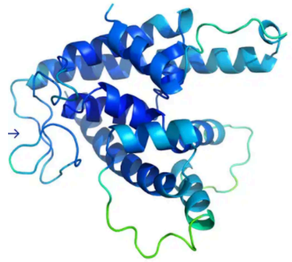
Answering questions about scenes

Plethora of Applications



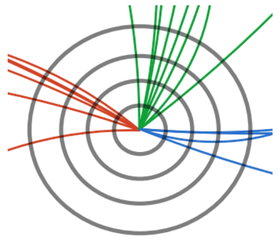
Drug Discovery

Target identification, property prediction, drug repurposing, protein engineering



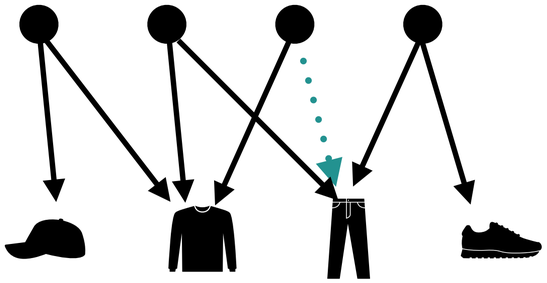
Protein Folding

How do amino acids fold to form proteins?



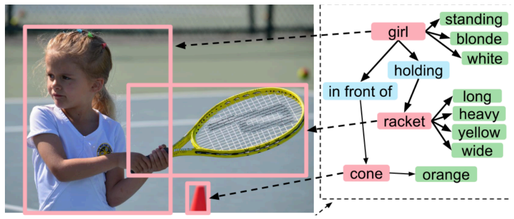
Jet Classification

What is the original object that gave rise to the jet?



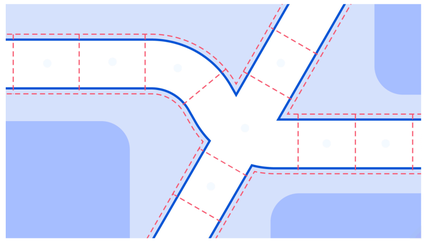
Recommender Systems

Realistic recommendations for users



Visual Question Answering

Answering questions about scenes

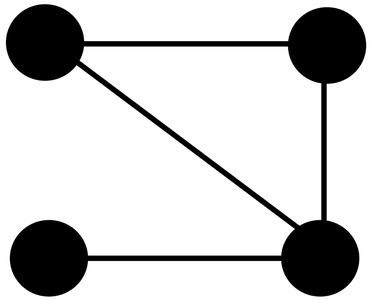


Traffic Forecasting

Estimating Times of Arrivals

Graph Representation Learning

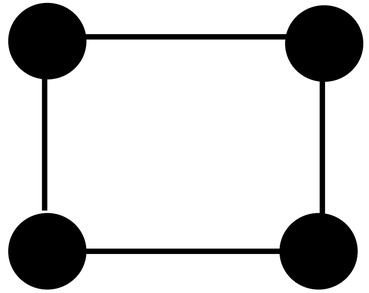
Graph Machine Learning



Node degrees?

Contains an odd-length cycle?

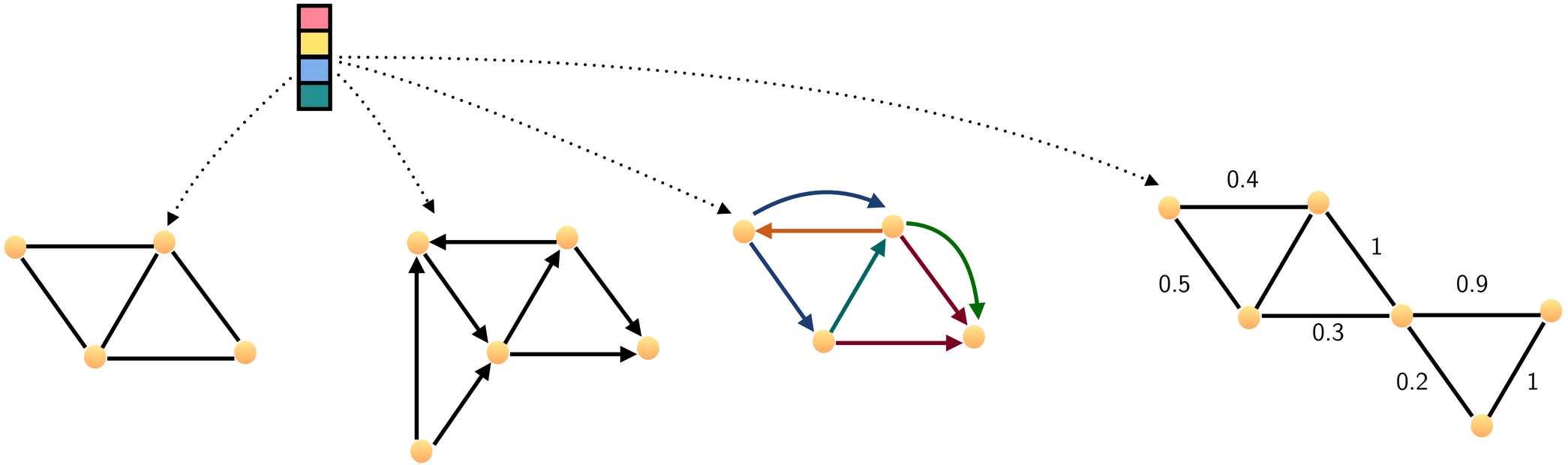
Minimum vertex cover size 1, 2?



Functions over graphs, or nodes, necessarily relate to graph properties, which carry valuable information: needs to be taken into account adequately.

Idea: Define similarity measures for nodes/graphs, and then use for the optimization task.

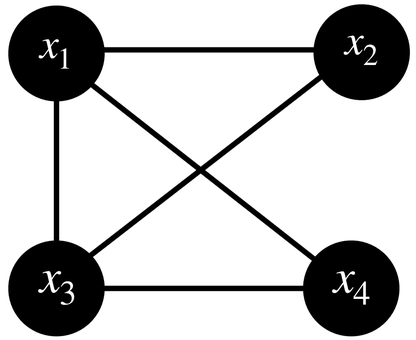
What Kind of Graphs?



Context: Simple, undirected, unweighted graphs $G = (V, E, \mathbf{X})$ attributed with node features.

- V: Set of vertices/nodes
- $E \subseteq V \times V$: Set of edges
- $\mathbf{X} \in \mathbb{R}^{d \times V}$: Node feature matrix, which stores a feature vector $\mathbf{x}_u = \mathbf{X}[u]^\top$ for each node u. domain-specific attributes, or node degrees, or simply one-hot encodings.

What Kind of Representations?



$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$\mathbf{A}$

Representations: We can represent the graph in terms of its adjacency matrix and feature matrix:

- $\mathbf{A}$ is the adjacency matrix of a graph $G = (V, E)$.
- $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ is a feature matrix of a graph $G = (V, E)$ where d is the embedding dimensionality.
- We sometimes write $G = (\mathbf{A}, \mathbf{X})$ instead of $G = (V, E, \mathbf{X})$.

What Kind of Functions?

$$f$$

Consider k-ary functions f, which define, for every node attributed graph $G = (V, E, \mathbf{X})$, a mapping of the form $f(G) : V^k \to \mathbb{D}$.

Graph-level functions:

$$f(G) \to \mathbb{B}$$

$$f(G) \to \mathbb{R}$$

$$f(G) \to \mathbb{R}^d$$

Graph functions ($k = 0$)

Node-level functions:

$$f(G) : V \to \mathbb{B}$$

$$f(G) : V \to \mathbb{R}$$

$$f(G) : V \to \mathbb{R}^d$$

Unary functions ($k = 1$)

Edge-level functions:

$$f(G) : V^2 \to \mathbb{B}$$

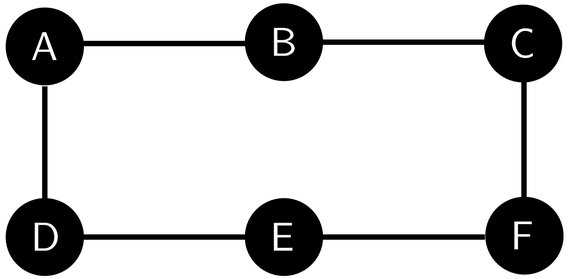
$$f(G) : V^2 \to \mathbb{R}$$

$$f(G) : V^2 \to \mathbb{R}^d$$

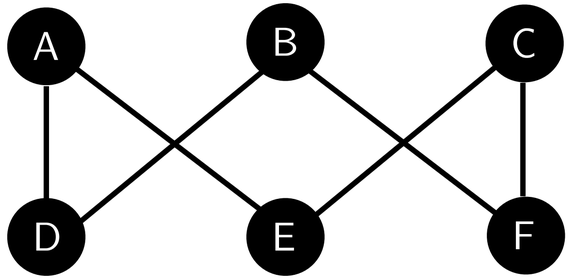
Binary functions ($k = 2$)

Graph Isomorphism

G



H



Graph Isomorphism

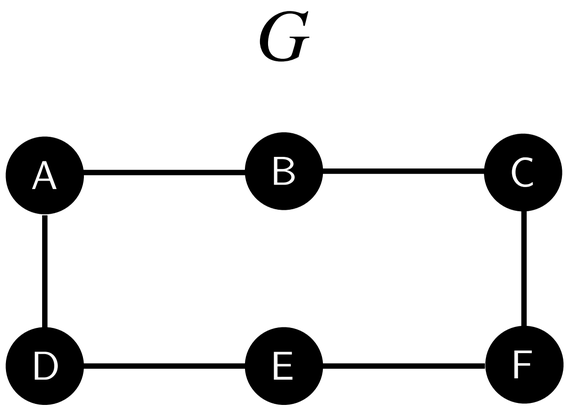
Isomorphism: Two graphs $G = (V_G, E_G, \mathbf{X}_G)$ and $H = (V_H, E_H, \mathbf{X}_G)$ with node features are isomorphic if there exists a bijection between the node sets V_G and V_H such that

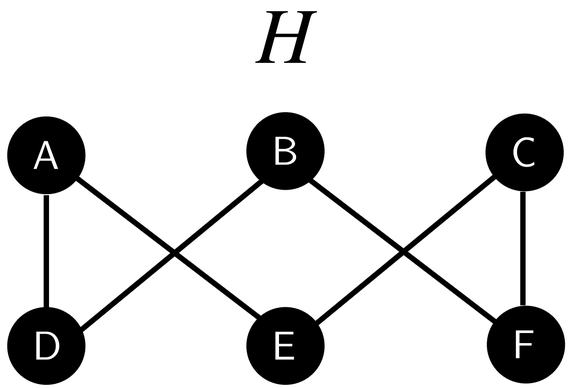
$$(u, v) \in E_G \text{ if and only if } (f(u), f(v)) \in E_H \text{ for all } u, v \in V_G,$$

and

$$\mathbf{X}_G[u] = \mathbf{X}_H[f(u)] \text{ for all } u \in V_G.$$

Inductive Bias: Invariance and Equivariance

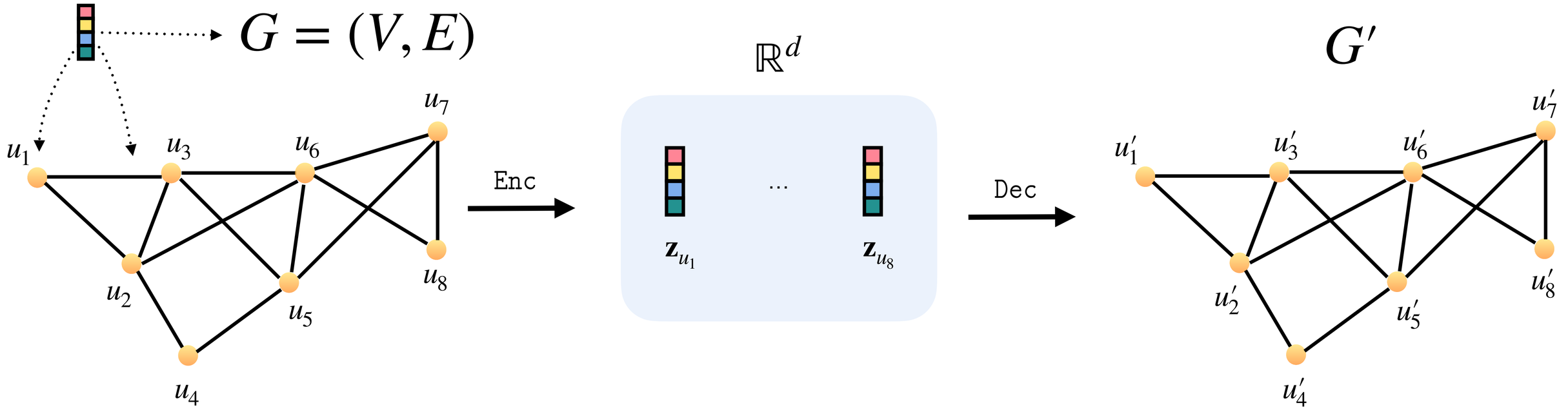




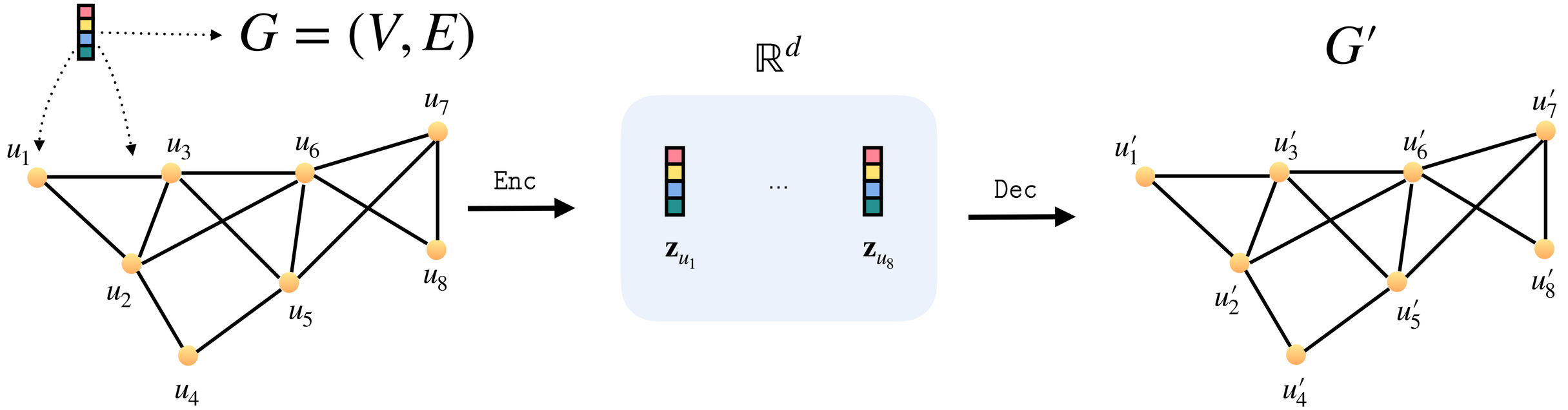
Invariance: A function f over graphs is permutation-invariant if for all isomorphic graphs G, H it holds that $f(G) = f(H)$, i.e., the function f does not depend on the ordering of the nodes in the graph.

Equivariance: A function f over graph nodes $f(G) : V \to \mathbb{R}^{|V|}$ is permutation-equivariant if for every graph G and, for every permutation π of V, it holds that $f(G)(V^{\pi}) = f(G)(V)^{\pi}$, i.e., the output of f is permuted in a consistent way when we permute the nodes in the graph.

An Encoder-Decoder Perspective



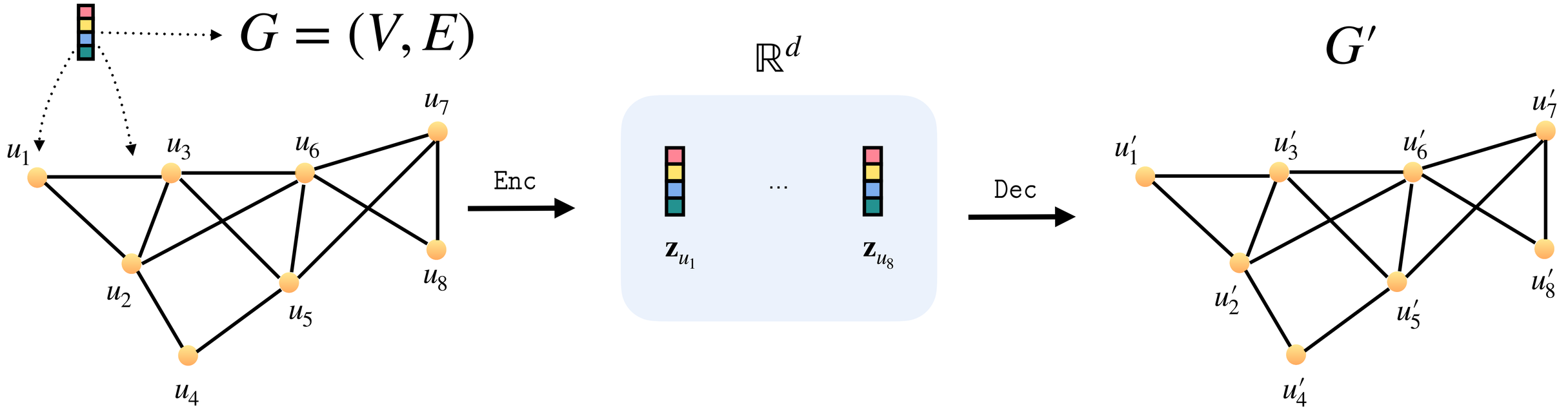
An Encoder-Decoder Perspective



Goal: Embedding nodes, edges, graphs, along with their features, and use these embeddings for predicting node-level, edge-level, or graph-level properties.

Intuition: Nodes/edges/graphs with "similar properties" should have representations closer to each other than nodes/edges/graphs with "dissimilar properties".

An Encoder-Decoder Perspective

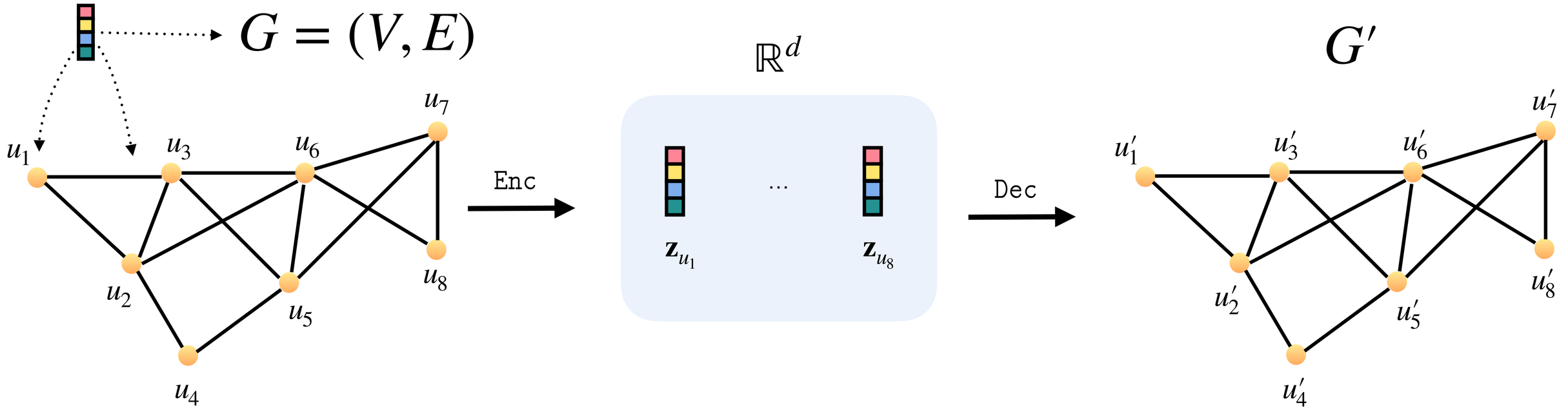


Training: Let $\mathbf{S}[u, v]$ be a similarity measure between the nodes u, v and suppose:

$$\texttt{Enc} : V \to \mathbb{R}^d \qquad \texttt{Dec} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$$

Optimization: $\forall u, v \in V : \texttt{Dec}(\texttt{Enc}(u), \texttt{Enc}(v)) = \texttt{Dec}(\mathbf{z_u}, \mathbf{z_v}) \sim \mathbf{S}[u, v]$, i.e., minimize the reconstruction loss.

An Encoder-Decoder Perspective



Graph representation learning tasks: Various node/edge/graph level tasks are of interest.

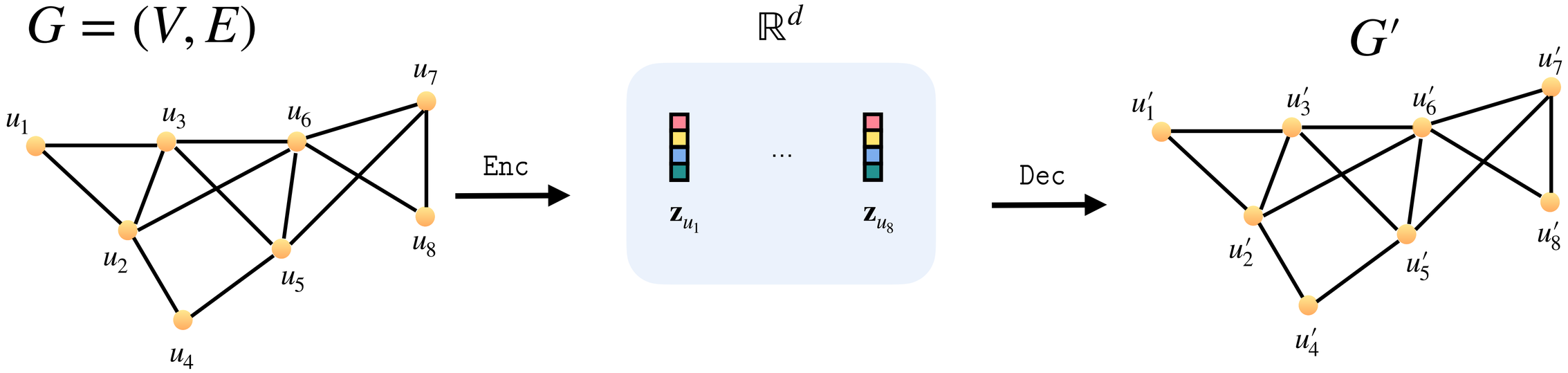
Node-level: Node classification/clustering/regression

Edge-level: Link prediction, knowledge graph completion

Graph-level: Graph classification/clustering/regression/generation

Shallow Node Embeddings

An Encoder-Decoder Perspective



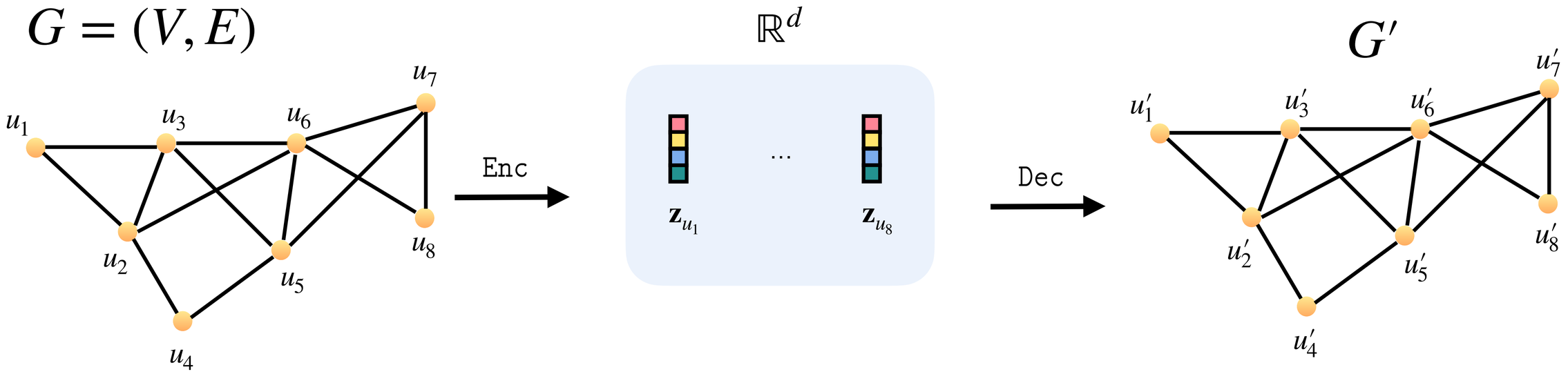
Encoder and Decoder: Let $\mathbf{S}[u, v]$ be a similarity measure between the nodes u, v and suppose:

$$\texttt{Enc}\colon V \to \mathbb{R}^d \qquad\qquad \texttt{Dec}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$$

Shallow encoder: A lookup function $\texttt{Enc}(\text{v}) = \mathbf{Z}[v]^\top$, where $\mathbf{Z}\colon \mathbb{R}^{|V| \times d}$ is a matrix of d-dimensional embeddings.

Unsupervised: We do not use node labels or features and the resulting embeddings are task-independent!

