Lecture 5: Expressive Power of Message Passing Neural Networks

Relational Learning

İsmail İlkan Ceylan

Advanced Topics in Machine Learning, University of Oxford

01.02.2021

Overview

Overview

- A journey into model representation capacity

Overview

- A journey into model representation capacity
- Graph isomorphism and colour refinement

Overview

- A journey into model representation capacity
- Graph isomorphism and colour refinement
- Expressive power of MPNNs

Overview

- A journey into model representation capacity
- Graph isomorphism and colour refinement
- Expressive power of MPNNs
- The logic of graphs

Overview

- A journey into model representation capacity
- Graph isomorphism and colour refinement
- Expressive power of MPNNs
- The logic of graphs
- Logical characterisation of MPNNs

Overview

- A journey into model representation capacity
- Graph isomorphism and colour refinement
- Expressive power of MPNNs
- The logic of graphs
- Logical characterisation of MPNNs
- Summary

A Journey into Model Representation Capacity

Model Representation Capacity

Model Representation Capacity

Informally, the expressive power, or the representation power, of a neural network describes its ability to approximate functions. Expressiveness results come with many flavours and assumptions, but the question is always the same: What class of functions can a neural network approximately represent?

Model Representation Capacity

Informally, the expressive power, or the representation power, of a neural network describes its ability to approximate functions. Expressiveness results come with many flavours and assumptions, but the question is always the same: What class of functions can a neural network approximately represent?

The celebrated universal approximation theorem states that an autoencoder network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact domain to any desired accuracy, under mild assumptions on the activation function.

Model Representation Capacity

Informally, the expressive power, or the representation power, of a neural network describes its ability to approximate functions. Expressiveness results come with many flavours and assumptions, but the question is always the same: What class of functions can a neural network approximately represent?

The celebrated universal approximation theorem states that an autoencoder network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact domain to any desired accuracy, under mild assumptions on the activation function.

(Cybenko, 1989) proved that a fully connected sigmoid neural network with one single hidden layer can universally approximate any continuous function on a bounded domain with arbitrarily small error; see also, e.g., (Hornik et al., 1989; Funahashi, 1989).

Model Representation Capacity

Informally, the expressive power, or the representation power, of a neural network describes its ability to approximate functions. Expressiveness results come with many flavours and assumptions, but the question is always the same: What class of functions can a neural network approximately represent?

The celebrated universal approximation theorem states that an autoencoder network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact domain to any desired accuracy, under mild assumptions on the activation function.

(Cybenko, 1989) proved that a fully connected sigmoid neural network with one single hidden layer can universally approximate any continuous function on a bounded domain with arbitrarily small error; see also, e.g., (Hornik et al., 1989; Funahashi, 1989).

In particular, MLPs can approximate any continuous function on a compact domain, i.e., for any such function, there is a parameter configuration for an MLP, corresponding to an approximation of the function.

Model Representation Capacity

Informally, the expressive power, or the representation power, of a neural network describes its ability to approximate functions. Expressiveness results come with many flavours and assumptions, but the question is always the same: What class of functions can a neural network approximately represent?

The celebrated universal approximation theorem states that an autoencoder network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact domain to any desired accuracy, under mild assumptions on the activation function.

(Cybenko, 1989) proved that a fully connected sigmoid neural network with one single hidden layer can universally approximate any continuous function on a bounded domain with arbitrarily small error; see also, e.g., (Hornik et al., 1989; Funahashi, 1989).

In particular, MLPs can approximate any continuous function on a compact domain, i.e., for any such function, there is a parameter configuration for an MLP, corresponding to an approximation of the function.

From a learning perspective, universal approximation is only the first step — it does not imply that the functions can be learned efficiently (e.g., we might need exponentially many neurons etc)!

Representations in The World of Graphs

Representations in The World of Graphs

How can we characterise expressive power in the world of graphs?

Representations in The World of Graphs

How can we characterise expressive power in the world of graphs?

Suppose we are interested in functions $f : \mathscr{G} \mapsto \mathbb{R}^{V_G}$.

Representations in The World of Graphs

How can we characterise expressive power in the world of graphs?

Suppose we are interested in functions $f : \mathcal{G} \mapsto \mathbb{R}^{V_G}$.

One way of characterising the expressive power would be through graph distinguishability.

Representations in The World of Graphs

How can we characterise expressive power in the world of graphs?

Suppose we are interested in functions $f : \mathcal{G} \mapsto \mathbb{R}^{V_G}$.

One way of characterising the expressive power would be through graph distinguishability.

In this case, we want to learn graph embeddings $\mathbf{z}_G$, $\mathbf{z}_H$ for graphs G and H such that

Representations in The World of Graphs

How can we characterise expressive power in the world of graphs?

Suppose we are interested in functions $f : \mathcal{G} \mapsto \mathbb{R}^{V_G}$.

One way of characterising the expressive power would be through graph distinguishability.

In this case, we want to learn graph embeddings $\mathbf{z}_G$, $\mathbf{z}_H$ for graphs G and H such that

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

Representations in The World of Graphs

How can we characterise expressive power in the world of graphs?

Suppose we are interested in functions $f : \mathcal{G} \mapsto \mathbb{R}^{V_G}$.

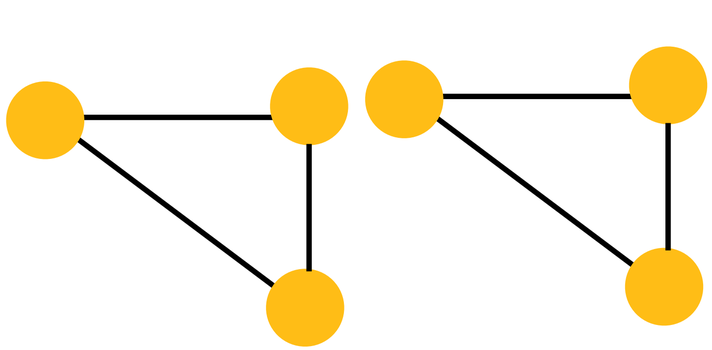
One way of characterising the expressive power would be through graph distinguishability.

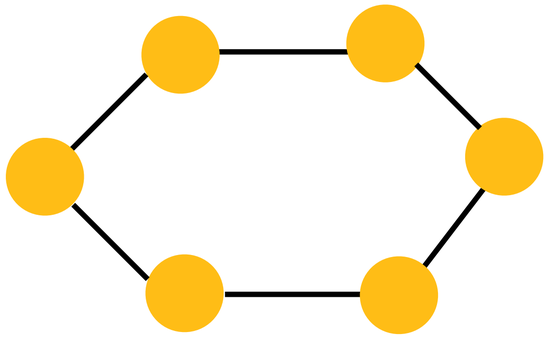
In this case, we want to learn graph embeddings $\mathbf{z}_G$, $\mathbf{z}_H$ for graphs G and H such that

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

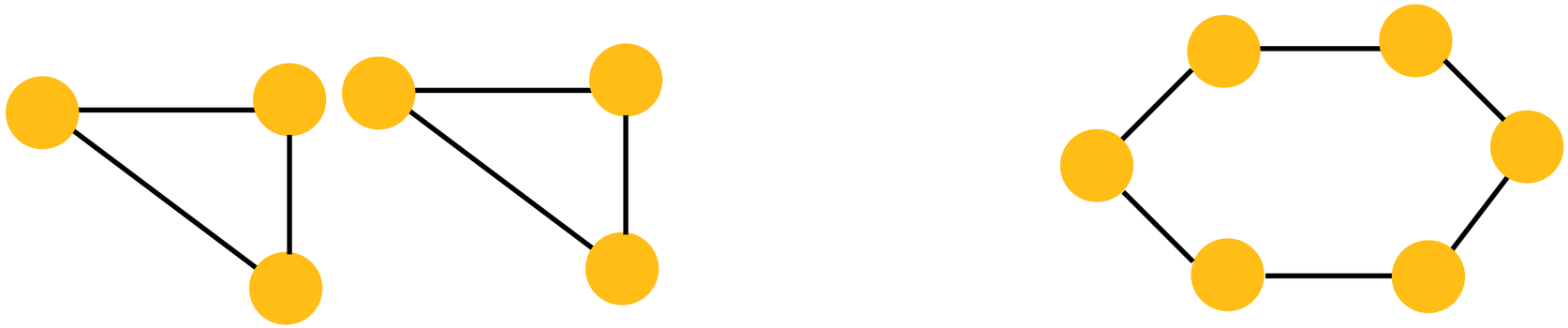
This would be desirable, as it implies we can distinguish all structures, and this paves the way for learning more general classes of functions over graphs.

