Lecture 6: Higher-Order Graph Neural Networks

Relational Learning

İsmail İlkan Ceylan Advanced Topics in Machine Learning, University of Oxford 05.02.2021

Overview

Overview

- Motivation

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks
    - Higher-order message passing neural networks: k-GNNs

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks
    - Higher-order message passing neural networks: k-GNNs
    - Invariant/Equivariant graph networks

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs
  - Invariant/Equivariant graph networks
  - Provably powerful graph networks

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs
  - Invariant/Equivariant graph networks
  - Provably powerful graph networks
- Expressive power in real-world data

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs
  - Invariant/Equivariant graph networks
  - Provably powerful graph networks
- Expressive power in real-world data
- Homophily and heterophily: Comparative perspectives

Overview

- Motivation
- The Weisfeiler-Lehman hierarchy
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs
  - Invariant/Equivariant graph networks
  - Provably powerful graph networks
- Expressive power in real-world data
- Homophily and heterophily: Comparative perspectives
- Summary

Motivation

Towards More General Graph Neural Networks

Towards More General Graph Neural Networks

We focused on a specific family of graph neural networks — MPNNs.

Towards More General Graph Neural Networks

We focused on a specific family of graph neural networks — MPNNs.

Graph neural networks are neural architectures dedicated to learning functions over graph-structured data, and this does not need to be through a specific message passing framework, or even, message passing.

Towards More General Graph Neural Networks

We focused on a specific family of graph neural networks — MPNNs.

Graph neural networks are neural architectures dedicated to learning functions over graph-structured data, and this does not need to be through a specific message passing framework, or even, message passing.

This leads us to a more general definition: In a GNN, nodes in the input graph are assigned vector representations, which are updated iteratively through series of invariant or equivariant computational layers — and their precise form is a design choice.

Towards More General Graph Neural Networks

We focused on a specific family of graph neural networks — MPNNs.

Graph neural networks are neural architectures dedicated to learning functions over graph-structured data, and this does not need to be through a specific message passing framework, or even, message passing.

This leads us to a more general definition: In a GNN, nodes in the input graph are assigned vector representations, which are updated iteratively through series of invariant or equivariant computational layers — and their precise form is a design choice.

We have seen several limitations of MPNNs, including their limitations in expressive power.

Towards More General Graph Neural Networks

We focused on a specific family of graph neural networks — MPNNs.

Graph neural networks are neural architectures dedicated to learning functions over graph-structured data, and this does not need to be through a specific message passing framework, or even, message passing.

This leads us to a more general definition: In a GNN, nodes in the input graph are assigned vector representations, which are updated iteratively through series of invariant or equivariant computational layers — and their precise form is a design choice.

We have seen several limitations of MPNNs, including their limitations in expressive power.

The focus of this lecture is the so-called higher-order graph neural networks, which use higher-order representations of the graphs, e.g., higher-order tensors, to be able to approximate a larger class of functions.

Towards More General Graph Neural Networks

We focused on a specific family of graph neural networks — MPNNs.

Graph neural networks are neural architectures dedicated to learning functions over graph-structured data, and this does not need to be through a specific message passing framework, or even, message passing.

This leads us to a more general definition: In a GNN, nodes in the input graph are assigned vector representations, which are updated iteratively through series of invariant or equivariant computational layers — and their precise form is a design choice.

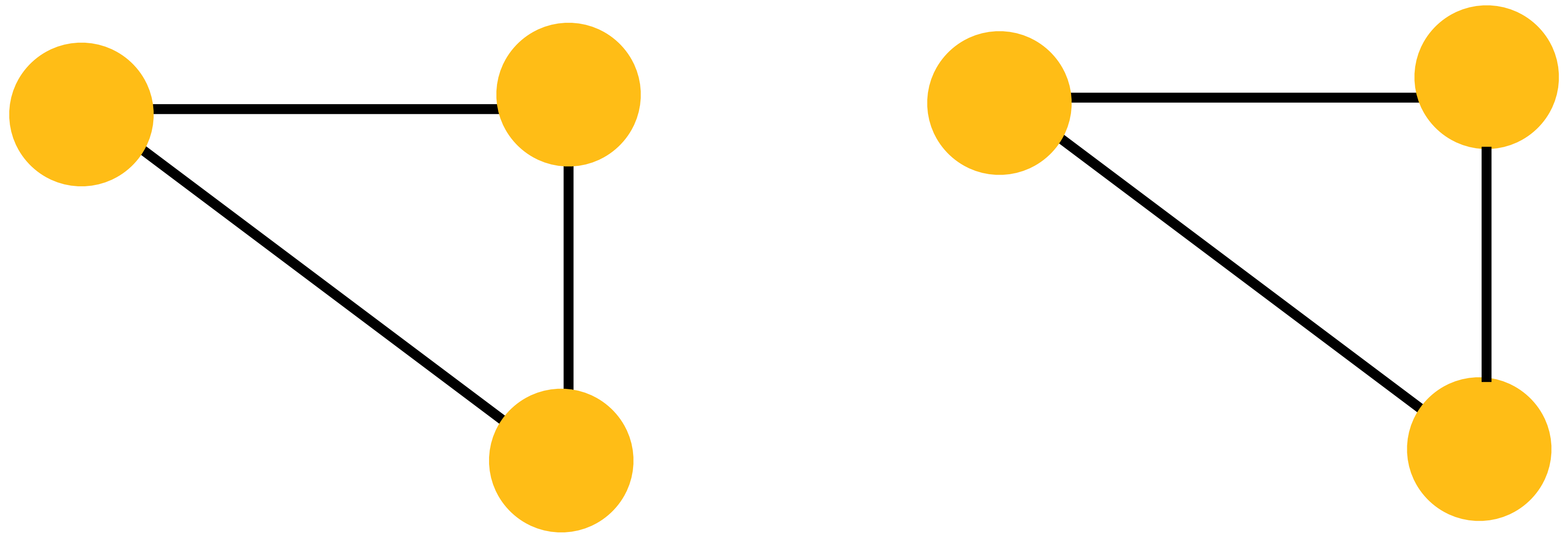
We have seen several limitations of MPNNs, including their limitations in expressive power.

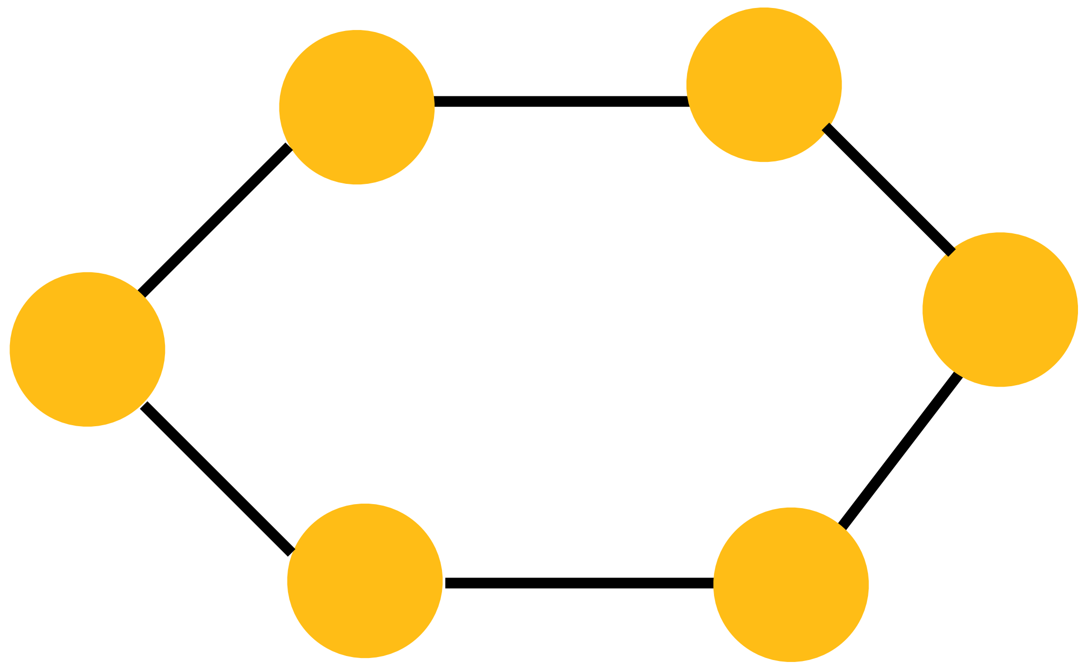
The focus of this lecture is the so-called higher-order graph neural networks, which use higher-order representations of the graphs, e.g., higher-order tensors, to be able to approximate a larger class of functions.

One way of achieving more expressive models is through a richer message passing approach — and this is related to Weisfeiler-Lehman hierarchy.

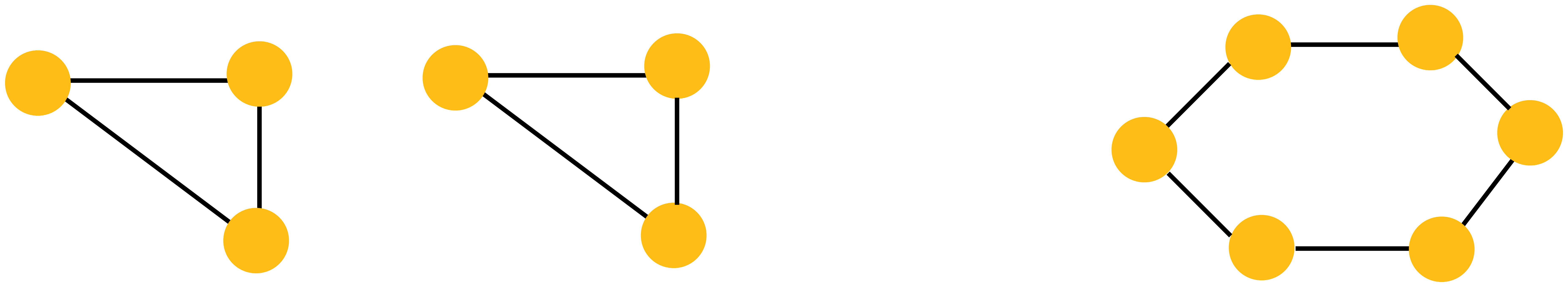
The Weisfeiler-Lehman Hierarchy

A Tale of Two Graphs



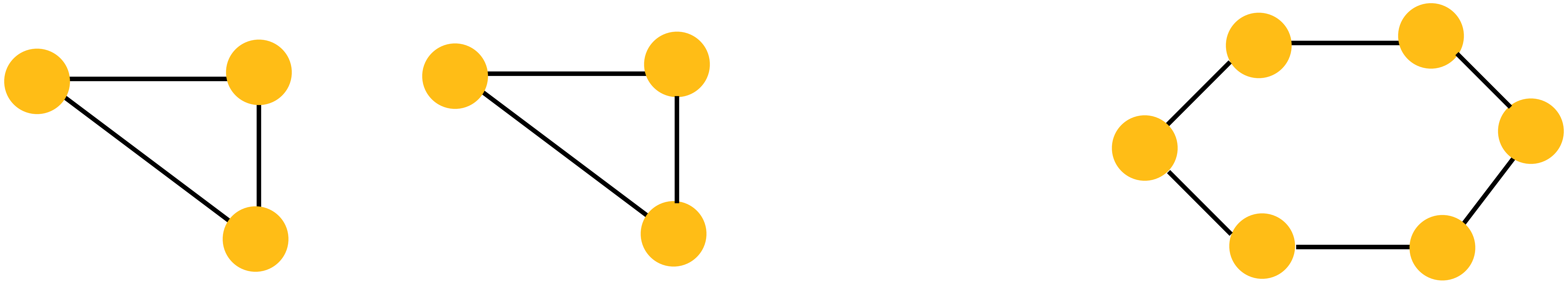


A Tale of Two Graphs



Problem: The embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side for MPNNs!

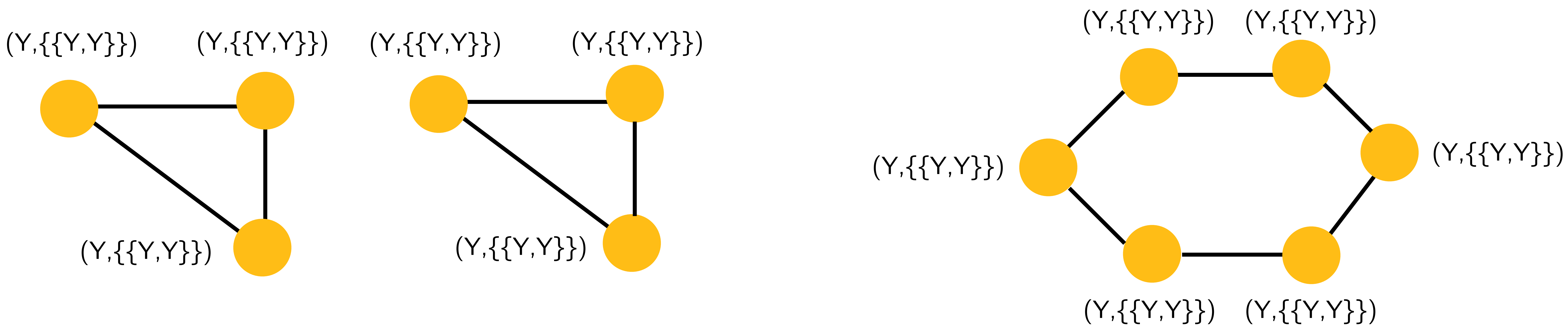
A Tale of Two Graphs



Problem: The embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side for MPNNs!

1-WL cannot distinguish the nodes in the respective graphs — and so neither can MPNNs.

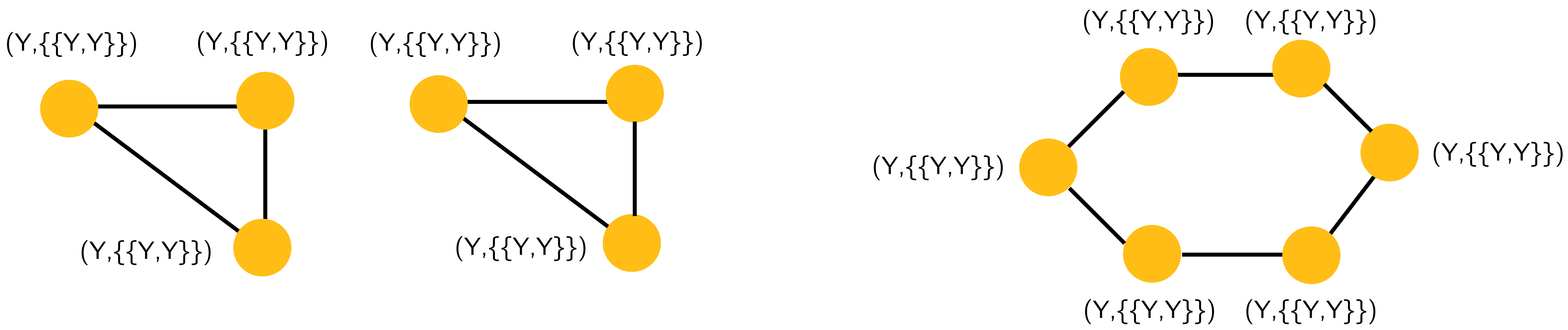
A Tale of Two Graphs



Problem: The embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side for MPNNs!

1-WL cannot distinguish the nodes in the respective graphs — and so neither can MPNNs.

A Tale of Two Graphs



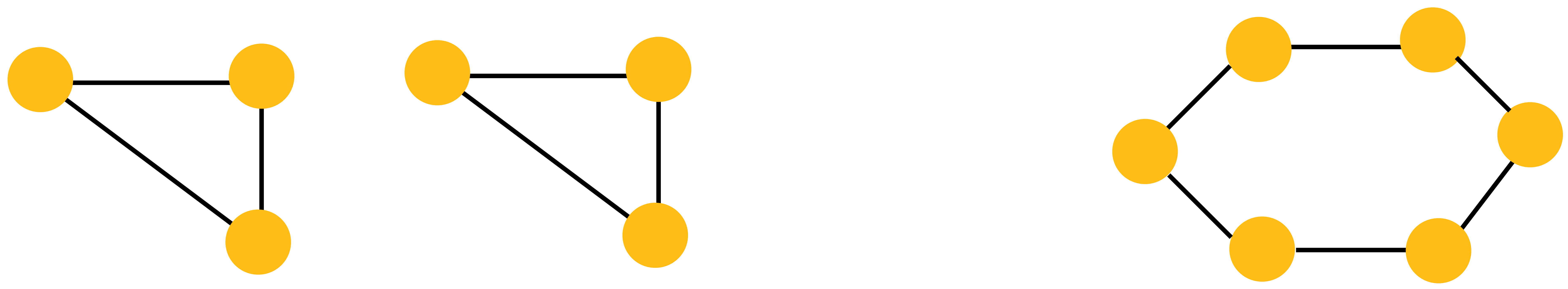
Problem: The embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side for MPNNs!

1-WL cannot distinguish the nodes in the respective graphs — and so neither can MPNNs.

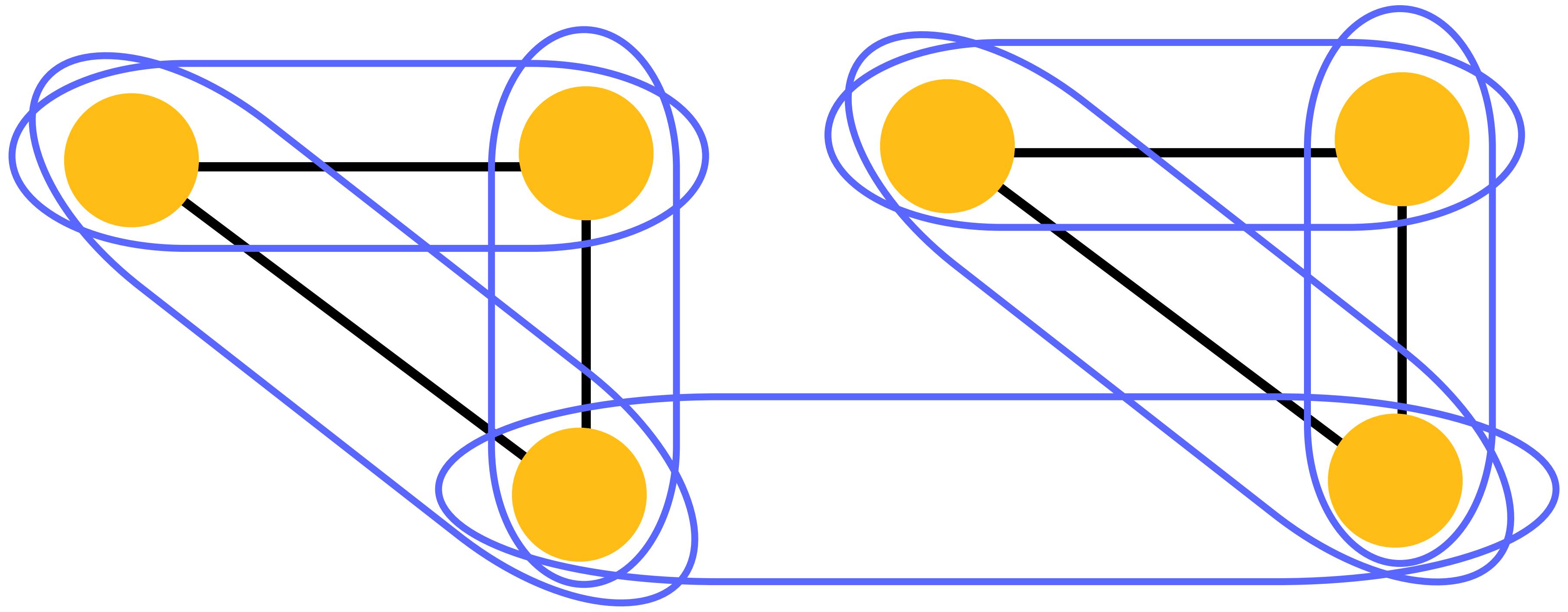
What if we extend the 1-WL algorithm to consider, e.g., pairs of nodes when colouring?