Optimization

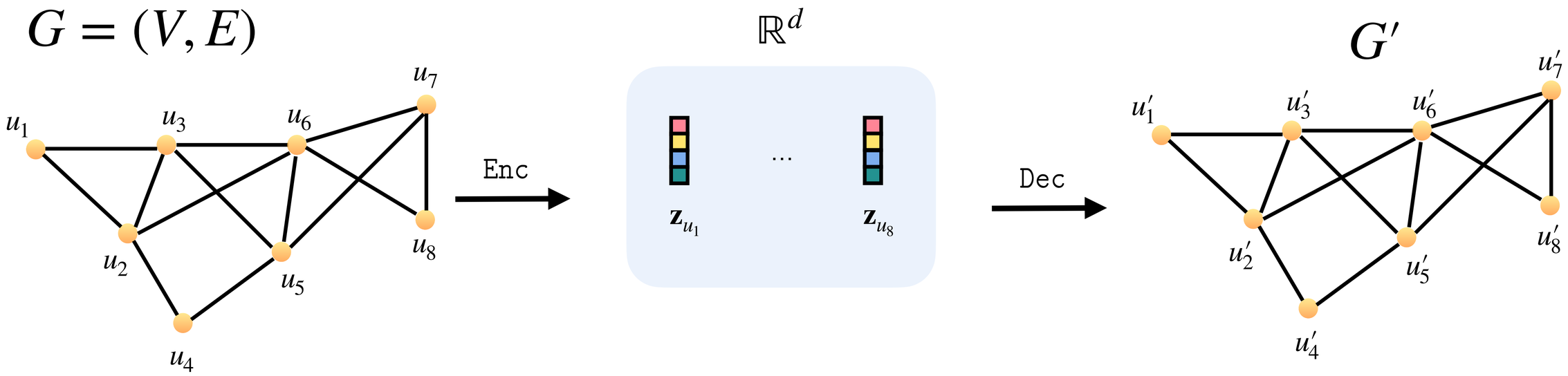


Optimization: Given a dataset $D = \{(u_i, v_i) \mid 1 \leq i \leq n\}$, minimize the loss:

$$\sum_{(u,v)\in D} f\Big(\texttt{Dec}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v]\Big),$$

where $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ (e.g., mean-squared error), measures the discrepancy between $\texttt{Dec}(\mathbf{z}_u, \mathbf{z}_v)$ and $\mathbf{S}[u, v]$.

Characterizing a Node Embedding Model



Idea: Node embedding models produce an embedding vector for each node such that nodes with similar properties are in close proximity to one another in the embedding space.

Intuition: Two nodes are similar if their neighbourhoods are similar according to some notion of neighbourhood.

(1) What kind of decoder? **(2) What kind of node/graph similarity?** **(3) Which loss function?**

Matrix Factorization Approaches: Inner Product

Decoder: Similarity between two nodes is proportional to the dot product of their embeddings:

$$\texttt{Dec}(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^\top \mathbf{z}_v$$

Loss:

$$\mathscr{L} = \sum_{(u,v)\in D} \|\texttt{Dec}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}(u,v)\|_2^2$$

Mapping back to the matrix $\mathbf{Z}$ of node embeddings, reveals the connection to matrix factorization:

$$\mathscr{L} \approx \|\mathbf{Z}\mathbf{Z}^\top - \mathbf{S}\|_2^2.$$

$$\mathbf{S} \approx \mathbf{Z}\mathbf{Z}^\top$$

Matrix Factorization Approaches: Inner Product

In its simplest form, we can set $\mathbf{S} = \mathbf{A}$ and minimise

$$\mathscr{L} = \sum_{(u,v)\in D} \|\texttt{Dec}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{A}(u,v)\|_2^2$$

This objective approximately recovers the graph:

$$\mathscr{L} \approx \|\mathbf{Z}\mathbf{Z}^\top - \mathbf{A}\|_2^2.$$

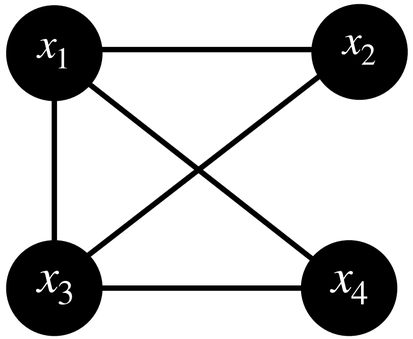
$$\mathbf{A} \approx \mathbf{Z}\mathbf{Z}^\top$$

To capture multi-hops, we can set a similarity defined over:

$$\mathbf{A}[u,v], \ldots, \mathbf{A}^k[u,v].$$

Decoder (i.e., any pairwise similarity) and accordingly the target similarity (neighbourhood overlap measures) can vary...

Other Approaches



$$\mathbf{D} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \end{array}$$

$$\mathbf{A} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathbf{L} = \mathbf{D} - \mathbf{A} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix} \end{array}$$

Similarity: In terms of generalizations of other matrices, i.e., the graph Laplacian.

Decode: We can decode differently, i.e., based on the L_2-distance: $\texttt{Dec}(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_u - \mathbf{z}_v\|_2^2$.

Random walk approaches: Models such as DeepWalk and node2vec, inspired by word2vec.

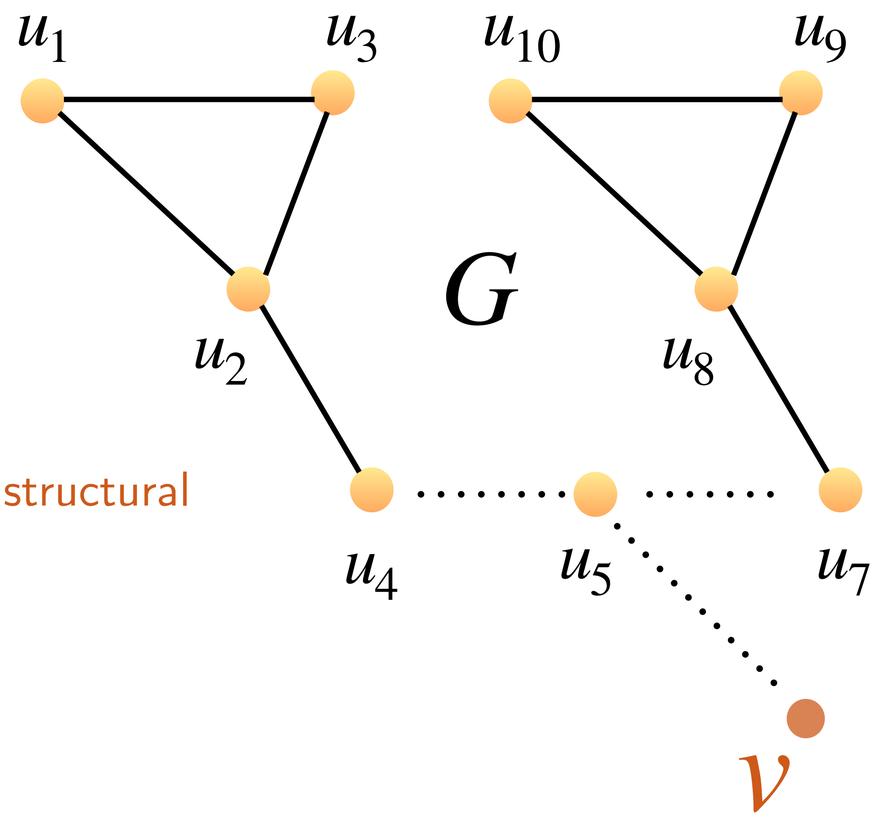
Beyond Shallow Embeddings

The embeddings of the nodes do not share any parameters, i.e., hard to model dependencies.

Stronger encoder to capture dependencies?

Node/graph-level features cannot be utilised effectively.

Encoder which can incorporate node features?



Hard to capture graph-level, global properties, hence worse on graph-level tasks.

Better capturing global properties?

Better encoder to capture structural properties?

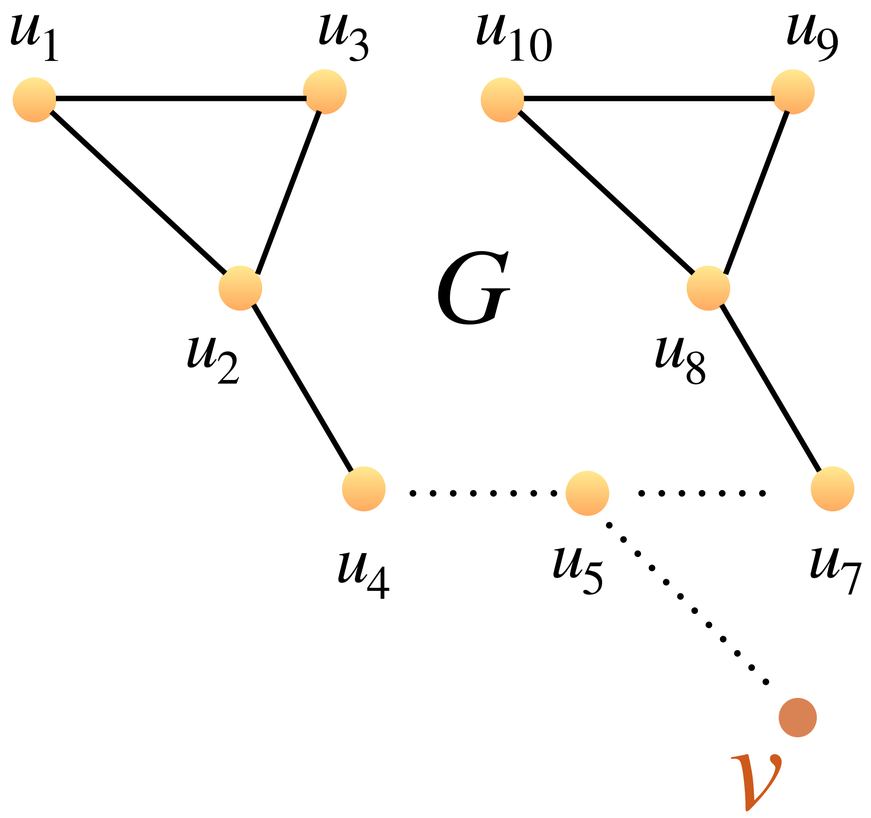
It is hard to capture certain structural similarities, e.g., u_1 and u_{10}.

Transductive: No embeddings for new nodes, unseen during training

Inductive models?

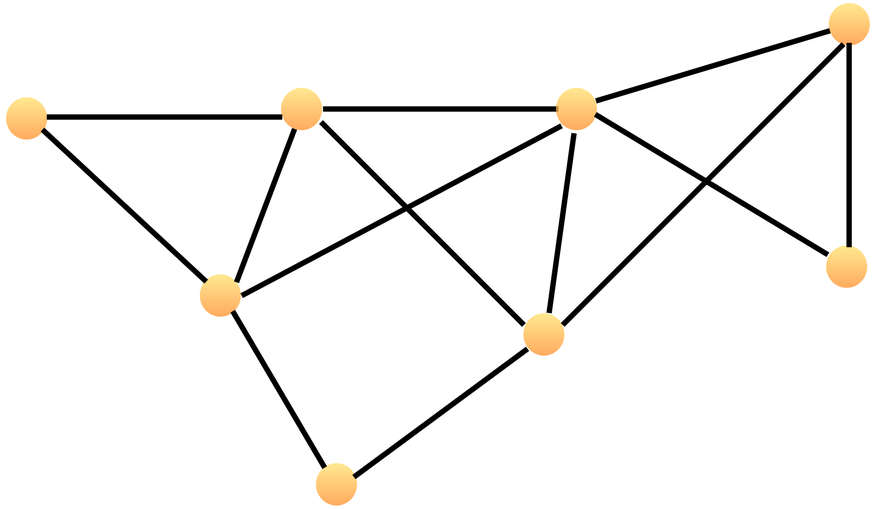
Message Passing Neural Networks

Goal: Designing neural architectures satisfying the desired desiderata!



Message Passing Neural Networks

Message Passing Neural Networks



Message passing neural networks (MPNNs) capture popular GNNs (Gilmer et al., 2017).

Idea: Iteratively update initial node features with the information received from their respective neighborhoods.

Notation: The representation of $u \in V$ at iteration t is $\mathbf{h}_u^{(t)}$, i.e., the initial representation is $\mathbf{h}_u^{(0)} = \mathbf{x}_u = \mathbf{X}[u]^\top$.

Message Passing Neural Networks

Given a graph $G = (V, E, \mathbf{X})$, an MPNN iteratively computes $\mathbf{h}_u^{(t)}$ for every node $u \in V$:

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u, \qquad \text{initialize}$$

$$\mathbf{m}_u^{(t)} = \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right), \qquad \text{aggregate}$$

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \mathbf{m}_u^{(t)}\right), \qquad \text{update/combine}$$

Message Passing Neural Networks

Given a graph $G = (V, E, \mathbf{X})$, an MPNN iteratively computes $\mathbf{h}_u^{(t)}$ for every node $u \in V$:

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u, \qquad \text{initialize}$$

$$\mathbf{m}_u^{(t)} = \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right), \qquad \text{aggregate}$$

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \mathbf{m}_u^{(t)}\right), \qquad \text{update/combine}$$

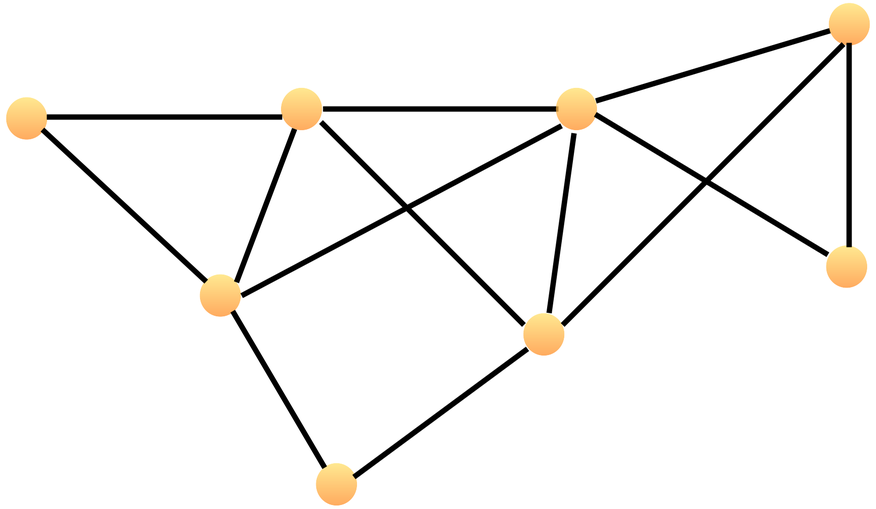
mean, sum, max, ...

message

non-linearity

$\phi^{(t)}$ and $\psi^{(t)}$ can be any differentiable function!

Message Passing Neural Networks

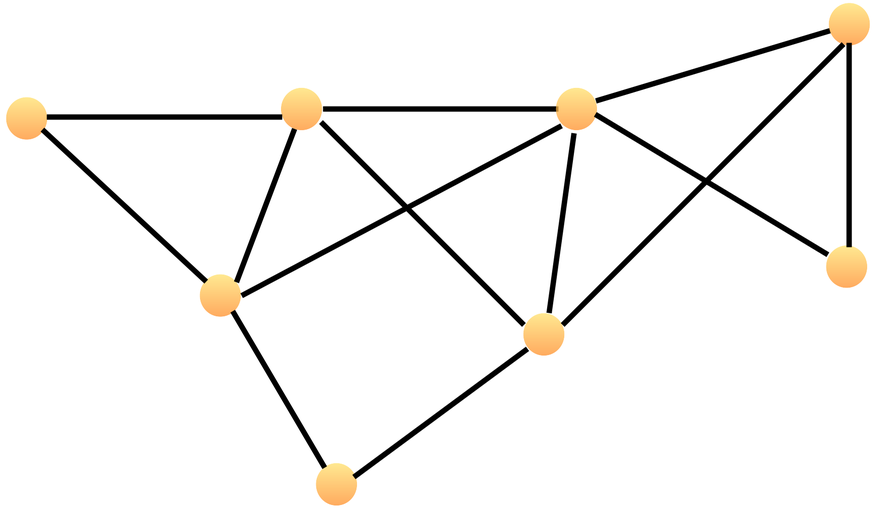


Given a graph $G = (V, E, \mathbf{X})$, an MPNN defines $\forall u \in V$ the features $\mathbf{h}_u^{(0)} = \mathbf{x}_u$, and iteratively updates them:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right)\right),$$

where $\phi^{(t)}$ and $\psi^{(t)}$ are differentiable functions.

Message Passing Neural Networks

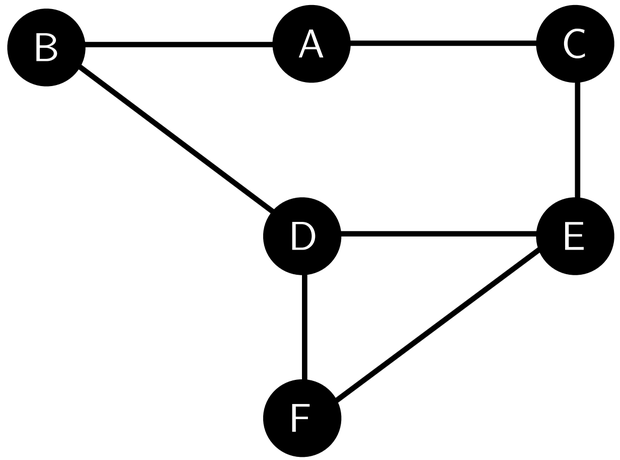


You may encounter variations, where a message computation function msg is defined w.r.t the source node:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\left\{\left\{\text{msg}(\mathbf{h}_u^{(t-1)}, \mathbf{h}_v^{(t-1)}) \mid v \in N(u)\right\}\right\}\right)\right),$$

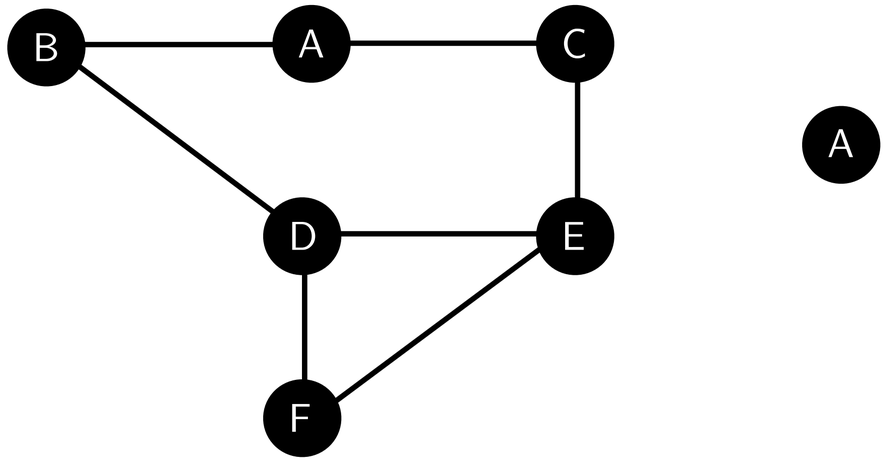
Remark: The function msg typically depends on the neighborhood - hard to decouple msg from $\psi^{(t)}$. Following a common convention, we view the message computation as part of aggregation.

Message Passing Neural Networks



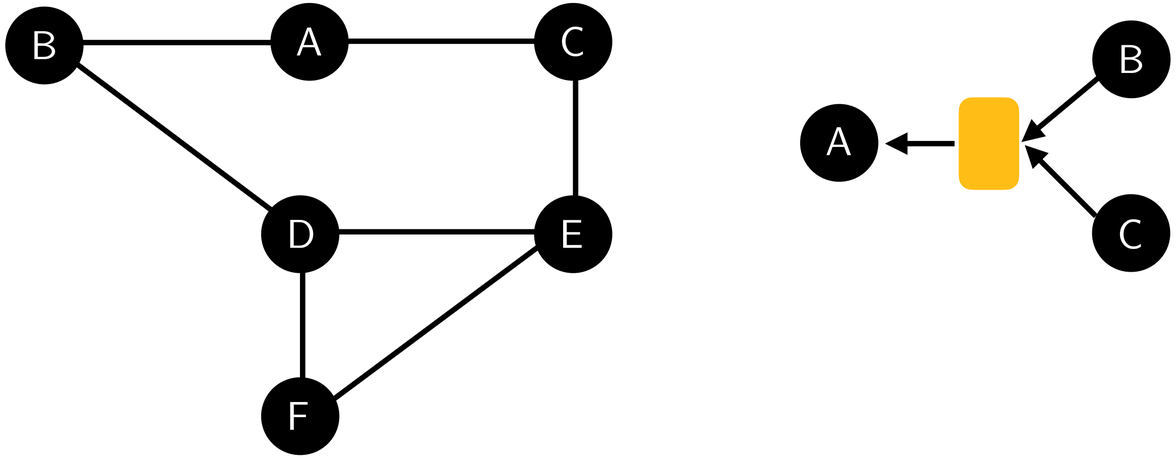
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



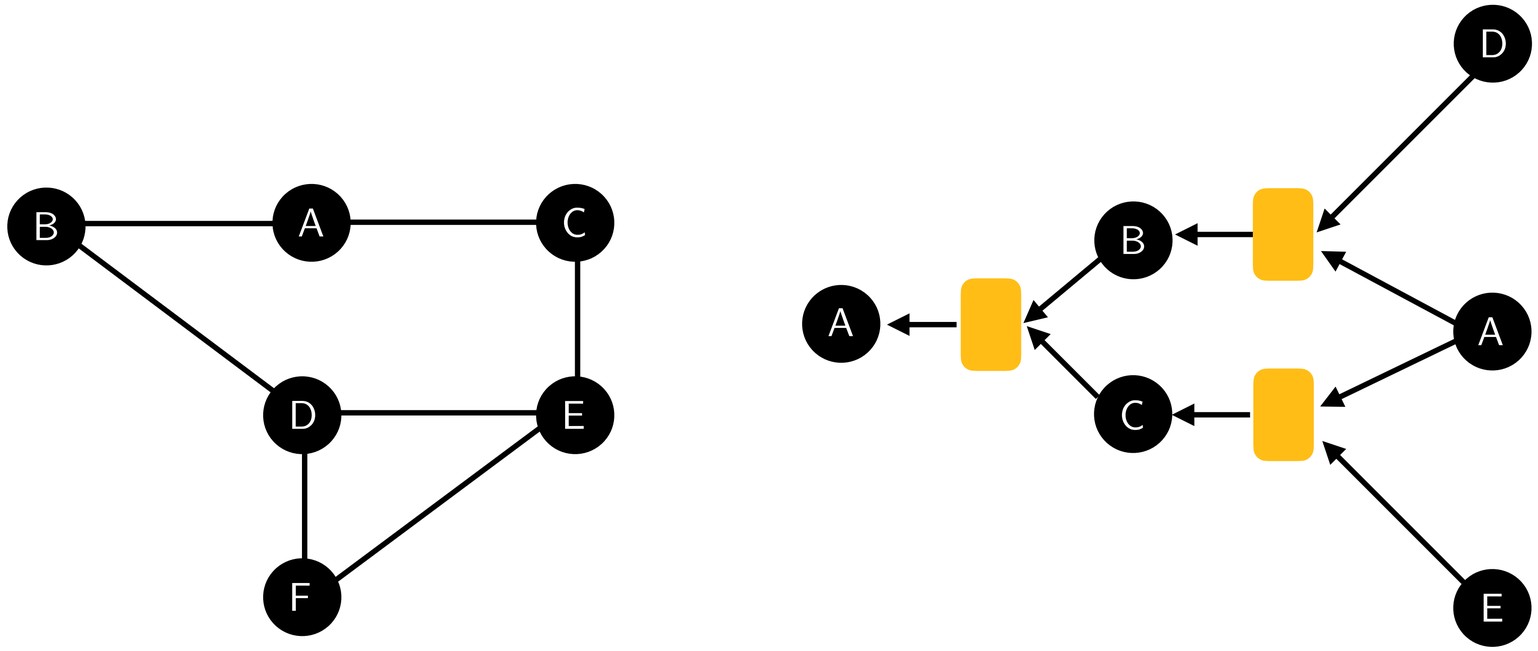
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



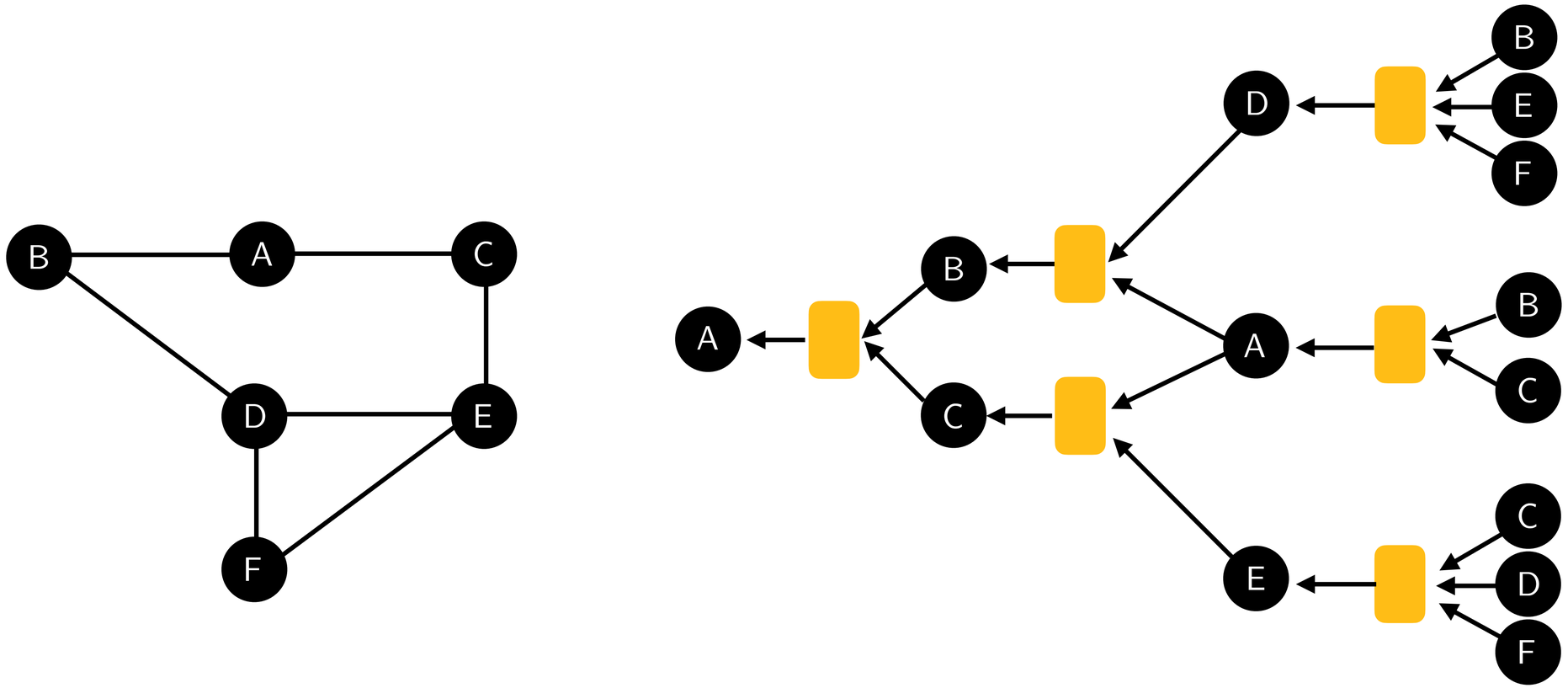
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