A Tale of Two Graphs



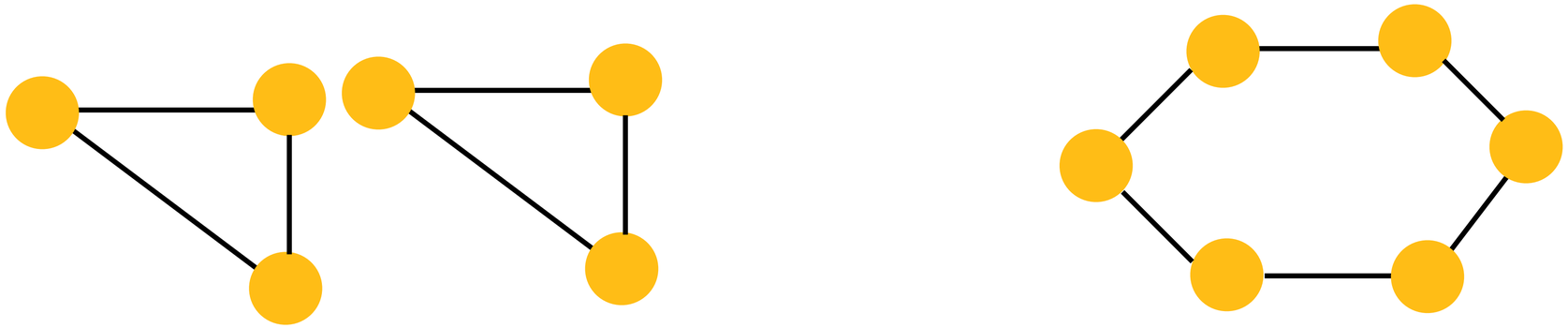


A Tale of Two Graphs



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!

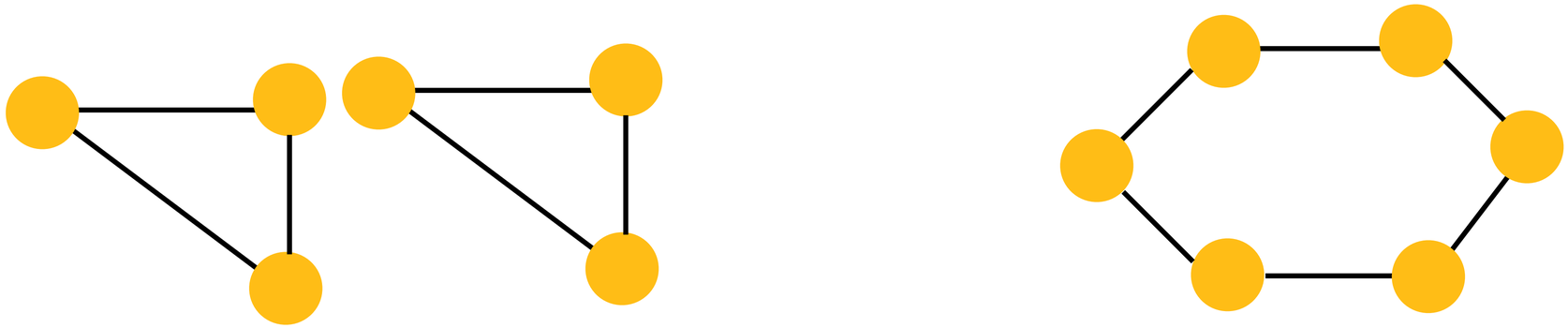
A Tale of Two Graphs



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!

MPNNs cannot distinguish between two triangles and a 6-cycle — severe limitation for graph classification, as the predictions for these graphs will be identical regardless of the function we are trying to learn!

A Tale of Two Graphs

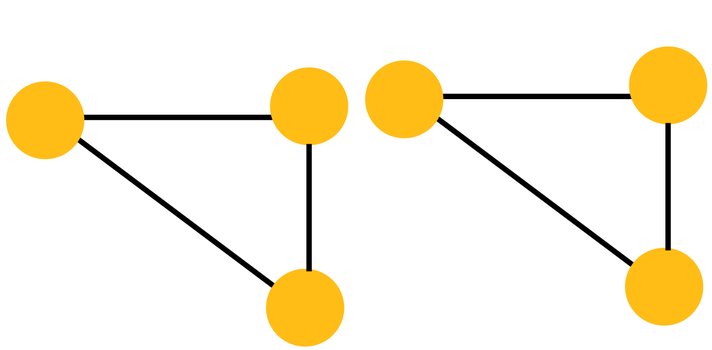


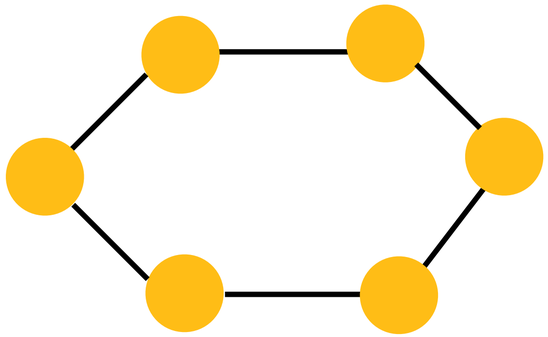
Problem: The embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side for MPNNs!

MPNNs cannot distinguish between two triangles and a 6-cycle — severe limitation for graph classification, as the predictions for these graphs will be identical regardless of the function we are trying to learn!

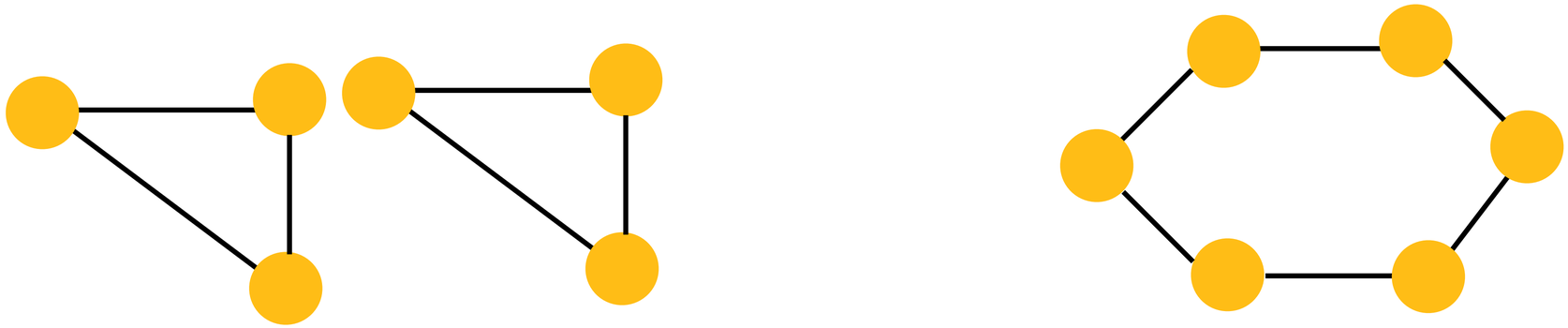
Is this only a problem for graph classification?

A Tale of Two Graphs



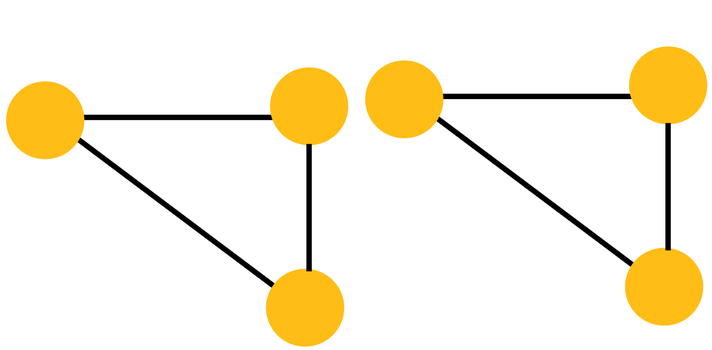


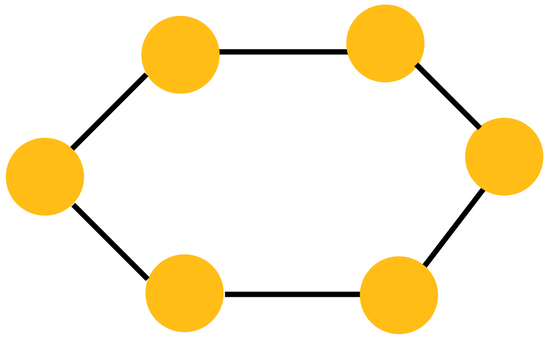
A Tale of Two Graphs



Consider a synthetic node classification task: Let's say that a node is a separator node, if it that has two neighbours that are non-adjacent, and we want to predict whether a node is a separator node or a non-separator node on the union of the graphs shown above.