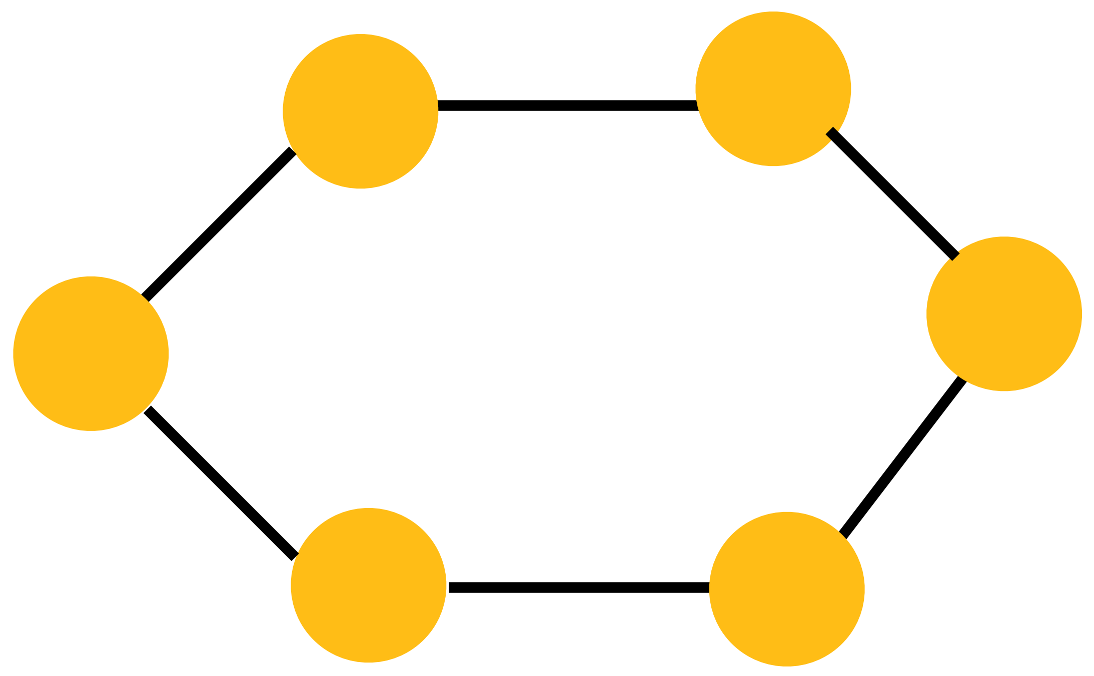
A Tale of Two Graphs

A Tale of Two Graphs

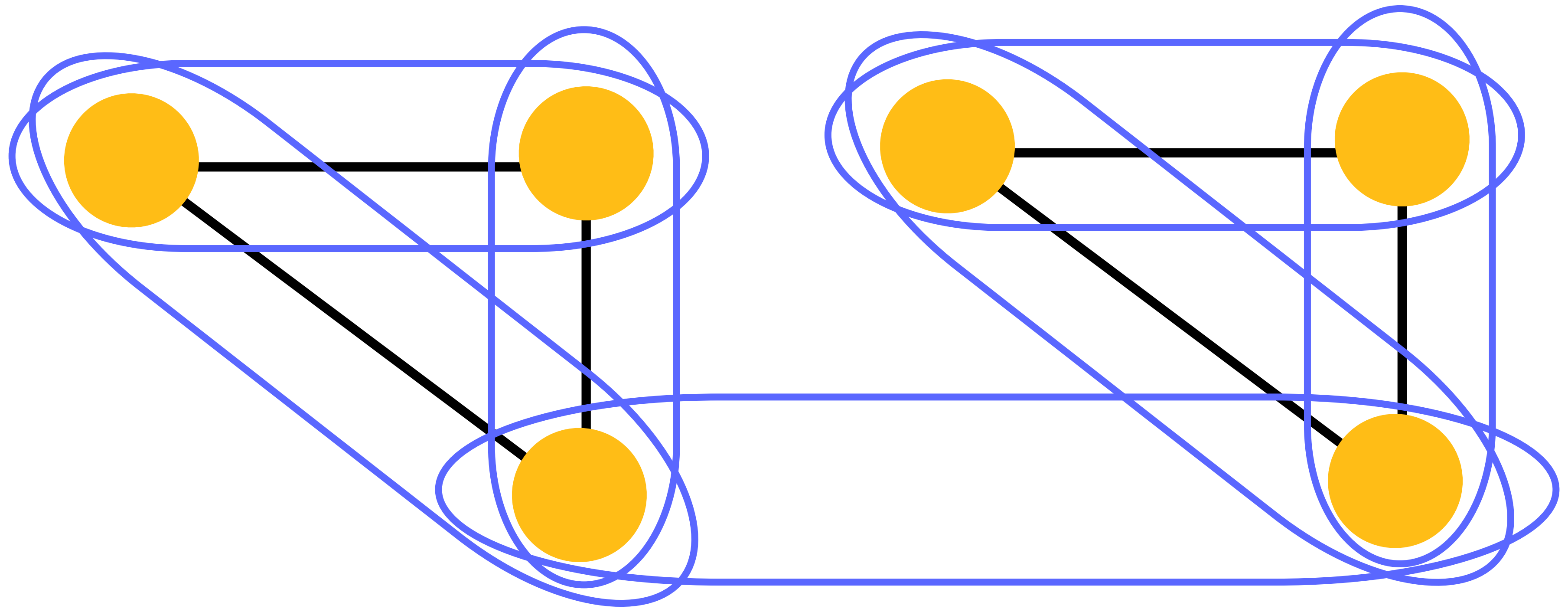


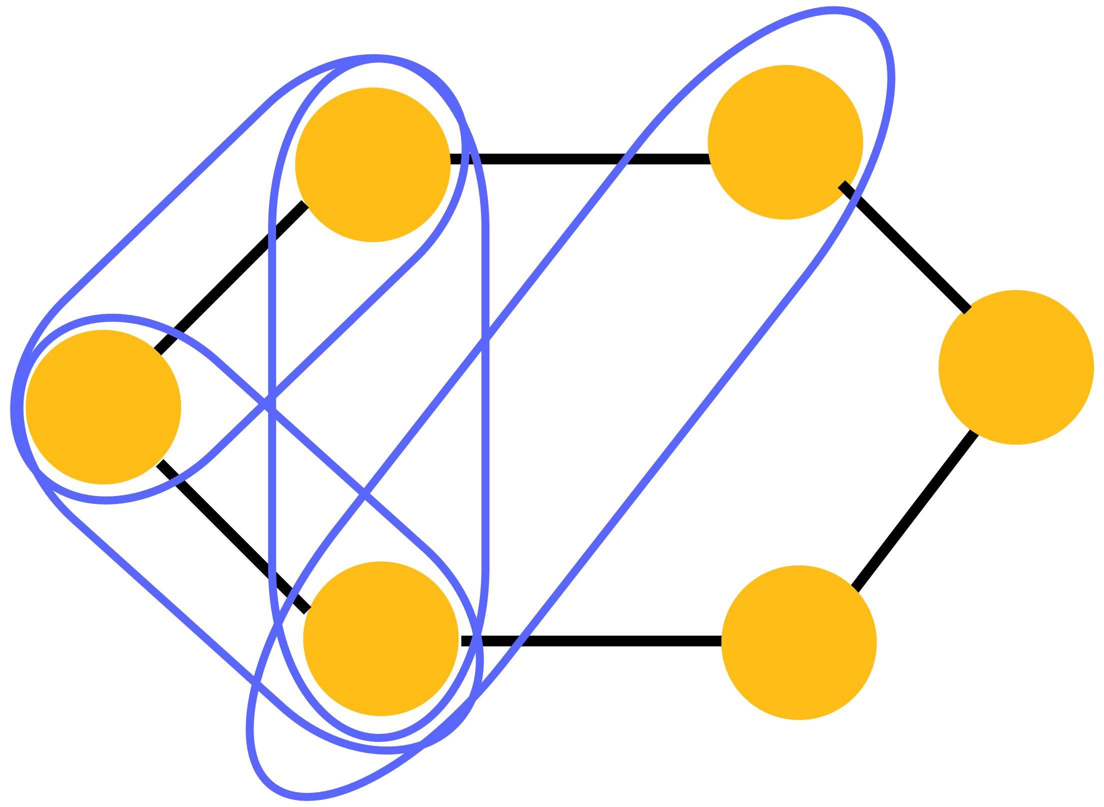
A Tale of Two Graphs



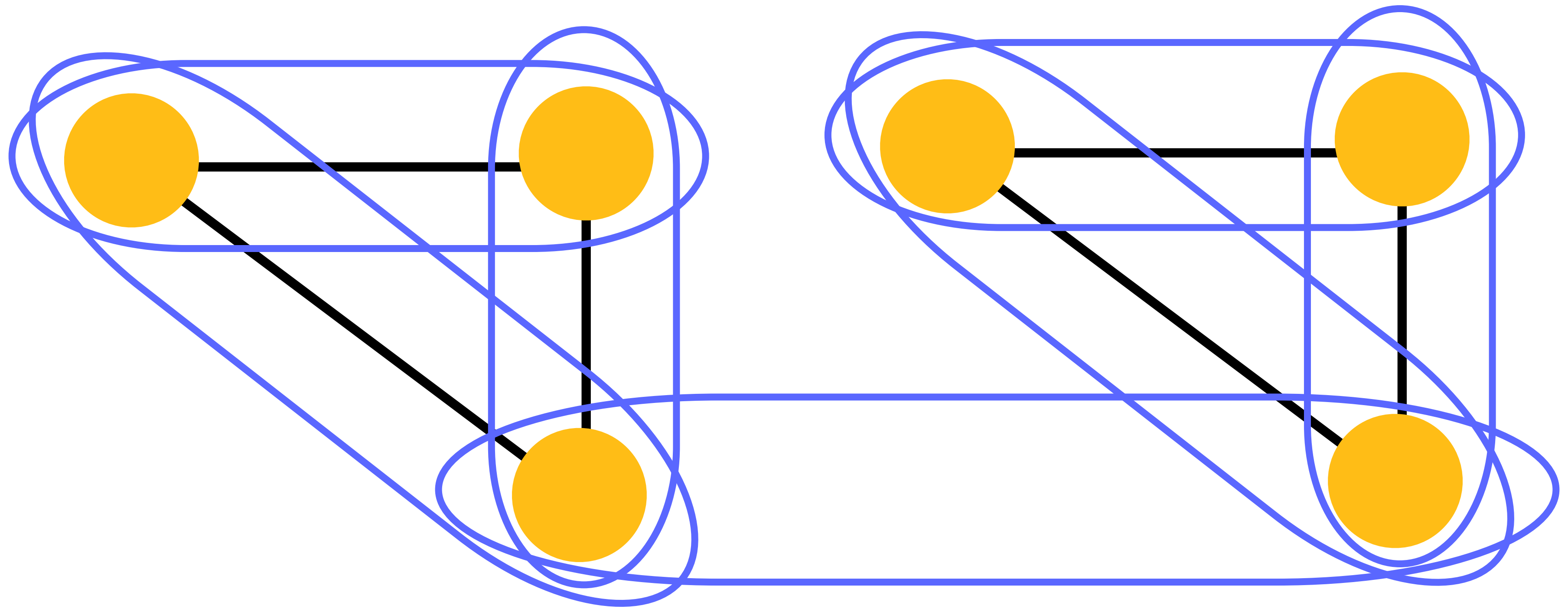


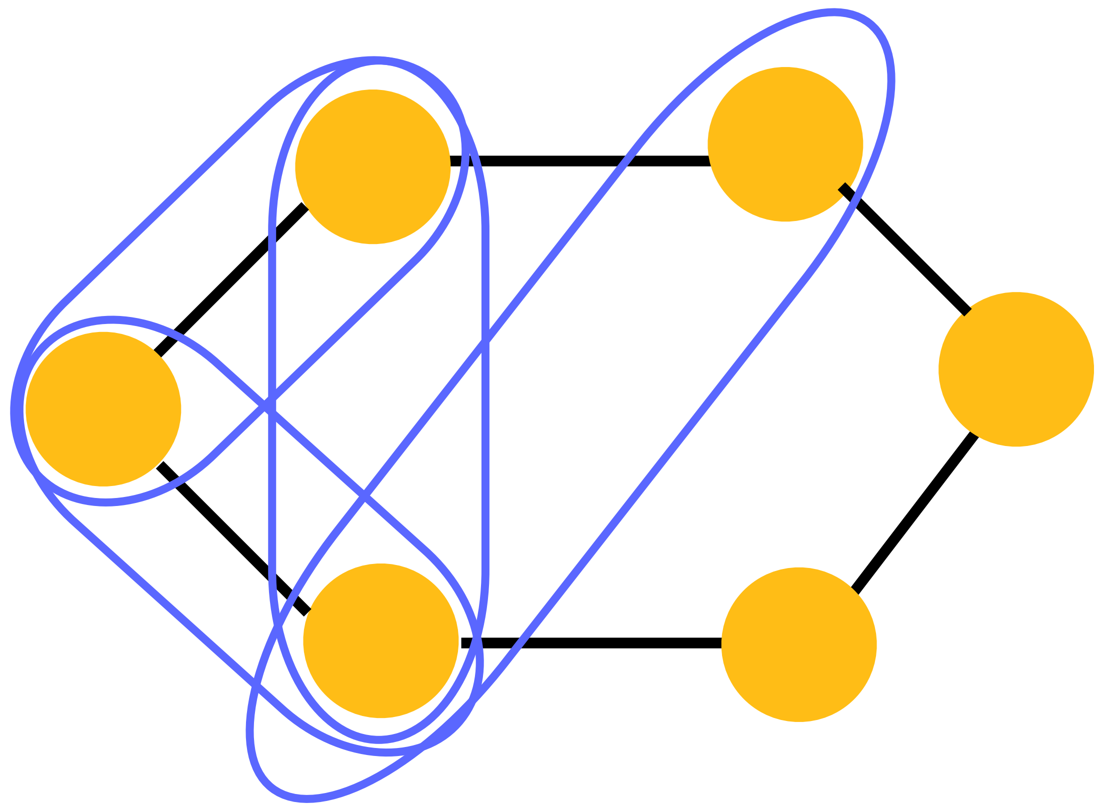
A Tale of Two Graphs





A Tale of Two Graphs





This extended algorithm is called the 2-dimensional WL algorithm, and it can distinguish these two graphs!

k-dimensional Weisfeiler-Lehman

k-dimensional Weisfeiler-Lehman

Given a graph $G = (V, E)$, the k-dimensional WL algorithm generalises colour refinement as follows:

k-dimensional Weisfeiler-Lehman

Given a graph $G = (V, E)$, the k-dimensional WL algorithm generalises colour refinement as follows:

- We consider k-tuples $(v_1, \ldots, v_k) \in V_G^k$ of nodes, where $k \in \mathbb{N}$ is the WL-dimension.

k-dimensional Weisfeiler-Lehman

Given a graph $G = (V, E)$, the k-dimensional WL algorithm generalises colour refinement as follows:

- We consider k-tuples $(v_1, \ldots, v_k) \in V_G^k$ of nodes, where $k \in \mathbb{N}$ is the WL-dimension.
- We consider a colouring function $\lambda : V_G^k \mapsto \mathbf{C}$ that colours each k-tuple of nodes of the graph with a colour from a set $\mathbf{C}$ of colours. This colour will depend on the isomorphism type of the tuple, e.g., a k-cycle and a k-tree will have different colours.

k-dimensional Weisfeiler-Lehman

Given a graph $G = (V, E)$, the k-dimensional WL algorithm generalises colour refinement as follows:

- We consider k-tuples $(v_1, \ldots, v_k) \in V_G^k$ of nodes, where $k \in \mathbb{N}$ is the WL-dimension.
- We consider a colouring function $\lambda : V_G^k \mapsto \mathbf{C}$ that colours each k-tuple of nodes of the graph with a colour from a set $\mathbf{C}$ of colours. This colour will depend on the isomorphism type of the tuple, e.g., a k-cycle and a k-tree will have different colours.

Partitions $\pi(\lambda)$ of V_G into vertex colour classes, and the refinement relation $\preceq$ is defined as before.

k-dimensional Weisfeiler-Lehman

Given a graph $G = (V, E)$, the k-dimensional WL algorithm generalises colour refinement as follows:

- We consider k-tuples $(v_1, \ldots, v_k) \in V_G^k$ of nodes, where $k \in \mathbb{N}$ is the WL-dimension.
- We consider a colouring function $\lambda : V_G^k \mapsto \mathbf{C}$ that colours each k-tuple of nodes of the graph with a colour from a set $\mathbf{C}$ of colours. This colour will depend on the isomorphism type of the tuple, e.g., a k-cycle and a k-tree will have different colours.

Partitions $\pi(\lambda)$ of V_G into vertex colour classes, and the refinement relation $\preceq$ is defined as before.

We denote a k-tuple as $t = (u_1, \ldots, u_k)$, and define a substitution as $t[v/i] = (u_1, \ldots, u_{i-1}, v, u_{i+1}, \ldots, u_k)$.

k-dimensional Weisfeiler-Lehman

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

1. **Initialisation**: All k-tuples $t \in V_G^k$, are initialised to their initial colours $\lambda^{(0)}(t)$.

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

1. **Initialisation**: All k-tuples $t \in V_G^k$, are initialised to their initial colours $\lambda^{(0)}(t)$.
2. **Refinement**: The colour of a k-tuple $t = (u_1, \ldots, u_k)$ is refined by combining the colours of its neighbourhood, which is defined as the set of all k-tuples in which at most one node differs from t:

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

1. **Initialisation**: All k-tuples $t \in V_G^k$, are initialised to their initial colours $\lambda^{(0)}(t)$.

2. **Refinement**: The colour of a k-tuple $t = (u_1, \ldots, u_k)$ is refined by combining the colours of its neighbourhood, which is defined as the set of all k-tuples in which at most one node differs from t:

$$\lambda^{(i+1)}(t) = \mathsf{HASH}\Big(\lambda^{(i)}(t), \{\{\big(\lambda^{(i)}(t[v/1]), \ldots, \lambda^{(i)}(t[v/k])\big) \mid v \in V_G\}\}\Big),$$

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

1. **Initialisation**: All k-tuples $t \in V_G^k$, are initialised to their initial colours $\lambda^{(0)}(t)$.

2. **Refinement**: The colour of a k-tuple $t = (u_1, \ldots, u_k)$ is refined by combining the colours of its neighbourhood, which is defined as the set of all k-tuples in which at most one node differs from t:

$$\lambda^{(i+1)}(t) = \mathsf{HASH}\Big(\lambda^{(i)}(t), \big\{\!\!\big\{ \big(\lambda^{(i)}(t[v/1]), \ldots, \lambda^{(i)}(t[v/k])\big) \mid v \in V_G \big\}\!\!\big\} \Big),$$

where double-braces denote a multiset, and HASH bijectively maps any pair to a unique value in $\mathbf{C}$.

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

1. **Initialisation**: All k-tuples $t \in V_G^k$, are initialised to their initial colours $\lambda^{(0)}(t)$.

2. **Refinement**: The colour of a k-tuple $t = (u_1, \ldots, u_k)$ is refined by combining the colours of its neighbourhood, which is defined as the set of all k-tuples in which at most one node differs from t:

$$\lambda^{(i+1)}(t) = \mathsf{HASH}\Big(\lambda^{(i)}(t), \{\{\big(\lambda^{(i)}(t[v/1]), \ldots, \lambda^{(i)}(t[v/k])\big) \mid v \in V_G)\}\}\Big),$$

   where double-braces denote a multiset, and HASH bijectively maps any pair to a unique value in $\mathbf{C}$.

3. **Stop**: The algorithm terminates when a stable colouring is reached: That is, at iteration j, where j is the minimal integer satisfying:

k-dimensional Weisfeiler-Lehman

For a given a graph $G = (V, E)$, and a dimension $k \geq 1$, and an initial colouring $\lambda^{(0)}$ of k-tuples:

1. **Initialisation**: All k-tuples $t \in V_G^k$, are initialised to their initial colours $\lambda^{(0)}(t)$.

2. **Refinement**: The colour of a k-tuple $t = (u_1, \ldots, u_k)$ is refined by combining the colours of its neighbourhood, which is defined as the set of all k-tuples in which at most one node differs from t:

$$\lambda^{(i+1)}(t) = \mathsf{HASH}\Big(\lambda^{(i)}(t), \{\{\big(\lambda^{(i)}(t[v/1]), \ldots, \lambda^{(i)}(t[v/k])\big) \mid v \in V_G)\}\}\Big),$$

   where double-braces denote a multiset, and HASH bijectively maps any pair to a unique value in $\mathbf{C}$.

3. **Stop**: The algorithm terminates when a stable colouring is reached: That is, at iteration j, where j is the minimal integer satisfying:

$$\forall t, t' \in V_G^k : \lambda^{(j+1)}(t) = \lambda^{(j+1)}(t') \text{ if and only if } \lambda^{(j)}(t) = \lambda^{(j)}(t').$$

k-dimensional Weisfeiler-Lehman

k-dimensional Weisfeiler-Lehman

There are different versions of the Weisfeiler-Lehman algorithm leading to inconsistent dimension counts. We follow the version of Cai et al. (1992), as it has been adopted as the standard in the literature on graph isomorphism testing. This version is also known as folklore Weisfeiler-Lehman algorithm, and sometimes denoted as k-FWL.

k-dimensional Weisfeiler-Lehman

There are different versions of the Weisfeiler-Lehman algorithm leading to inconsistent dimension counts. We follow the version of Cai et al. (1992), as it has been adopted as the standard in the literature on graph isomorphism testing. This version is also known as folklore Weisfeiler-Lehman algorithm, and sometimes denoted as k-FWL.

In the other version of the algorithm, the update step is defined slightly differently, and based on set of tuples, instead of ordered tuples. All in all:

k-dimensional Weisfeiler-Lehman

There are different versions of the Weisfeiler-Lehman algorithm leading to inconsistent dimension counts. We follow the version of Cai et al. (1992), as it has been adopted as the standard in the literature on graph isomorphism testing. This version is also known as folklore Weisfeiler-Lehman algorithm, and sometimes denoted as k-FWL.

In the other version of the algorithm, the update step is defined slightly differently, and based on set of tuples, instead of ordered tuples. All in all:

- Both variants of the algorithm lead to the same expressive power modulo the shift in dimension counts: For any $k \geq 2$, folklore k-WL is equivalent to $(k+1)$-WL (Grohe, 2017).

k-dimensional Weisfeiler-Lehman

There are different versions of the Weisfeiler-Lehman algorithm leading to inconsistent dimension counts. We follow the version of Cai et al. (1992), as it has been adopted as the standard in the literature on graph isomorphism testing. This version is also known as folklore Weisfeiler-Lehman algorithm, and sometimes denoted as k-FWL.

In the other version of the algorithm, the update step is defined slightly differently, and based on set of tuples, instead of ordered tuples. All in all:

- Both variants of the algorithm lead to the same expressive power modulo the shift in dimension counts: For any $k \geq 2$, folklore k-WL is equivalent to $(k+1)$-WL (Grohe, 2017).
- The folklore WL hierarchy is proper: For each $k \geq 1$ there is a pair of non-isomorphic graphs distinguishable by folklore $(k+1)$-WL but not by folklore k-WL.

k-dimensional Weisfeiler-Lehman

There are different versions of the Weisfeiler-Lehman algorithm leading to inconsistent dimension counts. We follow the version of Cai et al. (1992), as it has been adopted as the standard in the literature on graph isomorphism testing. This version is also known as folklore Weisfeiler-Lehman algorithm, and sometimes denoted as k-FWL.

In the other version of the algorithm, the update step is defined slightly differently, and based on set of tuples, instead of ordered tuples. All in all:

- Both variants of the algorithm lead to the same expressive power modulo the shift in dimension counts: For any $k \geq 2$, folklore k-WL is equivalent to $(k+1)$-WL (Grohe, 2017).
- The folklore WL hierarchy is proper: For each $k \geq 1$ there is a pair of non-isomorphic graphs distinguishable by folklore $(k+1)$-WL but not by folklore k-WL.
- 1-WL and 2-WL have the same expressive power — different from the folklore WL.

k-dimensional Weisfeiler-Lehman

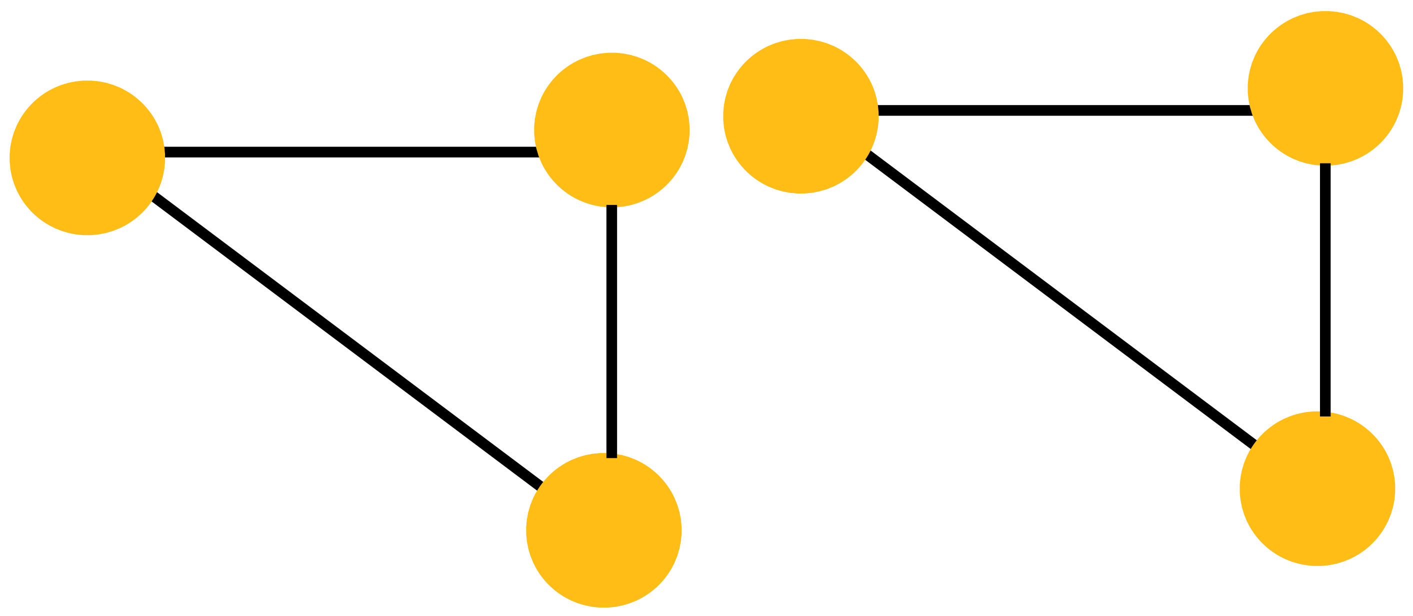
There are different versions of the Weisfeiler-Lehman algorithm leading to inconsistent dimension counts. We follow the version of Cai et al. (1992), as it has been adopted as the standard in the literature on graph isomorphism testing. This version is also known as folklore Weisfeiler-Lehman algorithm, and sometimes denoted as k-FWL.