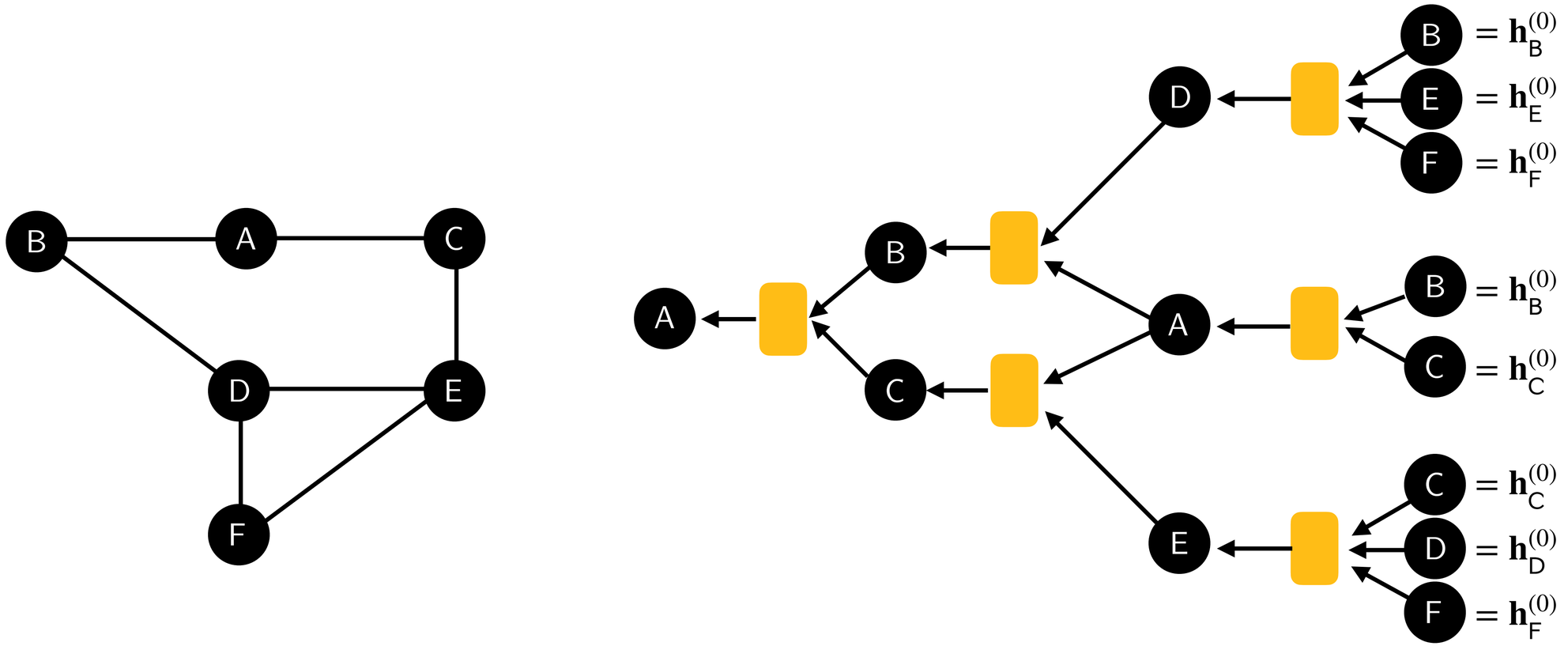
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



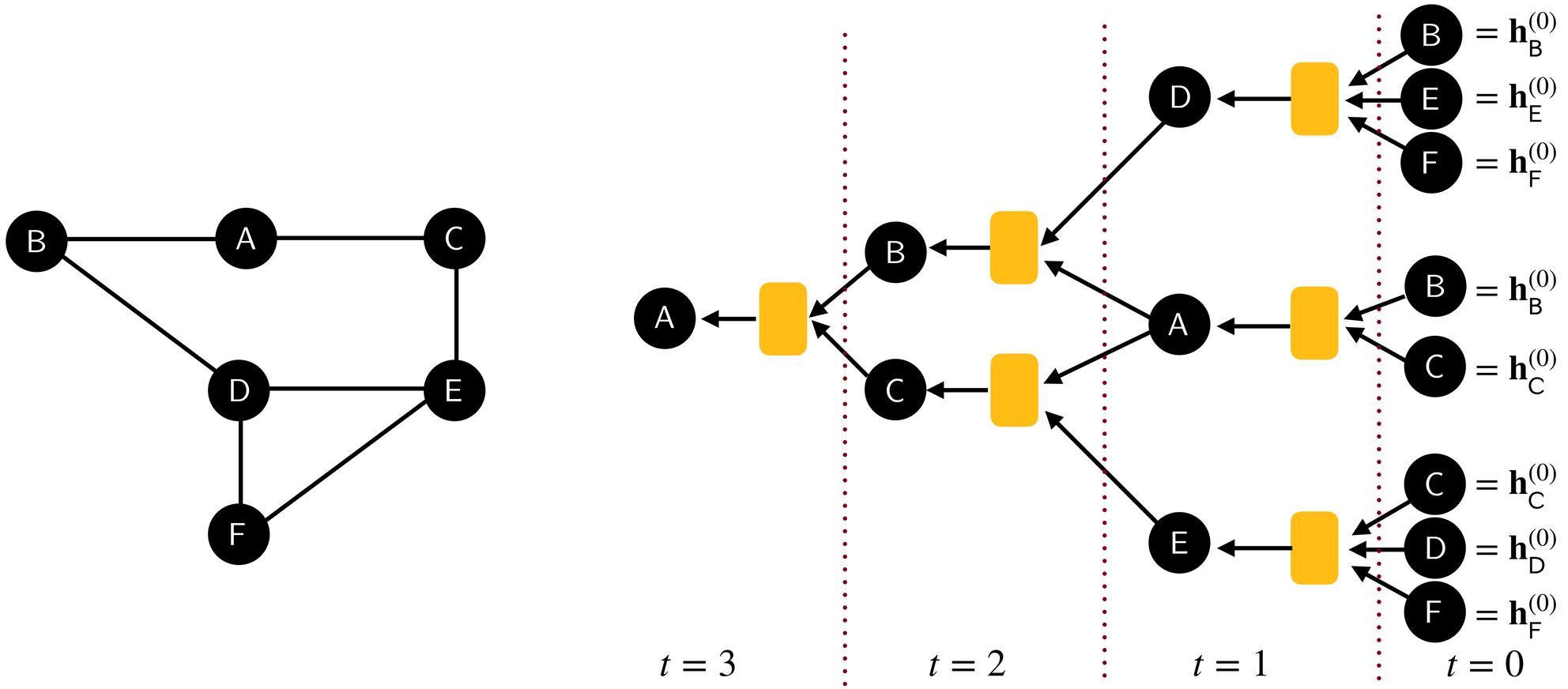
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



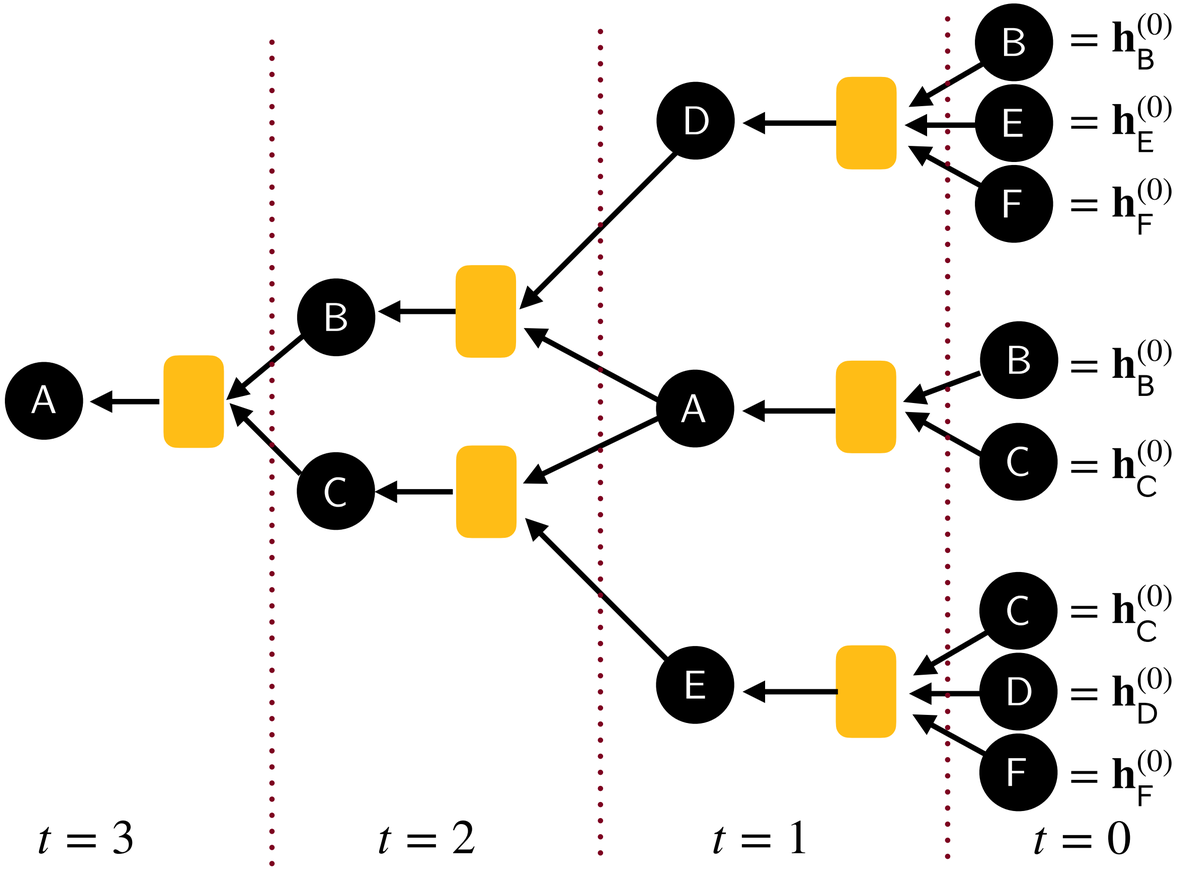
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks

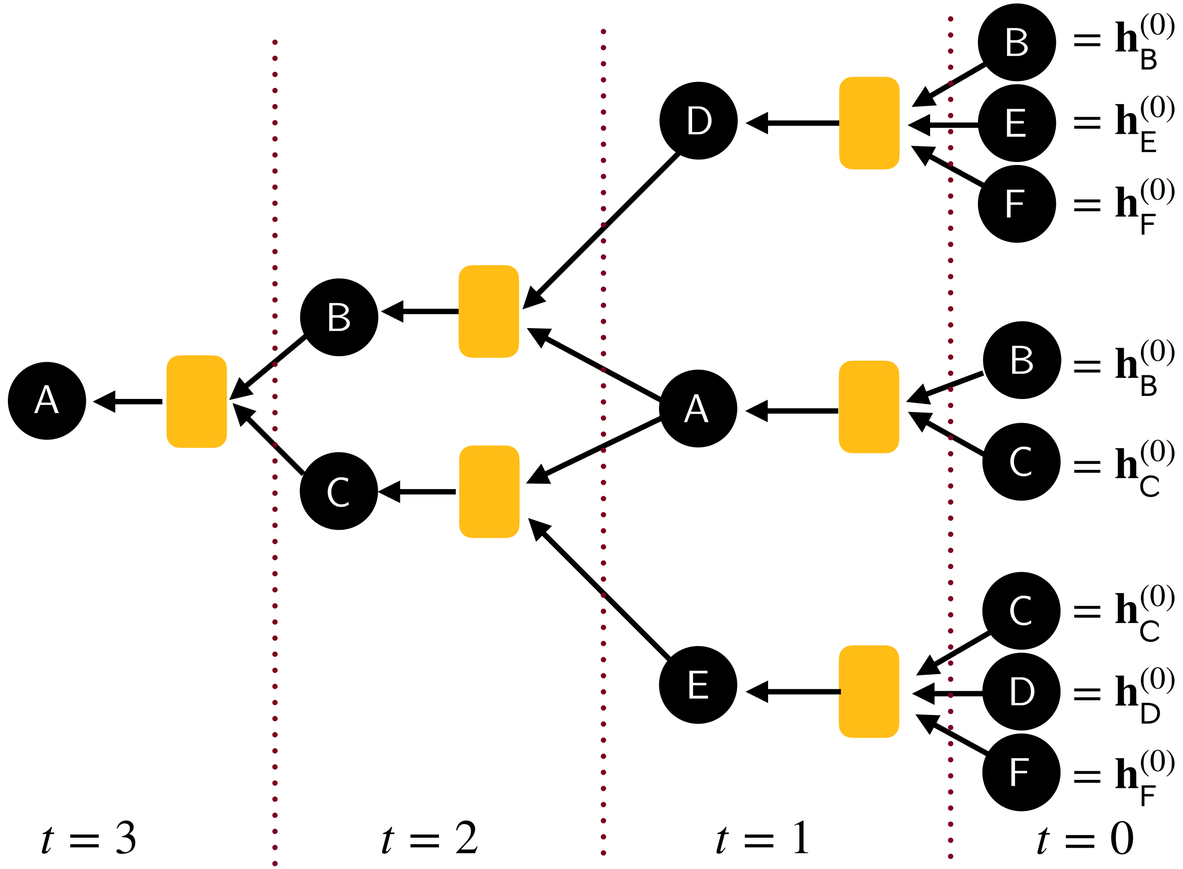


A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks

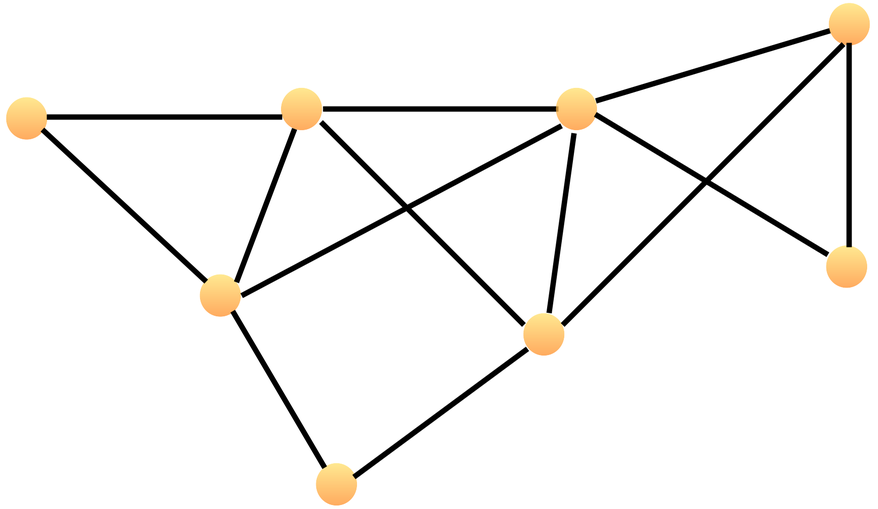


Message Passing Neural Networks



The i-th iteration is the i-th layer of the MPNN, since each iteration can be seen as an "unrolling" of the network. The #layers defines the depth, and the embedding dimensionality the width of the network.

Message Passing Neural Networks



Node-level final representation: The final node representations are denoted as $\mathbf{z}_u = \mathbf{h}_u^{(k)}$.

Graph-level final representation: A final graph embedding $\mathbf{z}_G$ for a graph G through a mapping from the multiset of all the node embeddings $\{\{\mathbf{z}_{u_1} \ldots \mathbf{z}_{u_n}\}\}$ to $\mathbf{z}_G$ known as relational pooling (Murphy et al., 2019).

Common choices are sum, or mean, which are normalized, e.g., w.r.t. number of the nodes.

Deriving a Basic Graph Neural Network Model

Model design space is very large: many possible choices for aggregate and update.

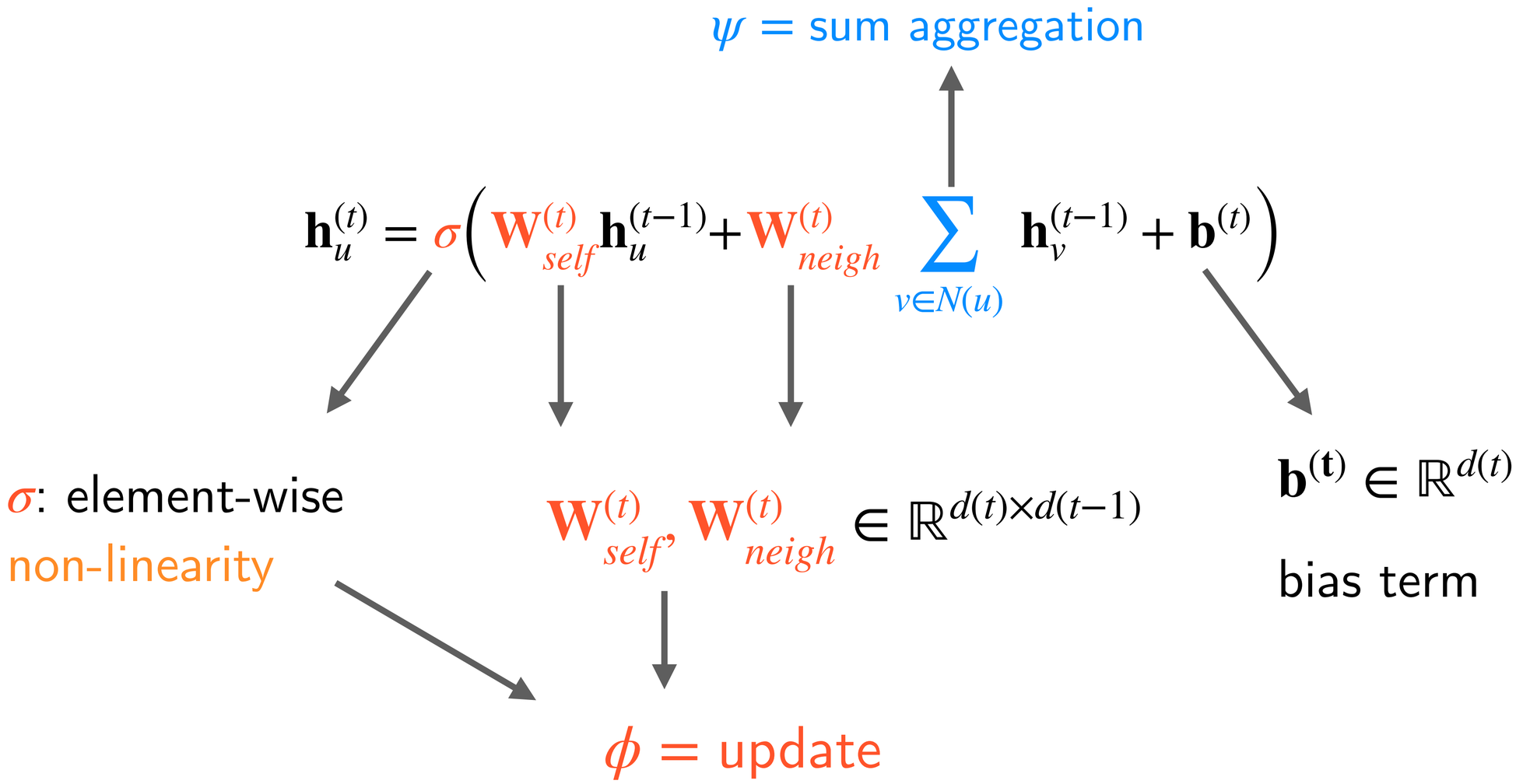
$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \{\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\}\}\right)\right)$$

$$= \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$ aggregate: sum

$$= \sigma\left(\mathbf{W}_{self}^{(t)} \mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

update: linear transformations
with a nonlinearity at the end

The Basic Graph Neural Network Model



Message Passing With Self-Loops

Message passing: Define an aggregate function which treats the source node also as a neighbor:

$$\mathbf{h}_u^{(t)} = \psi^{(t)}\left(\{\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\}\} \cup \{\{\mathbf{h}_u^{(t-1)}\}\}\right)$$

Self-loop: This can be thought as (implicitly) adding self-loops to the nodes, hence the name.

Message Passing With Self-Loops

Message passing: Define an aggregate function which treats the source node also as a neighbor:

$$\mathbf{h}_u^{(t)} = \psi^{(t)}\left(\left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\} \cup \left\{\left\{\mathbf{h}_u^{(t-1)}\right\}\right\}\right)$$

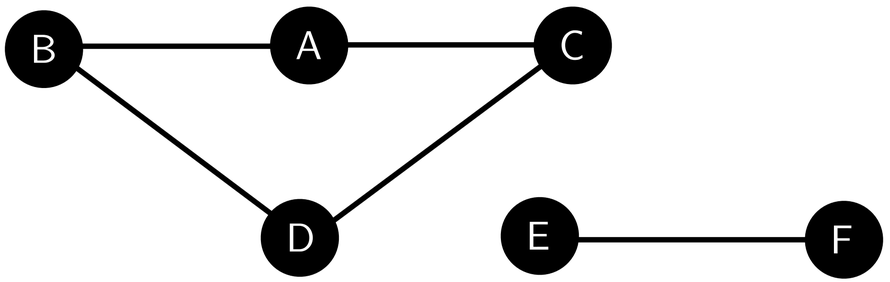
Self-loop: This can be thought as (implicitly) adding self-loops to the nodes, hence the name.

Basic model: Note that this further simplifies the base model:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(x)} \mathbf{h}_v^{(t-1)} + \mathbf{h}_u^{(t-1)}\right)$$

Expressivity: This limits the expressivity since the information coming from the node's neighbor's cannot be differentiated from the information from the node itself.

A Limitation of Message Passing

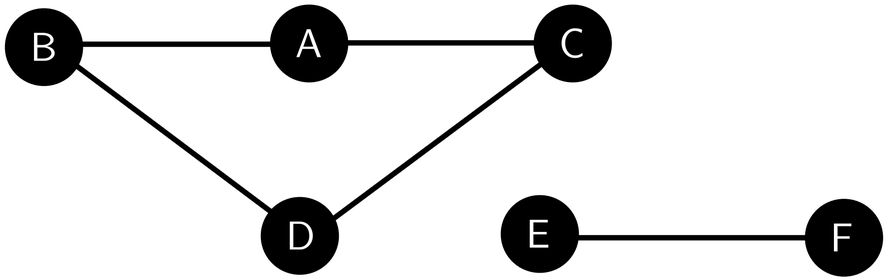


Problem: The presented message passing approach is local: no information flows across disjoint subgraphs.

Remark: Pooling yields a graph embedding, which is global, but there is still no communication between disjoint subgraphs during message passing, so the node embeddings are "blind" to disjoint subgraphs.

Solution: Global feature computation, or global readout, on each layer of the MPNN (Battaglia et al., 2018).

Message Passing with Global Readout

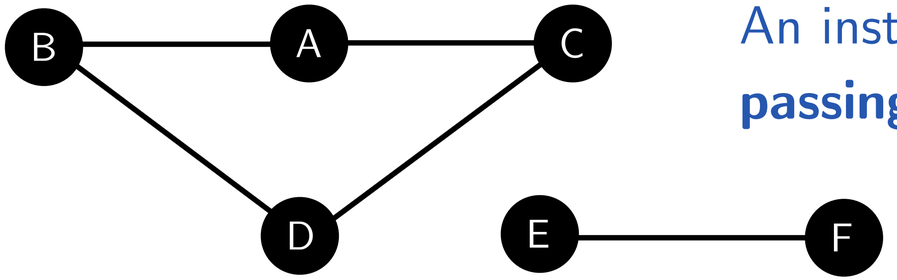


The representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right), \gamma^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_w^{(t-1)} \mid w \in G\right\}\right\}\right)\right),$$

where $\gamma^{(t)}$ is a differentiable function, and all aggregate functions are typical candidates also for $\gamma^{(t)}$.

Message Passing with Global Readout



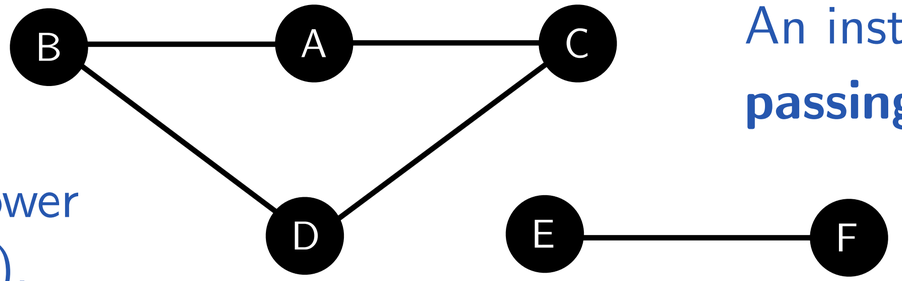
An instance of **generalized message passing** (Battaglia et al., 2018)

The representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right), \gamma^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_w^{(t-1)} \mid w \in G\right\}\right\}\right)\right),$$

where $\gamma^{(t)}$ is a differentiable function, and all aggregate functions are typical candidates also for $\gamma^{(t)}$.