A Tale of Two Graphs

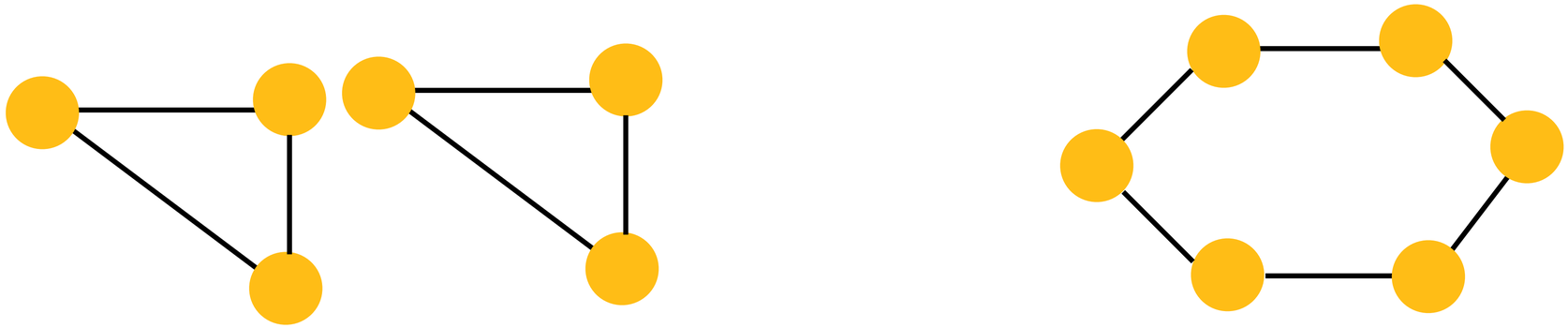




Consider a synthetic node classification task: Let's say that a node is a separator node, if it that has two neighbours that are non-adjacent, and we want to predict whether a node is a separator node or a non-separator node on the union of the graphs shown above.

It is easy to see that all nodes in the 6-cycle are separator nodes, and all nodes in the triangles are non-separator nodes.

A Tale of Two Graphs



Consider a synthetic node classification task: Let's say that a node is a separator node, if it that has two neighbours that are non-adjacent, and we want to predict whether a node is a separator node or a non-separator node on the union of the graphs shown above.

It is easy to see that all nodes in the 6-cycle are separator nodes, and all nodes in the triangles are non-separator nodes.

An MPNN will either predict all nodes to be separator nodes, or all of them as non-separator nodes, regardless of training choices etc — either case, a random answer with exactly $50\,\%$ accuracy.

Finding the Culprits

Finding the Culprits

Recall that we can define an embedding of a graph G as a multi-layer perceptron:

$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

Finding the Culprits

Recall that we can define an embedding of a graph G as a multi-layer perceptron:

$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

We did not prefer embedding graphs to an MLP — not permutation-invariant.

Finding the Culprits

Recall that we can define an embedding of a graph G as a multi-layer perceptron:

$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

We did not prefer embedding graphs to an MLP — not permutation-invariant.

MPNNs are superior on graph-tasks — strong inductive bias.

Finding the Culprits

Recall that we can define an embedding of a graph G as a multi-layer perceptron:

$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

We did not prefer embedding graphs to an MLP — not permutation-invariant.

MPNNs are superior on graph-tasks — strong inductive bias.

But we lost something — MLPs are universal and MPNNs are not!

Finding the Culprits

Recall that we can define an embedding of a graph G as a multi-layer perceptron:

$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

We did not prefer embedding graphs to an MLP — not permutation-invariant.

MPNNs are superior on graph-tasks — strong inductive bias.

But we lost something — MLPs are universal and MPNNs are not!

A step forward in terms of inductive bias, but a step backwards in terms in representation capacity!

Finding the Culprits

Recall that we can define an embedding of a graph G as a multi-layer perceptron:

$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

We did not prefer embedding graphs to an MLP — not permutation-invariant.

MPNNs are superior on graph-tasks — strong inductive bias.

But we lost something — MLPs are universal and MPNNs are not!

A step forward in terms of inductive bias, but a step backwards in terms in representation capacity!

This is a trade-off: We want to constrain the learning space (e.g., incorporating inductive bias) as much as possible, but not so much that we induce strong limitations in the representation capacity.

Graph Isomorphism and Color Refinement

Graph Isomorphism

Graph Isomorphism

Graph isomorphism testing is one of the most fundamental tasks in graph theory:

We say that two graphs G and H are isomorphic if there is a bijection between the vertex sets V_G and V_H:

$$f : V_G \mapsto V_H$$

such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ *and* $f(v)$ are adjacent in H.

Graph isomorphism is then the problem of deciding whether a given pair of graphs are isomorphic.

Graph Isomorphism

Graph isomorphism testing is one of the most fundamental tasks in graph theory:

We say that two graphs G and H are isomorphic if there is a bijection between the vertex sets V_G and V_H:

$$f : V_G \mapsto V_H$$

such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ *and* $f(v)$ are adjacent in H.

Graph isomorphism is then the problem of deciding whether a given pair of graphs are isomorphic.

We can restate this with features and using matrices:

If we have two graphs G and H, represented with adjacency matrices $\mathbf{A}^G$and $\mathbf{A}^H$, as well as associated with node features $\mathbf{X}^G$and $\mathbf{X}^H$, we say that two graphs are isomorphic if and only if there exists a permutation matrix $\mathbf{P}$ such that:

$$\mathbf{P}\mathbf{A}^G\mathbf{P}^\top = \mathbf{A}^H \text{ and } \mathbf{P}\mathbf{X}^G = \mathbf{X}^H.$$

Graph Isomorphism

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph isomorphism is widely suspected not to be NP-hard, but there is no known polynomial time algorithm for the problem.

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph isomorphism is widely suspected not to be NP-hard, but there is no known polynomial time algorithm for the problem.

A major breakthrough in theoretical computer science — (Babai, 2016) presented a quasi-polynomial time algorithm for GI!

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph isomorphism is widely suspected not to be NP-hard, but there is no known polynomial time algorithm for the problem.

A major breakthrough in theoretical computer science — (Babai, 2016) presented a quasi-polynomial time algorithm for GI!

Graph isomorphism is therefore usually referred as an NP-intermediate problem, and is arguably the most natural NP-intermediate problem.

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph isomorphism is widely suspected not to be NP-hard, but there is no known polynomial time algorithm for the problem.

A major breakthrough in theoretical computer science — (Babai, 2016) presented a quasi-polynomial time algorithm for GI!

Graph isomorphism is therefore usually referred as an NP-intermediate problem, and is arguably the most natural NP-intermediate problem.

Graph isomorphism is also known for its own complexity class GI — you may come across other problems which are also GI-complete.

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph isomorphism is widely suspected not to be NP-hard, but there is no known polynomial time algorithm for the problem.

A major breakthrough in theoretical computer science — (Babai, 2016) presented a quasi-polynomial time algorithm for GI!

Graph isomorphism is therefore usually referred as an NP-intermediate problem, and is arguably the most natural NP-intermediate problem.

Graph isomorphism is also known for its own complexity class GI — you may come across other problems which are also GI-complete.

Exact graph isomorphism testing is (unsurprisingly) beyond MPNNs.

Graph Isomorphism

Graph isomorphism has attracted a lot of attention in theoretical computer science, as determining the exact computational complexity of graph isomorphism is a long-standing open problem.

Graph isomorphism is widely suspected not to be NP-hard, but there is no known polynomial time algorithm for the problem.

A major breakthrough in theoretical computer science — (Babai, 2016) presented a quasi-polynomial time algorithm for GI!

Graph isomorphism is therefore usually referred as an NP-intermediate problem, and is arguably the most natural NP-intermediate problem.

Graph isomorphism is also known for its own complexity class GI — you may come across other problems which are also GI-complete.

Exact graph isomorphism testing is (unsurprisingly) beyond MPNNs.

There are, however, many practical — approximate — algorithms for graph isomorphism testing that work on broad classes of graphs, including colour refinement.

Graph Isomorphism Testing

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

1. **Initialisation**: All vertices in a graph are initialised to their initial colours.

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

1. **Initialisation**: All vertices in a graph are initialised to their initial colours.
2. **Refinement**: All vertices are recoloured depending on their current colour and the colours in their neighbourhoods.