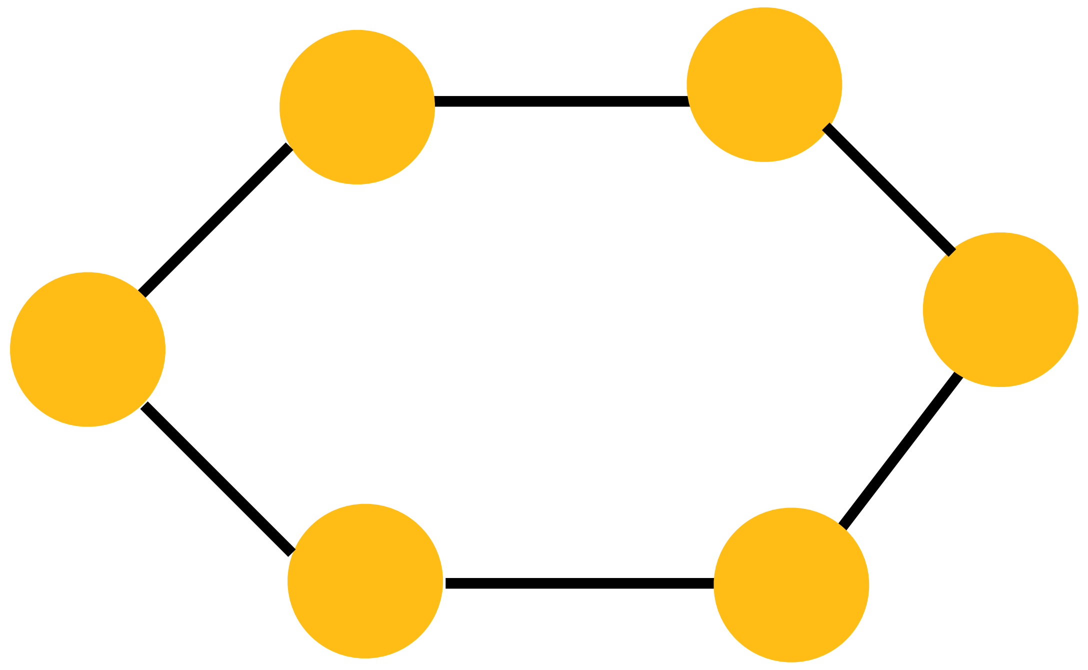
In the other version of the algorithm, the update step is defined slightly differently, and based on set of tuples, instead of ordered tuples. All in all:

- Both variants of the algorithm lead to the same expressive power modulo the shift in dimension counts: For any $k \geq 2$, folklore k-WL is equivalent to $(k+1)$-WL (Grohe, 2017).
- The folklore WL hierarchy is proper: For each $k \geq 1$ there is a pair of non-isomorphic graphs distinguishable by folklore $(k+1)$-WL but not by folklore k-WL.
- 1-WL and 2-WL have the same expressive power — different from the folklore WL.

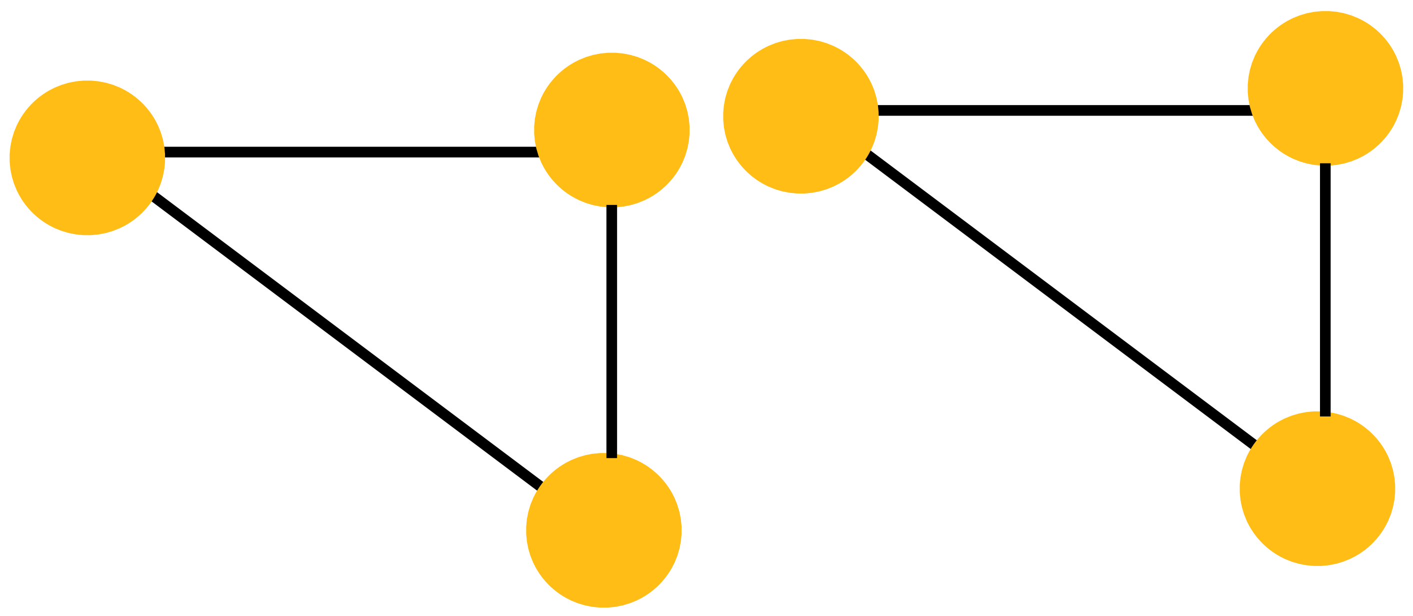
We will always refer to the folklore version of this algorithm in the sequel, unless stated explicitly otherwise.

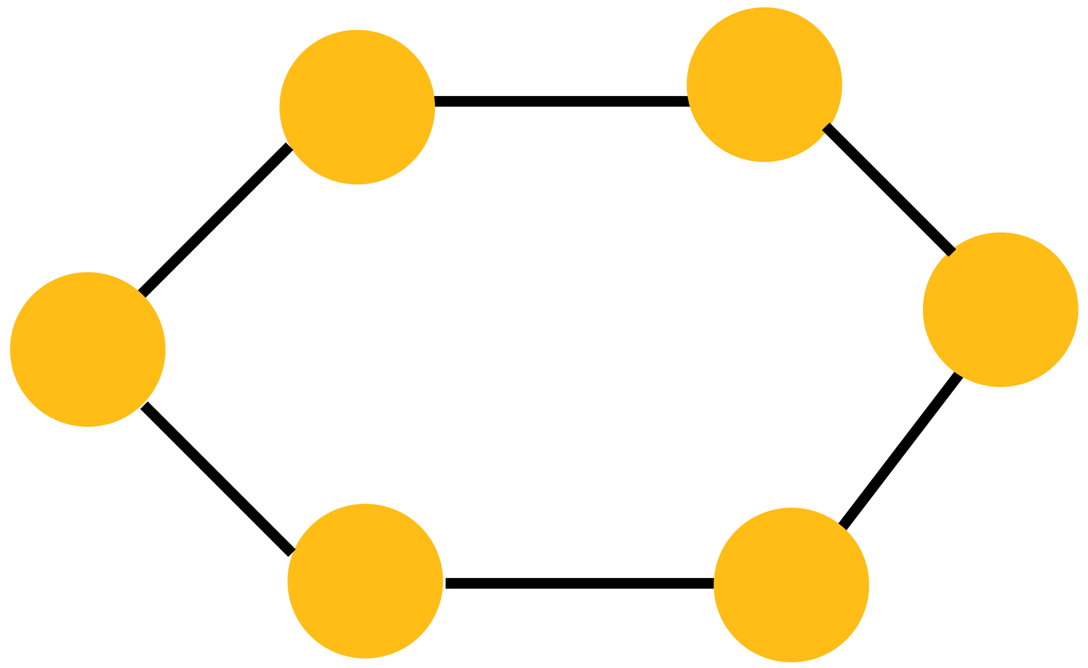
A Tale of Two Graphs





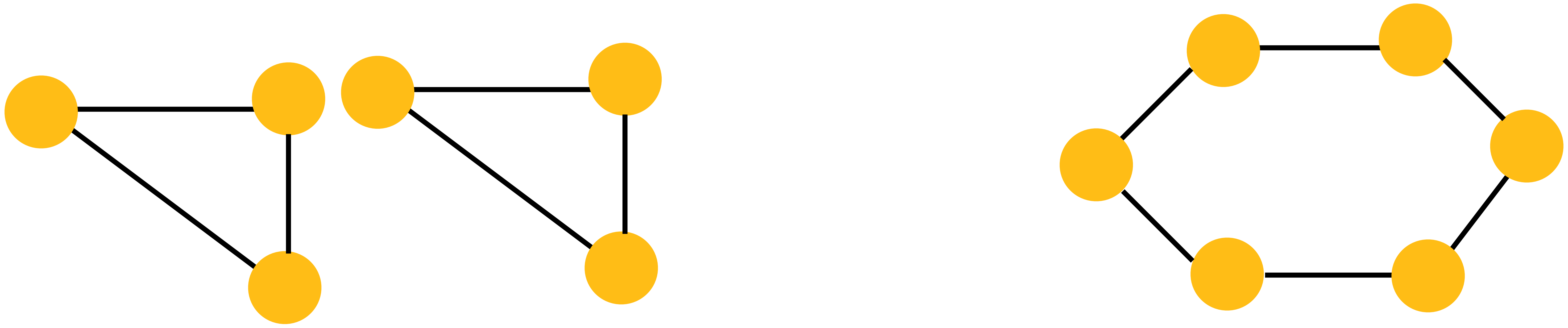
A Tale of Two Graphs





Recall that for all $k \geq 2$ two graphs G and H satisfy the same C^k-sentences if and only if $(k-1)$-WL does not distinguish them (Cai et al., 1992).

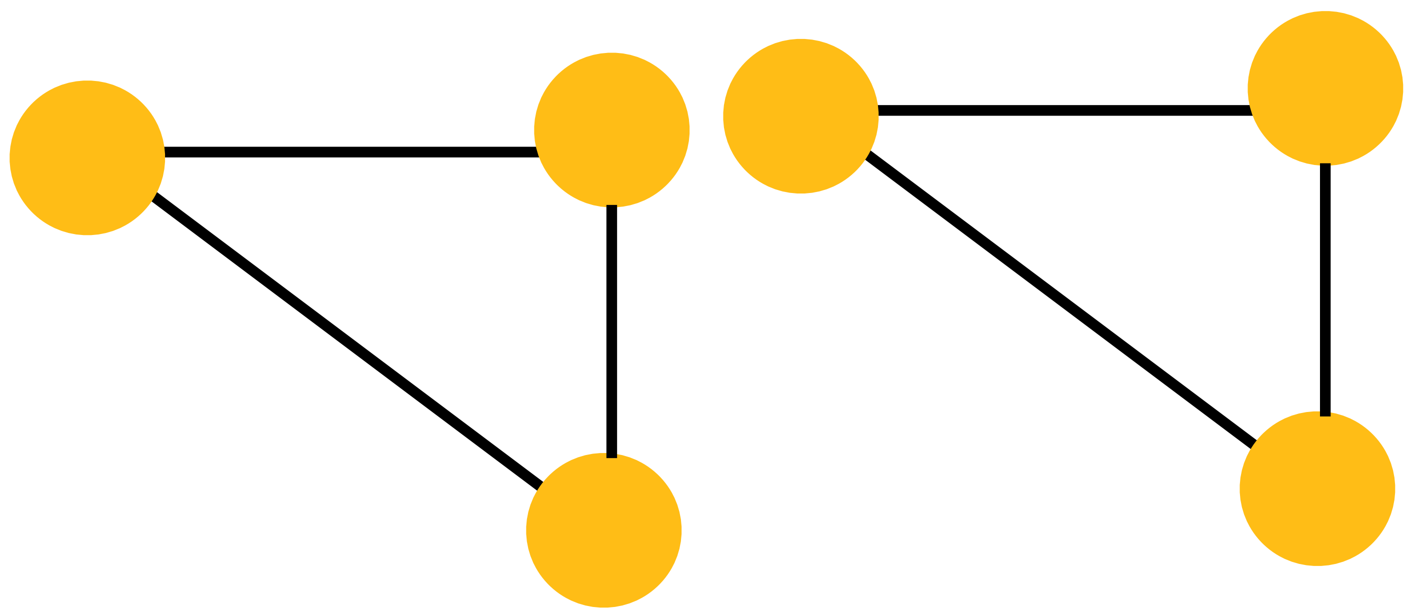
A Tale of Two Graphs

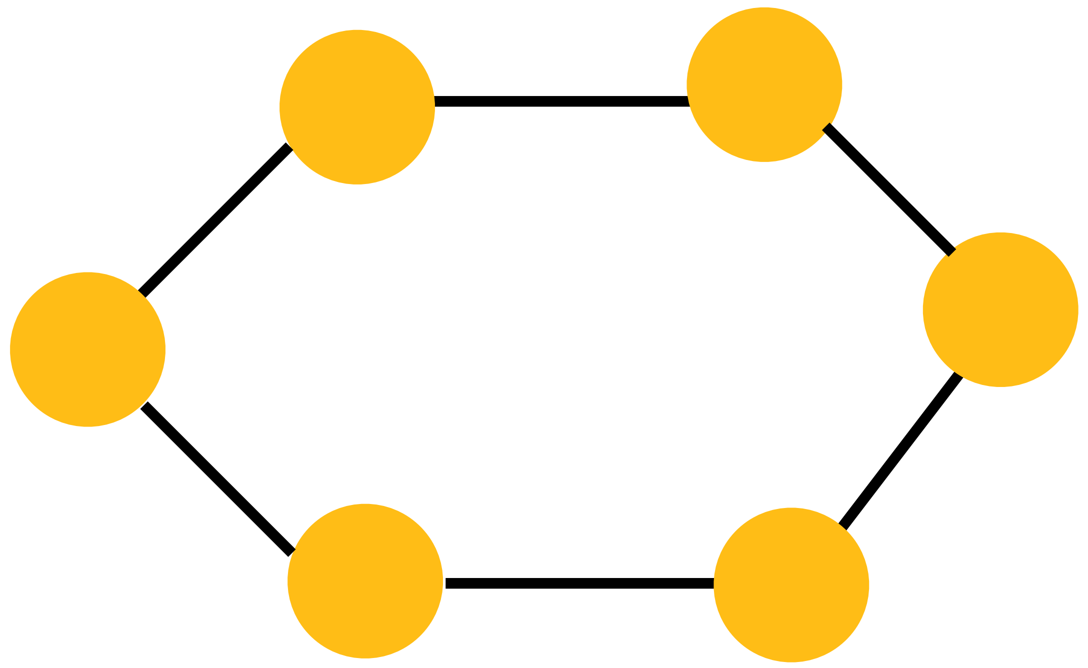


Recall that for all $k \geq 2$ two graphs G and H satisfy the same C^k-sentences if and only if $(k-1)$-WL does not distinguish them (Cai et al., 1992).

To see why separator and non-separator nodes can be distinguished for the given graphs with 2-WL, it is sufficient to formalise this as a logical node classifier in C^3:

A Tale of Two Graphs



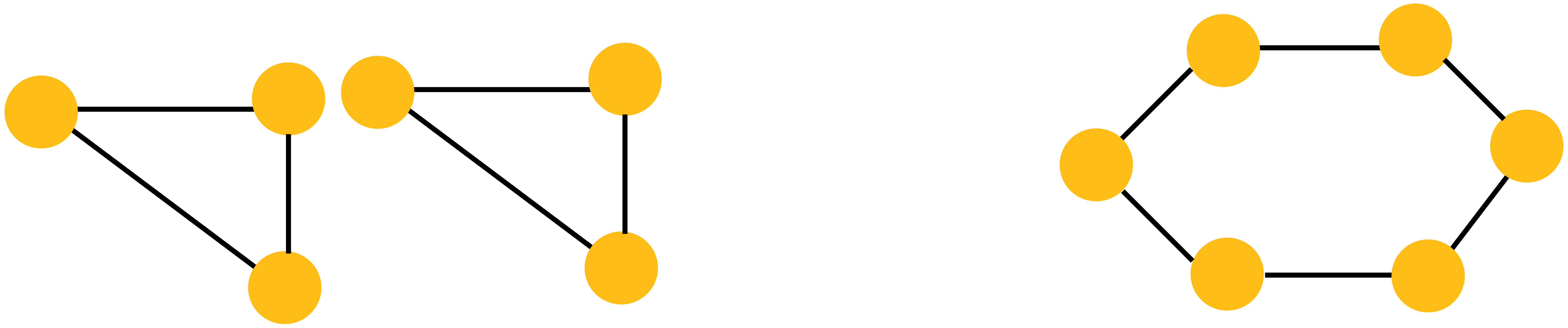


Recall that for all $k \geq 2$ two graphs G and H satisfy the same C^k-sentences if and only if $(k-1)$-WL does not distinguish them (Cai et al., 1992).

To see why separator and non-separator nodes can be distinguished for the given graphs with 2-WL, it is sufficient to formalise this as a logical node classifier in C^3:

$$\Phi(x) = \exists y, z \; E(x, y) \wedge E(y, z) \wedge E(x, z).$$

A Tale of Two Graphs



Recall that for all $k \geq 2$ two graphs G and H satisfy the same C^k-sentences if and only if $(k-1)$-WL does not distinguish them (Cai et al., 1992).

To see why separator and non-separator nodes can be distinguished for the given graphs with 2-WL, it is sufficient to formalise this as a logical node classifier in C^3:

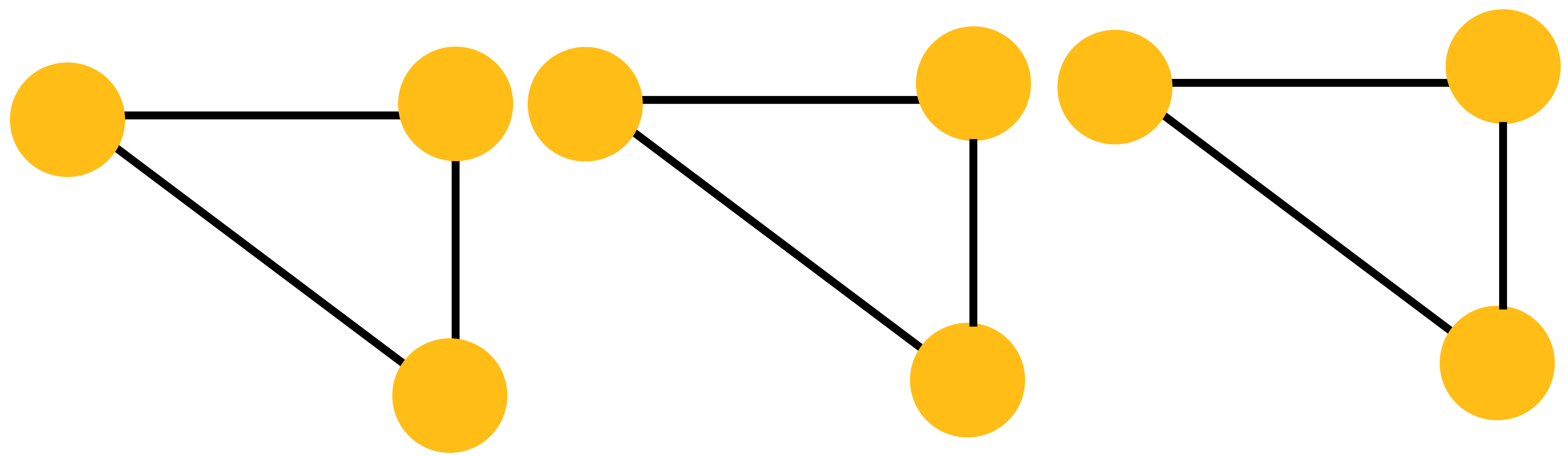
$$\Phi(x) = \exists y, z \; E(x, y) \wedge E(y, z) \wedge E(x, z).$$

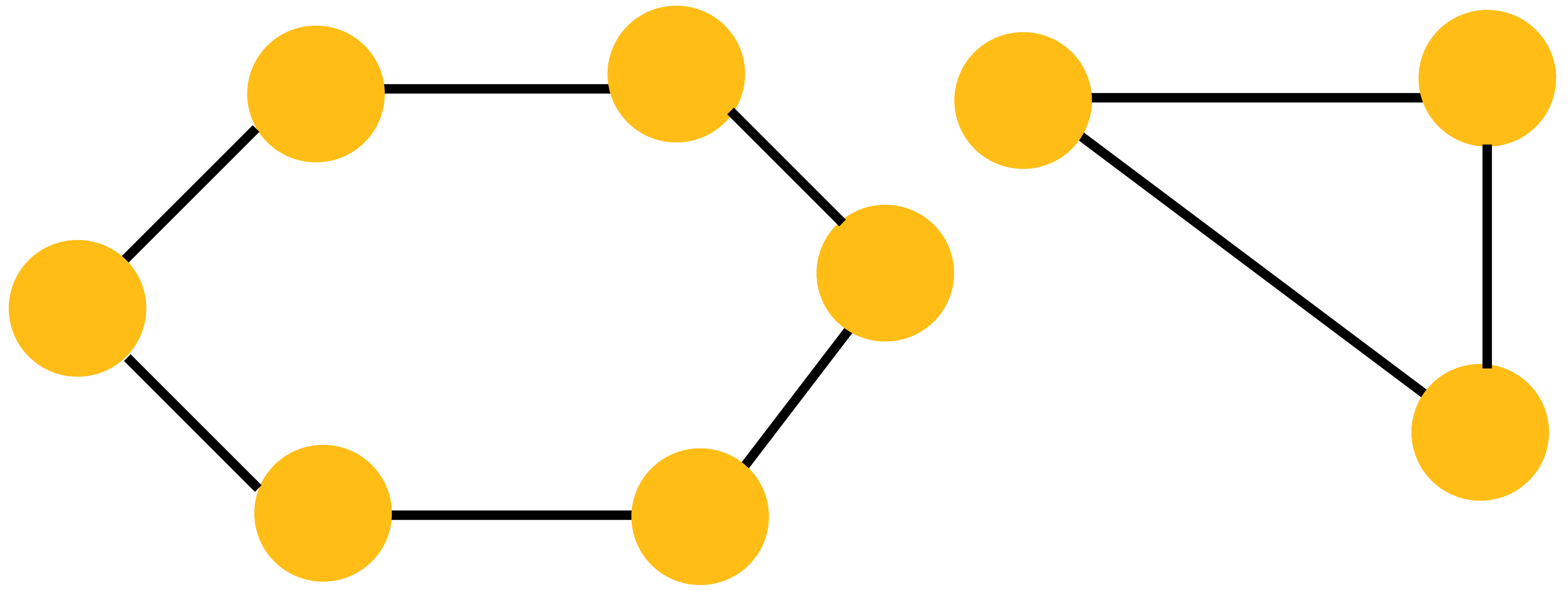
It is easy to see that the graph on the left hand side satisfies $\Phi(u)$ for any node u, and the graph on the right hand side does not. That is, there are C^3-sentences, distinguishing these graphs, and so must 2-WL.