Message Passing with Global Readout



An instance of **generalized message passing** (Battaglia et al., 2018)

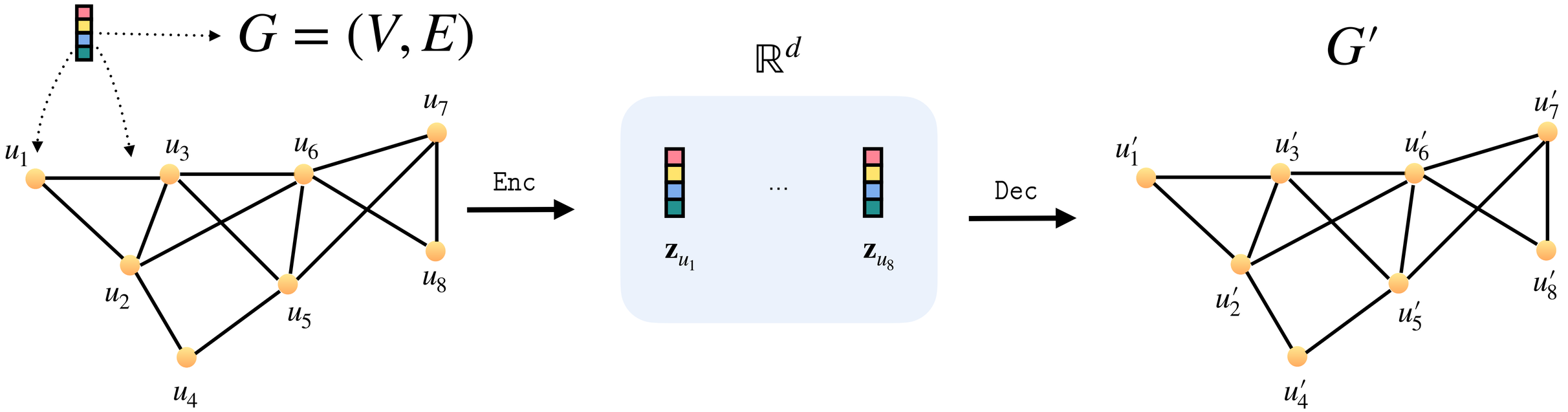
A difference in the expressive power of MPNNs (Barcelo et al., 2020).

The representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right), \gamma^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_w^{(t-1)} \mid w \in G\right\}\right\}\right)\right),$$

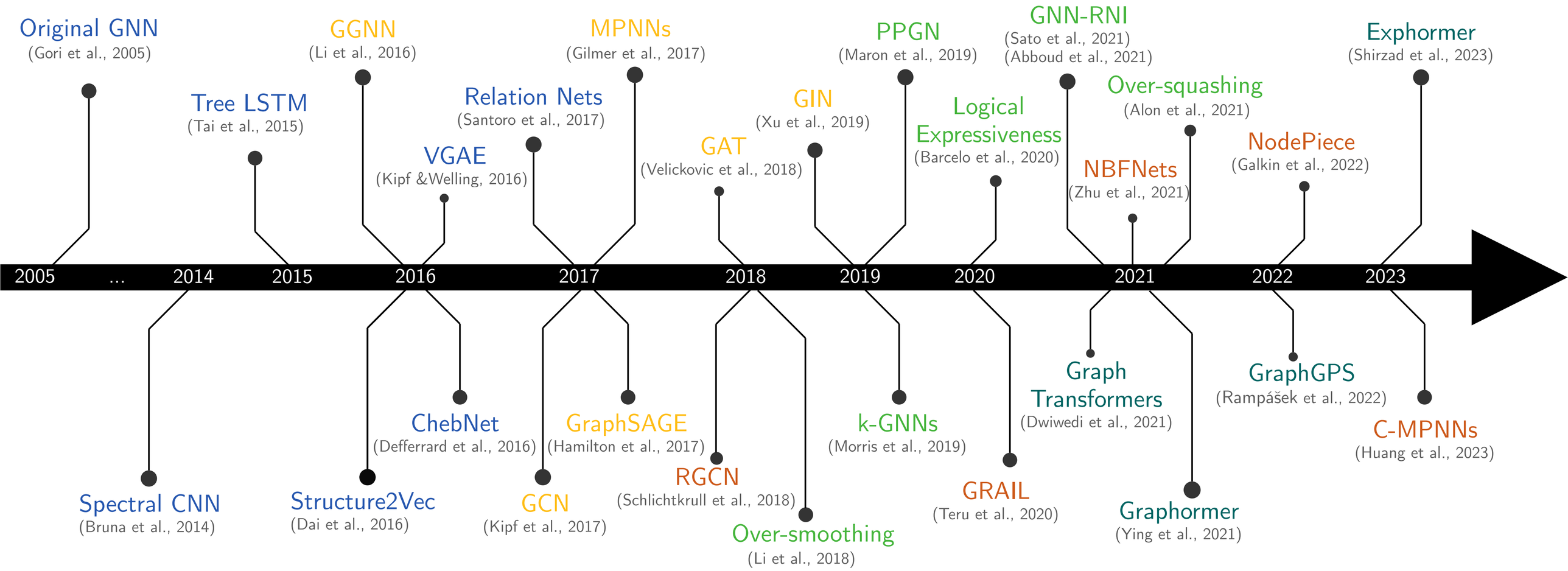
where $\gamma^{(t)}$ is a differentiable function, and all aggregate functions are typical candidates also for $\gamma^{(t)}$.

Encoder-Decoder



The learned embeddings can be used for many graph machine learning task, e.g., graph/node classification, graph/node regression, graph/node clustering, depending how they are learned.

Graph Neural Networks



Graph Convolutional Networks

Graph Convolutional Networks

The base GCN model (Kipf et al., 2017) can be seen as a self-loop message passing approach:

Sum aggregation over degree-normalized features

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u)N(v)}} \right)$$

What is the connection to convolutions?

Self-loop approach: single parameter matrix with a non-linearity

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \mathbf{h}_v^{(t-1)} \right)$$

Very similar to the basic self-loop approach

Revisiting the Basic Model

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)} \mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)} \right)$$

Node-level

$\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$: Node representations at layer t

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{H}^{(t-1)} \mathbf{W}_{self}^{(t)} + \mathbf{A}\, \mathbf{H}^{(t-1)} \mathbf{W}_{neigh}^{(t)} \right)$$

Graph-level

Identity

Adjacency

Revisiting the Basic Model

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)} \mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)} \right)$$

Node-level

$\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$: Node representations at layer t

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{H}^{(t-1)} \mathbf{W}_{self}^{(t)} + \mathbf{A}\, \mathbf{H}^{(t-1)} \mathbf{W}_{neigh}^{(t)} \right)$$

Graph-level

Identity

Adjacency

Filter: The layers apply a $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}$, combined with some weight matrices and a non-linearity.

Graph Convolutional Networks

$$\mathbf{H}^{(t)} = \sigma\left(\tilde{\mathbf{A}}_{sym}\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\right)$$

$$\left(\mathbf{D}+\mathbf{I}\right)^{-\frac{1}{2}}\left(\mathbf{I}+\mathbf{A}\right)\left(\mathbf{D}+\mathbf{I}\right)^{-\frac{1}{2}}$$

GCN is a local, first-order approximation of spectral graph convolution based on Chebyshev polynomials.

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u)N(v)}}\right)$$

GCN applies filters based on the symmetric normalized adjacency matrix, ensures values: [0,1]

Intuitively, in the base GCN model:

- $\tilde{\mathbf{A}}_{sym}$ enables messaging between neighbors and with node's self representation through the identity.
- Node's own embedding is treated identically to messages from other nodes: self-loops.

Graph Attention Networks

Learning Aggregation

$$\sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}$$

Fixed aggregation: sum

$$\sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u)N(v)}}$$

Fixed aggregation: sum over degree-normalized features

$$\sum_{v \in N(u)} \mathbf{W}\mathbf{h}_v^{(t-1)}$$

Sum aggregation with learnable transformation matrix on features

Goal: Learn to aggregate non-uniformly across neighbors?

Idea: Use attention as a means to non-uniformly aggregate over the neighborhood.

Background: Attention models obtained strong results in, e.g., machine translation (Bahdanau et al., 2015).

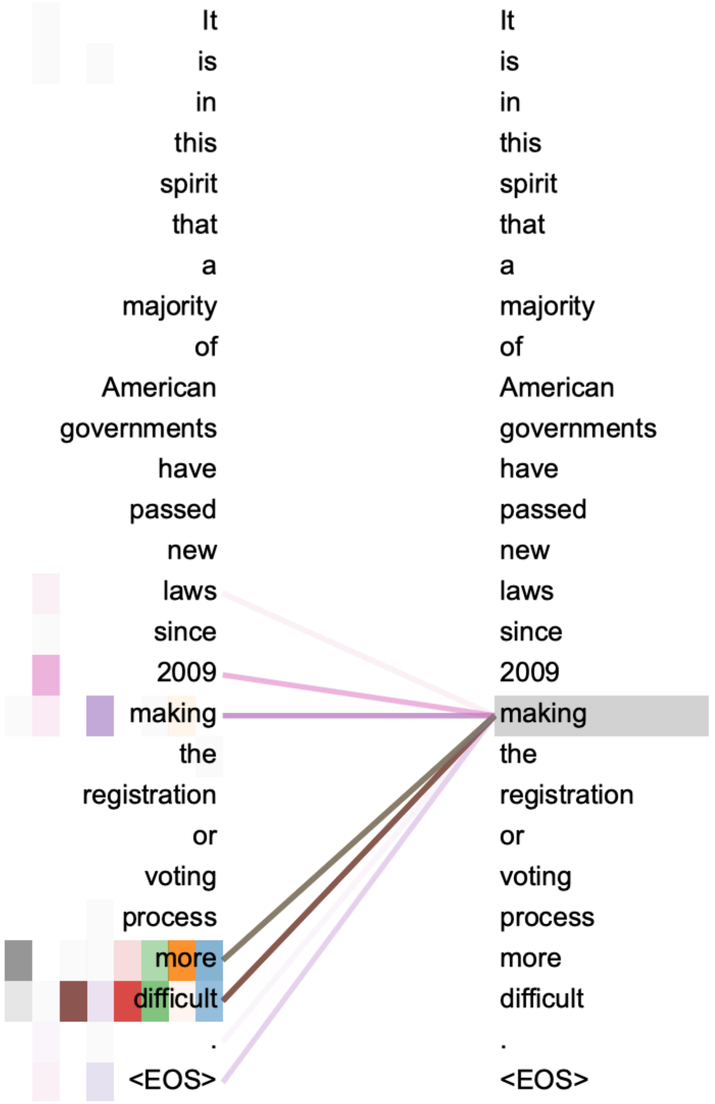
Attention

Attention: Allocate different weights to distinct inputs, based on their relevance to the learned task.

Transformer (Vaswani et al., 2017): Figure shows attention weights for the word 'making' encoding "making more difficult".

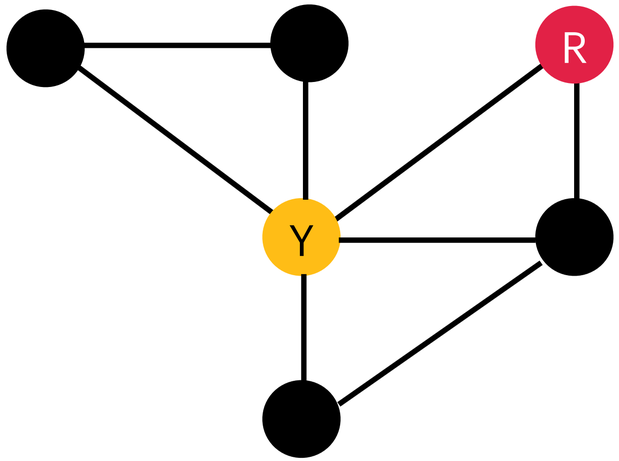
Breaking uniformity: Attend to more relevant tokens, rather than uniformly considering all possible tokens.

Graph attention: A node can benefit from weighing the relative importance of its neighbors.



Attention over Graphs

Example: Classify all nodes connected to a red node as true and every other node as false.

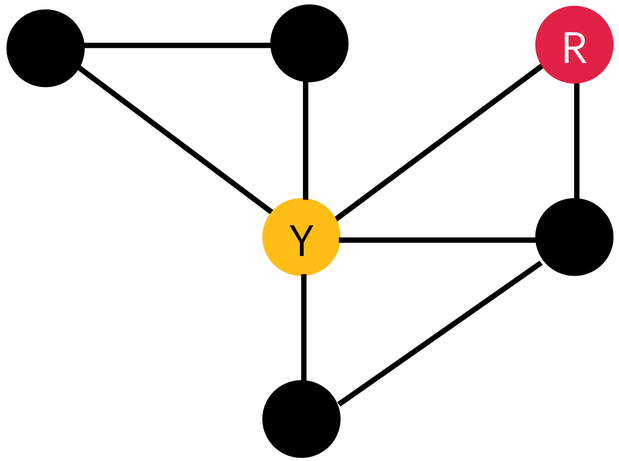


This task relies only to the fact that a node has a red neighbor.

Neighborhood attention: Richer weighing of a node's neighbors, which results in potentially more descriptive and task-specific aggregation schemes.

Idea: Learn an attention weight for each neighbor, which yields weighted aggregation functions.

Graph Attention Networks



Graph attention networks (GAT) (Velickovic et al., 2018) use a weighted sum aggregation, with a pairwise node attention mechanism during message passing (using a self-loop approach):

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \alpha_{(u,v)}\, \mathbf{h}_v^{(t-1)}\right),$$

where $\alpha_{u,v}$ is the attention weight on a node $v \in N(u) \cup \{u\}$ with respect to a source node u.

Graph Attention Networks

Aggregate: Weighted sum based on attention

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \alpha_{(u,v)} \, \mathbf{h}_v^{(t-1)} \right)$$

Update: linear transformation combined with non-linearity

$$\alpha_{u,v} = \frac{\exp(e_{u,v})}{\sum_{v' \in N(u)} \exp(e_{u,v'})}$$

$$e_{u,v} = \text{LeakyReLU}(\mathbf{h}_u^\top \mathbf{W} \mathbf{h}_v) \quad \text{Bilinear}$$

$$e_{u,v} = \text{LeakyReLU}(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]) \quad \text{GAT}$$

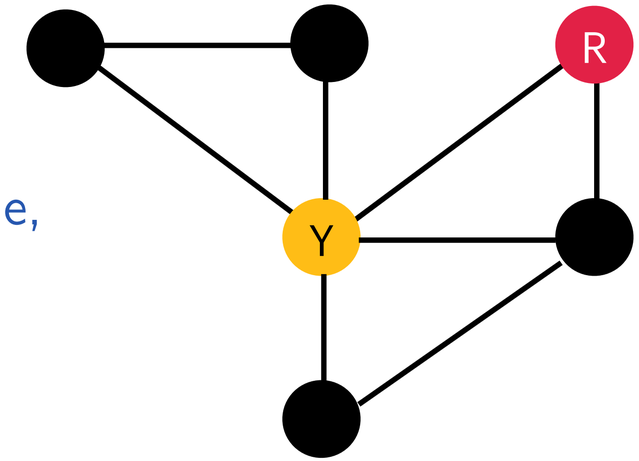
$$e_{u,v} = \mathbf{a}^\top \text{LeakyReLU}(\mathbf{W}[\mathbf{h}_u \oplus \mathbf{h}_v]) \quad \text{GATv2}$$

GAT applies $\mathbf{a}^\top$ and $\mathbf{W}$ consecutively and these can be collapsed into single linear layer!

GATv2 (Brody et al., 2022) avoids by first applying the non-linearity.

Graph Attention Networks

Multi-head attention: Learn multiple, distinct, independently parametrized attention weights.



Transformer: Multiple attention heads to compute attention weights between all pairs of positions in the input.

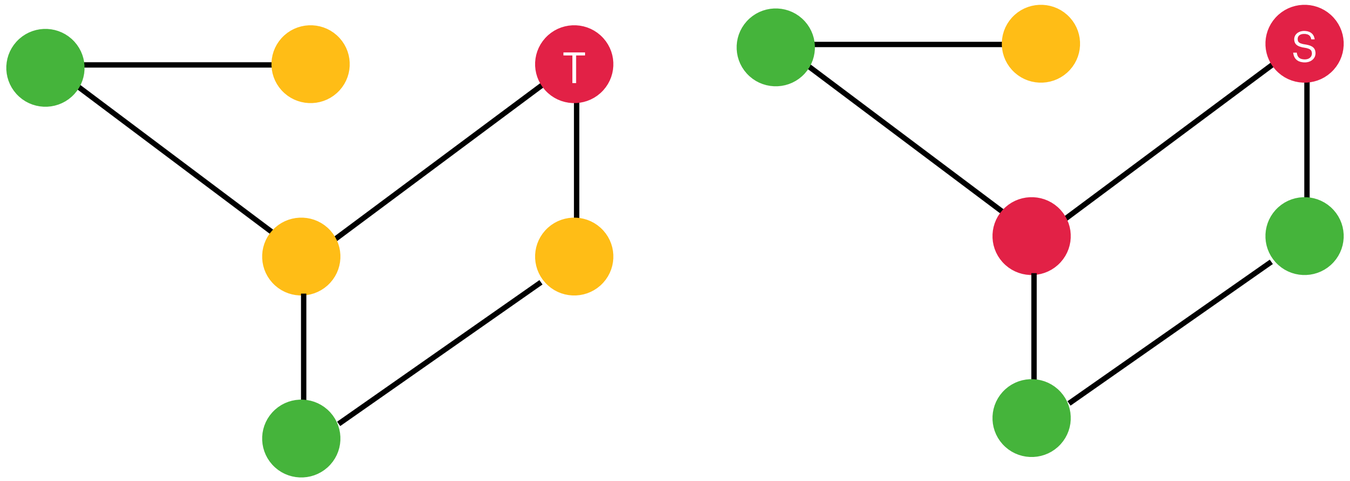
Coincides with GAT using multi-head attention on a fully connected graph.

- Learn k attention weights $\alpha_{u,v,1}, \ldots, \alpha_{u,v,k}$ for the nodes u, v.
- Concatenate resulting k node representations $\mathbf{h}_u[1], \ldots, \mathbf{h}_u[k]$ for each node u:

$$\mathbf{h}_u = \mathbf{h}_u[1] \oplus \ldots \oplus \mathbf{h}_u[k]$$

Graph Isomorphism Networks

A Closer Look at Aggregation

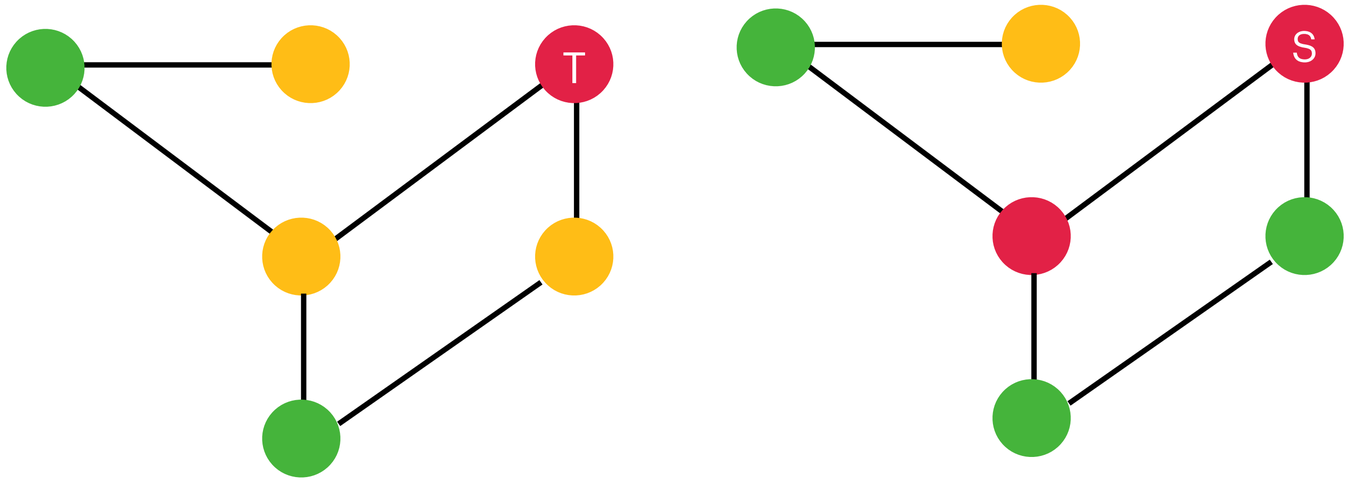


Question: What is the impact of different choices of aggregation on the discrimination ability of GNNs?

Task: Input graph with node types red, green and yellow, where the features are the RGB values.

Setup: Consider a red node to analyze how different functions aggregate neighbor messages.

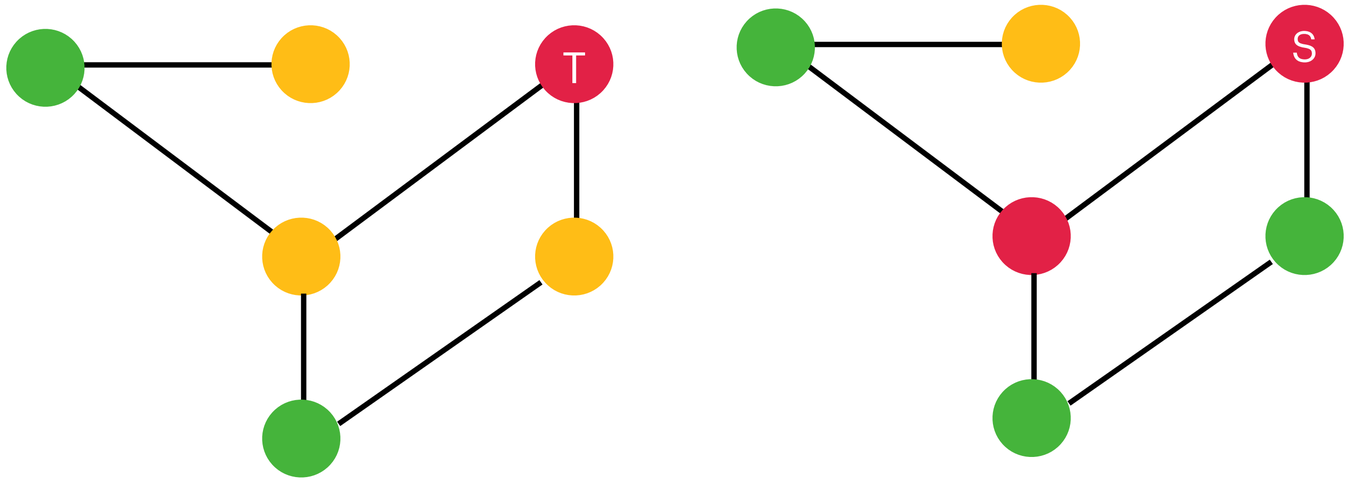
A Closer Look at Aggregation



Sum: Can discern between neighborhoods based on their sizes, but it can lead to false equality.

Example: Sum cannot distinguish between a 2-yellow and a red-green neighborhood.

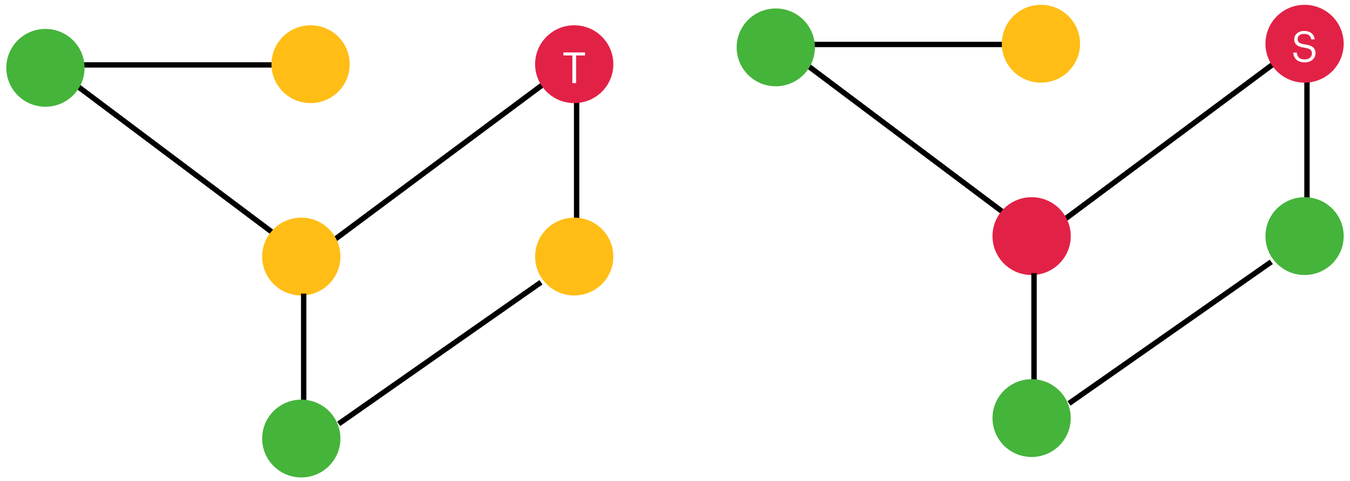
A Closer Look at Aggregation



Mean: Useful for bounding the range of aggregate messages, but cannot recognize multiplicities.

Example: 2-red or 3-red neighbours are indifferent, as the mean operation eliminates cardinality.