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

1. **Initialisation**: All vertices in a graph are initialised to their initial colours.
2. **Refinement**: All vertices are recoloured depending on their current colour and the colours in their neighbourhoods.
3. **Stop**: Terminate when the colouring stabilises.

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

1. **Initialisation**: All vertices in a graph are initialised to their initial colours.
2. **Refinement**: All vertices are recoloured depending on their current colour and the colours in their neighbourhoods.
3. **Stop**: Terminate when the colouring stabilises.

Formally, for a graph $G = (V, E)$, we say that a function $\lambda : V_G \mapsto \mathbf{C}$ colours each vertex of the graph with a colour from a set $\mathbf{C}$ of colours.

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

1. **Initialisation**: All vertices in a graph are initialised to their initial colours.
2. **Refinement**: All vertices are recoloured depending on their current colour and the colours in their neighbourhoods.
3. **Stop**: Terminate when the colouring stabilises.

Formally, for a graph $G = (V, E)$, we say that a function $\lambda : V_G \mapsto \mathbf{C}$ colours each vertex of the graph with a colour from a set $\mathbf{C}$ of colours.

Every graph colouring λ induces a partition $\pi(\lambda)$ of V_G into vertex colour classes. For two partitions $\pi(\lambda)$ and $\pi(\lambda')$ of a graph G, we say that $\pi(\lambda)$ refines $\pi(\lambda')$, denoted $\pi(\lambda) \preceq \pi(\lambda')$, if every element of $\pi(\lambda)$ is a (not necessarily proper) subset of an element of $\pi(\lambda')$.

Graph Isomorphism Testing

Colour refinement is a very simple and effective algorithm for graph isomorphism testing. Given a graph as input, colour refinement is the following procedure:

1. **Initialisation**: All vertices in a graph are initialised to their initial colours.
2. **Refinement**: All vertices are recoloured depending on their current colour and the colours in their neighbourhoods.
3. **Stop**: Terminate when the colouring stabilises.

Formally, for a graph $G = (V, E)$, we say that a function $\lambda : V_G \mapsto \mathbf{C}$ colours each vertex of the graph with a colour from a set $\mathbf{C}$ of colours.

Every graph colouring λ induces a partition $\pi(\lambda)$ of V_G into vertex colour classes. For two partitions $\pi(\lambda)$ and $\pi(\lambda')$ of a graph G, we say that $\pi(\lambda)$ refines $\pi(\lambda')$, denoted $\pi(\lambda) \preceq \pi(\lambda')$, if every element of $\pi(\lambda)$ is a (not necessarily proper) subset of an element of $\pi(\lambda')$.

We write $\pi(\lambda) \equiv \pi(\lambda')$ if $\pi(\lambda) \preceq \pi(\lambda')$ and $\pi(\lambda') \preceq \pi(\lambda)$.

Colour Refinement

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

2. **Refinement**: All vertices $u \in V$ are recursively recoloured depending on their colours and the colours in their neighbourhoods:

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

2. **Refinement**: All vertices $u \in V$ are recursively recoloured depending on their colours and the colours in their neighbourhoods:

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

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

2. **Refinement**: All vertices $u \in V$ are recursively recoloured depending on their colours and the colours in their neighbourhoods:

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

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

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

2. **Refinement**: All vertices $u \in V$ are recursively recoloured depending on their colours and the colours in their neighbourhoods:

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

   where double-braces denote a multiset, and HASH bijectively maps any pair (composed of a colour and a multiset of colours) 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:

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

2. **Refinement**: All vertices $u \in V$ are recursively recoloured depending on their colours and the colours in their neighbourhoods:

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

   where double-braces denote a multiset, and HASH bijectively maps any pair (composed of a colour and a multiset of colours) 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:

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

Colour Refinement

We can now define colour refinement formally for a given a graph $G = (V, E)$ with an initial colouring $\lambda^{(0)}$:

1. **Initialisation**: All vertices $u \in V$, are initialised to their initial colours $\lambda^{(0)}(u)$.

2. **Refinement**: All vertices $u \in V$ are recursively recoloured depending on their colours and the colours in their neighbourhoods:

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

   where double-braces denote a multiset, and HASH bijectively maps any pair (composed of a colour and a multiset of colours) 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:

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

Observe that the stopping condition is well-defined, since $\pi(\lambda^{(i+1)}) \preceq \pi(\lambda^{(i)})$ for any $i \geq 0$, i.e., each iteration corresponds to a refinement, and there exists a minimal integer j such that $\pi(\lambda^{j}) \equiv \pi(\lambda^{(j+1)})$.

Colour Refinement

Colour Refinement

Colour refinement can be used to check whether two given graphs G and H are non-isomorphic:

Colour Refinement

Colour refinement can be used to check whether two given graphs G and H are non-isomorphic:

- Compute the stable colouring $\lambda^{(k)}$ on the disjoint union of G and H.

Colour Refinement

Colour refinement can be used to check whether two given graphs G and H are non-isomorphic:

- Compute the stable colouring $\lambda^{(k)}$ on the disjoint union of G and H.
- If there is a colour $c \in \mathbf{C}$ in the stable colouring, such that the numbers of vertices of colour c differ in G and H, they are non-isomorphic.

Colour Refinement

Colour refinement can be used to check whether two given graphs G and H are non-isomorphic:

- Compute the stable colouring $\lambda^{(k)}$ on the disjoint union of G and H.
- If there is a colour $c \in \mathbf{C}$ in the stable colouring, such that the numbers of vertices of colour c differ in G and H, they are non-isomorphic.

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

Colour Refinement

Colour refinement can be used to check whether two given graphs G and H are non-isomorphic:

- Compute the stable colouring $\lambda^{(k)}$ on the disjoint union of G and H.
- If there is a colour $c \in \mathbf{C}$ in the stable colouring, such that the numbers of vertices of colour c differ in G and H, they are non-isomorphic.

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

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

Colour Refinement

Colour refinement can be used to check whether two given graphs G and H are non-isomorphic:

- Compute the stable colouring $\lambda^{(k)}$ on the disjoint union of G and H.
- If there is a colour $c \in \mathbf{C}$ in the stable colouring, such that the numbers of vertices of colour c differ in G and H, they are non-isomorphic.

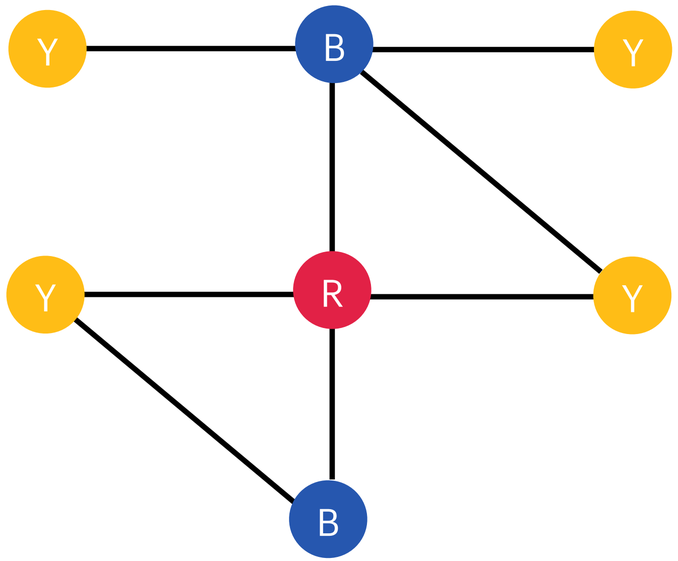
Colour refinement is sound for non-isomorphism checking: whenever it returns yes, for two graphs G and H, they are non-isomorphic.

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

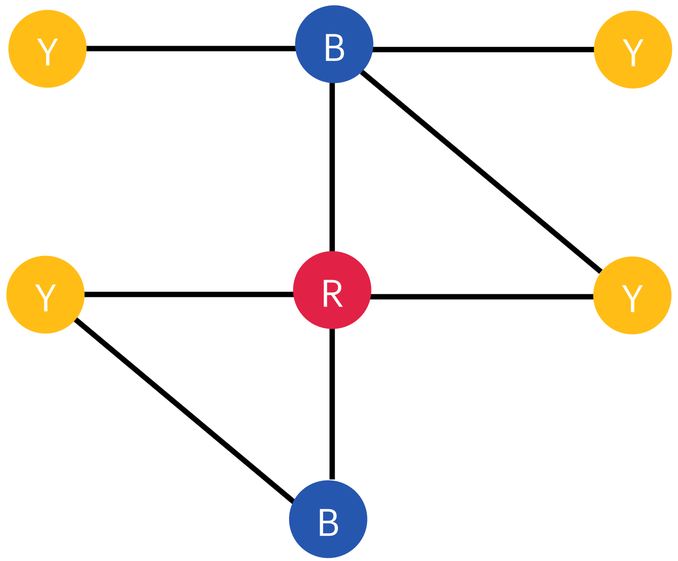
Colour refinement is also known as naive vertex refinement, or 1-dimensional Weisfeiler Lehman (1-WL) algorithm.