Higher-Order Graph Neural Networks

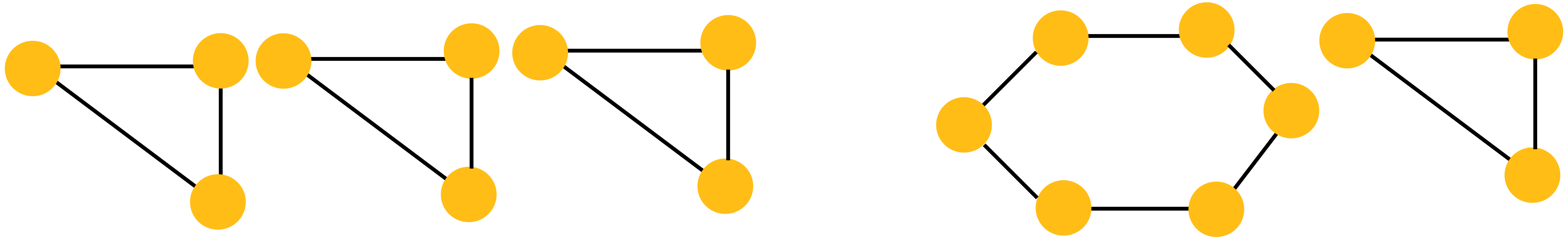
Higher-Order Message Passing Neural Networks

Weisfeiler-Lehman: From 1-GNNs to k-GNNs



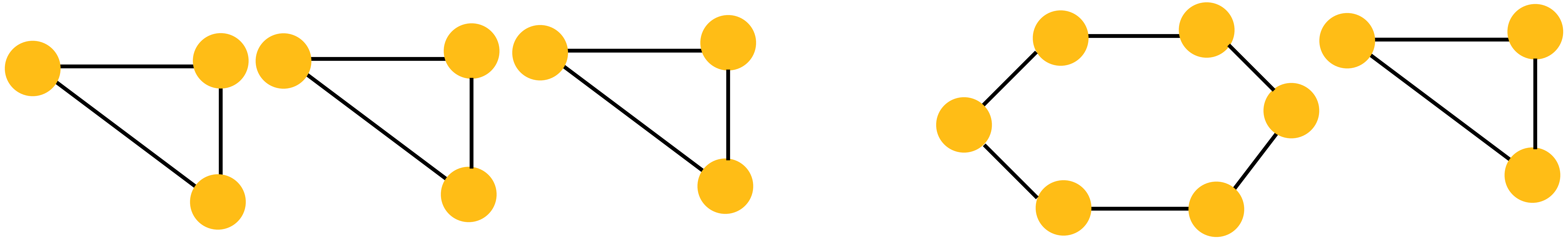


Weisfeiler-Lehman: From 1-GNNs to k-GNNs



The k-GNN model (Morris et al., 2019) is a generalisation of MPNNs based on the $(k-1)$-dimensional WL algorithm.

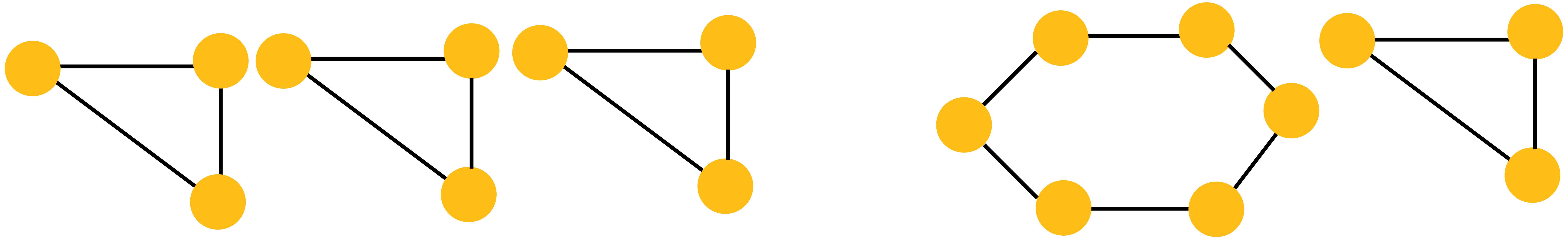
Weisfeiler-Lehman: From 1-GNNs to k-GNNs



The k-GNN model (Morris et al., 2019) is a generalisation of MPNNs based on the $(k-1)$-dimensional WL algorithm.

The idea is still based on message passing, but a higher-order form, where we perform message passing directly between subgraph structures, rather than individual nodes.

Weisfeiler-Lehman: From 1-GNNs to k-GNNs

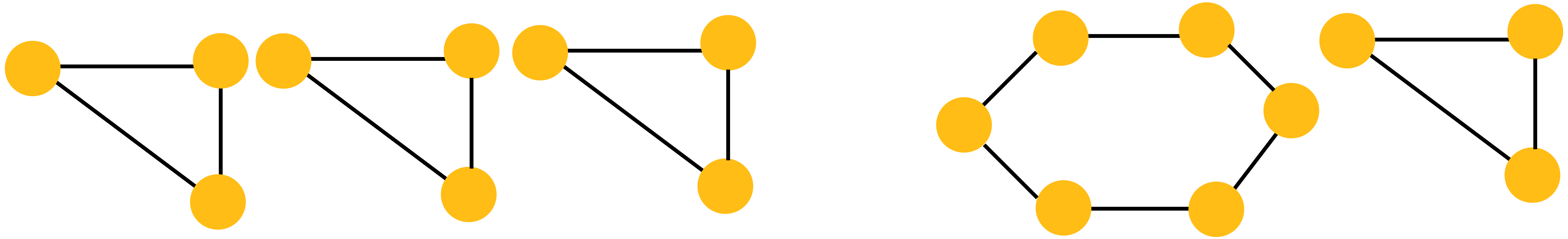


The k-GNN model (Morris et al., 2019) is a generalisation of MPNNs based on the $(k-1)$-dimensional WL algorithm.

The idea is still based on message passing, but a higher-order form, where we perform message passing directly between subgraph structures, rather than individual nodes.

We can therefore view this model as a higher-order MPNN.

Weisfeiler-Lehman: From 1-GNNs to k-GNNs



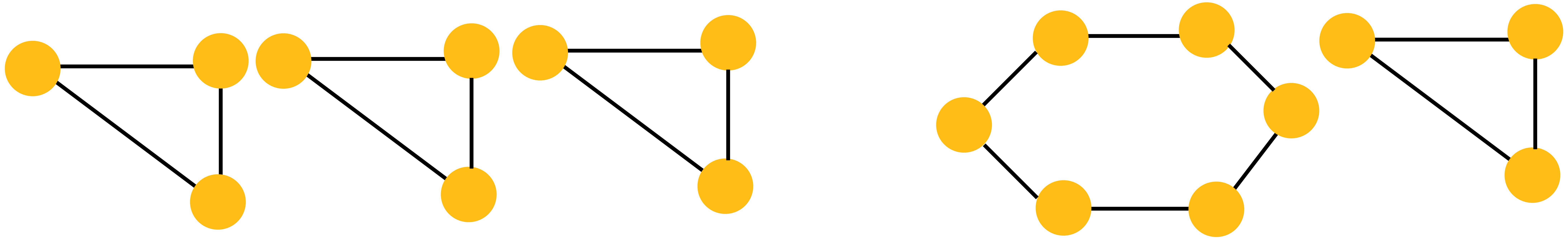
The k-GNN model (Morris et al., 2019) is a generalisation of MPNNs based on the $(k-1)$-dimensional WL algorithm.

The idea is still based on message passing, but a higher-order form, where we perform message passing directly between subgraph structures, rather than individual nodes.

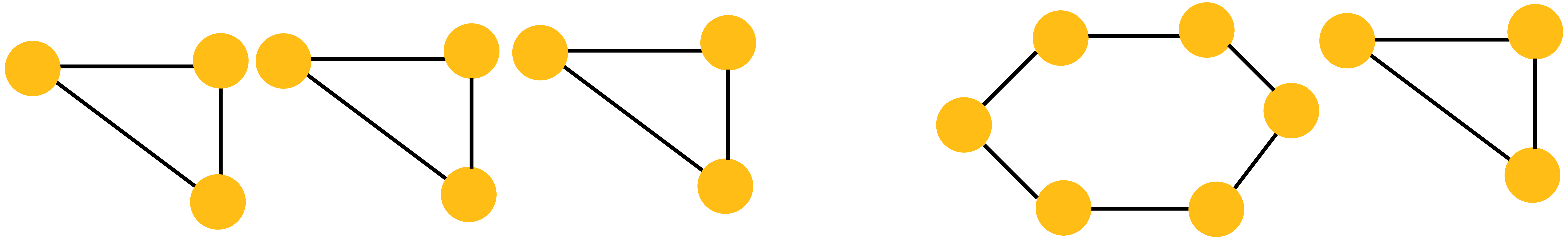
We can therefore view this model as a higher-order MPNN.

This form of message passing can capture structural information that is not visible at the node-level.

Weisfeiler-Lehman: From 1-GNNs to k-GNNs

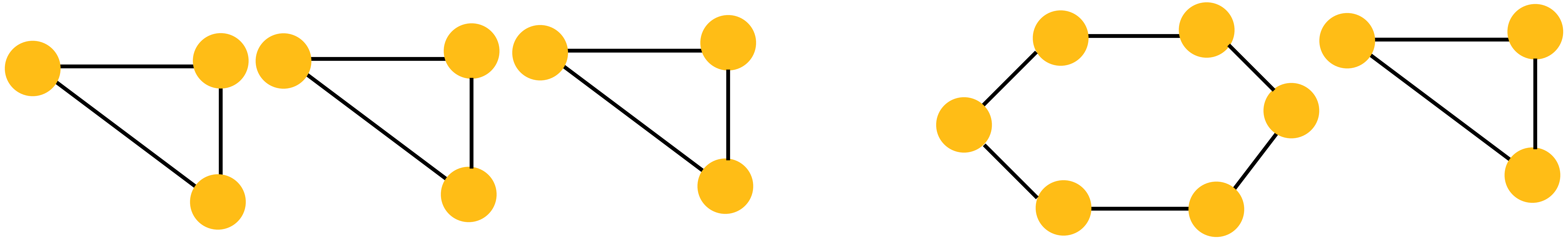


Weisfeiler-Lehman: From 1-GNNs to k-GNNs



3-GNNs of (Morris et al., 2019) have the same power as folklore 2-WL. For example, 3-GNNs can distinguish the two graphs shown above, by the #triangles they contain, given by the Boolean formula in C^3:

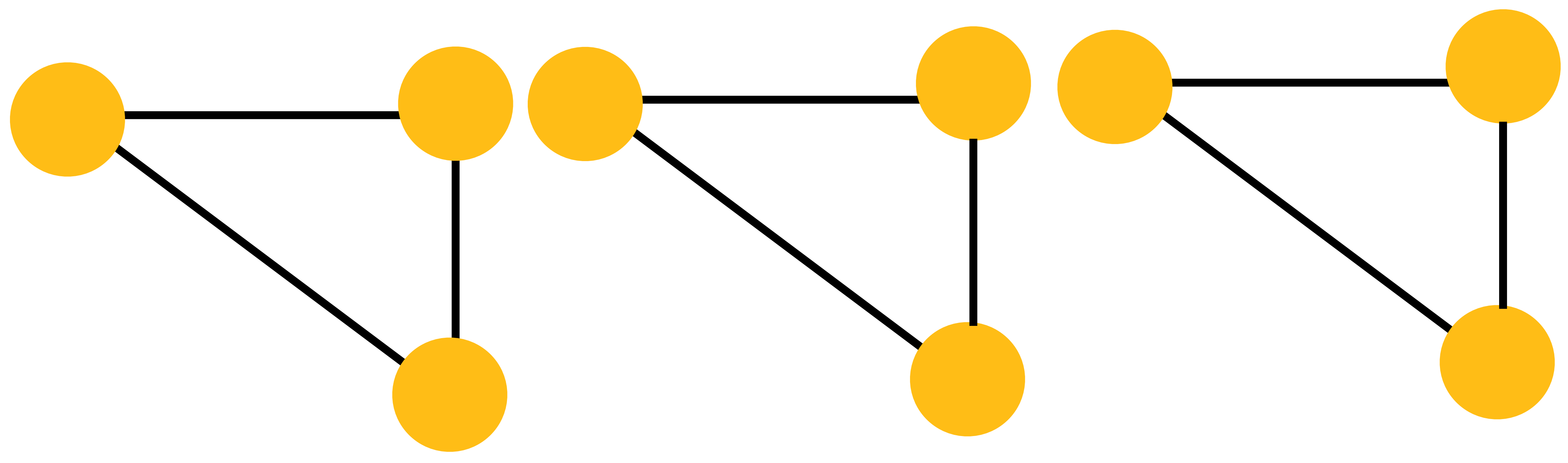
Weisfeiler-Lehman: From 1-GNNs to k-GNNs

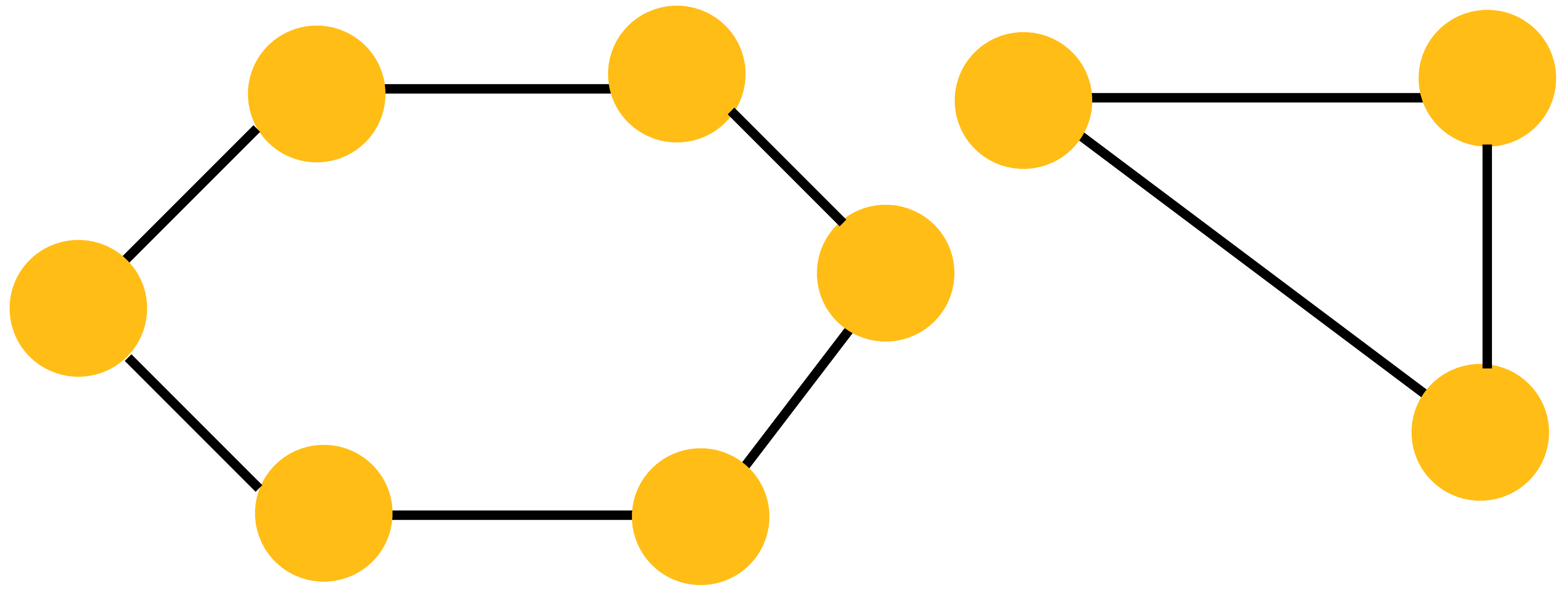


3-GNNs of (Morris et al., 2019) have the same power as folklore 2-WL. For example, 3-GNNs can distinguish the two graphs shown above, by the #triangles they contain, given by the Boolean formula in C^3:

$$\Phi = \exists^{\geq 9} x \Big(\exists y, z \; E(x, y) \wedge E(y, z) \wedge E(x, z) \Big)$$

Weisfeiler-Lehman: From 1-GNNs to k-GNNs





3-GNNs of (Morris et al., 2019) have the same power as folklore 2-WL. For example, 3-GNNs can distinguish the two graphs shown above, by the #triangles they contain, given by the Boolean formula in C^3:

$$\Phi = \exists^{\geq 9} x \Big(\exists y, z \; E(x,y) \wedge E(y,z) \wedge E(x,z) \Big)$$

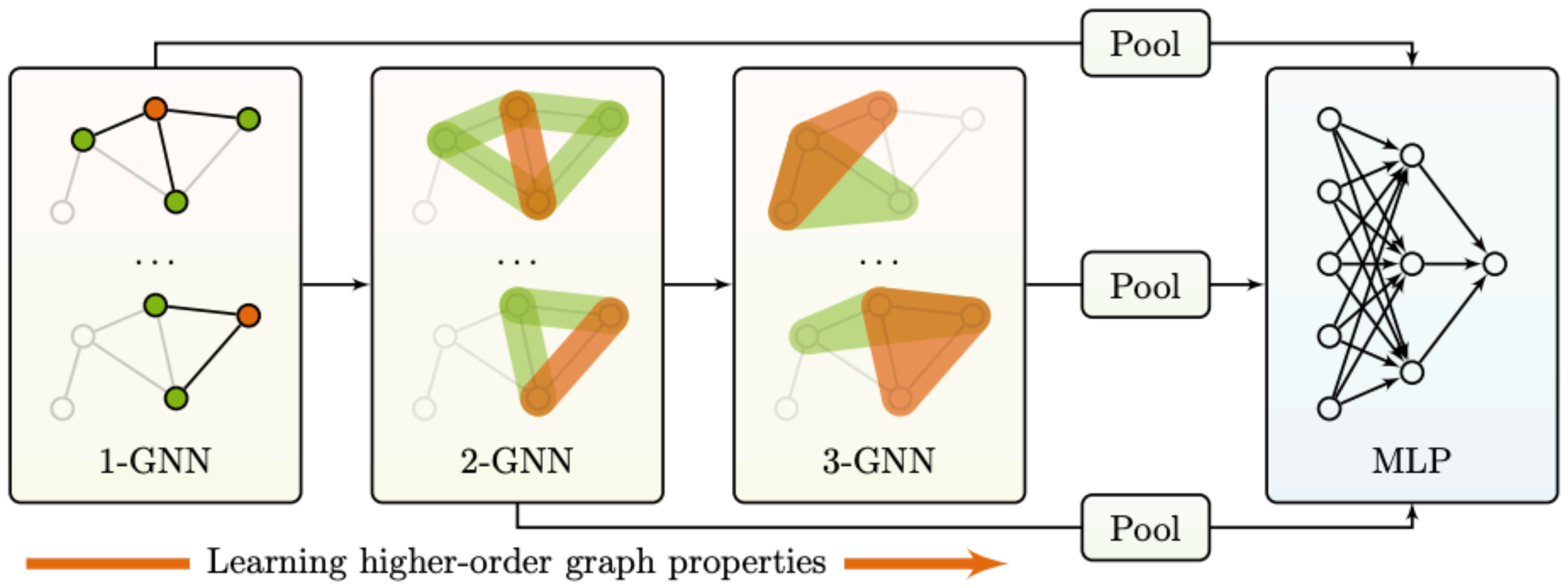
This formula states that there are at least three triangles, which is satisfied by the graph on the left hand side but not by the graph on the right hand side. k-GNNs with $k \geq 3$ are strictly more powerful than MPNNs.

Hierarchical Variants

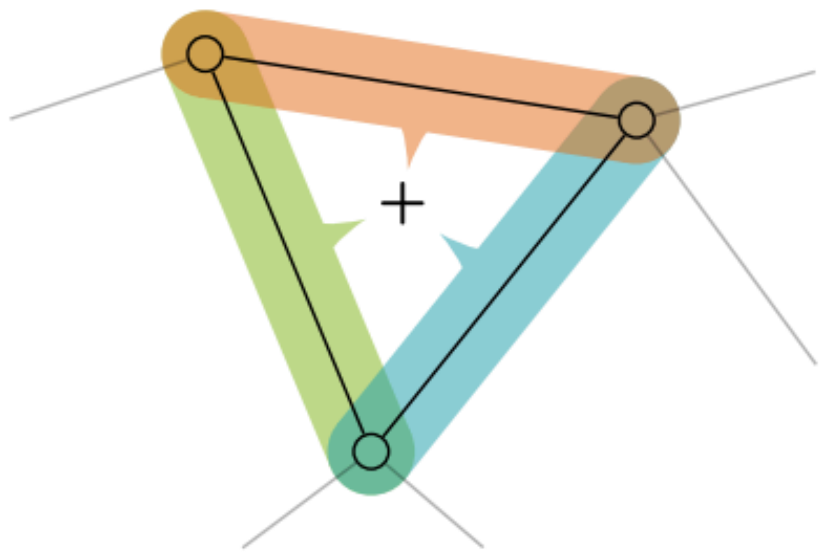
Hierarchical Variants

Hierarchical variants of k-GNNs, called 1-k-GNNs, aim to combine graph representations learned at different granularities. The idea is to apply message passing starting from one-hot indicator vectors as initial features, and applying the usual node-level message passing (1-WL), and afterwards using the resulting representations to learn better representations for pairs of nodes, with a higher-order message passing (2-WL), etc., illustrated in Figure 1 of (Morris et al., 2019), as shown above.