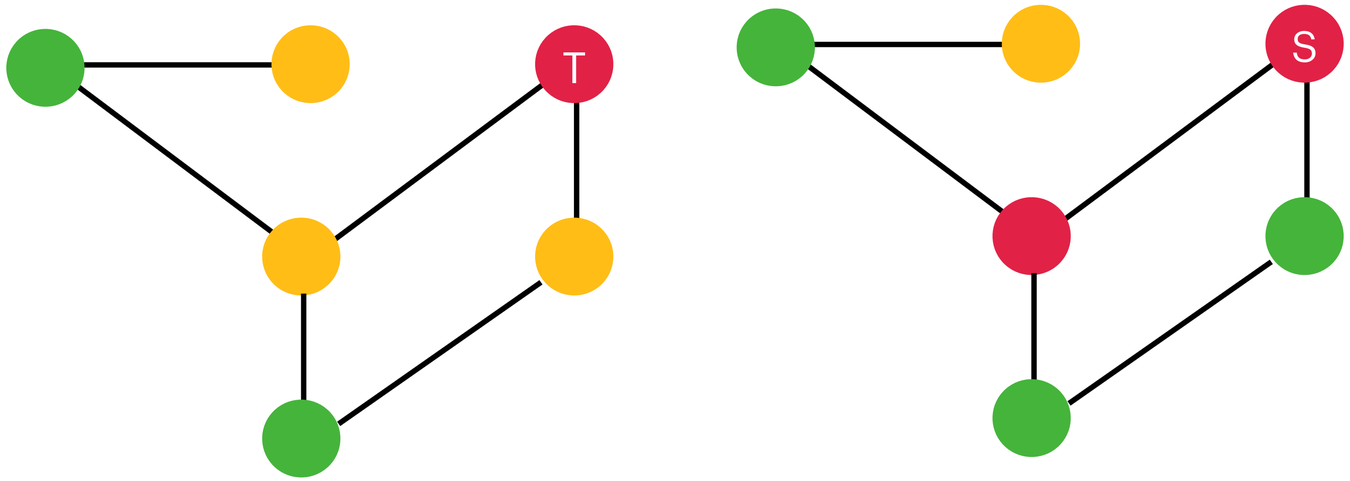
A Closer Look at Aggregation



Max: Highlights a relevant element, but limited in discriminative ability.

Example: Considering red $<$ yellow $<$ green, green is answer for any neighborhood involving at least 1 green node.

Aggregation and Expressiveness



Observation: An aggregation function must distinguish between distinct neighborhoods, and return different results given different neighborhood multisets.

Injective: The aggregation function must be injective relative to the neighborhood.

Expressive power: MPNNs are at their maximal expressiveness with injective functions (Xu et al.,2019).

Aggregation and Expressiveness

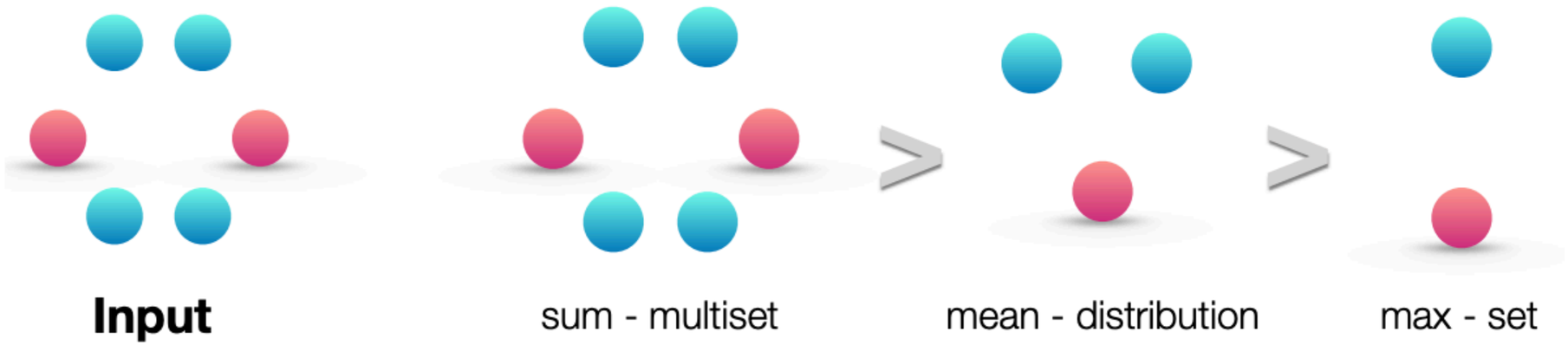
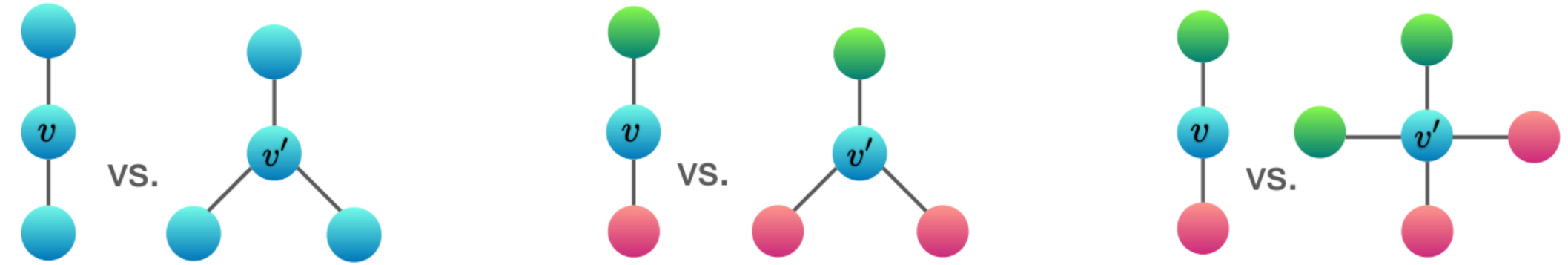


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset.** Left panel shows the input multiset, *i.e.*, the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities (reduces the multiset to a simple set). (Xu et al., 2019)

Aggregation and Expressiveness

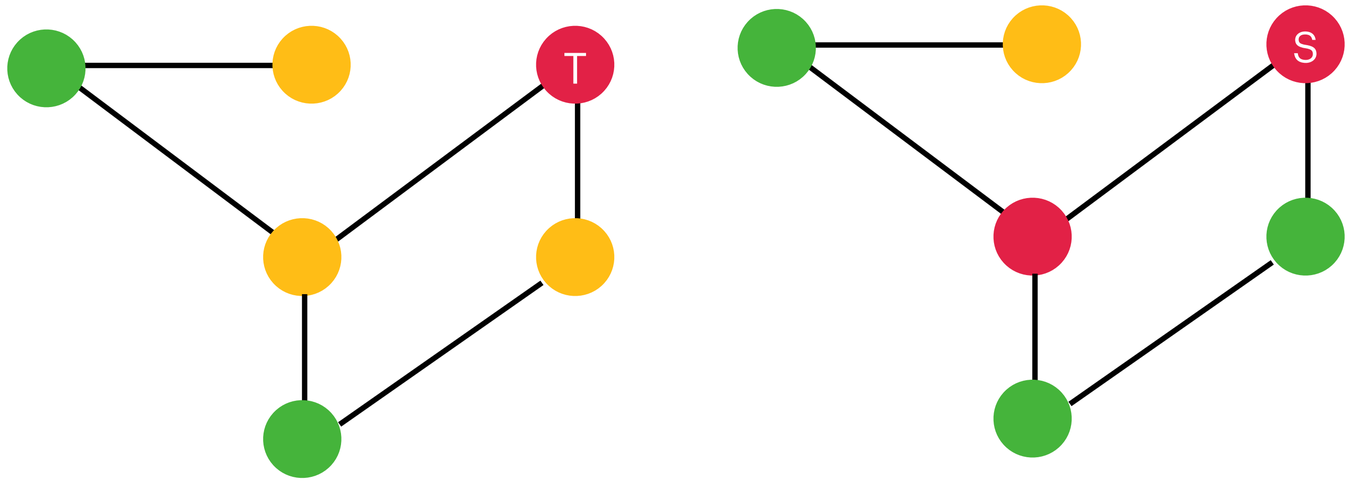


(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

Figure 3: **Examples of graph structures that mean and max aggregators fail to distinguish.** Between the two graphs, nodes v and v' get the same embedding even though their corresponding graph structures differ. Figure 2 gives reasoning about how different aggregators "compress" different multisets and thus fail to distinguish them.

(Xu et al., 2019)

Graph Isomorphism Networks

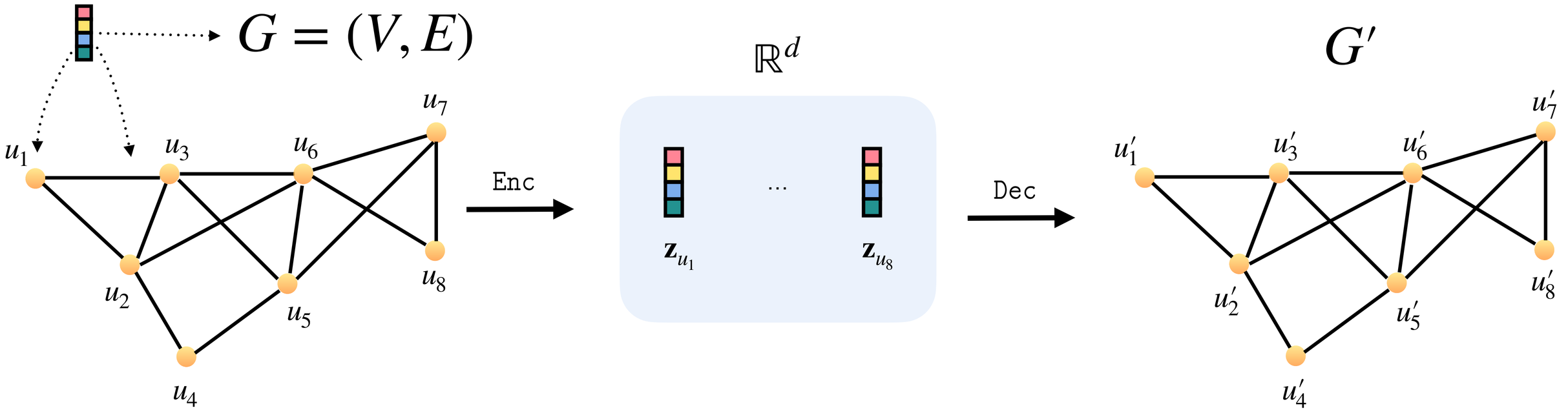


Graph isomorphism networks (GINs) (Xu et al., 2019) update the representation $\mathbf{h}_u$ for each node $u \in V$ as:

$$\mathbf{h}_u^{(t)} = MLP\left((1+\epsilon)\cdot \mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

...and GIN layers are injective.

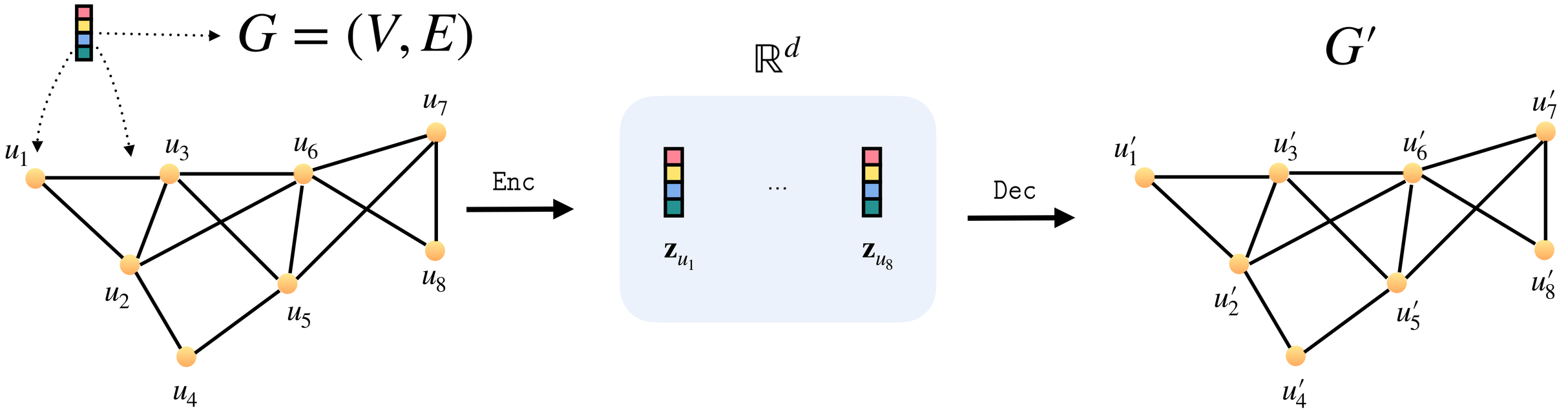
Graph Representation Learning



Graph representation learning with strong relational inductive bias

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

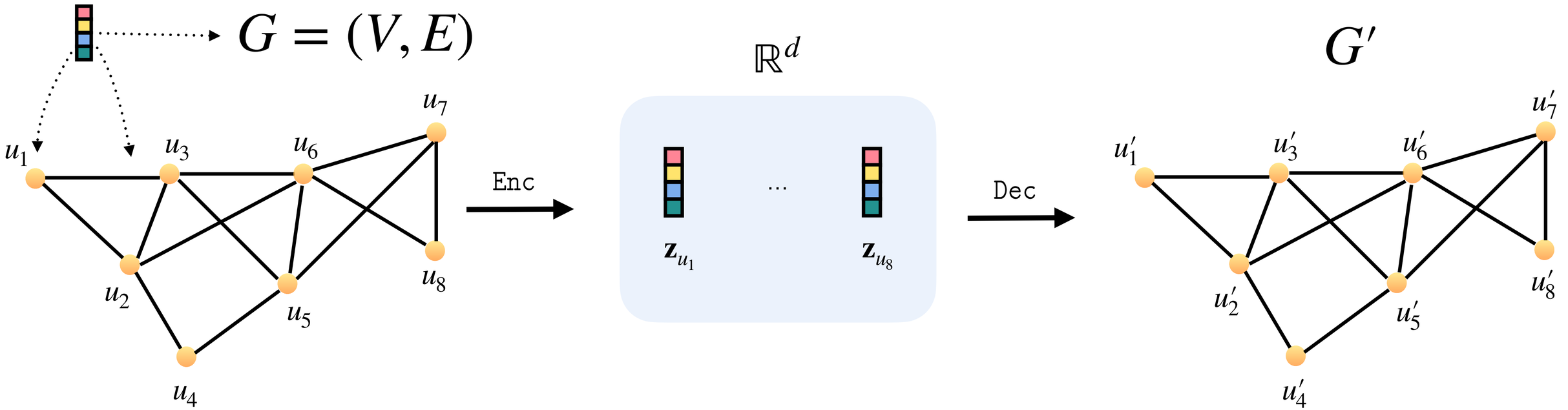
Graph Representation Learning



Learned parameters are independent of graph size

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

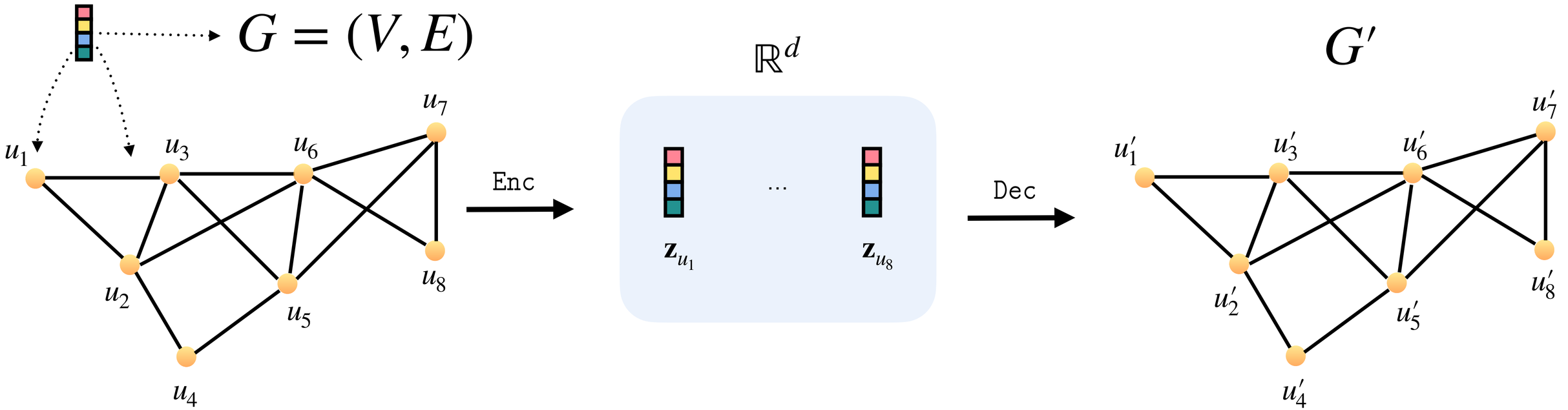
Graph Representation Learning



Applies to variable-size graphs

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

Graph Representation Learning

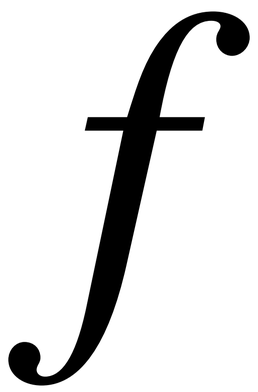


What is the expressive power?

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

A Journey into Model Representation Capacity

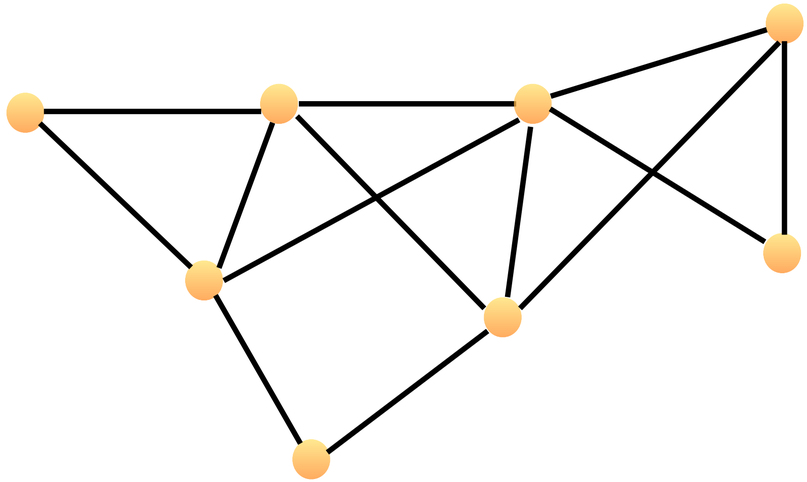
Model Representation Capacity



Expressive power: Capacity of a model (e.g., neural network) to approximate functions.

Feedforward networks: MLPs can approximate any continuous function f on a compact domain: for any such function, there is a parameter configuration for an MLP, corresponding to an approximation of the function (Cybenko, 1989; Funahashi, 1989; Hornik et al., 1989).

Model Representation Capacity

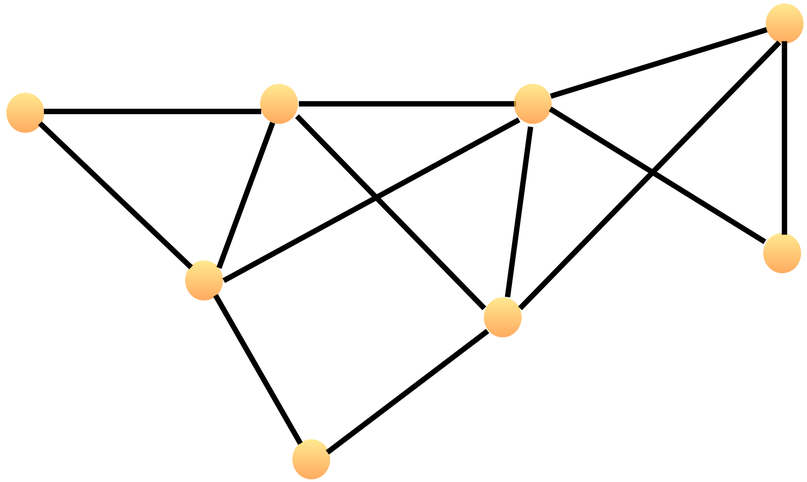


Expressive Power in the World of Graphs: One way of characterizing the expressive power would be through graph distinguishability. Learn graph embeddings $\mathbf{z}_G$, $\mathbf{z}_H$ for graphs G and H:

$$\mathbf{z}_G = \mathbf{z}_H \text{ if and only if } G \text{ is isomorphic to } H$$

Model Representation Capacity

Problem: Contains graph isomorphism testing, an NP-intermediate problem, where the best algorithm requires quasi-polynomial time (Babai, 2016).

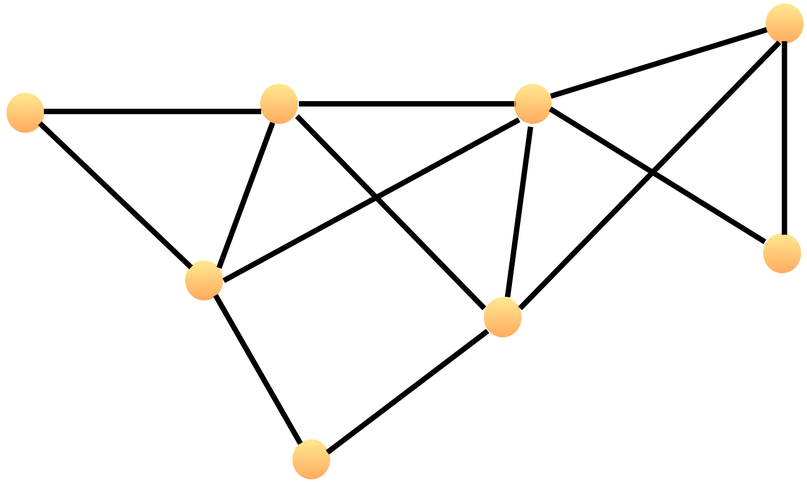


Expressive Power in the World of Graphs: One way of characterizing the expressive power would be through graph distinguishability. Learn graph embeddings $\mathbf{z}_G$, $\mathbf{z}_H$ for graphs G and H:

$$\mathbf{z}_G = \mathbf{z}_H \text{ if and only if } G \text{ is isomorphic to } H$$

Model Representation Capacity

Problem: Contains graph isomorphism testing, an NP-intermediate problem, where the best algorithm requires quasi-polynomial time (Babai, 2016).

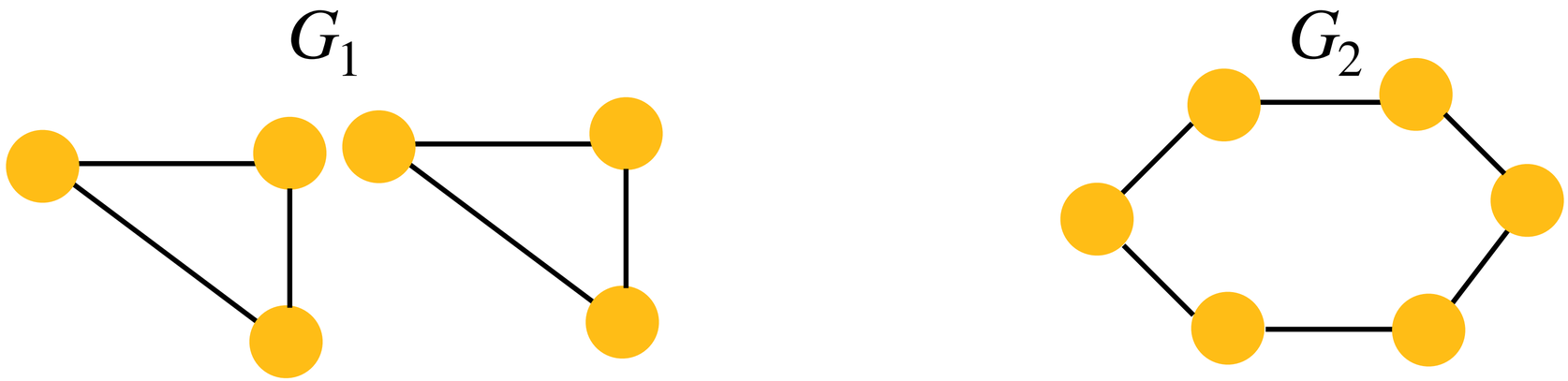


Question: Where do MPNNs stand in graph distinguishability?

Expressive Power in the World of Graphs: One way of characterizing the expressive power would be through graph distinguishability. Learn graph embeddings $\mathbf{z}_G$, $\mathbf{z}_H$ for graphs G and H:

$$\mathbf{z}_G = \mathbf{z}_H \text{ if and only if } G \text{ is isomorphic to } H$$

A Tale of Two Graphs



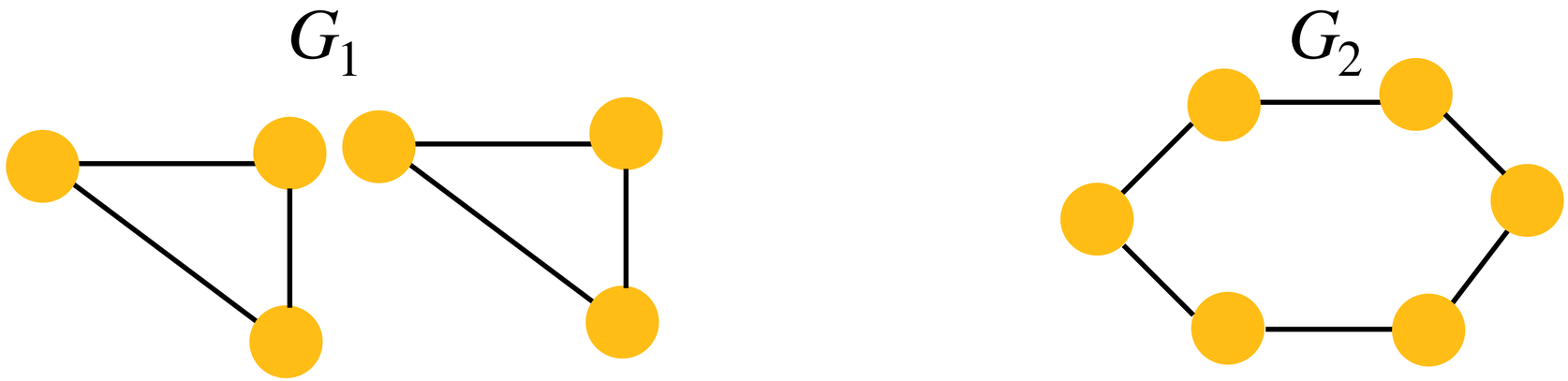
Problem: Any MPNN will learn identical representations for the graphs G_1 and G_2.