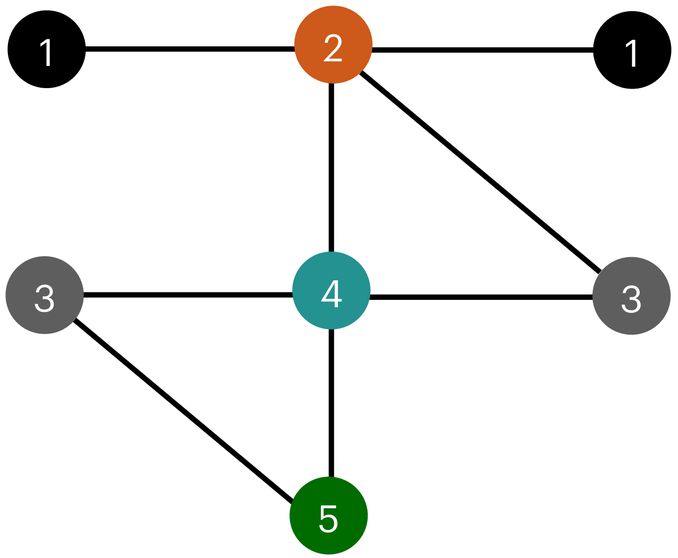
Colour Refinement: Example

Colour Refinement: Example

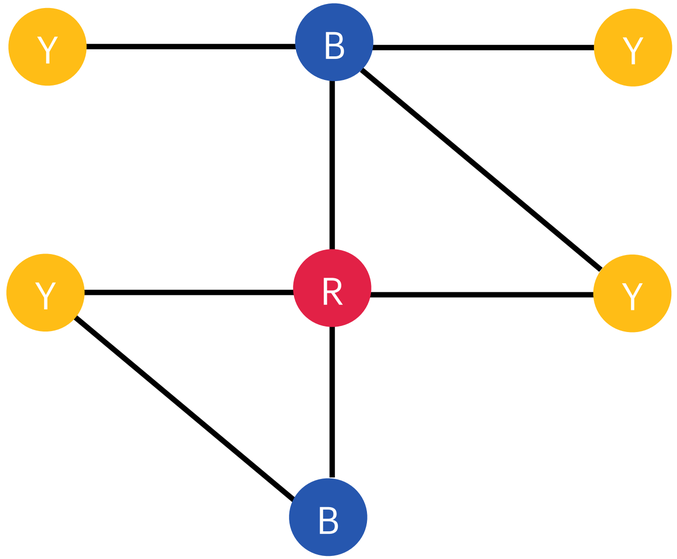


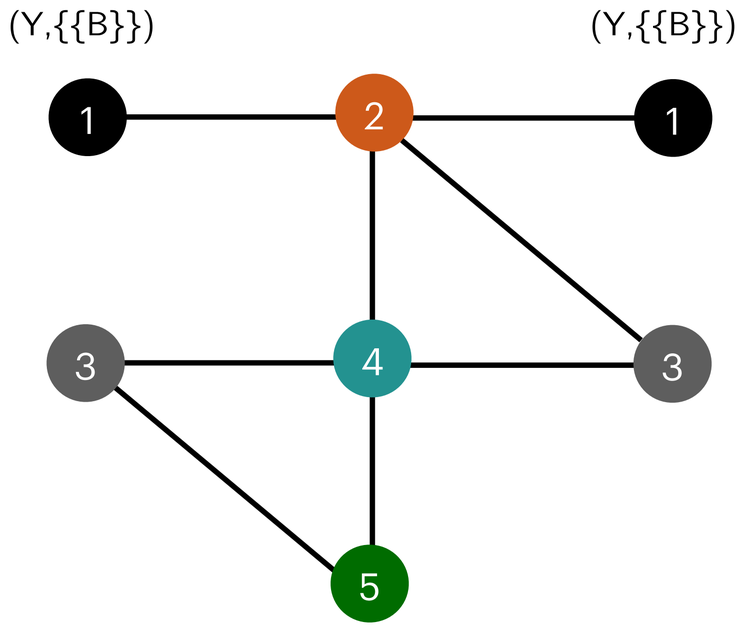
Colour Refinement: Example



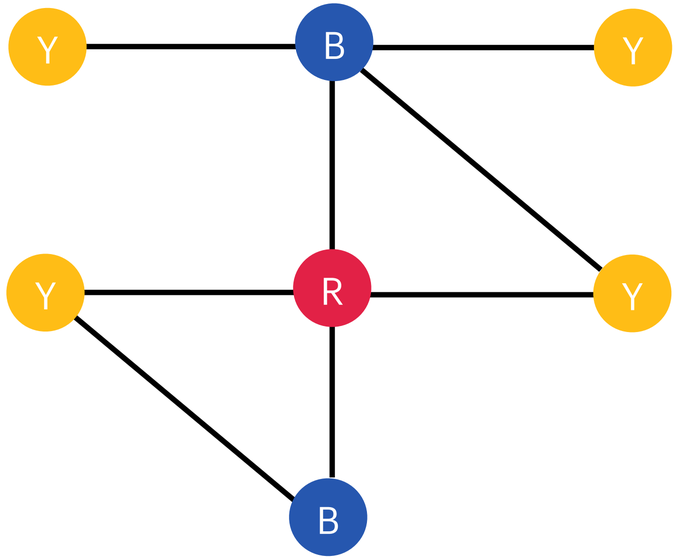


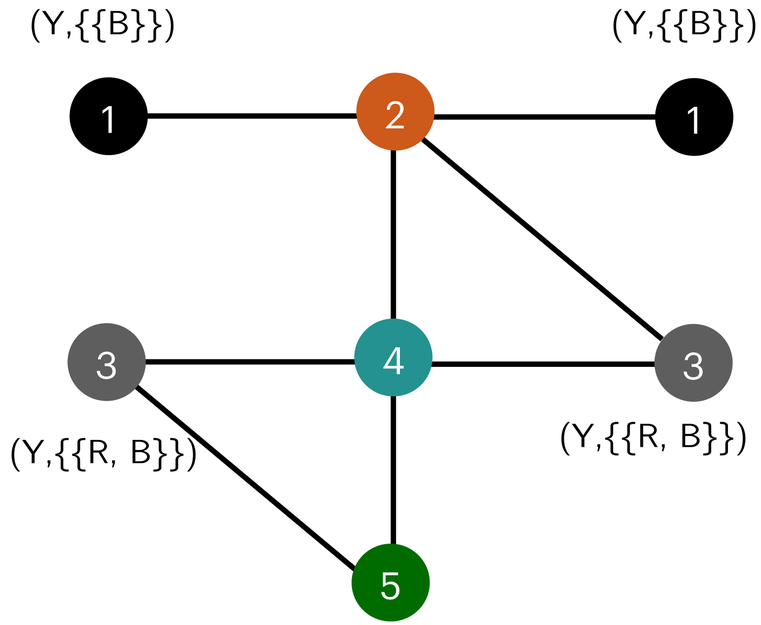
Colour Refinement: Example