Hierarchical Variants



(a) Hierarchical 1-2-3-GNN network architecture

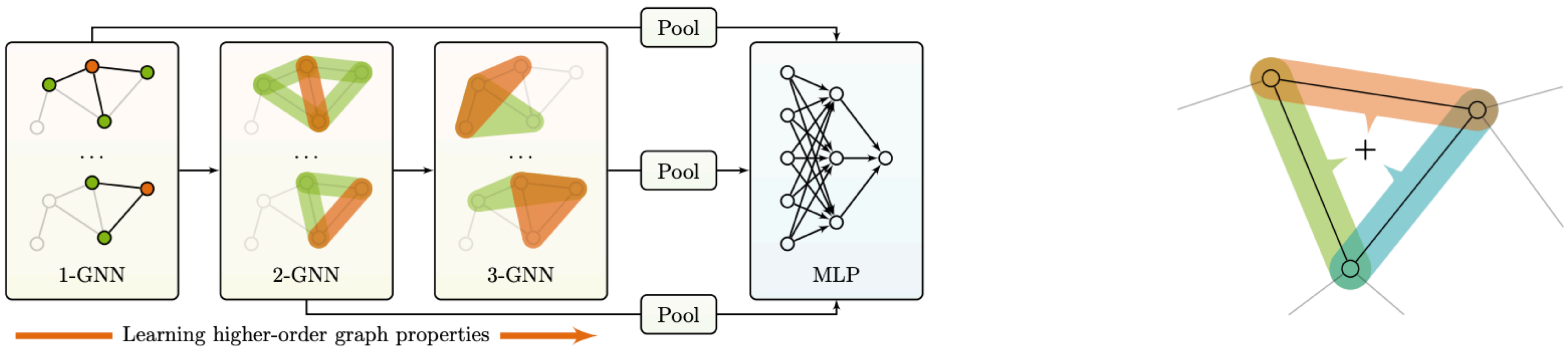


(b) Pooling from 2- to 3-GNN.

Figure 1: Illustration of the proposed hierarchical variant of the k-GNN layer. For each subgraph S on k nodes a feature f is learned, which is initialized with the learned features of all $(k-1)$-element subgraphs of S. Hence, a hierarchical representation of the input graph is learned.

Hierarchical variants of k-GNNs, called 1-k-GNNs, aim to combine graph representations learned at different granularities. The idea is to apply message passing starting from one-hot indicator vectors as initial features, and applying the usual node-level message passing (1-WL), and afterwards using the resulting representations to learn better representations for pairs of nodes, with a higher-order message passing (2-WL), etc., illustrated in Figure 1 of (Morris et al., 2019), as shown above.

Hierarchical Variants

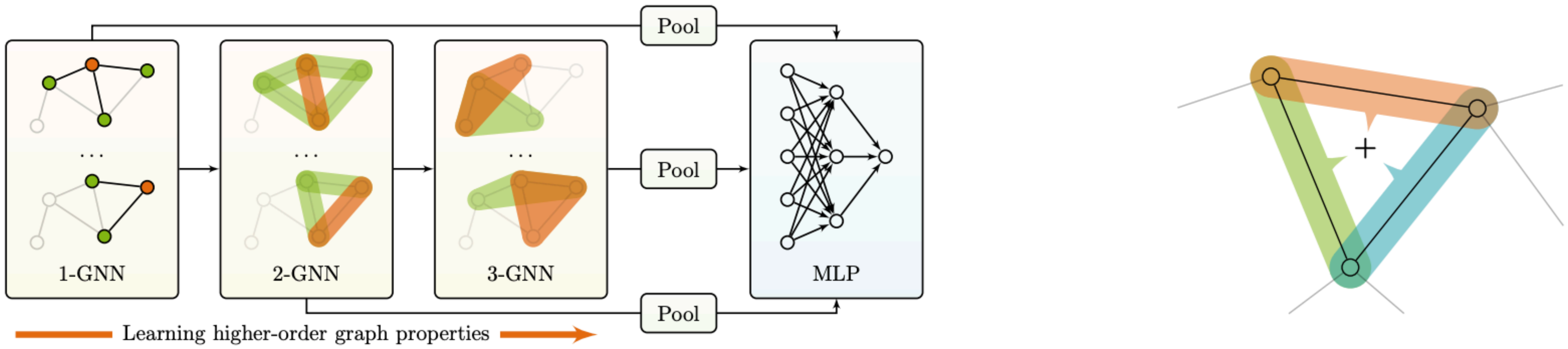


(a) Hierarchical 1-2-3-GNN network architecture

(b) Pooling from 2- to 3-GNN.

Figure 1: Illustration of the proposed hierarchical variant of the k-GNN layer. For each subgraph S on k nodes a feature f is learned, which is initialized with the learned features of all $(k-1)$-element subgraphs of S. Hence, a hierarchical representation of the input graph is learned.

Hierarchical Variants



(a) Hierarchical 1-2-3-GNN network architecture

(b) Pooling from 2- to 3-GNN.

Figure 1: Illustration of the proposed hierarchical variant of the k-GNN layer. For each subgraph S on k nodes a feature f is learned, which is initialized with the learned features of all $(k-1)$-element subgraphs of S. Hence, a hierarchical representation of the input graph is learned.

In this hierarchical approach the initial messages in a k-GNN are based on the output of lower-dimensional GNNs, which allows the model to effectively capture graph structures of varying granularity.

Many real-world graphs inherit a hierarchical structure in this sense, and so a hierarchical message passing approach is potentially helpful — and this is empirically confirmed in the evaluation of (Morris et al., 2019).

Limitations of k-GNNs

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

In fact, it is implemented only up to 3-GNNs (corresponding to 2-WL expressiveness), which already requires cubic memory allocations — already intractable on existing benchmarks.

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

In fact, it is implemented only up to 3-GNNs (corresponding to 2-WL expressiveness), which already requires cubic memory allocations — already intractable on existing benchmarks.

Time complexity of the message passing also increases combinatorially in k!

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

In fact, it is implemented only up to 3-GNNs (corresponding to 2-WL expressiveness), which already requires cubic memory allocations — already intractable on existing benchmarks.

Time complexity of the message passing also increases combinatorially in k!

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

In fact, it is implemented only up to 3-GNNs (corresponding to 2-WL expressiveness), which already requires cubic memory allocations — already intractable on existing benchmarks.

Time complexity of the message passing also increases combinatorially in k!

k-GNNs are more expressive than MPNNs, but still limited in their expressive power, as $(k+1)$-GNN is strictly more expressive than k-GNN for any $k \geq 2$.

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

In fact, it is implemented only up to 3-GNNs (corresponding to 2-WL expressiveness), which already requires cubic memory allocations — already intractable on existing benchmarks.

Time complexity of the message passing also increases combinatorially in k!

k-GNNs are more expressive than MPNNs, but still limited in their expressive power, as $(k+1)$-GNN is strictly more expressive than k-GNN for any $k \geq 2$.

Though being permutation-invariant, the non-hierarchical version of the algorithm is somewhat limited in that the explicit connection to node-level information is lost and only k-tuples are considered.

Limitations of k-GNNs

The higher-order k-GNNs have $(k-1)$-WL expressive power, but need $O(|V|^k)$ memory to run — excessive memory requirements. These higher-order models require intractably-sized intermediate tensors in practice.

In fact, it is implemented only up to 3-GNNs (corresponding to 2-WL expressiveness), which already requires cubic memory allocations — already intractable on existing benchmarks.

Time complexity of the message passing also increases combinatorially in k!

k-GNNs are more expressive than MPNNs, but still limited in their expressive power, as $(k+1)$-GNN is strictly more expressive than k-GNN for any $k \geq 2$.

Though being permutation-invariant, the non-hierarchical version of the algorithm is somewhat limited in that the explicit connection to node-level information is lost and only k-tuples are considered.

This can hurt the inductive bias, especially when node level features are very important for the task at hand!

Invariant/Equivariant Graph Networks

Invariant/Equivariant Graph Networks

Invariant/Equivariant Graph Networks

Invariant (resp., equivariant) graph networks (Maron et al., 2019a) represent graphs as a higher-order tensor, where node adjacency is directly encoded.

Invariant/Equivariant Graph Networks

Invariant (resp., equivariant) graph networks (Maron et al., 2019a) represent graphs as a higher-order tensor, where node adjacency is directly encoded.

Let us restate the properties invariance and equivariance using features and based on MPNNs.

Invariant/Equivariant Graph Networks

Invariant (resp., equivariant) graph networks (Maron et al., 2019a) represent graphs as a higher-order tensor, where node adjacency is directly encoded.

Let us restate the properties invariance and equivariance using features and based on MPNNs.

MPNNs are permutation-invariant:

$$\text{POOL}\big(MPNN(\mathbf{P}\mathbf{A}\mathbf{P}^\top, \mathbf{P}\mathbf{X})\big) = \text{POOL}\big(\mathbf{P}(MPNN(\mathbf{A}, \mathbf{X}))\big)$$

for any permutation matrix $\mathbf{P}$, adjacency matrix $\mathbf{A}$, and feature matrix $\mathbf{X}$.

Invariant/Equivariant Graph Networks

Invariant (resp., equivariant) graph networks (Maron et al., 2019a) represent graphs as a higher-order tensor, where node adjacency is directly encoded.

Let us restate the properties invariance and equivariance using features and based on MPNNs.

MPNNs are permutation-invariant:

$$\mathsf{POOL}\big(MPNN(\mathbf{PAP}^\top, \mathbf{PX})\big) = \mathsf{POOL}\big(\mathbf{P}(MPNN(\mathbf{A}, \mathbf{X}))\big)$$

for any permutation matrix $\mathbf{P}$, adjacency matrix $\mathbf{A}$, and feature matrix $\mathbf{X}$.

MPNNs are permutation-equivariant:

$$\mathbf{P}(MPNN(\mathbf{A}, \mathbf{X})) = MPNN(\mathbf{PAP}^\top, \mathbf{PX})$$

for any permutation matrix $\mathbf{P}$, adjacency matrix $\mathbf{A}$, and feature matrix $\mathbf{X}$.

Invariant/Equivariant Graph Networks

Invariant/Equivariant Graph Networks

Idea: Based on a tensor representation of graphs, define a GNN model based on permutation equivariant/ invariant tensor operations.

Invariant/Equivariant Graph Networks

Idea: Based on a tensor representation of graphs, define a GNN model based on permutation equivariant/ invariant tensor operations.

Formally, we consider an order $(k+1)$-tensor $\mathbf{T} \in \mathbb{R}^{|V|^k \times d}$ where the first k channels of this tensor are indexed by the nodes of the graph. We write $\mathbf{P} \star \mathbf{T}$ to denote a permutation of the first k channels of this tensor according the node permutation matrix $\mathbf{P}$.

Invariant/Equivariant Graph Networks

Idea: Based on a tensor representation of graphs, define a GNN model based on permutation equivariant/ invariant tensor operations.

Formally, we consider an order $(k+1)$-tensor $\mathbf{T} \in \mathbb{R}^{|V|^k \times d}$ where the first k channels of this tensor are indexed by the nodes of the graph. We write $\mathbf{P} \star \mathbf{T}$ to denote a permutation of the first k channels of this tensor according the node permutation matrix $\mathbf{P}$.

A linear invariant layer can be defined as $\mathscr{L} : \mathbb{R}^{|V|^k \times d_1} \mapsto \mathbb{R}^{d_2}$ such that for all permutations $\mathbf{P}$:

$$\mathscr{L} \times (\mathbf{P} \star \mathbf{T}) = (\mathscr{L} \times \mathbf{T}).$$

Invariant/Equivariant Graph Networks

Idea: Based on a tensor representation of graphs, define a GNN model based on permutation equivariant/invariant tensor operations.

Formally, we consider an order $(k+1)$-tensor $\mathbf{T} \in \mathbb{R}^{|V|^k \times d}$ where the first k channels of this tensor are indexed by the nodes of the graph. We write $\mathbf{P} \star \mathbf{T}$ to denote a permutation of the first k channels of this tensor according the node permutation matrix $\mathbf{P}$.

A linear invariant layer can be defined as $\mathscr{L} : \mathbb{R}^{|V|^k \times d_1} \mapsto \mathbb{R}^{d_2}$ such that for all permutations $\mathbf{P}$:

$$\mathscr{L} \times (\mathbf{P} \star \mathbf{T}) = (\mathscr{L} \times \mathbf{T}).$$

A linear equivariant layer can be defined as $\mathscr{L} : \mathbb{R}^{|V|^{k_1} \times d_1} \mapsto \mathbb{R}^{|V|^{k_2} \times d_2}$ such that for all permutations $\mathbf{P}$:

$$\mathscr{L} \times (\mathbf{P} \star \mathbf{T}) = \mathbf{P} \star (\mathscr{L} \times \mathbf{T}).$$

Invariant/Equivariant Graph Networks

Invariant/Equivariant Graph Networks

Based on this abstraction, invariant k-order GNN model (Maron et al., 2019b), or k-IGNs, is defined as:

$$F = \text{MLP} \circ \mathscr{H} \circ \mathscr{L}_d \circ \sigma \circ \cdots \circ \sigma \circ \mathscr{L}_1,$$

where $\mathscr{L}_1, \ldots, \mathscr{L}_d$ are equivariant linear layers (with up to k different channels), $\mathscr{H}$ is an invariant layer, and σ denotes element-wise non-linearity. Figure 1 of (Maron et al., 2019c) illustrates the model.

Invariant/Equivariant Graph Networks

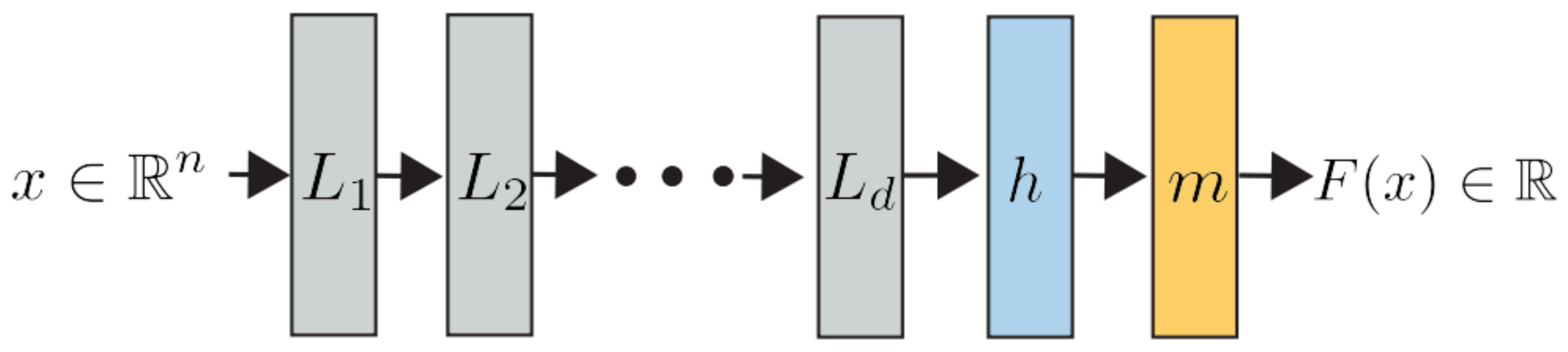


Figure 1. Illustration of invariant network architecture. The function is composed of multiple linear G-equivariant layers (gray), possibly of high order, and ends with a linear G-invariant function (light blue) followed by a Multi Layer Perceptron (yellow).

Based on this abstraction, invariant k-order GNN model (Maron et al., 2019b), or k-IGNs, is defined as:

$$F = \mathrm{MLP} \circ \mathscr{H} \circ \mathscr{L}_d \circ \sigma \circ \cdots \circ \sigma \circ \mathscr{L}_1,$$

where $\mathscr{L}_1, \ldots, \mathscr{L}_d$ are equivariant linear layers (with up to k different channels), $\mathscr{H}$ is an invariant layer, and σ denotes element-wise non-linearity. Figure 1 of (Maron et al., 2019c) illustrates the model.

Invariant/Equivariant Graph Networks

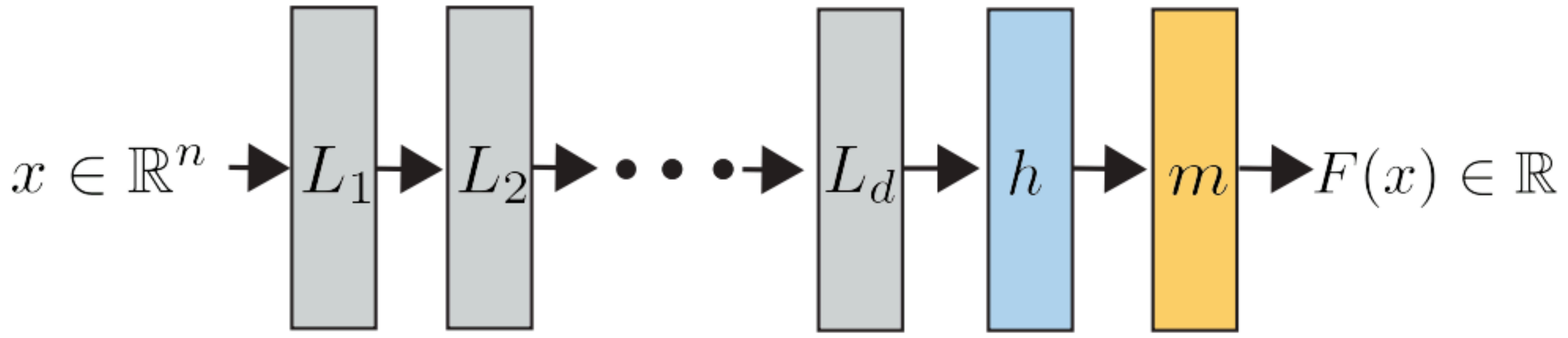


Figure 1. Illustration of invariant network architecture. The function is composed of multiple linear G-equivariant layers (gray), possibly of high order, and ends with a linear G-invariant function (light blue) followed by a Multi Layer Perceptron (yellow).

The input to the k-order invariant GNN is a tensor $\mathbf{T} \in \mathbb{R}^{|V|^2 \times d}$, where the first two channels correspond to the adjacency matrix of the graph and the remaining channels encode the initial node features.

The model is called k-order, as it allows equivariant layers with k channels, and this directly correlates with the expressive power of the model.

Expressive Power of Invariant Graph Networks

Expressive Power of Invariant Graph Networks

It has been shown that k-IGNs are as powerful as k-WL test.

Expressive Power of Invariant Graph Networks

It has been shown that k-IGNs are as powerful as k-WL test.

Theorem 1 (Maron et al., 2019a). Given two graphs G, G' that can be distinguished by the k-WL graph isomorphism test, there exists a k-order network F so that $F(G) \neq F(G')$. On the other direction for every two isomorphic graphs G, G' and k-order network F, $F(G) = F(G')$.

Expressive Power of Invariant Graph Networks

It has been shown that k-IGNs are as powerful as k-WL test.

Theorem 1 (Maron et al., 2019a). Given two graphs G, G' that can be distinguished by the k-WL graph isomorphism test, there exists a k-order network F so that $F(G) \neq F(G')$. On the other direction for every two isomorphic graphs G, G' and k-order network F, $F(G) = F(G')$.

If we bound the size of the input graphs with n, measured in the number of nodes, then n-th order invariant networks can distinguish any pair of non-isomorphic graphs. Note that invariant networks with order-2 tensors could already be computationally challenging!

Expressive Power of Invariant Graph Networks

It has been shown that k-IGNs are as powerful as k-WL test.

Theorem 1 (Maron et al., 2019a). Given two graphs G, G' that can be distinguished by the k-WL graph isomorphism test, there exists a k-order network F so that $F(G) \neq F(G')$. On the other direction for every two isomorphic graphs G, G' and k-order network F, $F(G) = F(G')$.

If we bound the size of the input graphs with n, measured in the number of nodes, then n-th order invariant networks can distinguish any pair of non-isomorphic graphs. Note that invariant networks with order-2 tensors could already be computationally challenging!

Invariant networks are shown to be universal (Maron et al., 2019c), but with tensor orders of $O(|V|^2)$!

Expressive Power of Invariant Graph Networks

It has been shown that k-IGNs are as powerful as k-WL test.

Theorem 1 (Maron et al., 2019a). Given two graphs G, G' that can be distinguished by the k-WL graph isomorphism test, there exists a k-order network F so that $F(G) \neq F(G')$. On the other direction for every two isomorphic graphs G, G' and k-order network F, $F(G) = F(G')$.

If we bound the size of the input graphs with n, measured in the number of nodes, then n-th order invariant networks can distinguish any pair of non-isomorphic graphs. Note that invariant networks with order-2 tensors could already be computationally challenging!

Invariant networks are shown to be universal (Maron et al., 2019c), but with tensor orders of $O(|V|^2)$!

More specifically, invariant networks are universal with tensor order $\dfrac{n(n-1)}{2}$. An alternative proof is given by (Keriven and Peyré, 2019), who also showed a universality result for the equivariant case.

Limitations of Invariant Graph Networks

Limitations of Invariant Graph Networks

Similarly to k-GNNs, k-IGNs may lose the inductive bias of node information relative to standard MPNNs.

Limitations of Invariant Graph Networks

Similarly to k-GNNs, k-IGNs may lose the inductive bias of node information relative to standard MPNNs.

They also only use adjacency information implicitly, via features in the initial tensor, but do not limit interactions solely to edge-connected nodes. This may hurt inductive bias when the property being learned is local.

Limitations of Invariant Graph Networks

Similarly to k-GNNs, k-IGNs may lose the inductive bias of node information relative to standard MPNNs.

They also only use adjacency information implicitly, via features in the initial tensor, but do not limit interactions solely to edge-connected nodes. This may hurt inductive bias when the property being learned is local.

Indeed, k-IGNs are inherently designed for graph-level computations: the correspondence with node tuples, is only implicit, unlike k-GNNs, where tuples have representations that are explicitly maintained and updated.

Limitations of Invariant Graph Networks

Similarly to k-GNNs, k-IGNs may lose the inductive bias of node information relative to standard MPNNs.

They also only use adjacency information implicitly, via features in the initial tensor, but do not limit interactions solely to edge-connected nodes. This may hurt inductive bias when the property being learned is local.

Indeed, k-IGNs are inherently designed for graph-level computations: the correspondence with node tuples, is only implicit, unlike k-GNNs, where tuples have representations that are explicitly maintained and updated.

Finally, these models are also prohibitive to run for large values of k, due to their very large memory and computational requirements.