MPNNs cannot distinguish between two triangles and a 6-cycle: severe limitation for graph classification!

Predictions for these graphs will be identical regardless of the function we are trying to learn!

Is this only a problem for graph classification?

A Tale of Two Graphs



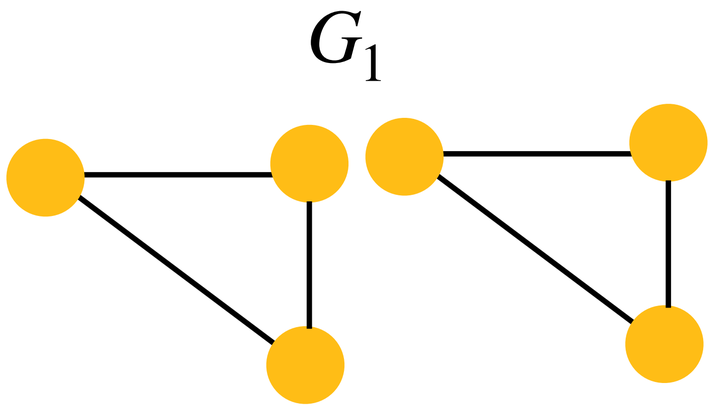
Separator: A node is a separator node if it has two neighbors which are non-adjacent to one another.

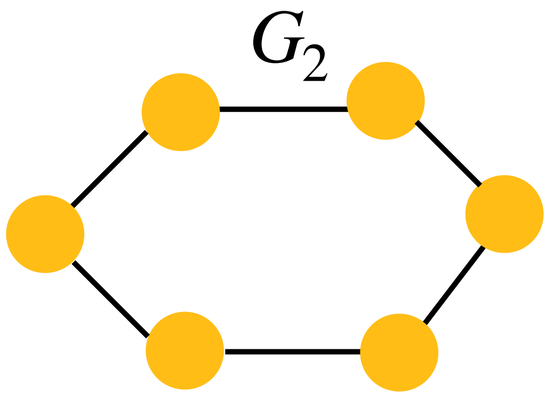
Input: Consider the graph G that is the disjoint union of the graphs G_1 and G_2.

Node classification task: Classify the nodes of G as separator or non-separator.

An MPNN randomly predicts all nodes to be separator nodes, or all of them as non-separator nodes.

A Tale of Two Graphs





All nodes are non-separator

Separator: A node is a separator node if it has two neighbors which are non-adjacent to one another.

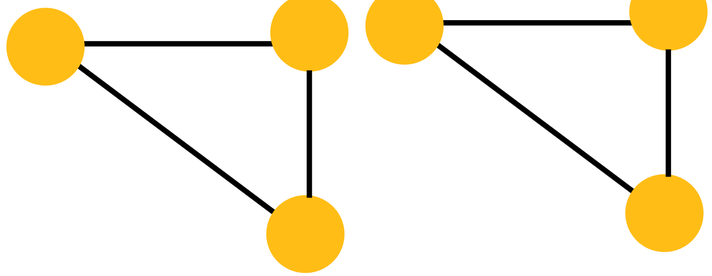
Input: Consider the graph G that is the disjoint union of the graphs G_1 and G_2.

Node classification task: Classify the nodes of G as separator or non-separator.

An MPNN randomly predicts all nodes to be separator nodes, or all of them as non-separator nodes.

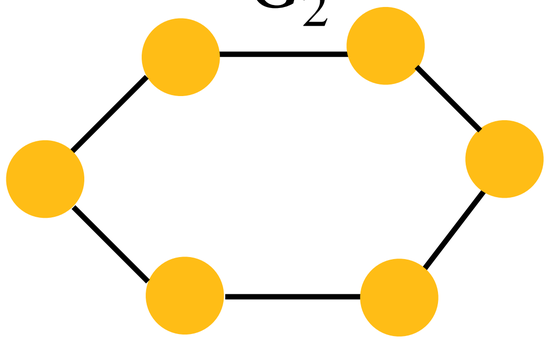
A Tale of Two Graphs

G_1



All nodes are non-separator

G_2



All nodes are separator

Separator: A node is a separator node if it has two neighbors which are non-adjacent to one another.

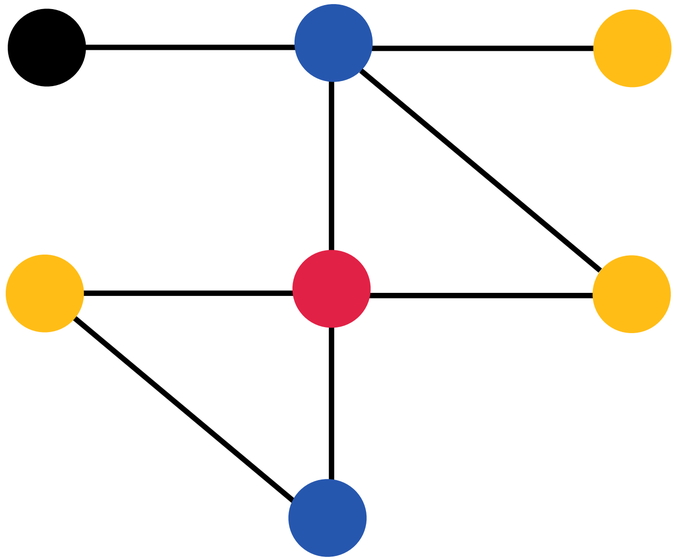
Input: Consider the graph G that is the disjoint union of the graphs G_1 and G_2.

Node classification task: Classify the nodes of G as separator or non-separator.

An MPNN randomly predicts all nodes to be separator nodes, or all of them as non-separator nodes.

Graph Isomorphism and Color Refinement

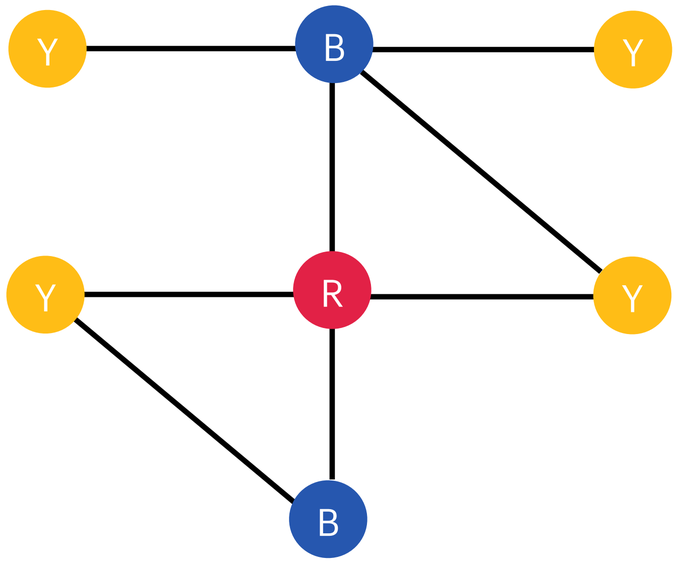
Color Refinement



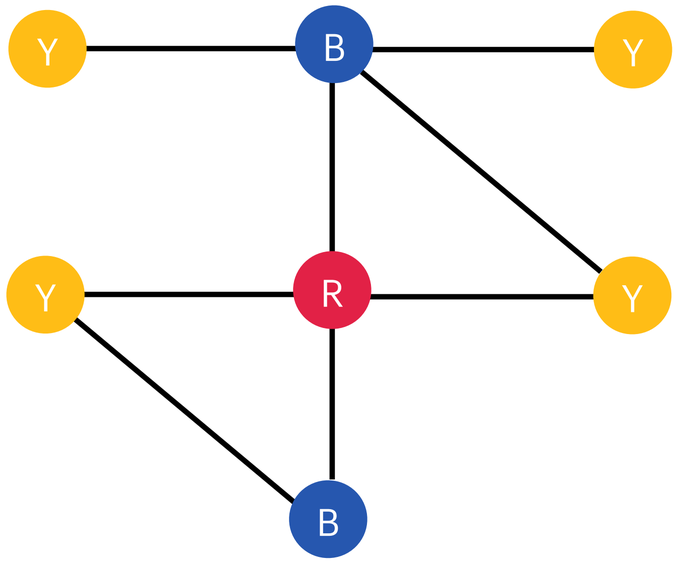
Color refinement is a simple and effective algorithm for graph isomorphism testing:

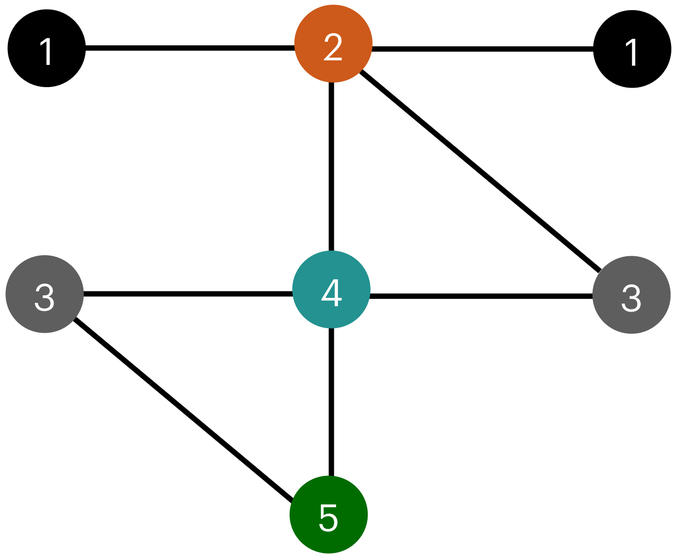
1. **Initialization**: All nodes in a graph are initialized to their initial colors.
2. **Refinement**: All nodes are re-colored depending on their current color and the colors in their neighborhoods.
3. **Stop**: Terminate when the coloring stabilizes.