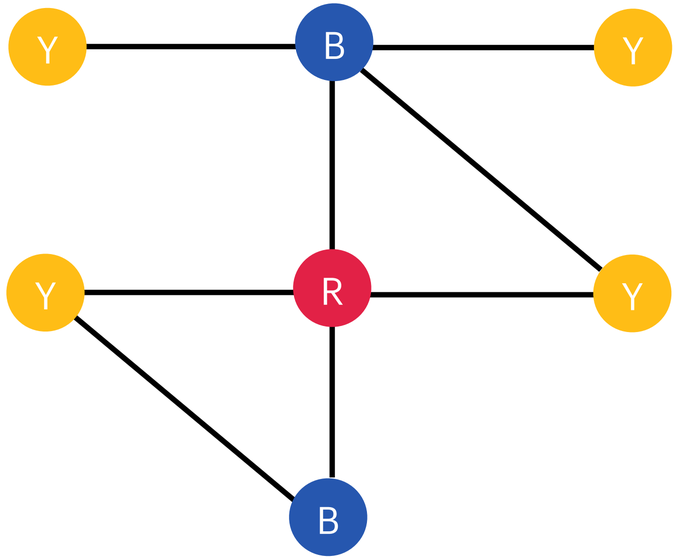


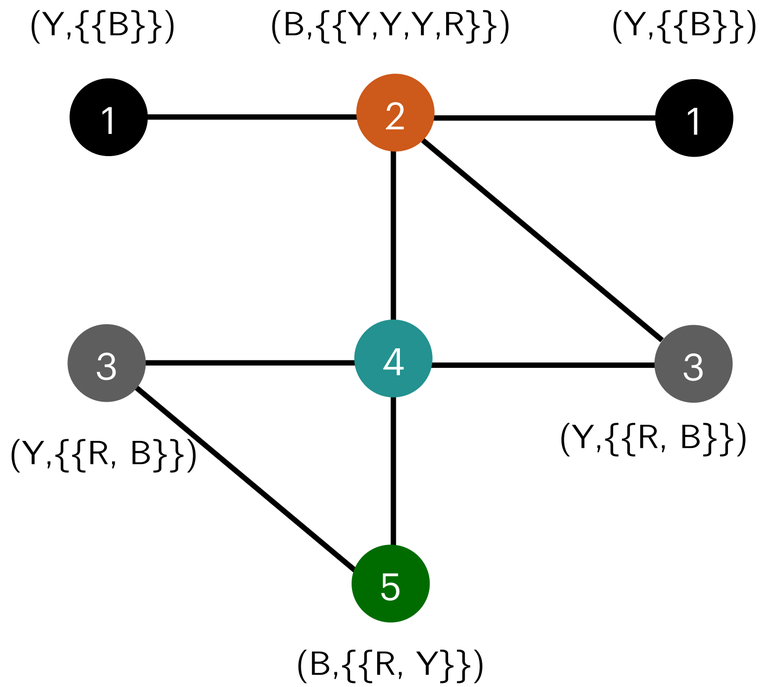
Colour Refinement: Example



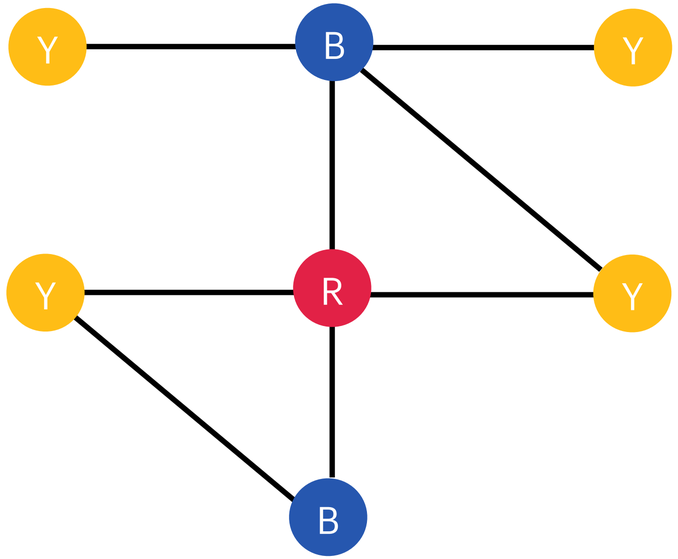


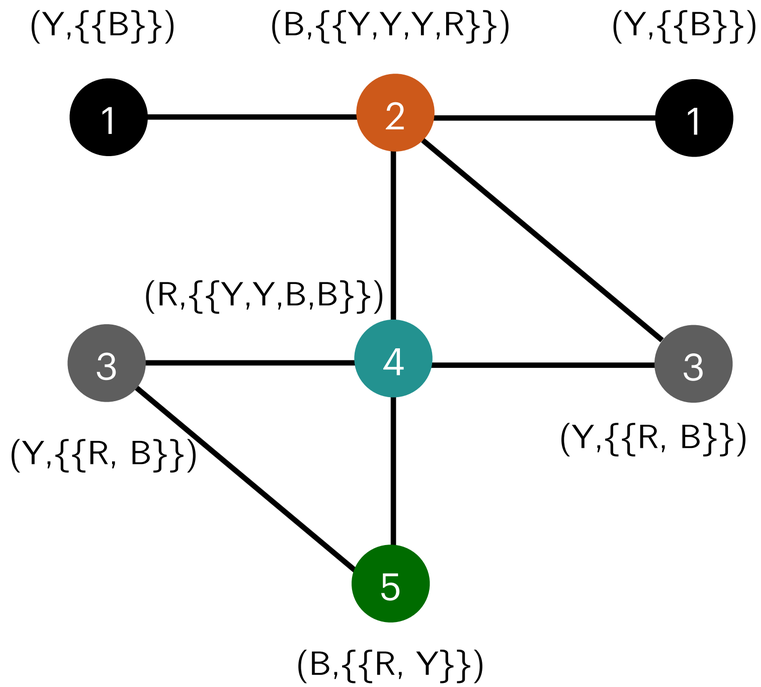
Colour Refinement: Example



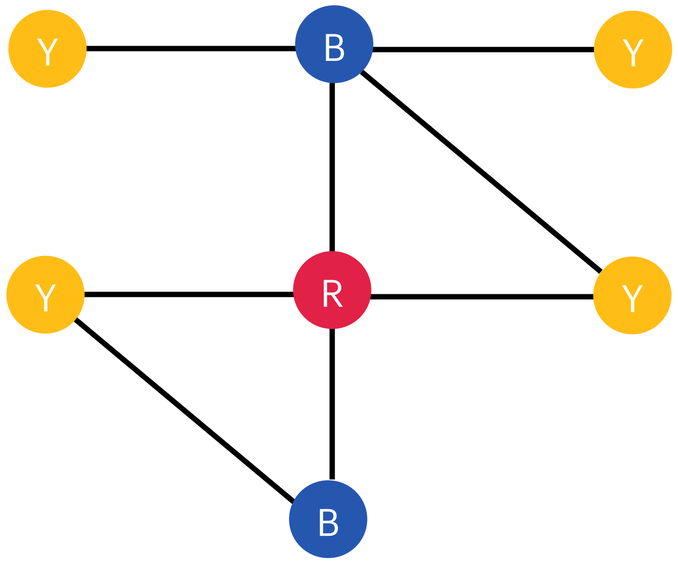


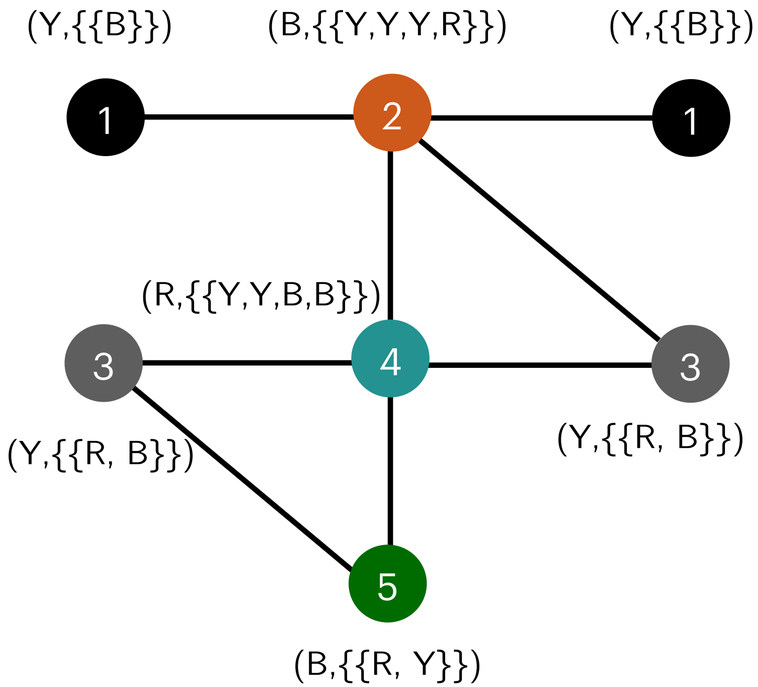