Provably Powerful Graph Networks

Provably Powerful Graph Networks

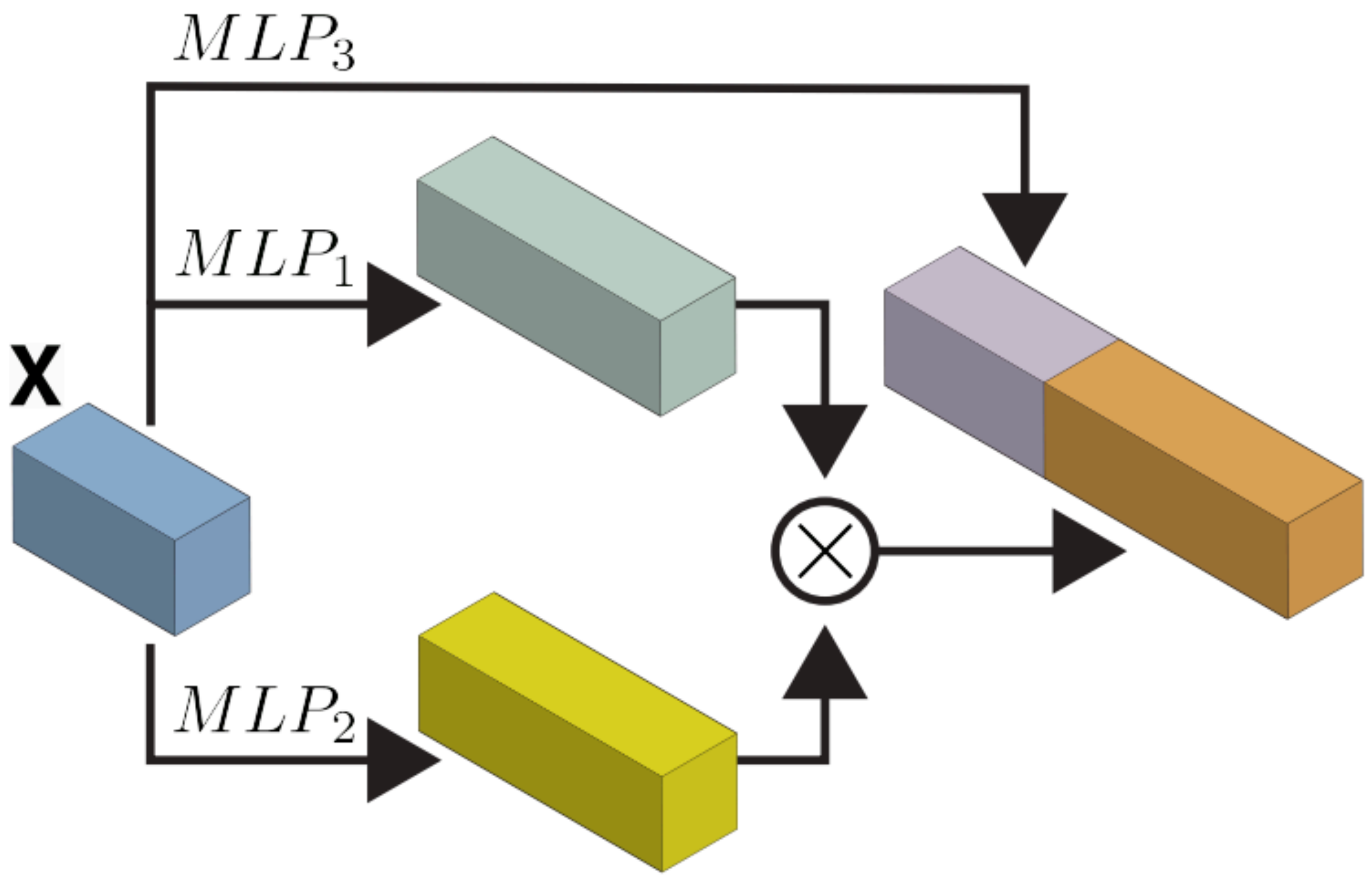


Figure 2: Block structure.

Provably Powerful Graph Networks

Provably powerful graph networks (PPGNs) are special type of invariant networks, motivated by the search for more expressive, yet still scalable, GNN models

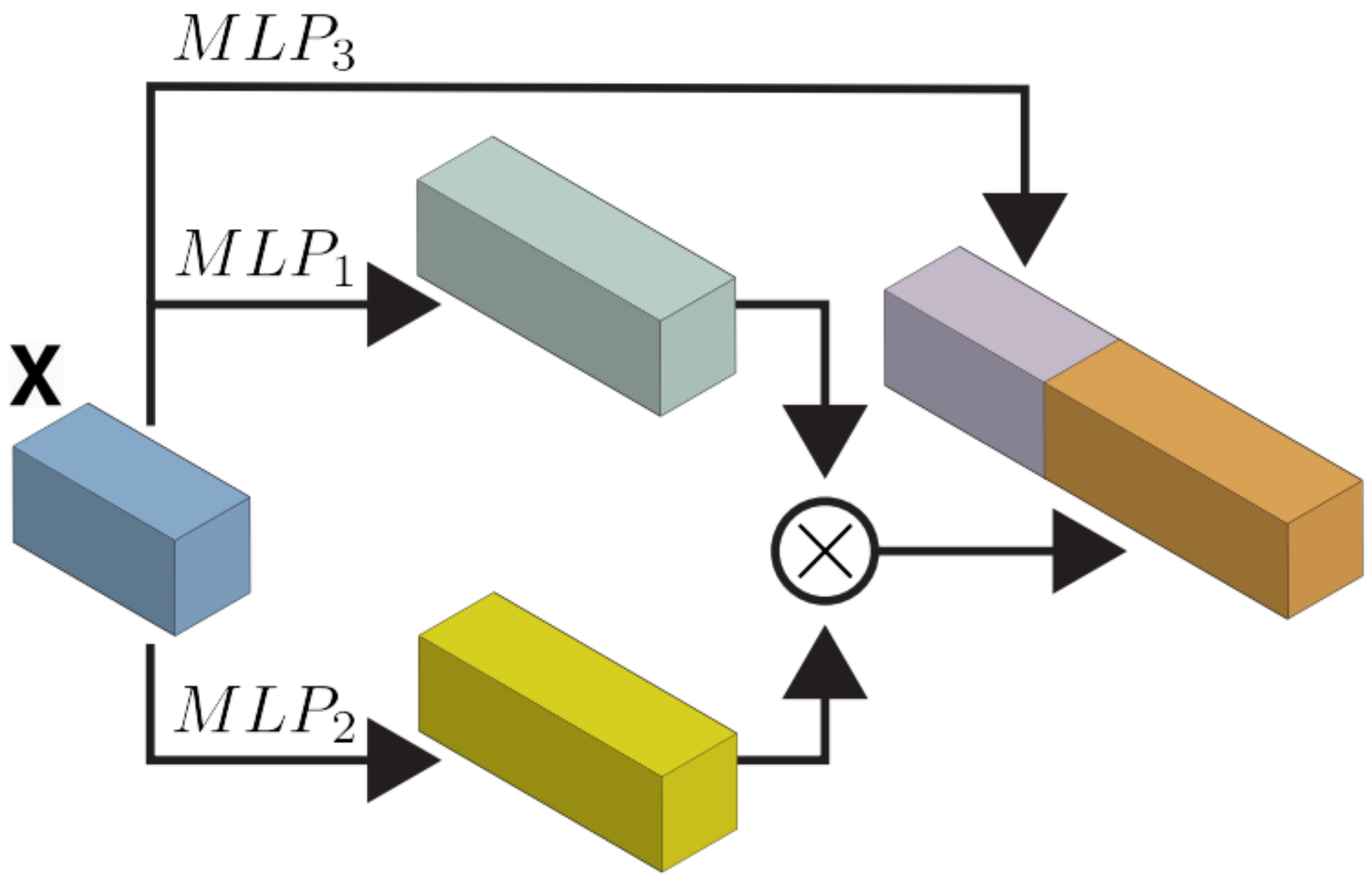


Figure 2: Block structure.

Provably Powerful Graph Networks

Provably powerful graph networks (PPGNs) are special type of invariant networks, motivated by the search for more expressive, yet still scalable, GNN models

PPGN works with 2 tensors, and is defined as follows:

$$F = \mathrm{MLP} \circ \mathscr{H} \circ \mathscr{B}_d \circ \cdots \circ \mathscr{B}_1,$$

where, as in k-IGNs, $\mathscr{H}$ is an invariant layer, and $\mathscr{B}_1, \ldots, \mathscr{B}_d$ are blocks have the structure shown in Figure 2 of (Maron et al., 2019a).

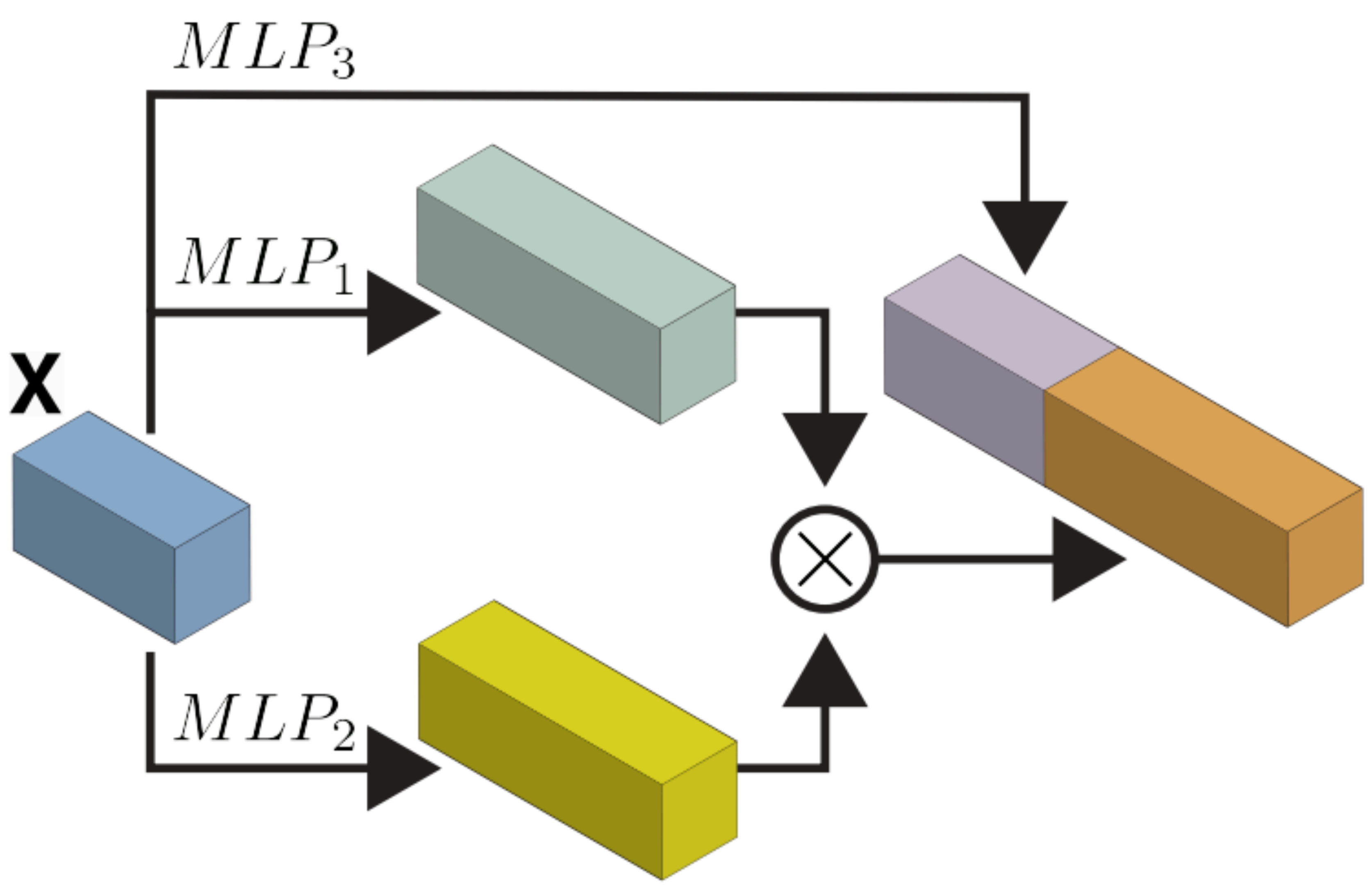


Figure 2: Block structure.

Provably Powerful Graph Networks

Provably powerful graph networks (PPGNs) are special type of invariant networks, motivated by the search for more expressive, yet still scalable, GNN models

PPGN works with 2 tensors, and is defined as follows:

$$F = \text{MLP} \circ \mathscr{H} \circ \mathscr{B}_d \circ \cdots \circ \mathscr{B}_1,$$

where, as in k-IGNs, $\mathscr{H}$ is an invariant layer, and $\mathscr{B}_1, \ldots, \mathscr{B}_d$ are blocks have the structure shown in Figure 2 of (Maron et al., 2019a).

Briefly, given an input $\mathbf{T} \in \mathbb{R}^{|V| \times |V| \times d}$ the idea is to apply MLP to each feature of the input tensor independently (i.e., 3 MLPs), and then perform matrix multiplication between matching features.

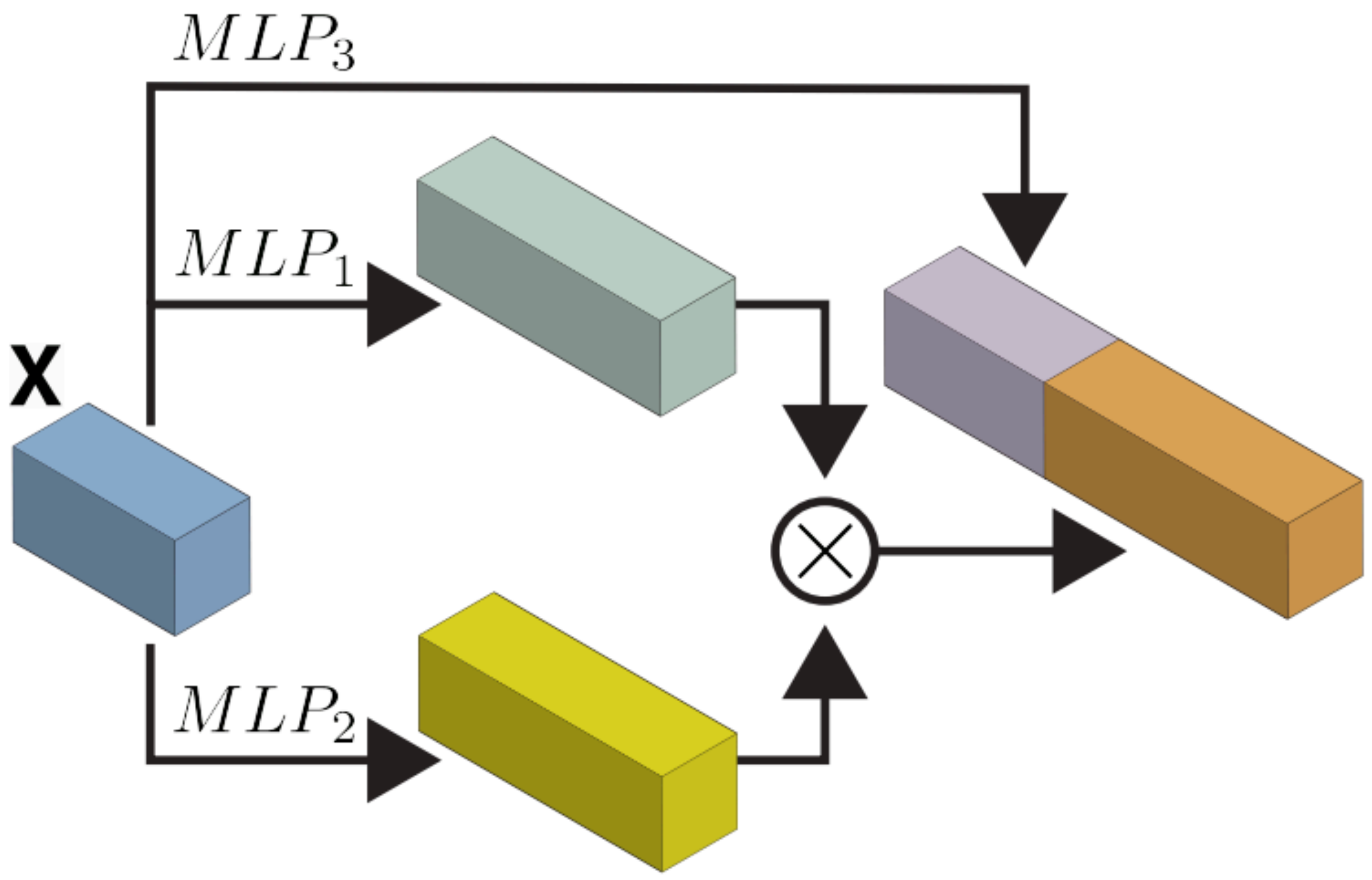


Figure 2: Block structure.

Provably Powerful Graph Networks

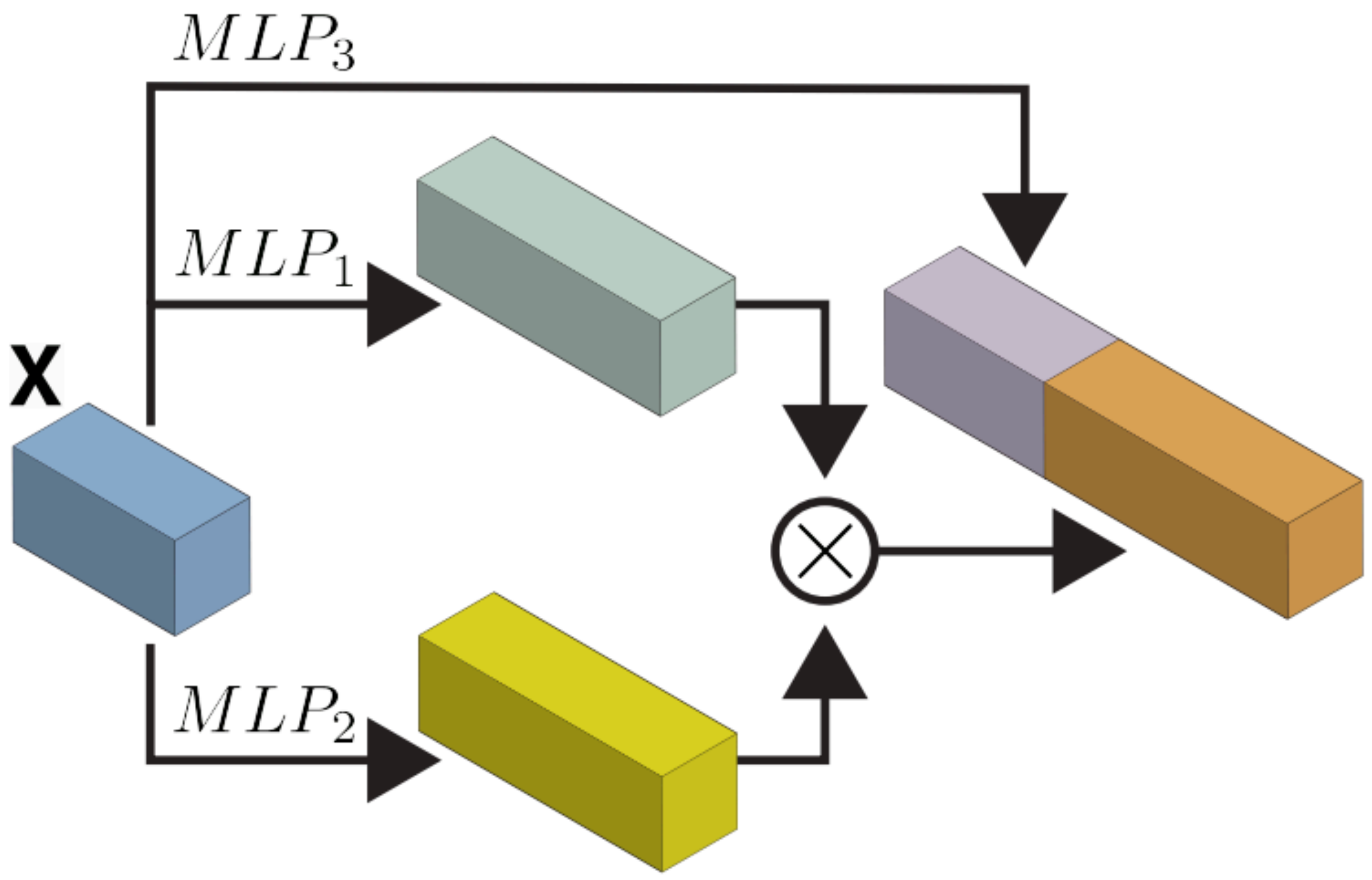


Figure 2: Block structure.

Provably Powerful Graph Networks

Matrix multiplication is equivariant, and thus the PPGN building block is equivariant, which makes the overall PPGN, represented by function F, invariant.

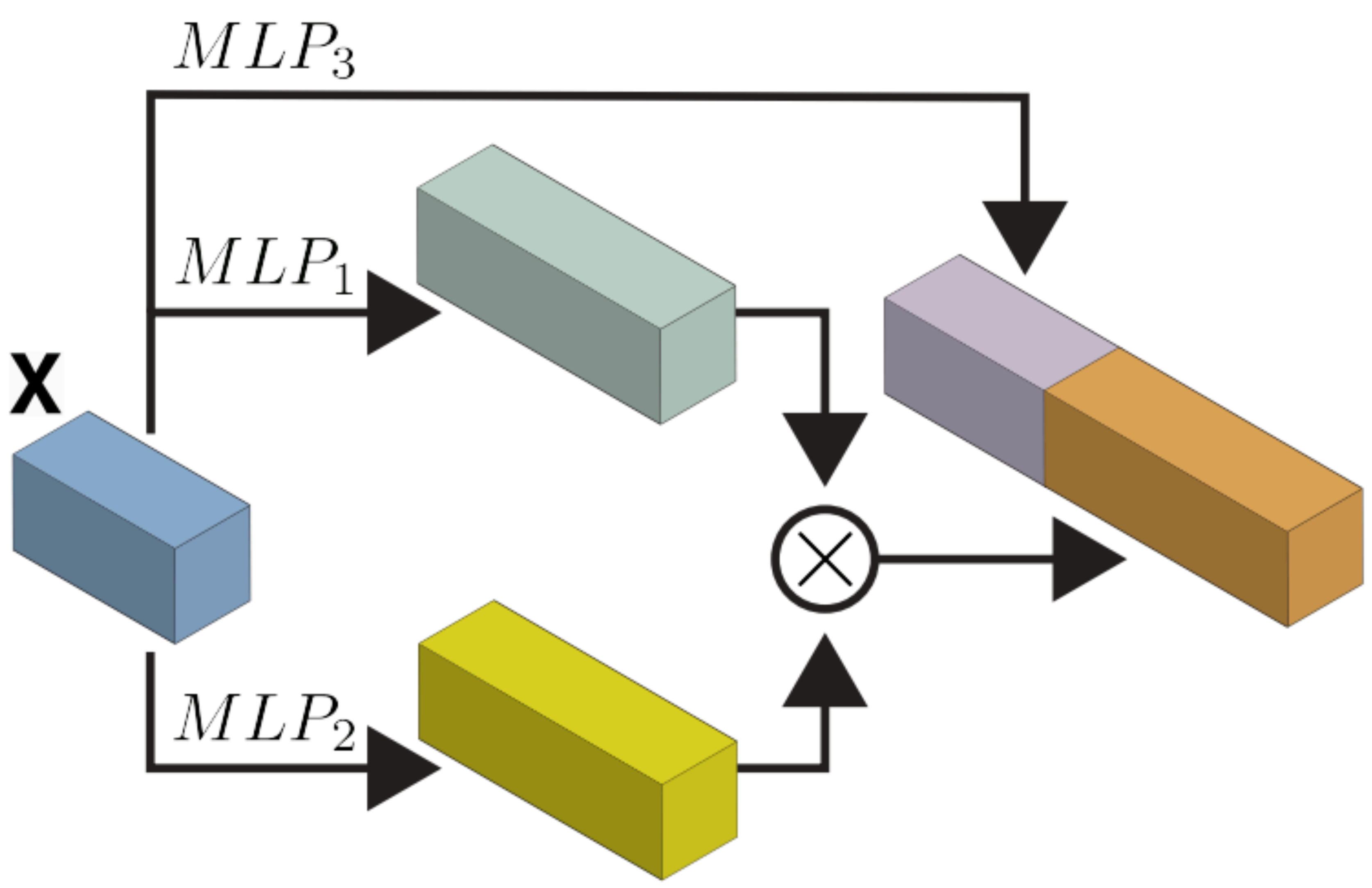


Figure 2: Block structure.