Color Refinement: Example

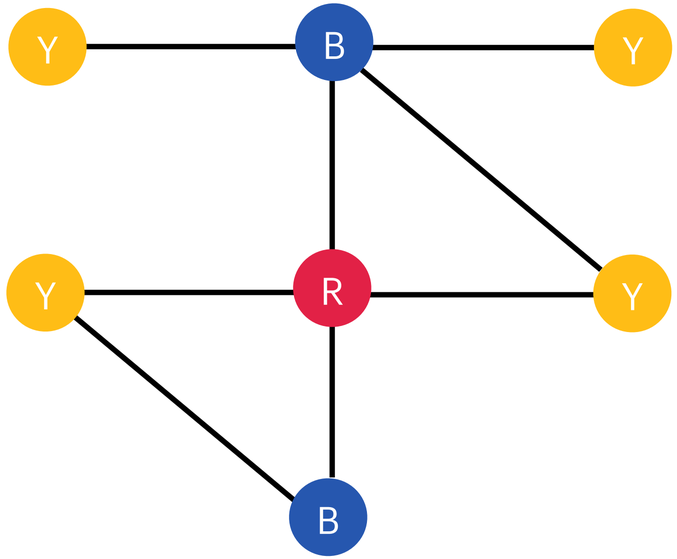


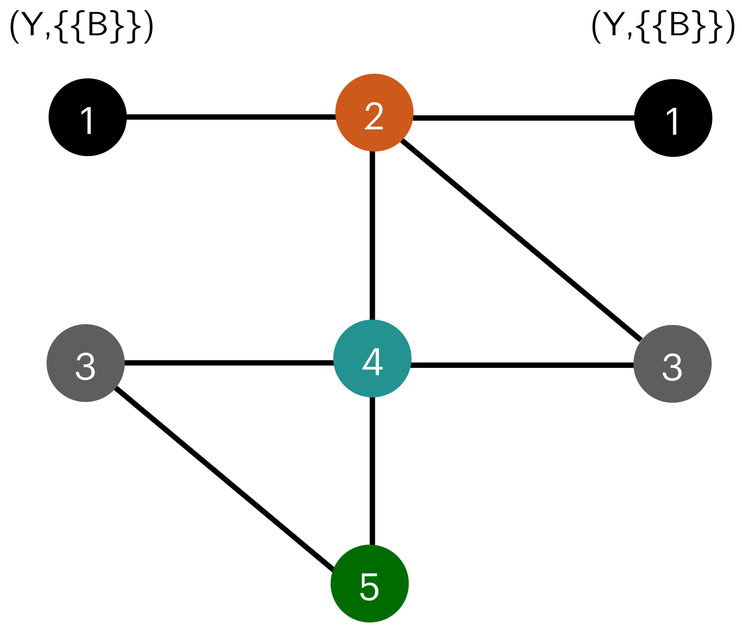
Color Refinement: Example



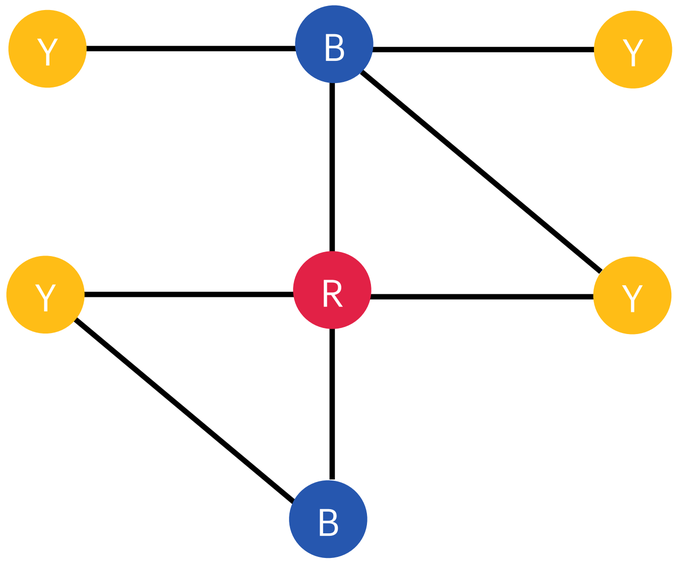


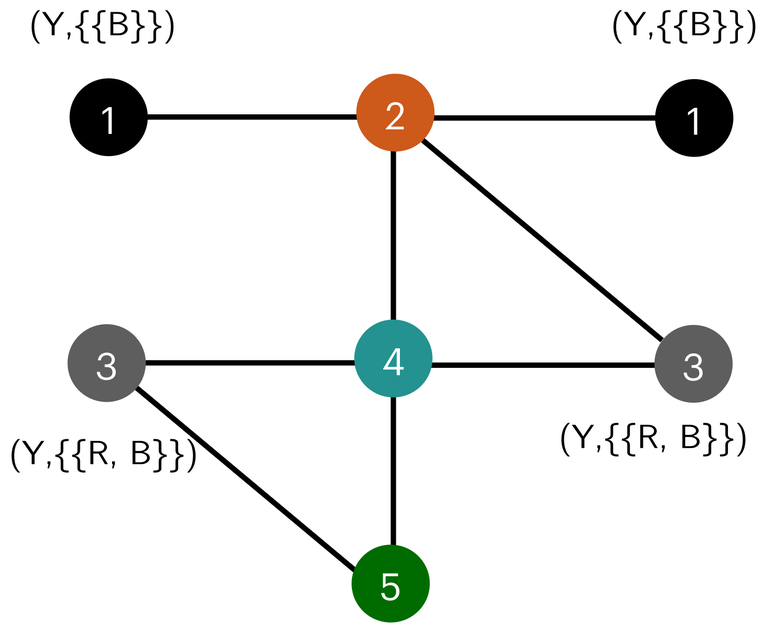
Color Refinement: Example



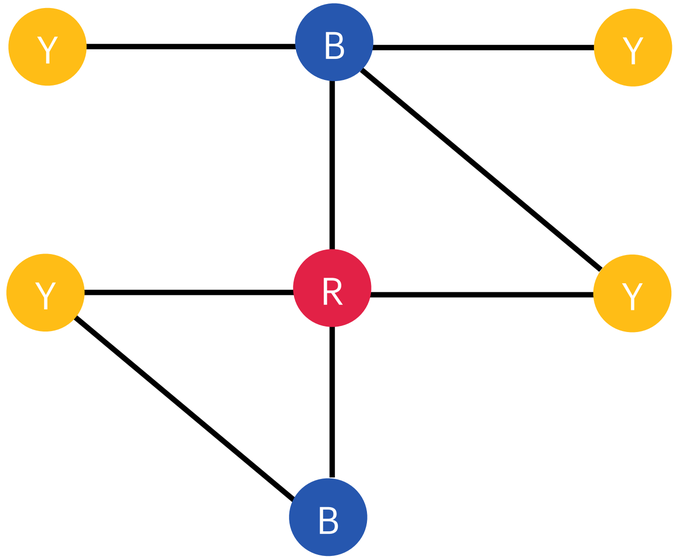


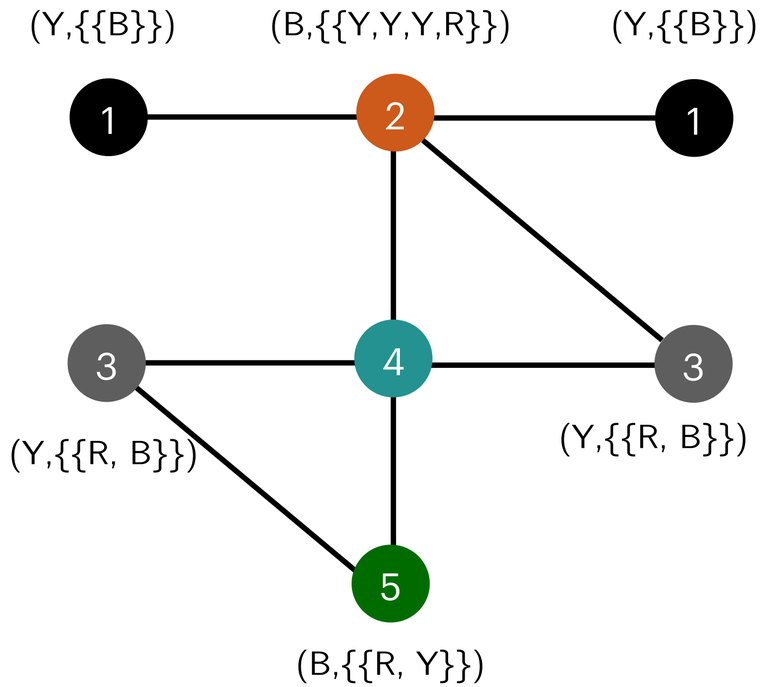
Color Refinement: Example



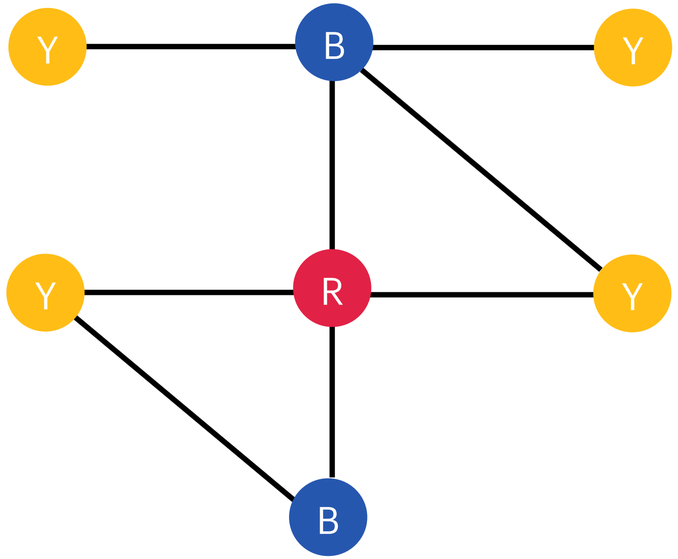


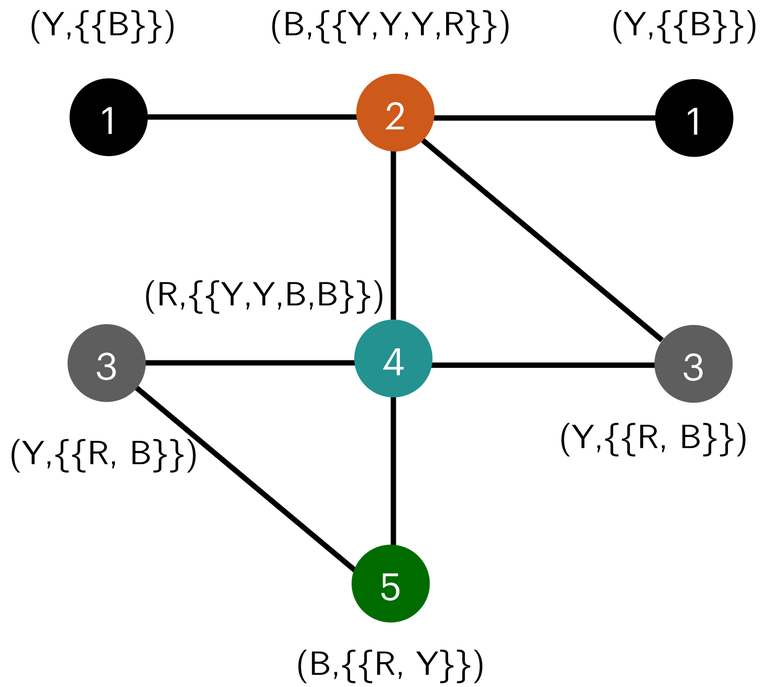
Color Refinement: Example



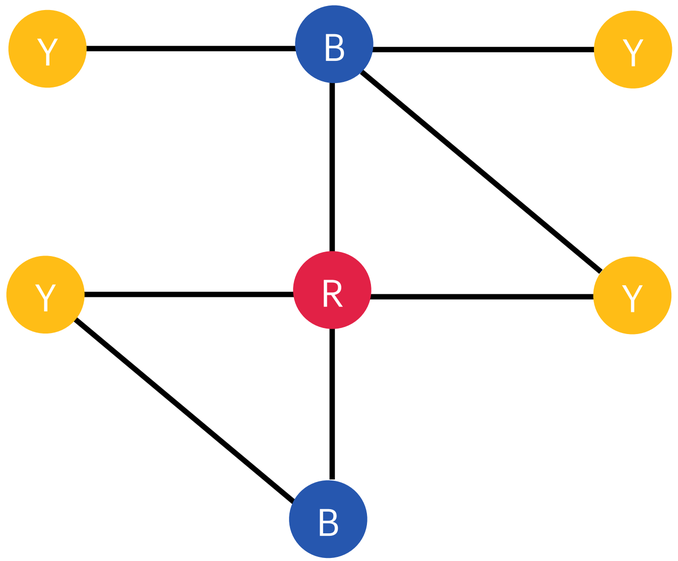


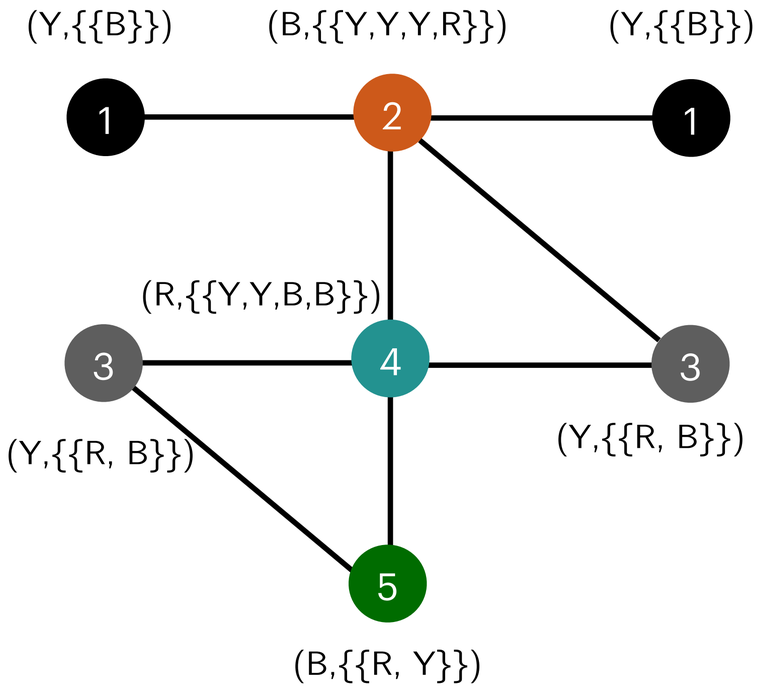
Color Refinement: Example

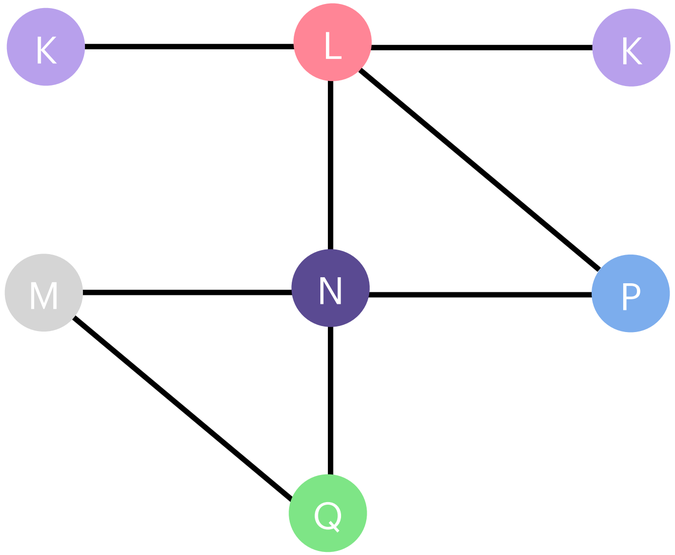




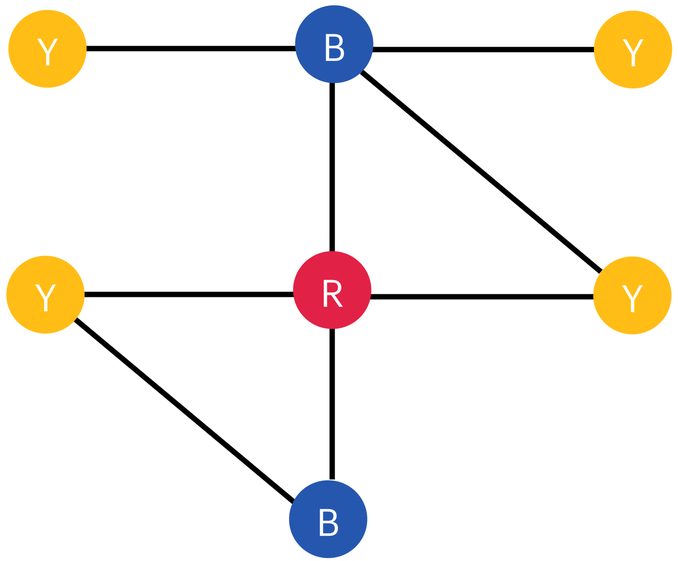
Color Refinement: Example

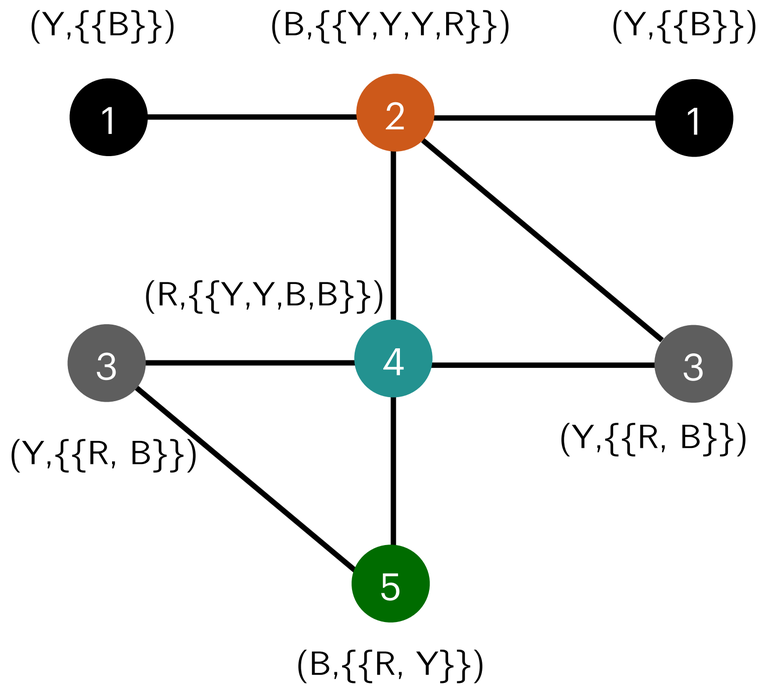


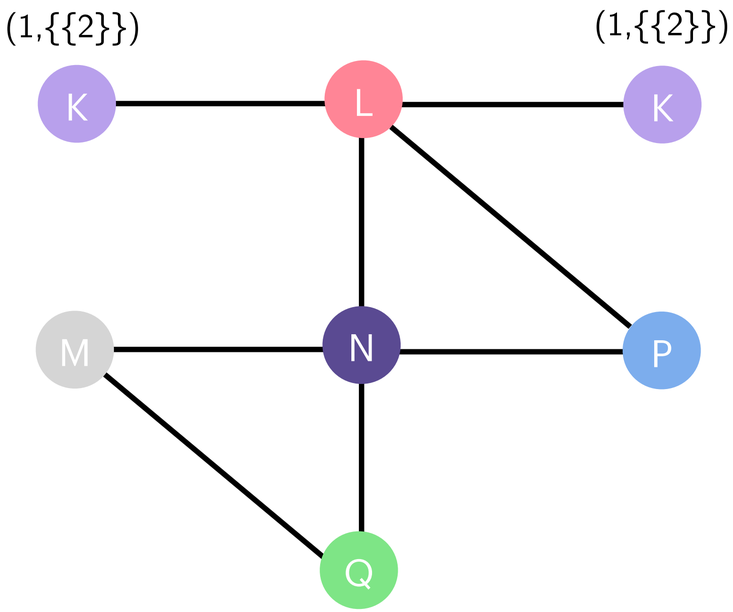




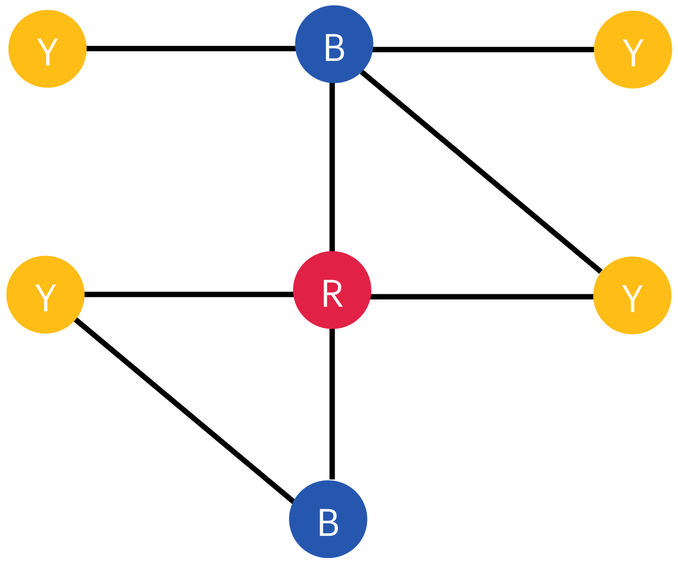
Color Refinement: Example

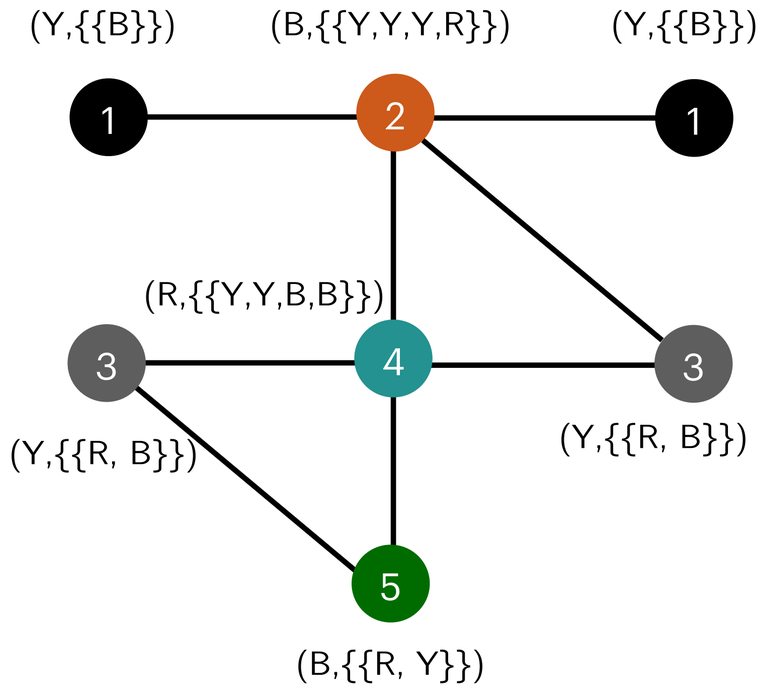


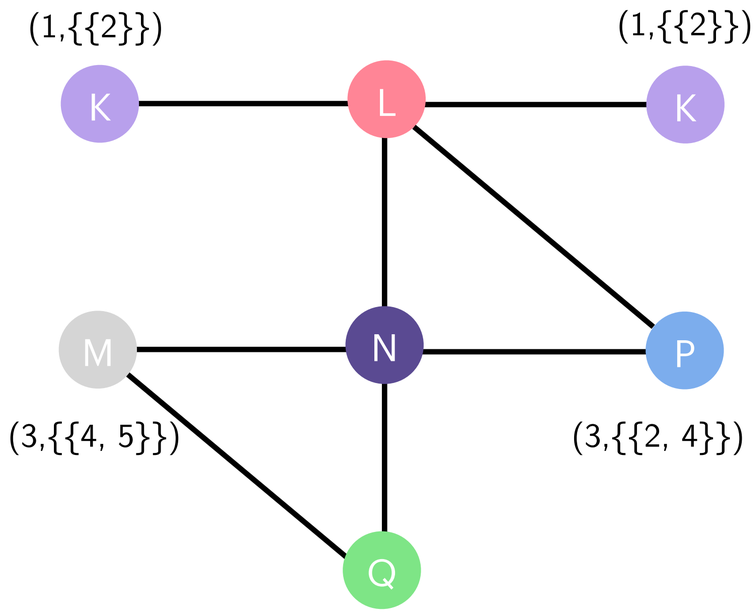




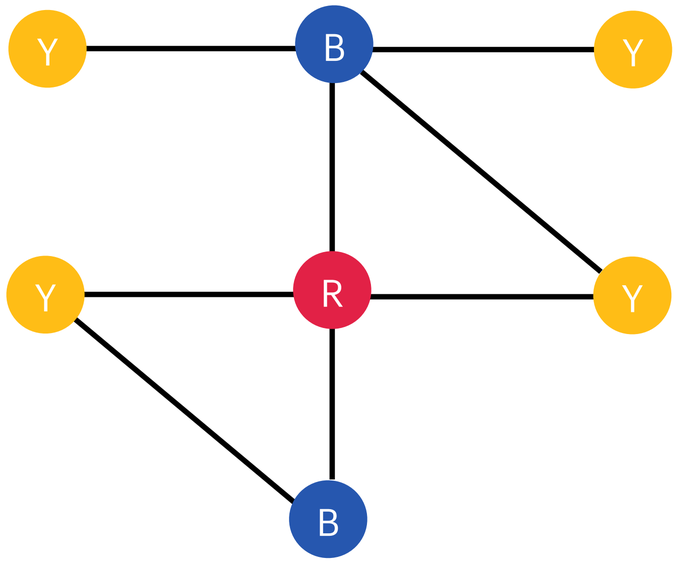
Color Refinement: Example

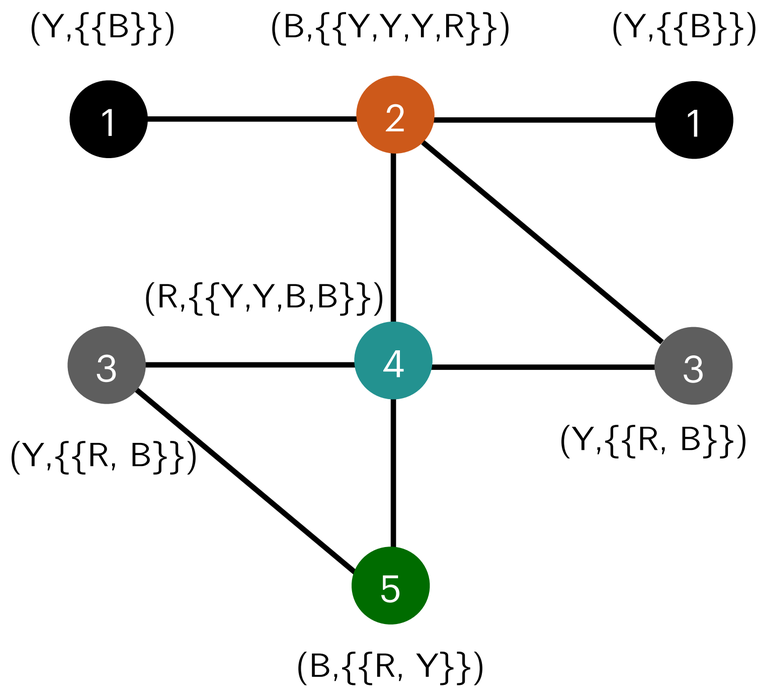


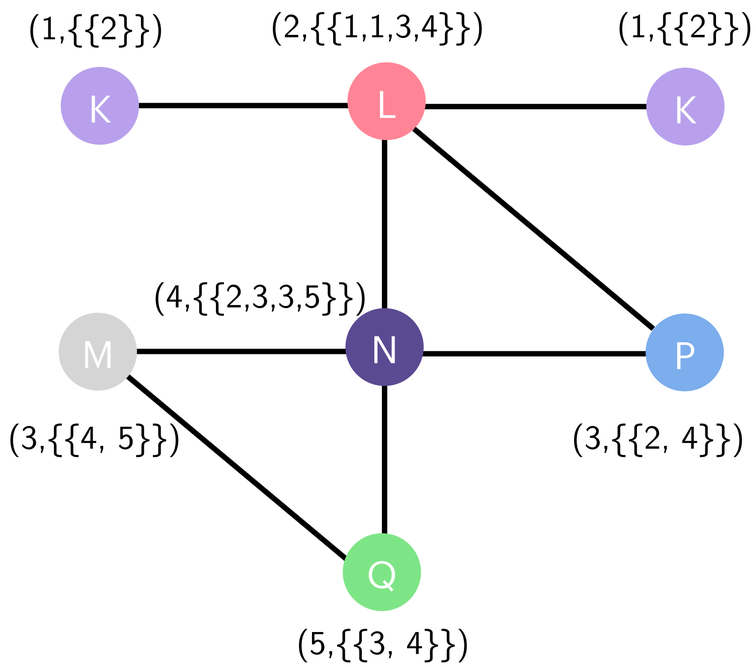




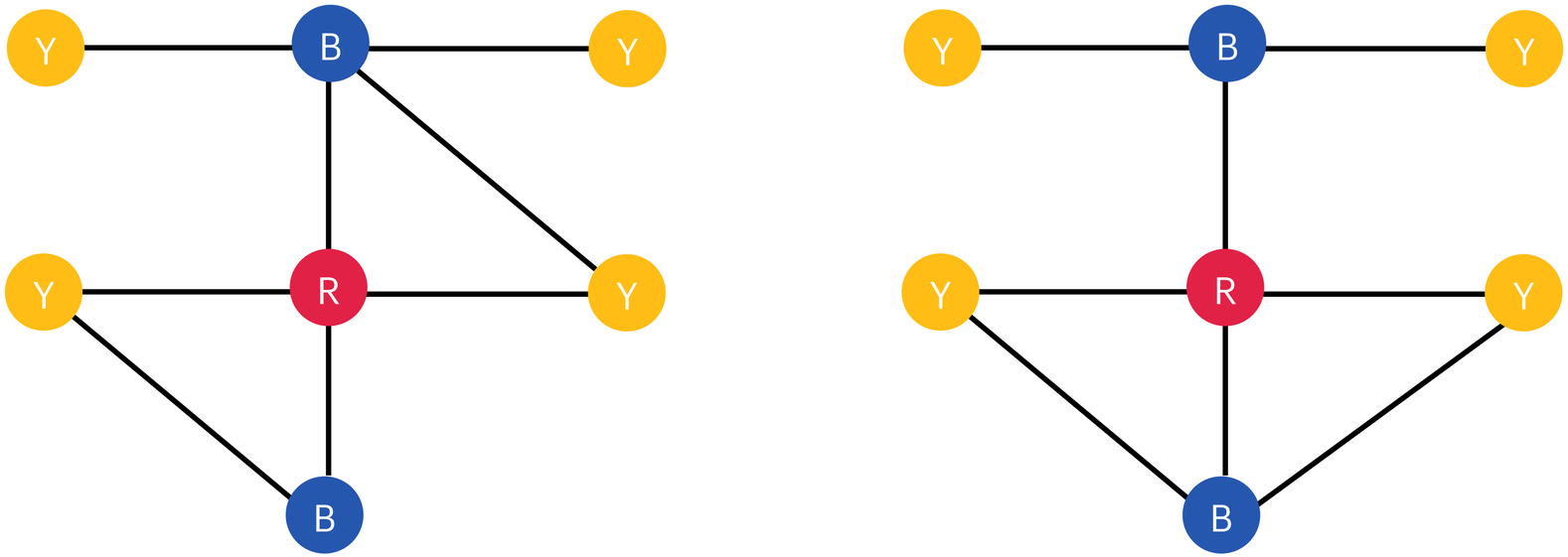
Color Refinement: Example





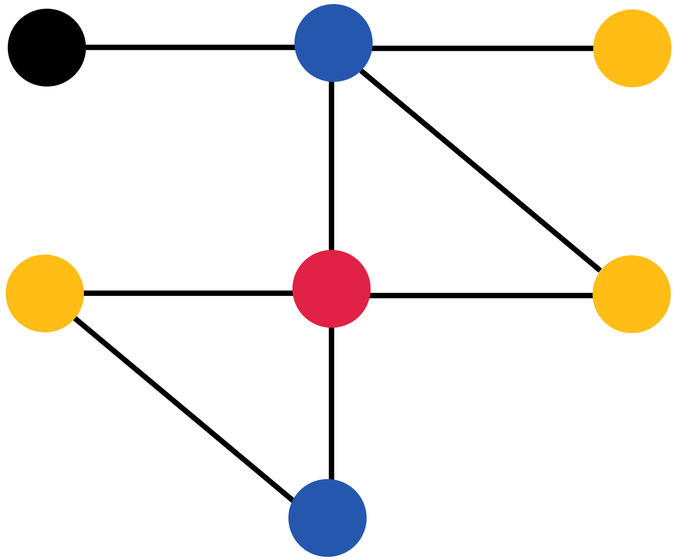


Color Refinement: Example



Two graphs: Node color classes differ for these graphs - color refinement can distinguish...

Color Refinement

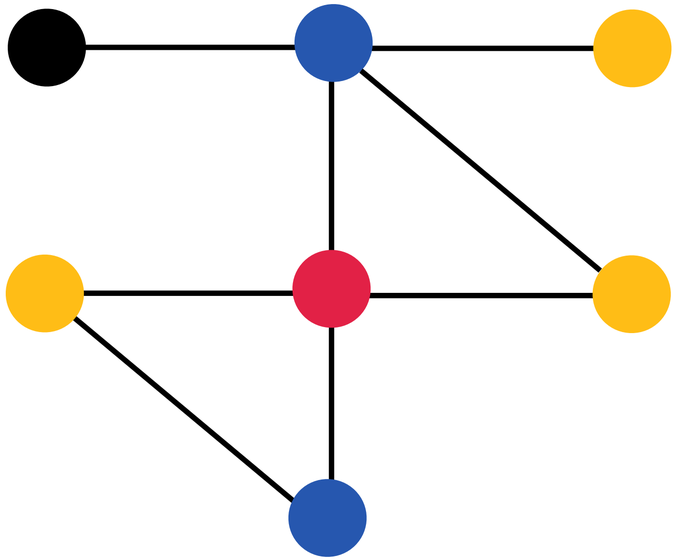


Given a graph $G = (V, E)$, and a set C of colors, we define a coloring function over the nodes of the graph:

$$\lambda : V_G \mapsto C$$

Each such λ colors the nodes of the graph and hence induces a partition of V_G into node color classes.

Color Refinement

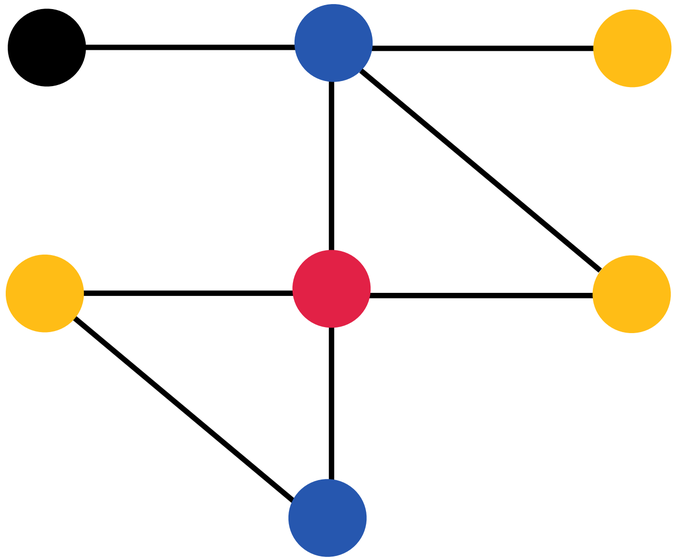


Refinement: A coloring λ refines a coloring λ', denoted as $\lambda \preceq \lambda'$, if for any $u, v \in V_G$ the following holds:

$$\lambda(u) = \lambda(v) \text{ implies } \lambda'(u) = \lambda'(v)$$

Equivalence: A coloring λ is equivalent to a coloring λ', denoted as $\lambda \equiv \lambda'$, if and only if $\lambda \preceq \lambda'$ and $\lambda' \preceq \lambda$.

Color Refinement



We respect the following notation:

- We can apply this function to different graphs, and therefore we will write $\lambda(G)(u)$ instead of $\lambda(u)$.
- We also need to refer to different coloring functions (at different iterations), which will be denoted by $\lambda^{(t)}(G)(u)$.

Color Refinement

Input: A graph $G = (V_G, E_G)$ with an initial coloring $\lambda^{(0)}(G) : V_G \to C$.

1. **Initialization**: All nodes $u \in V_G$ are initialized to their initial colors $\lambda^{(0)}(G)(u)$.

2. **Refinement**: All nodes $u \in V_G$ are recursively re-colored:

$$\lambda^{(i+1)}(G)(u) = \tau\Big(\lambda^{(i)}(G)(u), \{\{\lambda^{(i)}(G)(v) \mid v \in N(u)\}\}\Big),$$

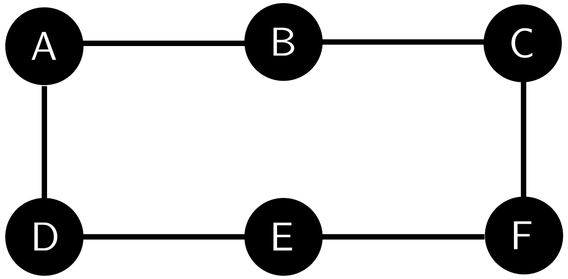
   where double-braces denote a multiset, and τ bijectively maps any pair (composed of a color and a multiset of colors) to a unique color.

3. **Stop**: The algorithm terminates at iteration j, where j is the minimal integer satisfying:

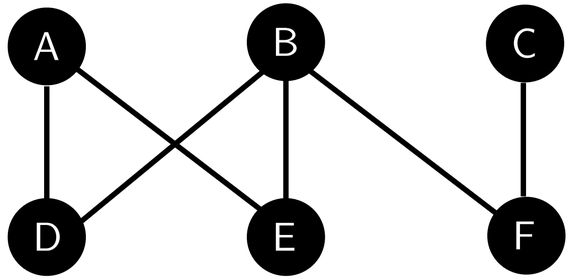
$$\forall u, v \in V_G : \lambda^{(j+1)}(G)(u) = \lambda^{(j+1)}(G)(v) \text{ if and only if } \lambda^{(j)}(G)(u) = \lambda^{(j)}(G)(v).$$

Color Refinement: Graph-Level

G



H



To apply the color refinement algorithm for isomorphism testing, we need graph-level colors:

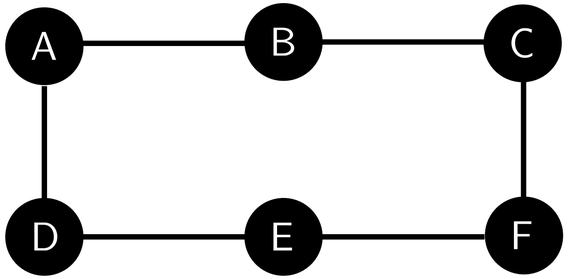
$$\lambda^{(t)}(G) = \tau\Big(\{\{\lambda^{(t)}(G)(u) \mid u \in V_G\}\}\Big)$$

Colour refinement can then be used to distinguish graphs. In particular, we can state the following:

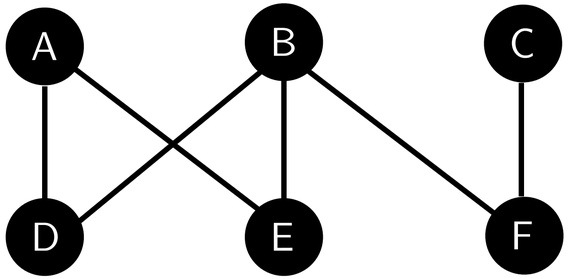
G and H are non-isomorphic iff $\lambda^{(t)}(G) \neq \lambda^{(t')}(H)$ for stable colorings $\lambda^{(t)}$ and $\lambda^{(t')}$.

Color Refinement

G



H



Soundness: Color refinement is sound for non-isomorphism checking: whenever it returns yes for two graphs G and H, they are non-isomorphic.

Incompleteness: Colour refinement is incomplete for non-isomorphism checking: even if G and H agree in every color class size in the stable coloring, the graphs might not be isomorphic.

1-WL Algorithm for Graph Isomorphism Testing

1-dimensional Weisfeiler Lehman algorithm (1-WL): A popular algorithm for graph isomorphism testing.

1-WL is very similar to color refinement, where the refinement considers both neighbors and non-neighbors:

$$\mathrm{wl}_1^{(i+1)}(G)(u) = \tau\Big(\mathrm{wl}_1^{(i)}(G)(u), \{\{\mathrm{wl}_1^{(i)}(G)(v) \mid v \in N(u)\}\}, \{\{\mathrm{wl}_1^{(i)}(G)(v) \mid v \in V_G \backslash N(u)\}\}\Big)$$

Remark: 1-WL and color refinement coincide on the graph-level:

$$\mathrm{wl}_1^{(t)}(G_1) \neq \mathrm{wl}_1^{(t)}(G_2) \text{ and } \lambda^{(t)}(G_1) \neq \lambda^{(t)}(G_2)$$

1-WL Algorithm for Graph Isomorphism Testing

1-dimensional Weisfeiler Lehman algorithm (1-WL): A popular algorithm for graph isomorphism testing.