Colour Refinement: Example

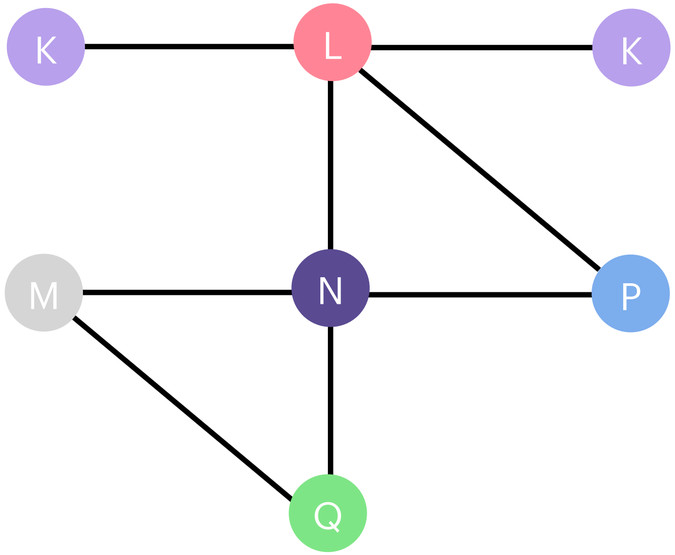




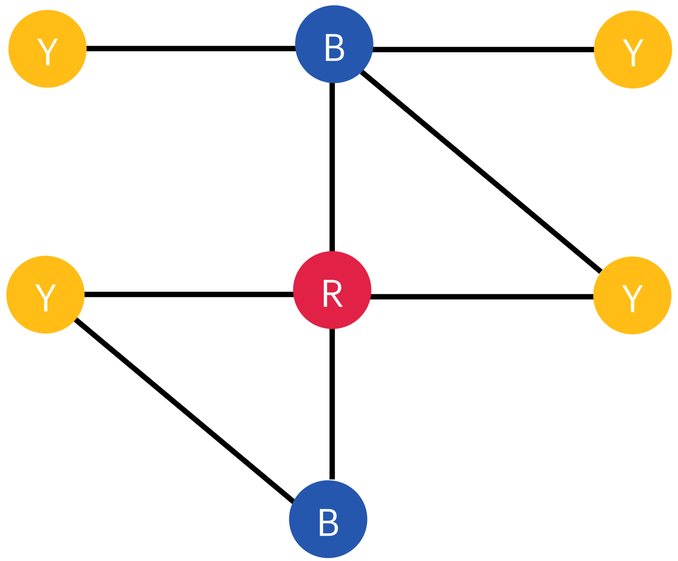
Colour Refinement: Example

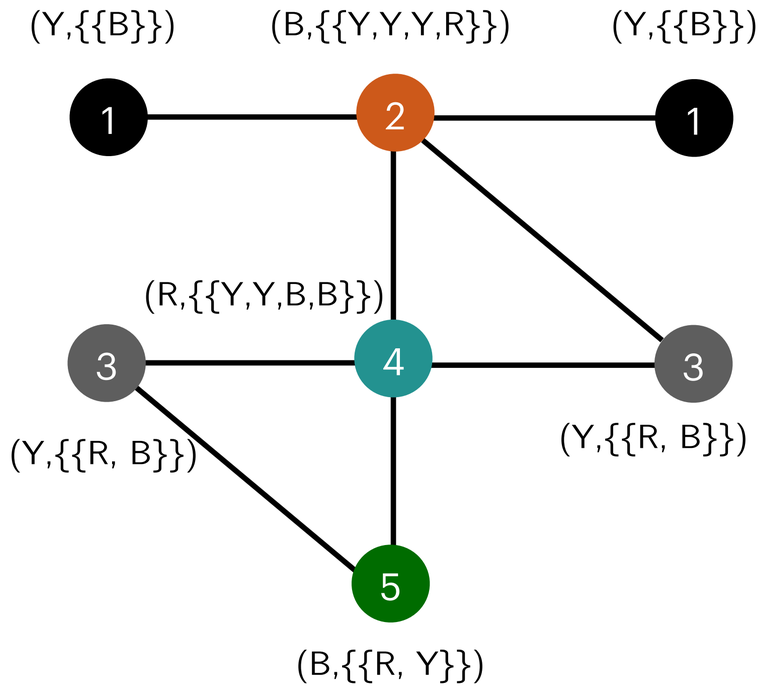


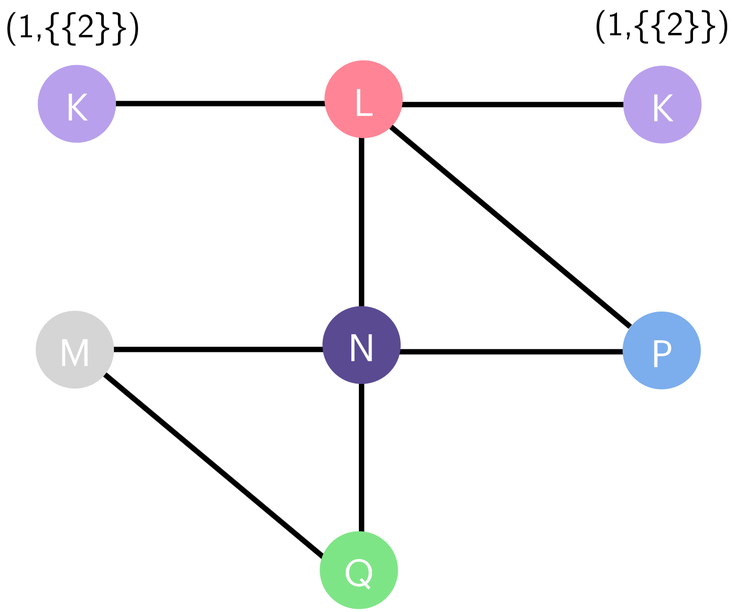




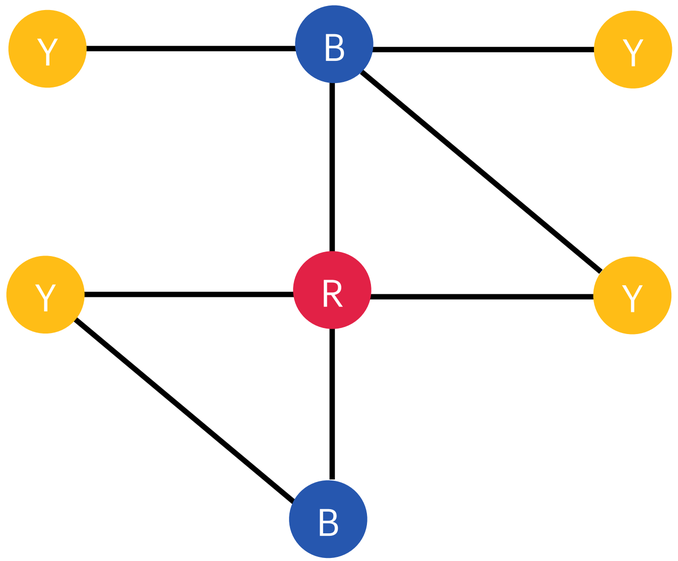
Colour Refinement: Example

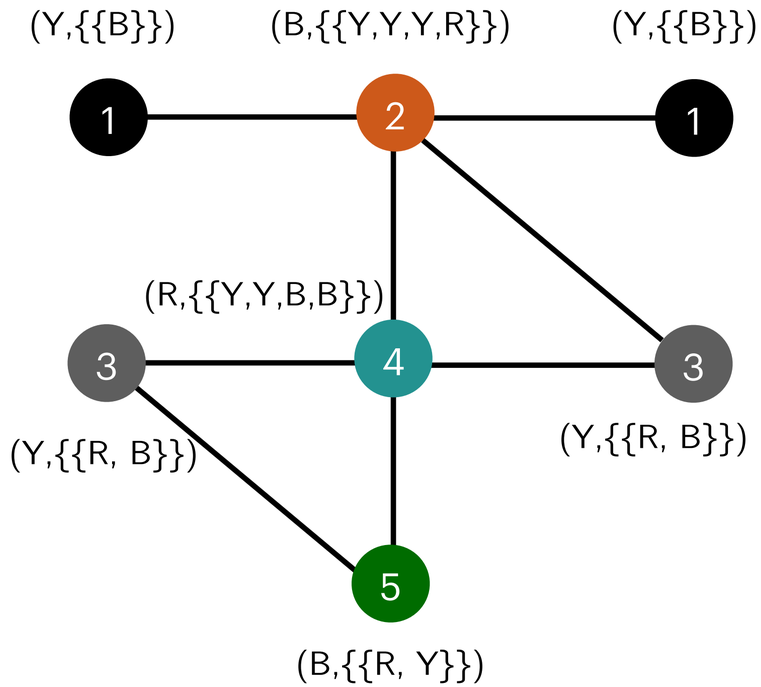