Provably Powerful Graph Networks

Matrix multiplication is equivariant, and thus the PPGN building block is equivariant, which makes the overall PPGN, represented by function F, invariant.

Furthermore, PPGNs are strictly more powerful than MPNNs. In fact, PPGNs can distinguish any pair of graphs that can be distinguished by folklore 2-WL.

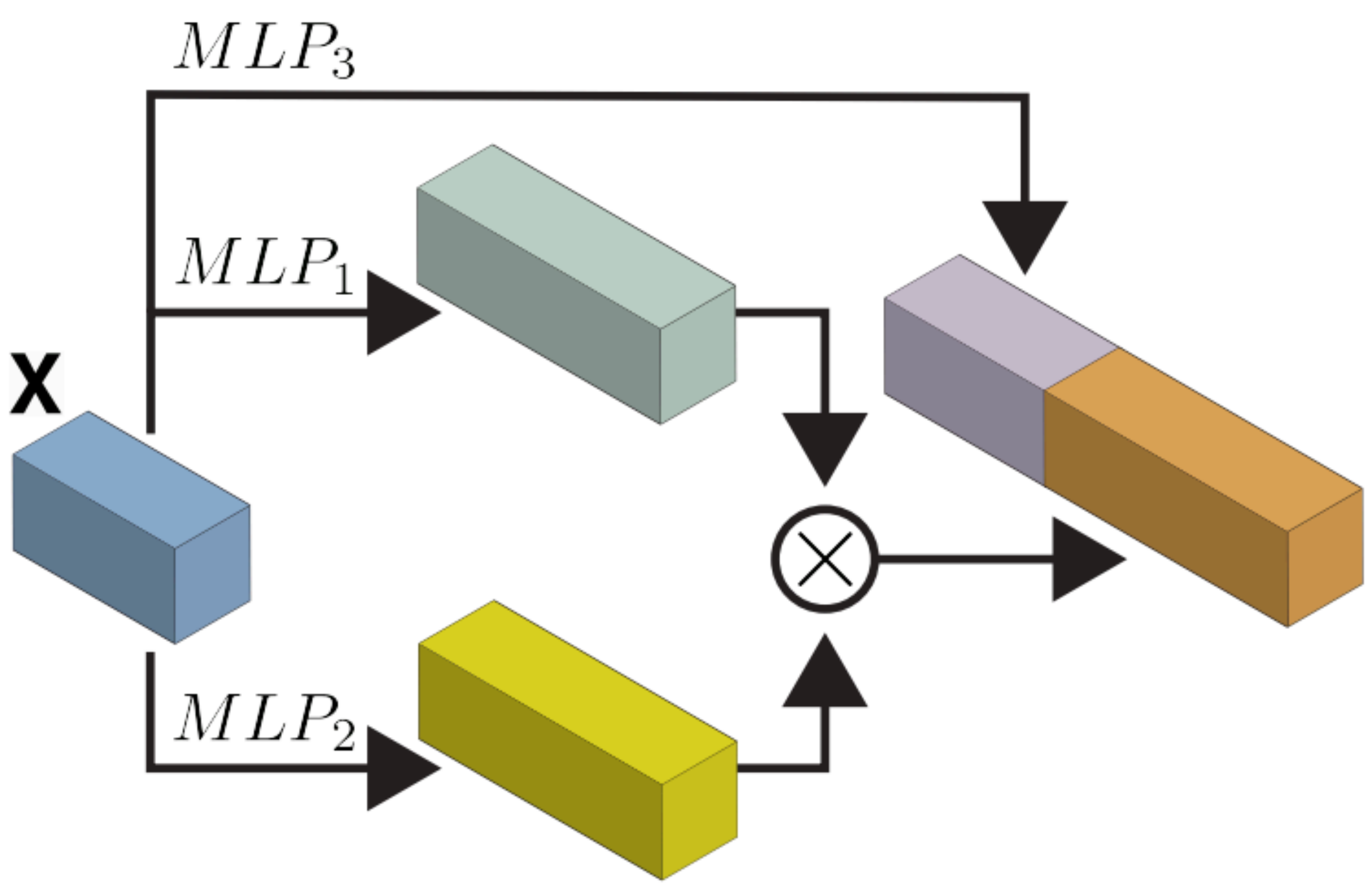


Figure 2: Block structure.

Provably Powerful Graph Networks

Matrix multiplication is equivariant, and thus the PPGN building block is equivariant, which makes the overall PPGN, represented by function F, invariant.

Furthermore, PPGNs are strictly more powerful than MPNNs. In fact, PPGNs can distinguish any pair of graphs that can be distinguished by folklore 2-WL.

Intuitively, the matrix multiplication yields a richer aggregation, which corresponds to 2-WL aggregation.

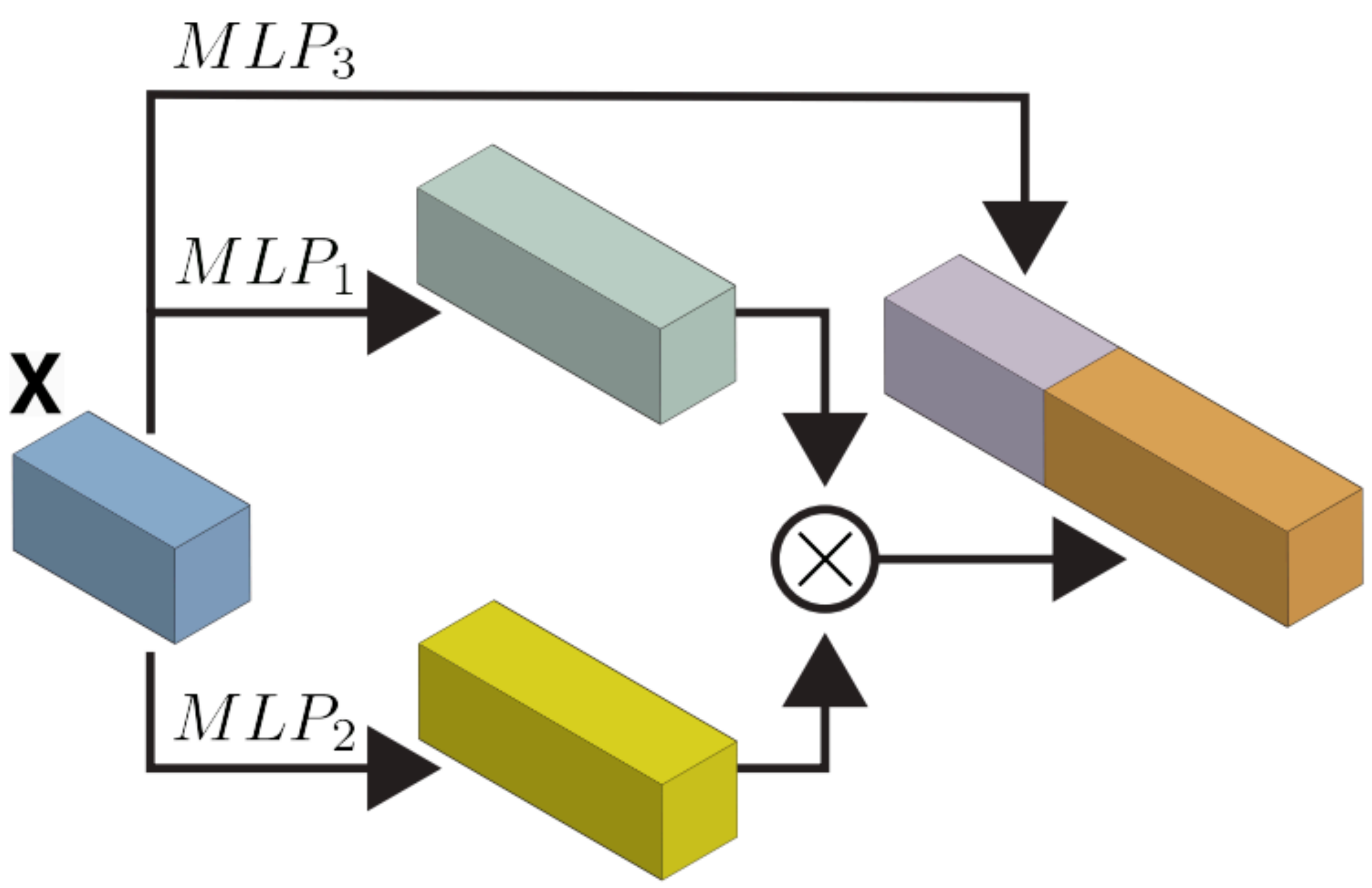


Figure 2: Block structure.

Provably Powerful Graph Networks

Matrix multiplication is equivariant, and thus the PPGN building block is equivariant, which makes the overall PPGN, represented by function F, invariant.

Furthermore, PPGNs are strictly more powerful than MPNNs. In fact, PPGNs can distinguish any pair of graphs that can be distinguished by folklore 2-WL.

Intuitively, the matrix multiplication yields a richer aggregation, which corresponds to 2-WL aggregation.

PPGNs have therefore the same power as 3-GNNs, but the strong point is that they maintain only $O(n^2)$ embeddings, which makes them more memory-efficient than 3-GNNs.

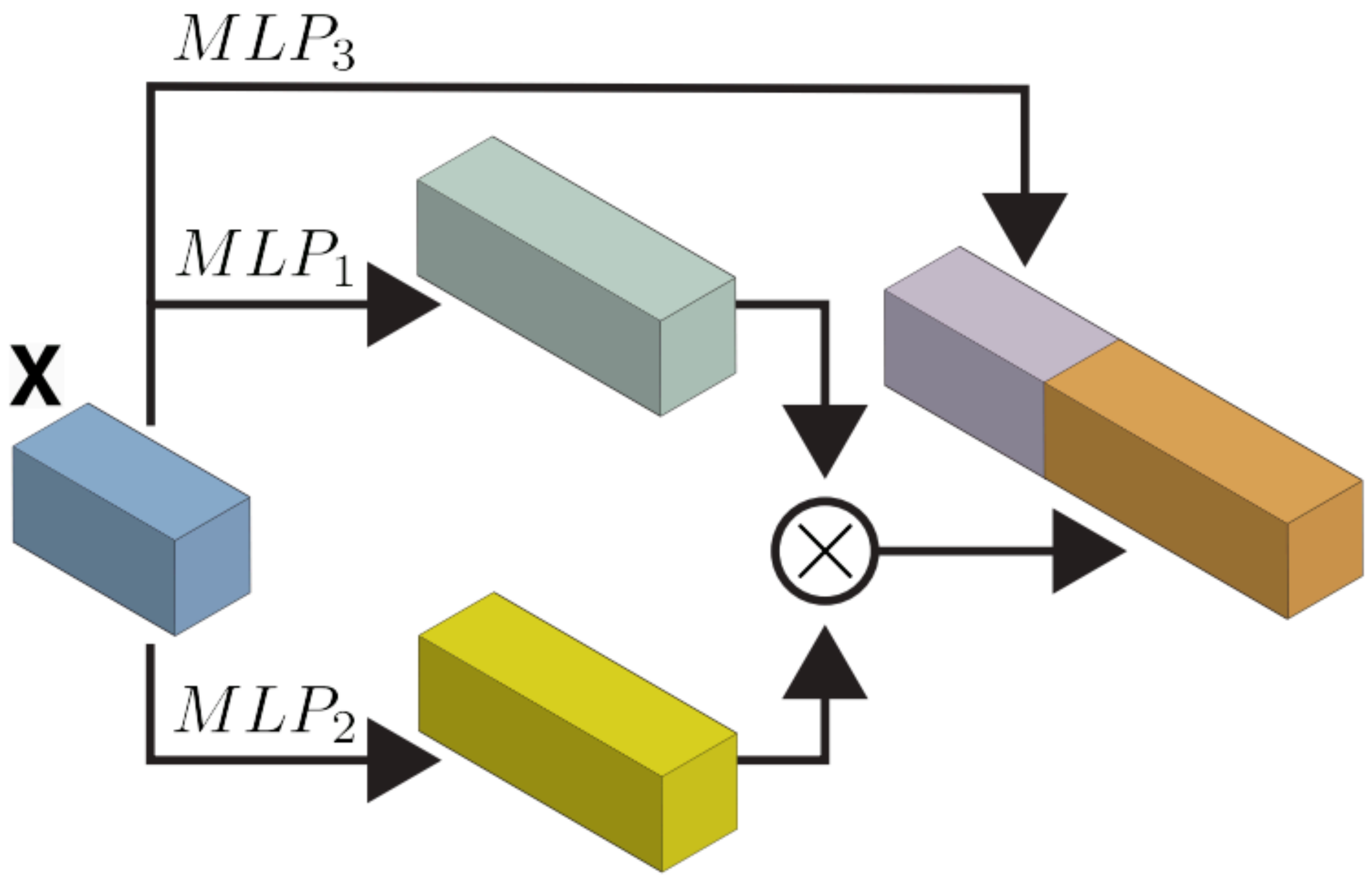


Figure 2: Block structure.

Expressive Power in the Real World

Expressive Power in Real-World Data

Expressive Power in Real-World Data

MPNNs cannot distinguish very basic graph pairs, but we also observe that this limitation is not very pronounced empirically, as modern-day benchmarks are unlikely to include limiting cases.

Expressive Power in Real-World Data

MPNNs cannot distinguish very basic graph pairs, but we also observe that this limitation is not very pronounced empirically, as modern-day benchmarks are unlikely to include limiting cases.

This can intuitively be explained by the following factors:

Expressive Power in Real-World Data

MPNNs cannot distinguish very basic graph pairs, but we also observe that this limitation is not very pronounced empirically, as modern-day benchmarks are unlikely to include limiting cases.

This can intuitively be explained by the following factors:

1. 1-WL edge cases typically correspond to data that is highly regular, whereas real-world data is overwhelmingly uneven and variable, e.g., knowledge graphs, where some entities are connected to hundreds of other entities, and others connect to very few, if any.

Expressive Power in Real-World Data

MPNNs cannot distinguish very basic graph pairs, but we also observe that this limitation is not very pronounced empirically, as modern-day benchmarks are unlikely to include limiting cases.

This can intuitively be explained by the following factors:

1. 1-WL edge cases typically correspond to data that is highly regular, whereas real-world data is overwhelmingly uneven and variable, e.g., knowledge graphs, where some entities are connected to hundreds of other entities, and others connect to very few, if any.

2. Real-world graphs are also typically large, and involve thousands, and potentially millions, of nodes. At this scale, the limitations of 1-WL are less likely to surface, as it is highly probable that some local substructure within the large graph can help distinguish it. In fact, 1-WL can distinguish almost all graphs as the number of graph nodes tends to infinity (Babai et al., 1980), i.e., it can distinguish these graphs with probability almost 1.

Expressive Power in Real-World Data

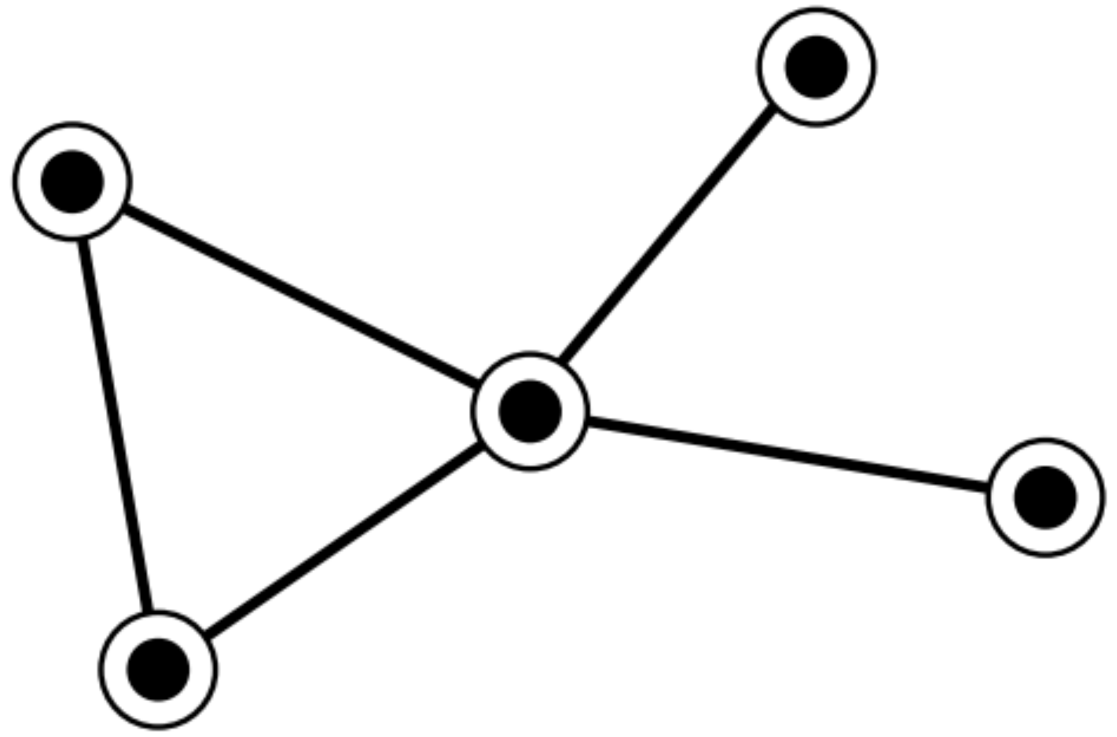


Figure 5 of (Newman, 2013)

Expressive Power in Real-World Data

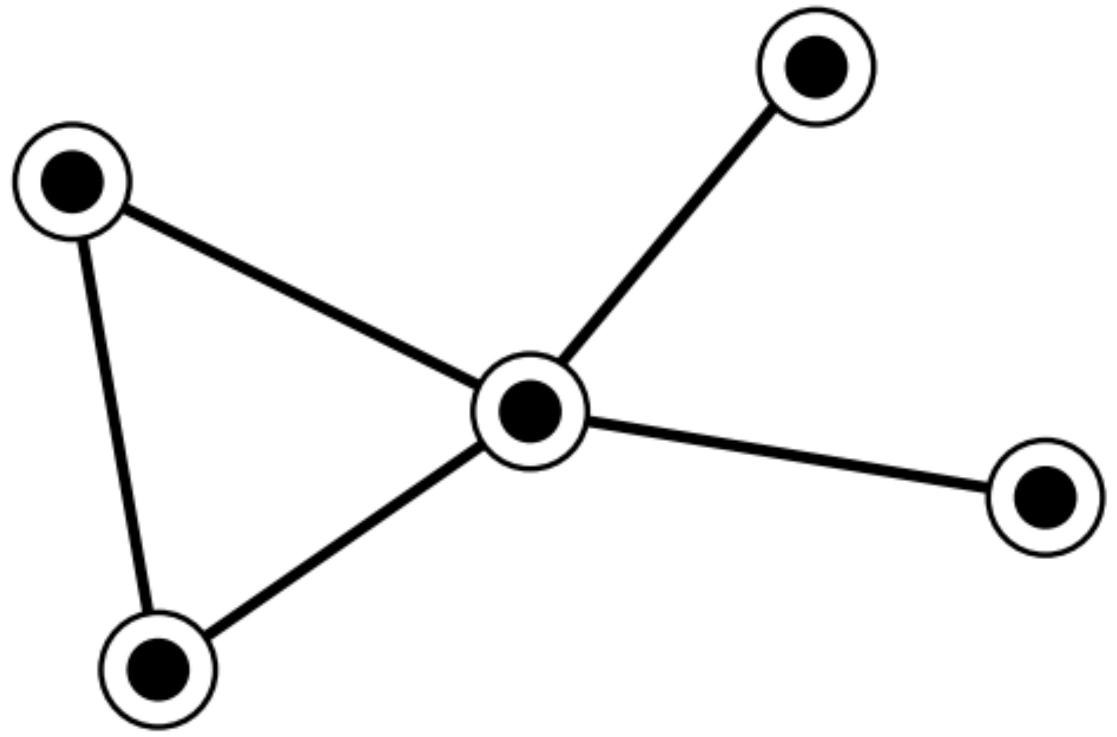


Figure 5 of (Newman, 2013)

Hence, it is hard to quantitatively evaluate the expressiveness of existing models using existing benchmarks — as it is not very likely to hit pairs of indistinguishable graphs.

Expressive Power in Real-World Data

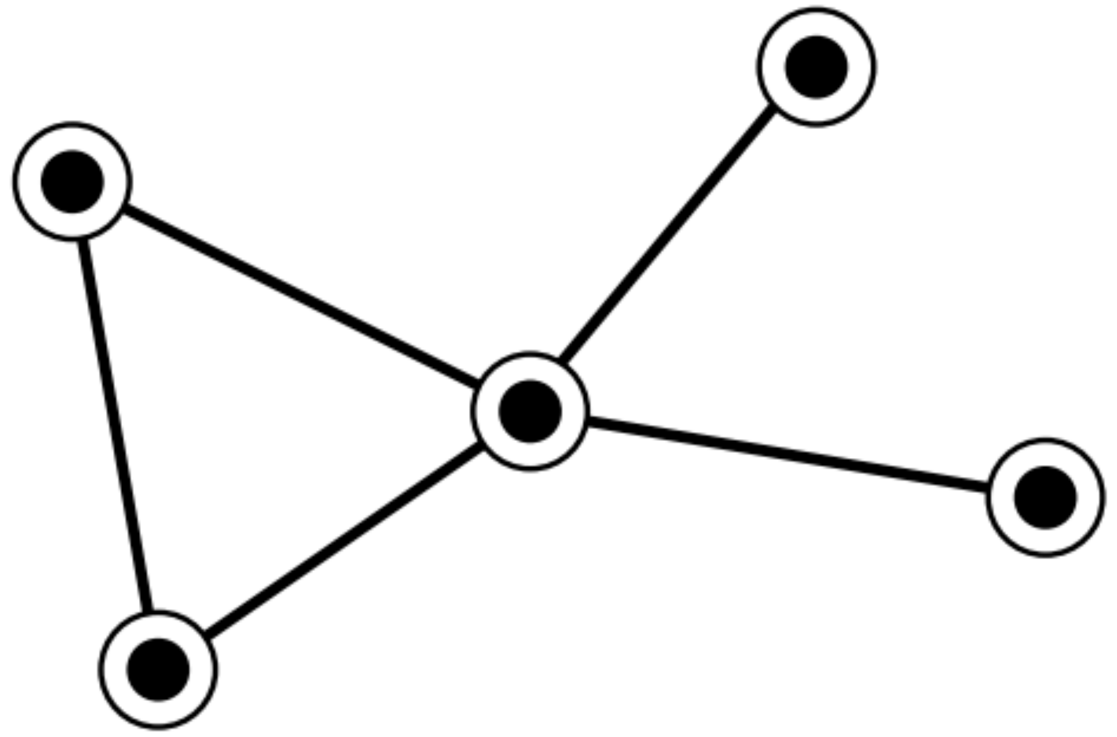


Figure 5 of (Newman, 2013)

Hence, it is hard to quantitatively evaluate the expressiveness of existing models using existing benchmarks — as it is not very likely to hit pairs of indistinguishable graphs.

This does not suggest, however, that lack of expressiveness cannot be an issue in practice.