1-WL is very similar to color refinement, where the refinement considers both neighbors and non-neighbors:

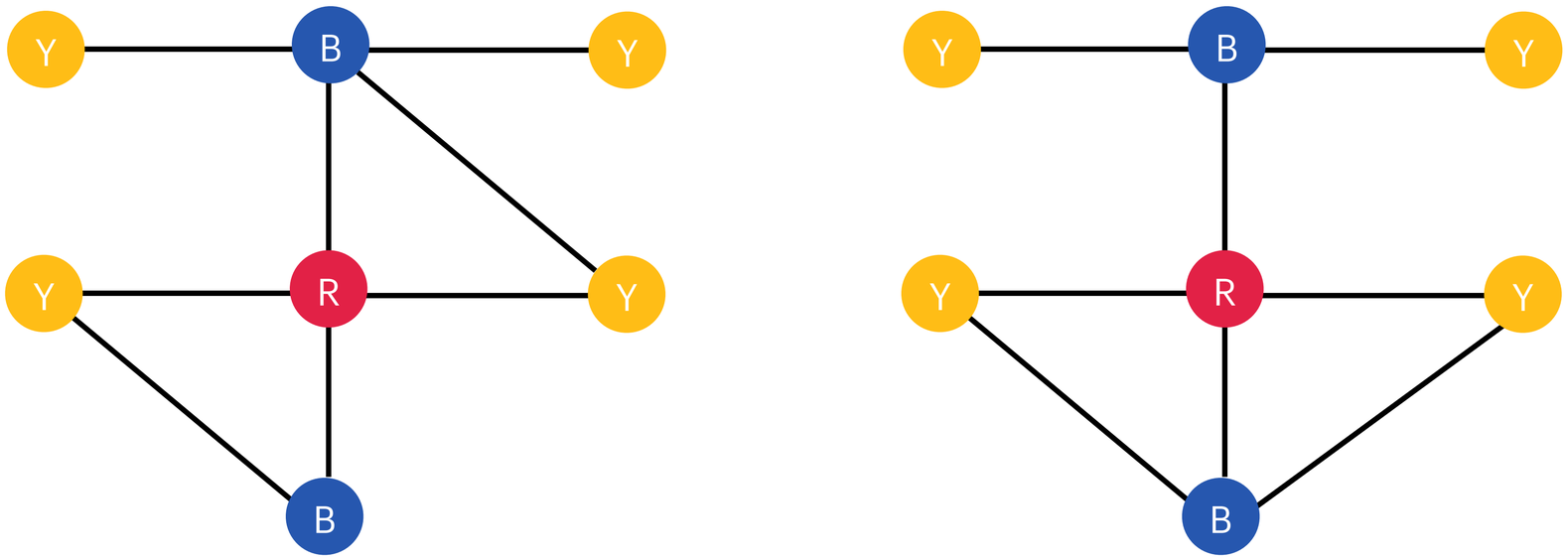
$$\mathrm{wl}_1^{(i+1)}(G)(u) = \tau\Big(\mathrm{wl}_1^{(i)}(G)(u), \{\{\mathrm{wl}_1^{(i)}(G)(v) \mid v \in N(u)\}\}, \{\{\mathrm{wl}_1^{(i)}(G)(v) \mid v \in V_G \backslash N(u)\}\}\Big)$$

Remark: They are different when we look at node-level refinements on different graphs:

$$\mathrm{wl}_1^{(t)}(G_1)(u) \neq \mathrm{wl}_1^{(t)}(G_2)(v) \text{ while } \lambda^{(t)}(G_1)(u) = \lambda^{(t)}(G_2)(v)$$

Expressive Power of MPNNs

Color Refinement: Example



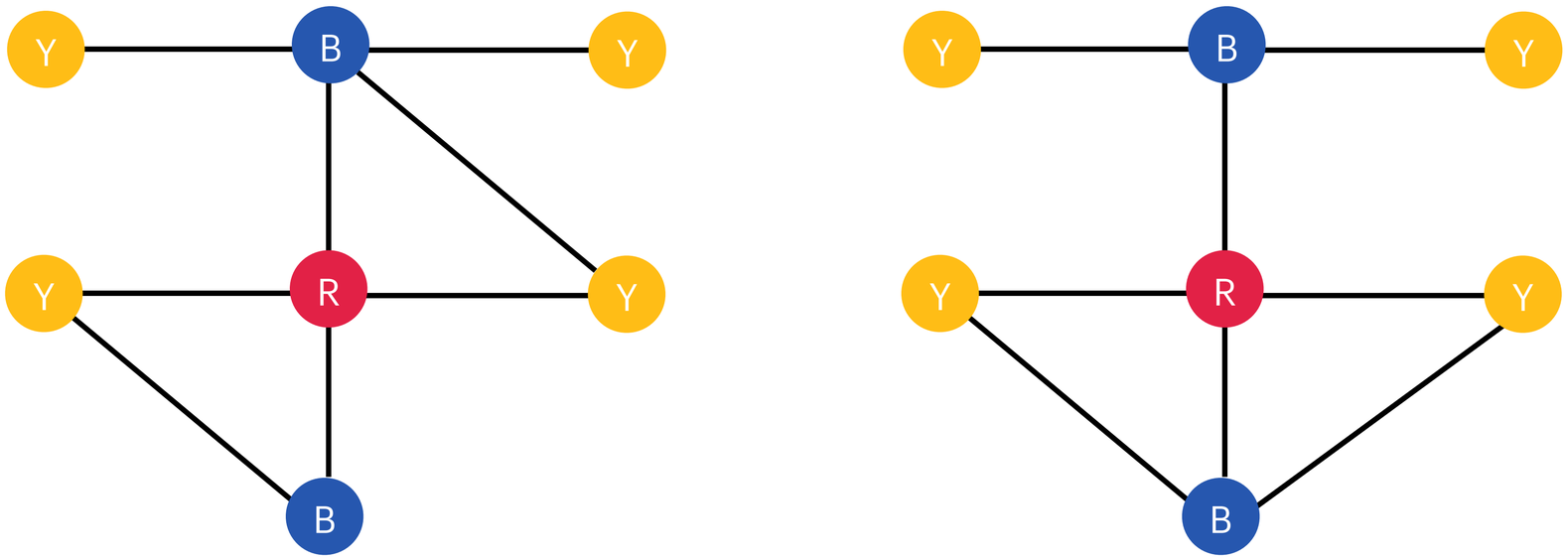
Color refinement and MPNNs aggregate information from the neighborhoods and update accordingly:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right)\right)$$

MPNN layers are feature maps over graphs: $\forall G$ and $\forall t, 1 \leq t \leq L$, we have the mapping $\mathbf{h}^{(t)}(G) : V \to \mathbb{R}^d$

Taking this perspective, we can view $\mathbf{h}_u^{(t)}$ as an abbreviation of $\mathbf{h}^{(t)}(G)(u)$.

Color Refinement: Example

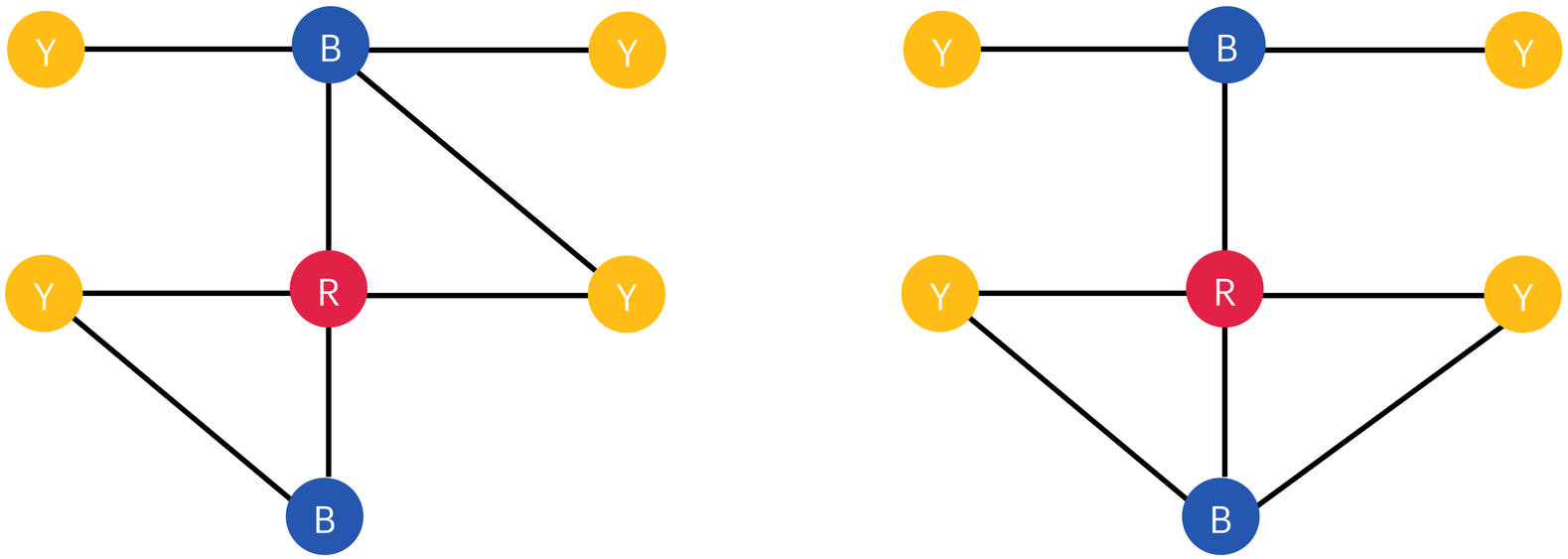


Color refinement and MPNNs aggregate information from the neighborhoods and update accordingly:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right)\right)$$

$$\lambda^{(i+1)}(G)(u) = \tau\left(\lambda^{(i)}(G)(u), \{\{\lambda^{(i)}(G)(v) \mid v \in N(u)\}\}\right)$$

Color Refinement: Example

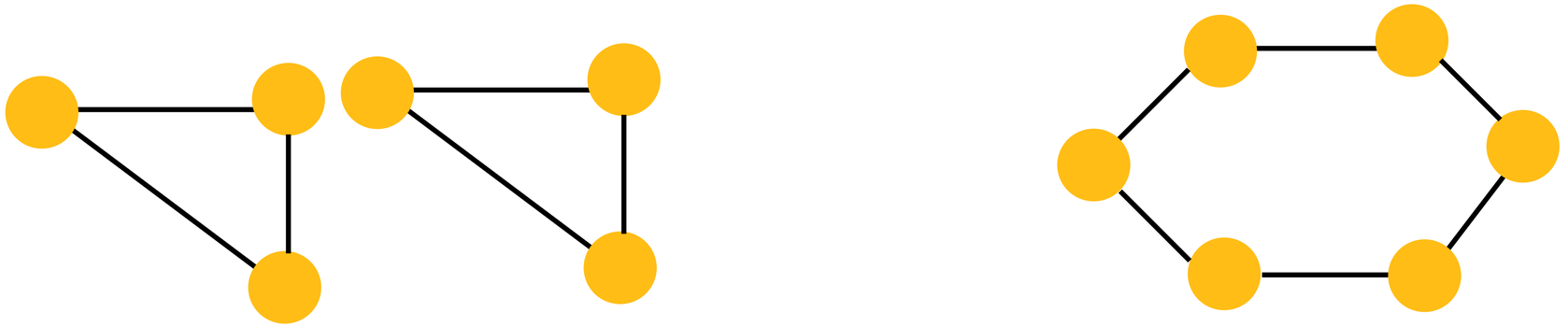


Can we view the rounds of the color refinement algorithm as the layers of an MPNN?

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right)\right)$$

$$\lambda^{(i+1)}(G)(u) = \tau\left(\lambda^{(i)}(G)(u), \{\{\lambda^{(i)}(G)(v) \mid v \in N(u)\}\}\right)$$

An Upper Bound for Expressiveness of MPNNs

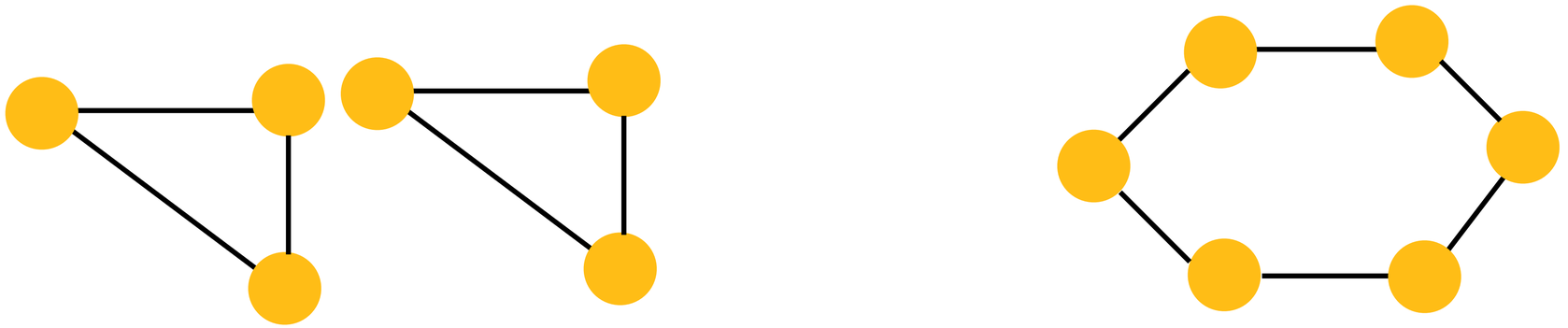


Theorem ([Morris et al., 2019, Xu et al., 2019]). Consider any MPNN that consists of k message-passing layers:

$$\mathbf{h}_u^{(t)} = \phi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \psi^{(t)}\left(\mathbf{h}_u^{(t-1)}, \left\{\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right\}\right)\right)$$

Given a graph $G = (V, E, \mathbf{X})$ with only discrete input features $\mathbf{h}_u^{(0)} = \mathbf{x_u} \in \mathbb{Z}^d$, we have that $\mathbf{h}_u^{(k)} \neq \mathbf{h}_v^{(k)}$ only if the nodes u and v in G have different labels after k iterations of the 1-WL algorithm.

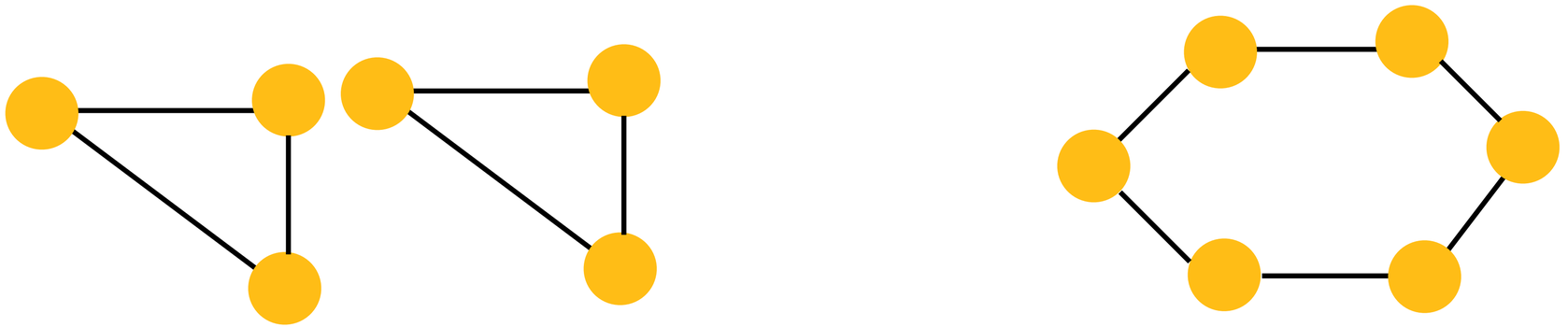
An Upper Bound for Expressiveness of MPNNs



MPNNs are at most as powerful as the 1-WL test:

- If the 1-WL algorithm assigns the same label to two nodes, then any MPNN will also assign the same embedding to these two nodes.
- If the 1-WL test cannot distinguish between two graphs, then an MPNN is also incapable of distinguishing between these two graphs.

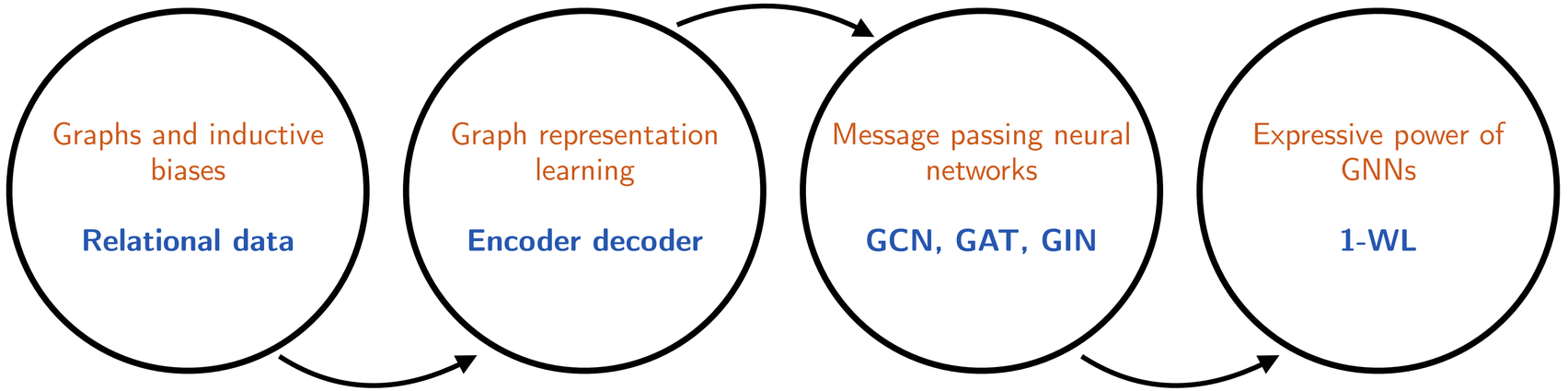
A Lower Bound for Expressiveness of MPNNs



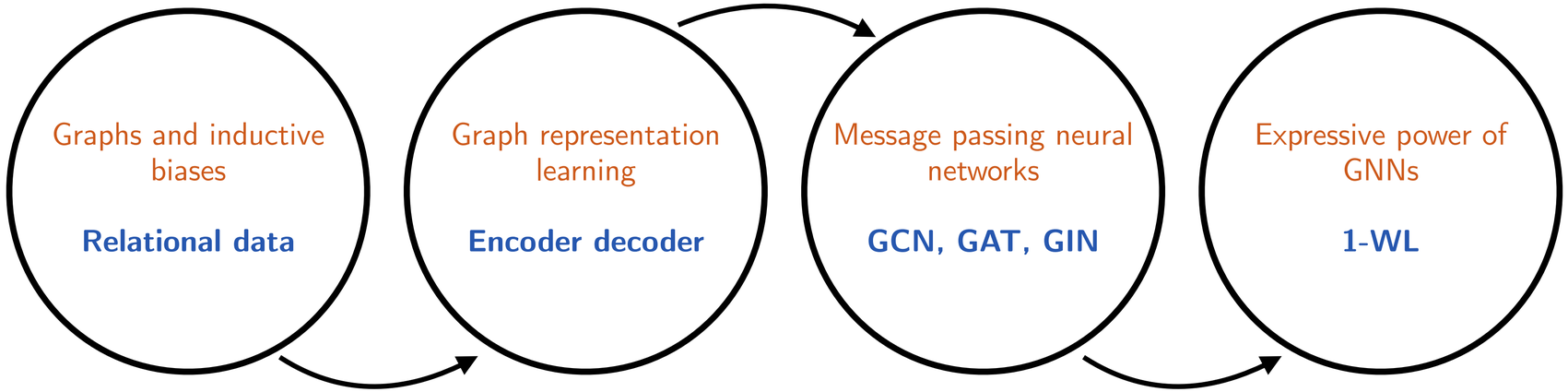
Theorem ([Morris et al., 2019, Xu et al., 2019]). Given a graph $G = (V, E, \mathbf{X})$, there exists an MPNN such that $\mathbf{h}_u^{(k)} = \mathbf{h}_v^{(k)}$ if and only if the two nodes u and v in G have the same label after k iterations of the 1-WL algorithm. In particular, the basic MPNN model is as powerful as 1-WL:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

Summary and Outlook



Summary and Outlook



A Journey into Graph Representation Learning

- Inductive learning via MPNNs.
- Expressiveness limitations are at the origin of many other problems.
- Expressiveness studies: uniformity conditions are necessary.
- Other limitations: related to information bottlenecks.

References

- Ganapathiraju, M. K., Thahir, M., Handen, A., Sarkar, S. N., Sweet, R. A., Nimgaonkar, V. L., Loscher, C. E., Bauer, E. M., & Chaparala, S. (2016). Schizophrenia interactome with 504 novel protein-protein interactions. *NPJ schizophrenia*, *2*, 16012.
- Rao, P.P., Kabir, S.N., & Mohamed, T.S. (2010). Nonsteroidal Anti-Inflammatory Drugs (NSAIDs): Progress in Small Molecule Drug Development. *Pharmaceuticals, 3*, 1530 - 1549.
- J. Johnson, R. Krishna, M. Stark, L. Li, D. A. Shamma, M. S. Bernstein, L. Fei-Fei. Image Retrieval using Scene Graphs, *CVPR*, 2015.
- Allamanis, Miltiadis, Graph Neural Networks on Program Analysis. 2021.
- Ganapathiraju, M. K., Thahir, M., Handen, A., Sarkar, S. N., Sweet, R. A., Nimgaonkar, V. L., Loscher, C. E., Bauer, E. M., & Chaparala, S. (2016). Schizophrenia interactome with 504 novel protein-protein interactions. *NPJ schizophrenia*, *2*, 16012.
- Hewett M, Oliver DE, Rubin DL, et al. Pharmgkb: the pharmacogenetics knowledge base. *Nucleic Acids Res* 2002;30(1):163–5.
- B. Weisfeiler and A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 1968.

References

- J. Gilmer, S.S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. 2017. Neural message passing for Quantum chemistry. *ICML*, 2017.
- R.L.Murphy, B. Srinivasan, V.A. Rao, and B. Ribeiro, Relational Pooling for Graph Representations. *ICML*, 2019.
- Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vin´ıcius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulc¸ehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish ¨ Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. CoRR, abs/1806.01261, 2018.
- P. Barcelo, E. Kostylev, M. Monet, J. Perez, J. Reutter, and J. Silva. The logical expressiveness of graph neural networks. *ICLR*, 2020.
- T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
- M. D. Zeiler, R. Fergus. Visualizing and Understanding Convolutional Networks, *ECCV*, 2014.

References

- Y. Li, D. Tarlow, M. Brockschmidt, and R.S. Zemel. Gated graph sequence neural networks. *ICLR*, 2016.
- P. Velickovic, G. Cucurull, A. Casanova, A. Romero,P. Lio, and Y. Bengio. Graph attention networks. *ICLR* 2018.
- Shaked Brody, Uri Alon, Eran Yahav. How Attentive are Graph Attention Networks? ICLR 2022.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS*, 2017.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.
- W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NIPS*, 2017.
- J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *ICML*, 2017.
- M.Defferrard, X. Bresson, P. Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *NIPS*, 2016.
- J. Bruna, W. Zaremba, A. Szlam, Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *ICLR*, 2014.
- Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, 2018.

References

- K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *IJCNLP*, 2015.
- H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. *ICML*, 2016.
- A. Santoro, D. Raposo, D.G.T.Barrett, M. Malinowski, R. Pascanu, P.W. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *NIPS*, 2017.
- R.L. Murphy, B. Srinivasan, V.A. Rao, and B. Ribeiro. Relational Pooling for Graph Representations. *ICML*, 2019.
- H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *NeurIPS*, 2019.
- C. Morris, M. Ritzert, M. Fey, W. Hamilton,J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. *AAAI*, 2019.
- W. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 2017.
- R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. *SDM*, 2021.
- R. Abboud, İ. İ. Ceylan, M. Grohe, T. Lukasiewicz, The Surprising Power of Graph Neural Networks with Random Node Initialization, *IJCAI*, 2021

References

- M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. *IJCNN*, 2005.
- F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks* 20(1):61–80, 2009.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Uri Alon, Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications, *ICLR*, 2021.
- Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, 2018.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.
- M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van den Berg, I. Titov, M. Welling, Modeling Relational Data with Graph Convolutional Networks, *ESWC*, 2018.
- K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In ICML, 2018.

References

- Komal K. Teru, Etienne G. Denis, and William L. Hamilton. 2020. Inductive relation prediction by subgraph reasoning. ICML.
- Zhu, Zhaocheng and Zhang, Zuobai and Xhonneux, Louis-Pascal and Tang, Jian. Neural bellman-ford networks: A general graph neural network framework for link prediction, NeurIPS 2021.
- Mikhail Galkin, Etienne Denis, Jiapeng Wu, William L. Hamilton. NodePiece: Compositional and Parameter-Efficient Representations of Large Knowledge Graphs, ICLR 2023.
- Xingyue Huang, Miguel Romero Orth, İsmail İlkan Ceylan and Pablo Barcelo. A Theory of Link Prediction via Relational Weisfeiler–Leman on Knowledge Graphs, NeurIPS 2023.
- Vijay Prakash Dwivedi, Xavier Bresson, Graph Transformer, DLG Workshop at AAAI 2021.
- Chengxuan Ying, Tinade Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, Tie-Yan Liu, Do Transformers Really Perform Bad for Graph Representation? NeurIPS 2021.
- Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. NeurIPS 2022.
- H Shirzad, A Velingker, B Venkatachalam, DJ Sutherland, AK Sinop. EXPHORMER: Sparse Transformers for Graphs. ICML 2023.