Expressive Power in Real-World Data

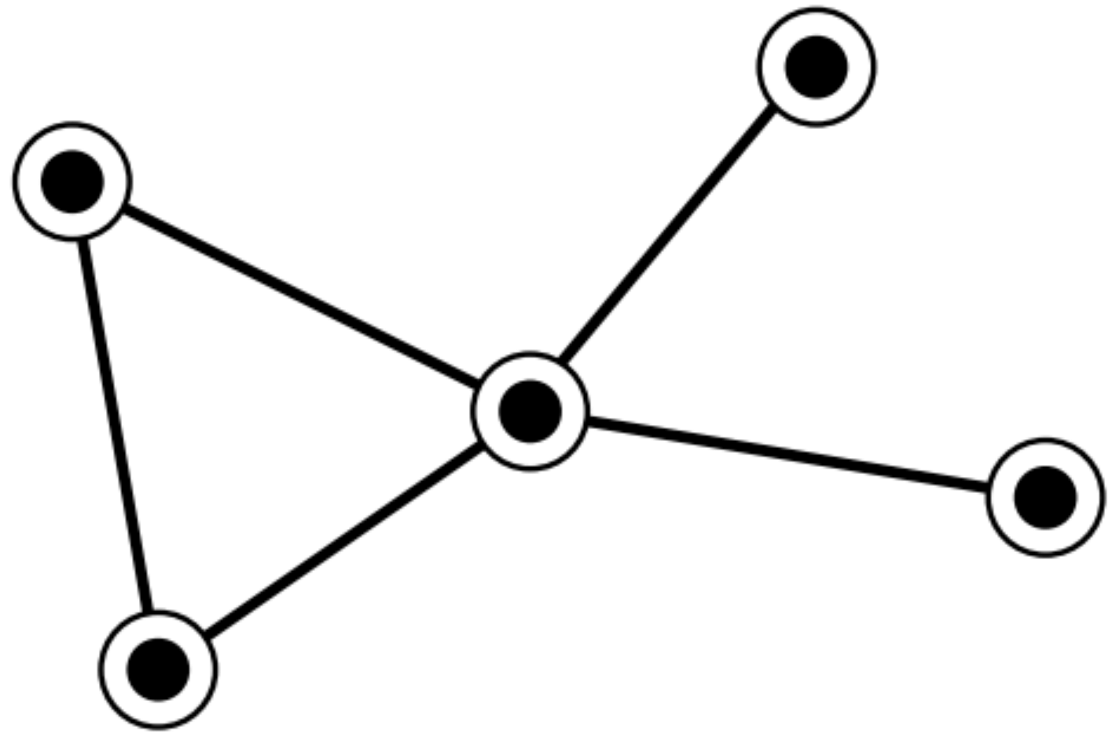


Figure 5 of (Newman, 2013)

Hence, it is hard to quantitatively evaluate the expressiveness of existing models using existing benchmarks — as it is not very likely to hit pairs of indistinguishable graphs.

This does not suggest, however, that lack of expressiveness cannot be an issue in practice.

This has been noted and new synthetic datasets dedicated to quantify the effect of expressive power are proposed (Abboud et al., 2020) with a detailed comparison against higher-order models, as we will see in more detail in Lecture 7.

Expressive Power in Real-World Data

Expressive Power in Real-World Data

There is an excellent survey covering types of graphs observed in real-world data (Newman, 2013):

Expressive Power in Real-World Data

There is an excellent survey covering types of graphs observed in real-world data (Newman, 2013):

“In many networks it is found that if vertex A is connected to vertex B and vertex B to vertex C, then there is a heightened probability that vertex A will also be connected to vertex C. In the language of social networks, the friend of your friend is likely also to be your friend.

Expressive Power in Real-World Data

There is an excellent survey covering types of graphs observed in real-world data (Newman, 2013):

“In many networks it is found that if vertex A is connected to vertex B and vertex B to vertex C, then there is a heightened probability that vertex A will also be connected to vertex C. In the language of social networks, the friend of your friend is likely also to be your friend.

In terms of network topology, transitivity means the presence of a heightened number of triangles in the network — sets of three vertices each of which is connected to each of the others. It can be quantified by defining a clustering coefficient C thus:

Expressive Power in Real-World Data

There is an excellent survey covering types of graphs observed in real-world data (Newman, 2013):

"In many networks it is found that if vertex A is connected to vertex B and vertex B to vertex C, then there is a heightened probability that vertex A will also be connected to vertex C. In the language of social networks, the friend of your friend is likely also to be your friend.

In terms of network topology, transitivity means the presence of a heightened number of triangles in the network — sets of three vertices each of which is connected to each of the others. It can be quantified by defining a clustering coefficient C thus:

$$C = 3 \times \frac{\#\text{triangles in the network}}{\#\text{connected triples of vertices}},$$

Expressive Power in Real-World Data

There is an excellent survey covering types of graphs observed in real-world data (Newman, 2013):

"In many networks it is found that if vertex A is connected to vertex B and vertex B to vertex C, then there is a heightened probability that vertex A will also be connected to vertex C. In the language of social networks, the friend of your friend is likely also to be your friend.

In terms of network topology, transitivity means the presence of a heightened number of triangles in the network — sets of three vertices each of which is connected to each of the others. It can be quantified by defining a clustering coefficient C thus:

$$C = 3 \times \frac{\#\text{triangles in the network}}{\#\text{connected triples of vertices}},$$

where a "connected triple" means a single vertex with edges running to an unordered pair of others."

Expressive Power in Real-World Data

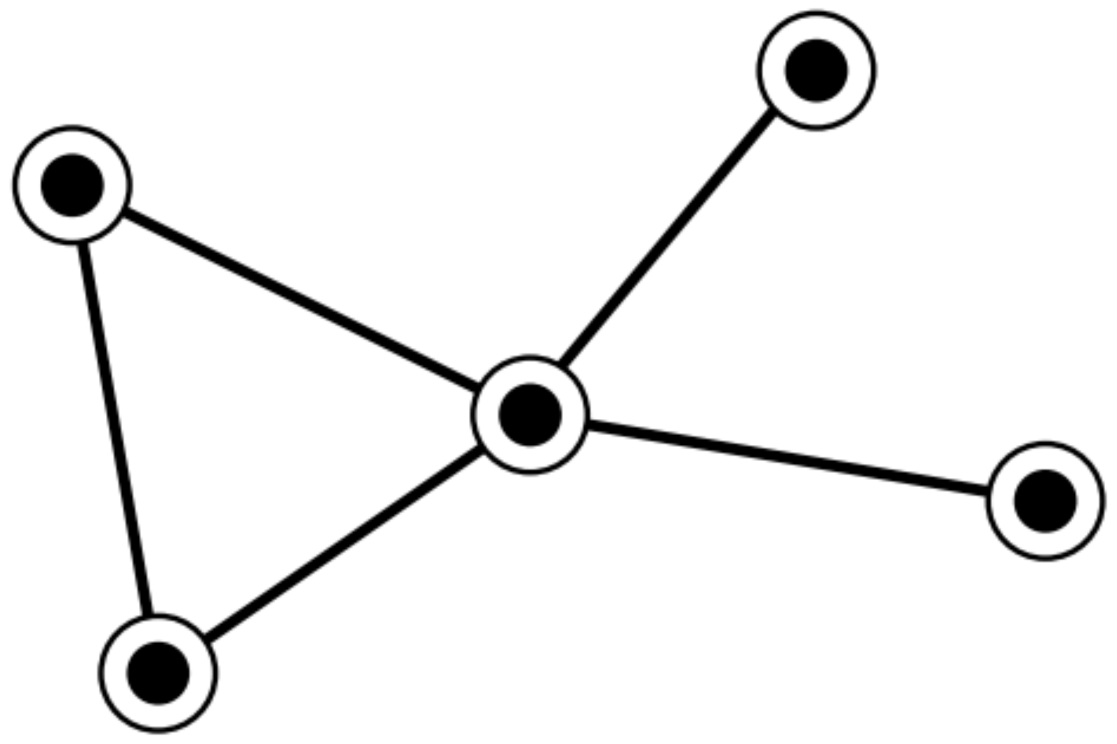


Figure 5 of (Newman, 2013)

Expressive Power in Real-World Data

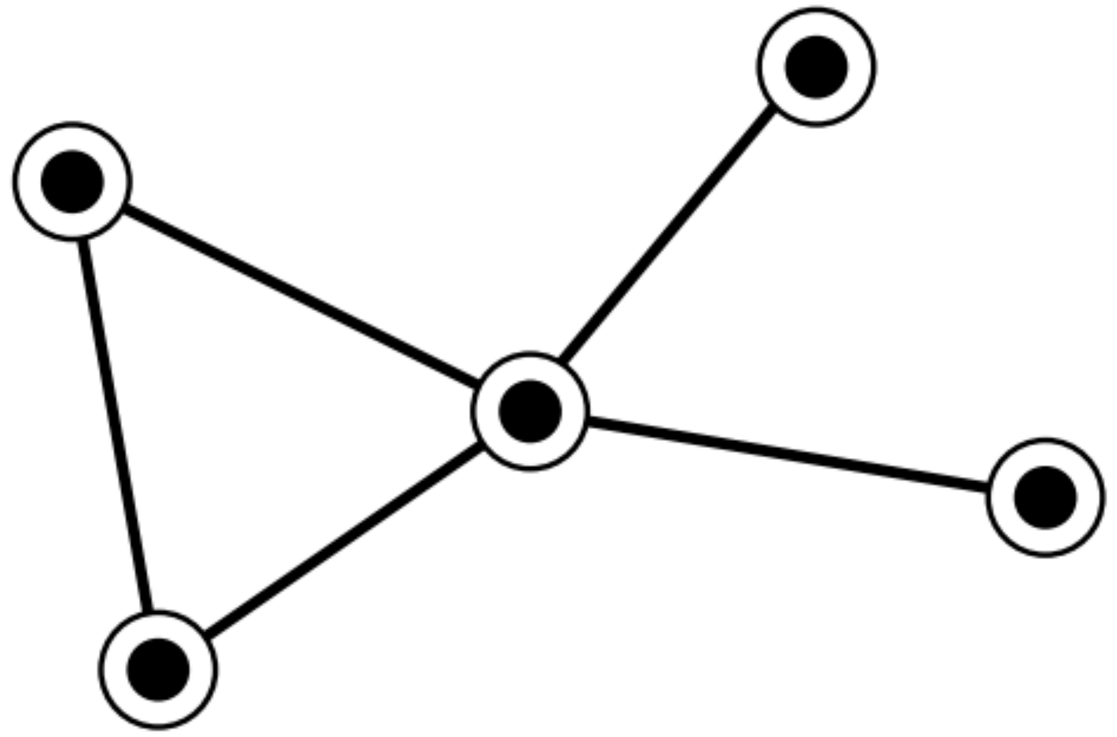


Figure 5 of (Newman, 2013)

"In simple terms, C is the mean probability that two vertices that are network neighbours of the same other vertex will themselves be neighbours." (Newman, 2013)

Expressive Power in Real-World Data

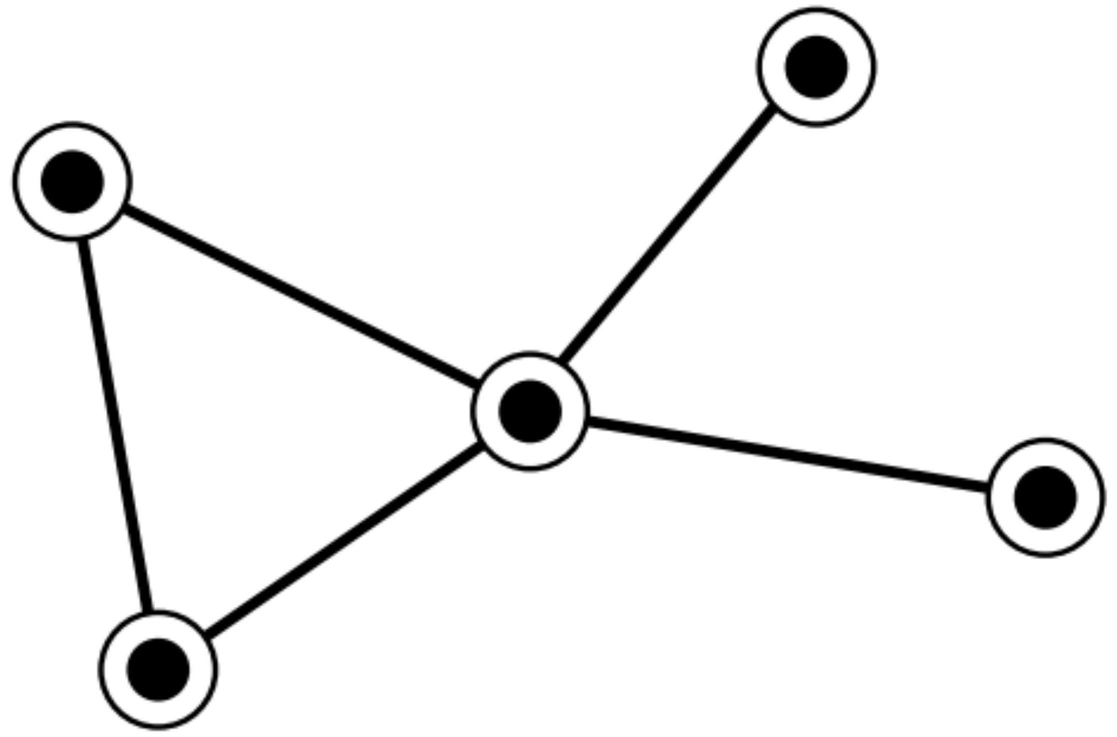


Figure 5 of (Newman, 2013)

"In simple terms, C is the mean probability that two vertices that are network neighbours of the same other vertex will themselves be neighbours." (Newman, 2013)

The graph shown above has 1 triangle and 8 connected triples, and so has a clustering coefficient of 3/8 .

Expressive Power in Real-World Data

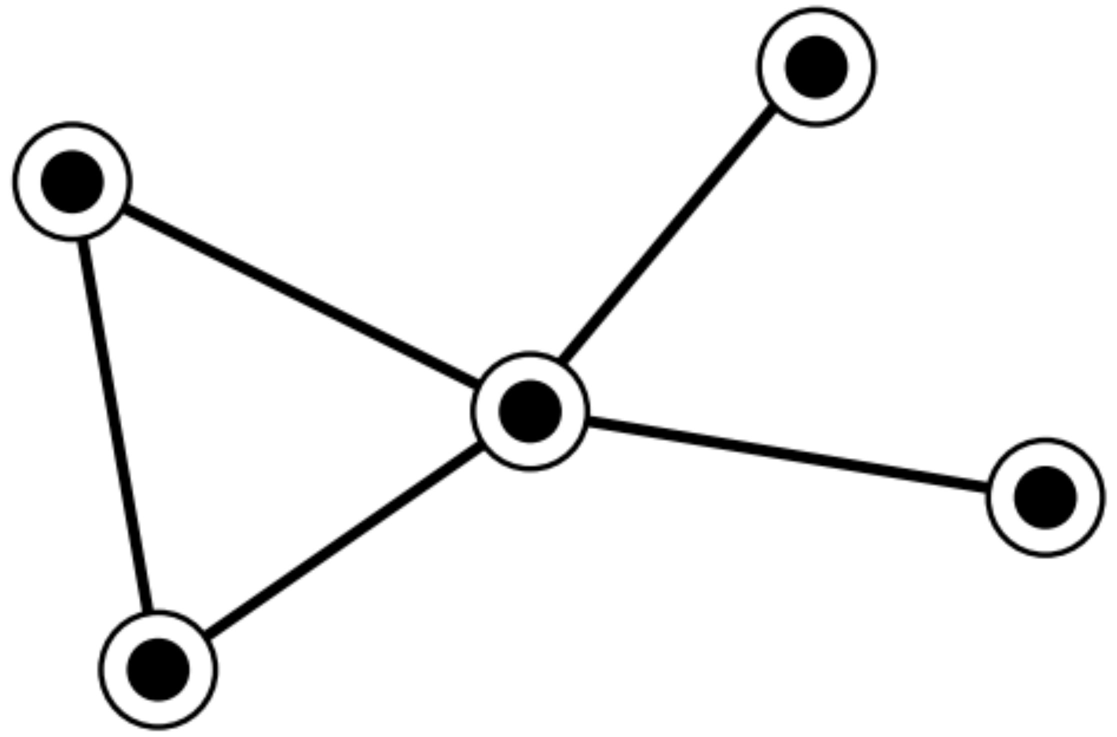


Figure 5 of (Newman, 2013)

"In simple terms, C is the mean probability that two vertices that are network neighbours of the same other vertex will themselves be neighbours." (Newman, 2013)

The graph shown above has 1 triangle and 8 connected triples, and so has a clustering coefficient of 3/8 .

There are other ways of defining cluster coefficient but they rely on being able to detect triangles.

Homophily and Heterophily

Homophily and Heterophily

Homophily and Heterophily

Within a graph, homophily intuitively describes a strong positive correlation between nodes and their neighbours.

Homophily and Heterophily

Within a graph, homophily intuitively describes a strong positive correlation between nodes and their neighbours.

More specifically, homophily implies that a node is highly likely to share features and attributes with its neighbours in the graph.

Homophily and Heterophily

Within a graph, homophily intuitively describes a strong positive correlation between nodes and their neighbours.

More specifically, homophily implies that a node is highly likely to share features and attributes with its neighbours in the graph.

For example, homophily is prominent in citation networks, where connected papers (i.e., papers citing one another) tend to tackle similar research areas.

Homophily and Heterophily

Within a graph, homophily intuitively describes a strong positive correlation between nodes and their neighbours.

More specifically, homophily implies that a node is highly likely to share features and attributes with its neighbours in the graph.

For example, homophily is prominent in citation networks, where connected papers (i.e., papers citing one another) tend to tackle similar research areas.

Conversely, heterophily describes negative correlations between nodes and their neighbours: A node tends to have contrasting features relative to its neighbours.

Homophily and Heterophily

Within a graph, homophily intuitively describes a strong positive correlation between nodes and their neighbours.

More specifically, homophily implies that a node is highly likely to share features and attributes with its neighbours in the graph.

For example, homophily is prominent in citation networks, where connected papers (i.e., papers citing one another) tend to tackle similar research areas.

Conversely, heterophily describes negative correlations between nodes and their neighbours: A node tends to have contrasting features relative to its neighbours.

For instance, protein graphs exhibit heterophily, as the proteins that interact with one another are usually different from a composition perspective.

MPNNs vs Higher-Order Models

MPNNs vs Higher-Order Models

Both homophily and heterophily are data-driven inductive biases. That is, the inductive bias they provide, unlike permutation-invariance and local message passing, does not rely on structural properties of graphs, but on the application domain and the specific input instances.

MPNNs vs Higher-Order Models

Both homophily and heterophily are data-driven inductive biases. That is, the inductive bias they provide, unlike permutation-invariance and local message passing, does not rely on structural properties of graphs, but on the application domain and the specific input instances.

These biases are prominent in real-world applications, and are commonly exploited.

MPNNs vs Higher-Order Models

Both homophily and heterophily are data-driven inductive biases. That is, the inductive bias they provide, unlike permutation-invariance and local message passing, does not rely on structural properties of graphs, but on the application domain and the specific input instances.

These biases are prominent in real-world applications, and are commonly exploited.

MPNNs, by virtue of their structure, are well-suited to model homophily and heterophily.

MPNNs vs Higher-Order Models

Both homophily and heterophily are data-driven inductive biases. That is, the inductive bias they provide, unlike permutation-invariance and local message passing, does not rely on structural properties of graphs, but on the application domain and the specific input instances.

These biases are prominent in real-world applications, and are commonly exploited.

MPNNs, by virtue of their structure, are well-suited to model homophily and heterophily.

Indeed, their computations are restricted to local operations, and neighbour aggregation. Thus, they can more easily capture correlations by simply adjusting combination and aggregation weights.

MPNNs vs Higher-Order Models

Both homophily and heterophily are data-driven inductive biases. That is, the inductive bias they provide, unlike permutation-invariance and local message passing, does not rely on structural properties of graphs, but on the application domain and the specific input instances.

These biases are prominent in real-world applications, and are commonly exploited.

MPNNs, by virtue of their structure, are well-suited to model homophily and heterophily.

Indeed, their computations are restricted to local operations, and neighbour aggregation. Thus, they can more easily capture correlations by simply adjusting combination and aggregation weights.

By contrast, higher-order models are more global, and so cannot naturally be restricted to this setting, unless empowered with some local variants.

MPNNs vs Higher-Order Models

Both homophily and heterophily are data-driven inductive biases. That is, the inductive bias they provide, unlike permutation-invariance and local message passing, does not rely on structural properties of graphs, but on the application domain and the specific input instances.

These biases are prominent in real-world applications, and are commonly exploited.

MPNNs, by virtue of their structure, are well-suited to model homophily and heterophily.

Indeed, their computations are restricted to local operations, and neighbour aggregation. Thus, they can more easily capture correlations by simply adjusting combination and aggregation weights.

By contrast, higher-order models are more global, and so cannot naturally be restricted to this setting, unless empowered with some local variants.

For example, k-GNN, for larger k, would require non-uniform handling of its connected tuples, based on local neighbourhoods, and k-IGN processes all nodes simultaneously, and thus must learn to filter out non-local features!

Summary

Summary

- The WL hierarchy and its relevance to GNNs

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs, hierarchical variants, limitations

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs, hierarchical variants, limitations
  - Invariant/Equivariant graph networks: universality, limitations

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs, hierarchical variants, limitations
  - Invariant/Equivariant graph networks: universality, limitations
  - Provably powerful graph neural networks: expressive power, scalability

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs, hierarchical variants, limitations
  - Invariant/Equivariant graph networks: universality, limitations
  - Provably powerful graph neural networks: expressive power, scalability
- Expressive power may not surface in existing benchmarks, but it still is relevant!

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs, hierarchical variants, limitations
  - Invariant/Equivariant graph networks: universality, limitations
  - Provably powerful graph neural networks: expressive power, scalability
- Expressive power may not surface in existing benchmarks, but it still is relevant!
- Homophily and heterophily: MPNNs vs higher-order models

Summary

- The WL hierarchy and its relevance to GNNs
- Higher-order graph neural networks
  - Higher-order message passing neural networks: k-GNNs, hierarchical variants, limitations
  - Invariant/Equivariant graph networks: universality, limitations
  - Provably powerful graph neural networks: expressive power, scalability
- Expressive power may not surface in existing benchmarks, but it still is relevant!
- Homophily and heterophily: MPNNs vs higher-order models
- There are other extensions of MPNNs, particularly with random features, yielding more expressive power without the need for higher-order tensors — Lecture 7.

References

- H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *NeurIPS*, 2019a.
- H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. *ICLR*, 2019b.
- H. Maron, E. Fetaya, N. Segol, and Y. Lipman. On the universality of invariant networks. *ICML*, 2019c.
- N. Keriven and G. Peyré, (2019). Universal invariant and equivariant graph neural 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.
- G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- Martin Grohe. Descriptive Complexity, Canonisation, and Definable Graph Structure Theory. *Cambridge University Press,* 2017.
- Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 2003.
- Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, Thomas Lukasiewicz, The Surprising Power of Graph Neural Networks with Random Node Initialization, arXiv:2010.01179, 2020