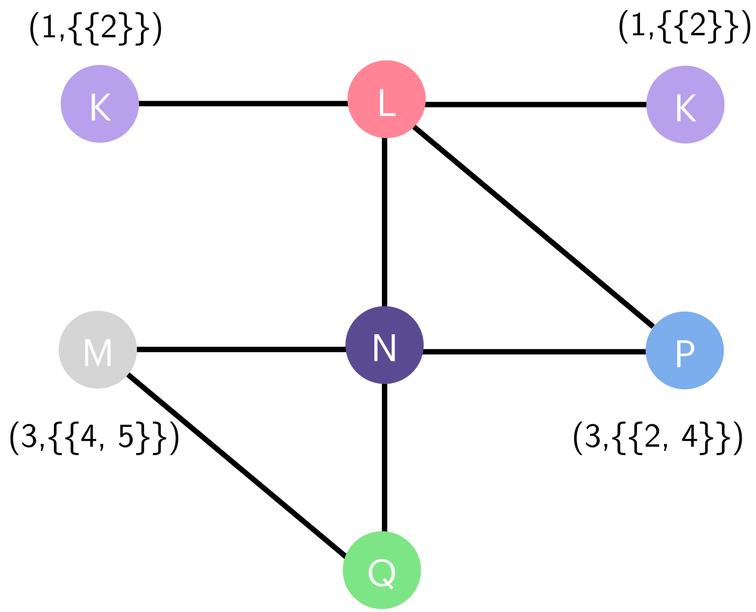




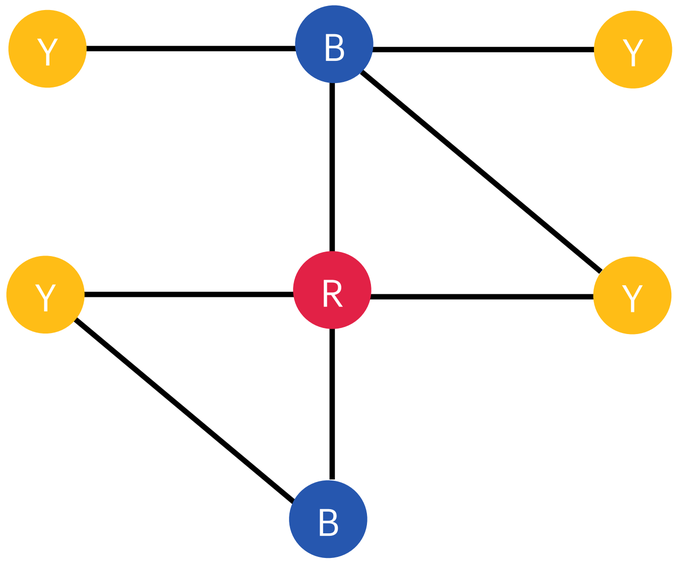
Colour Refinement: Example

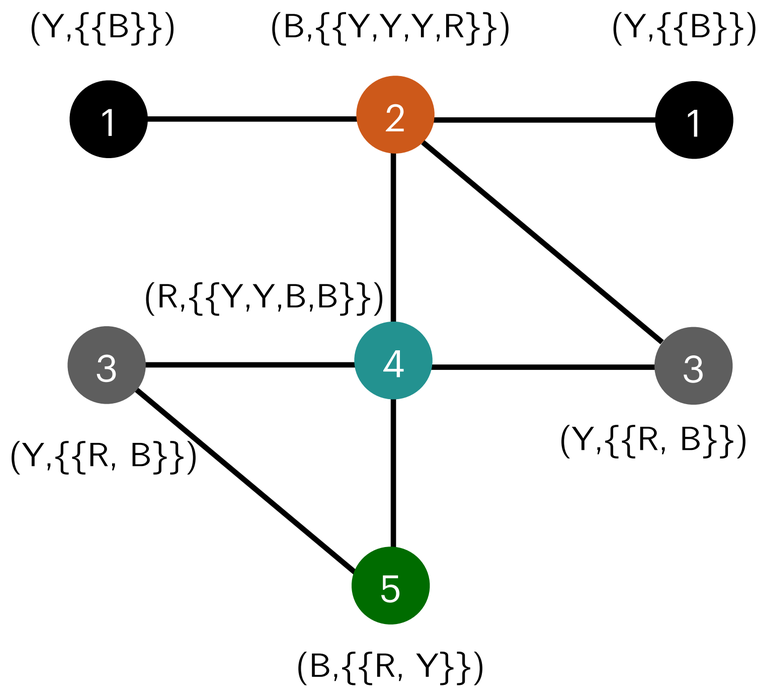


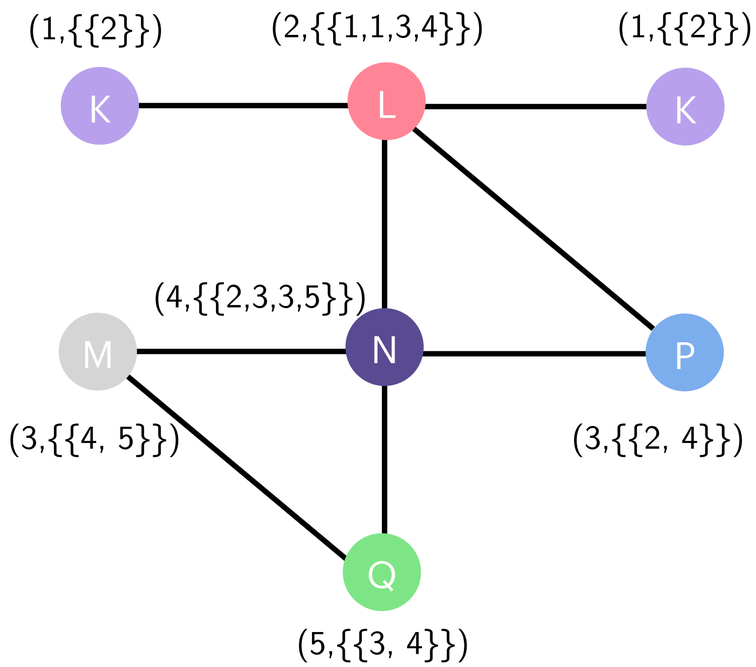




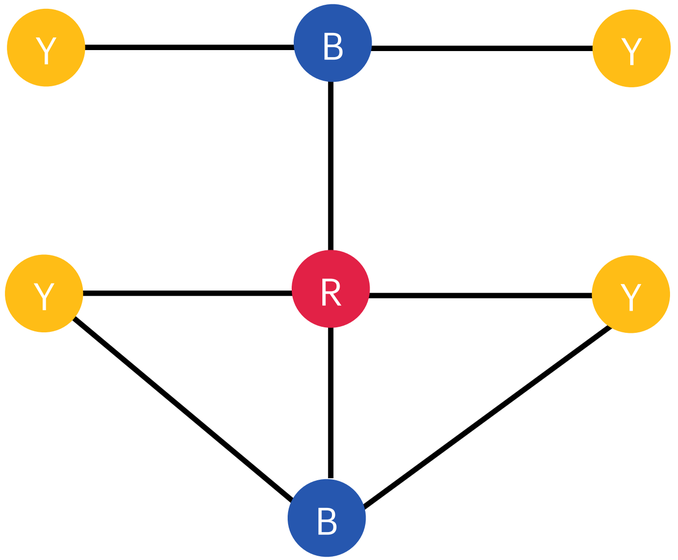
Colour Refinement: Example



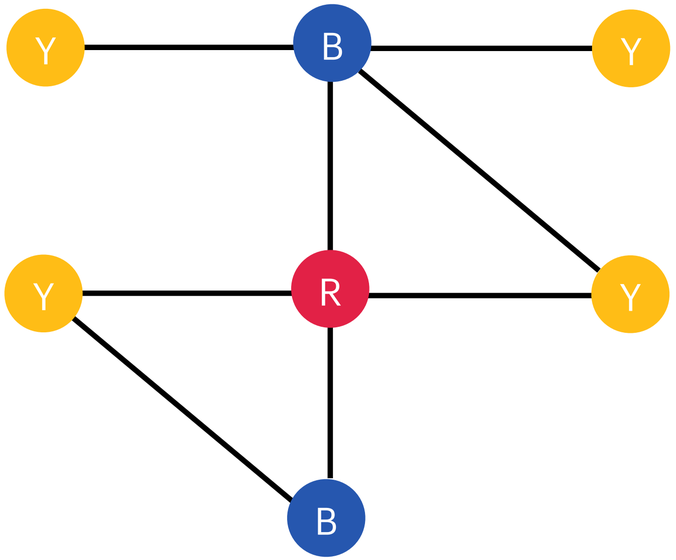


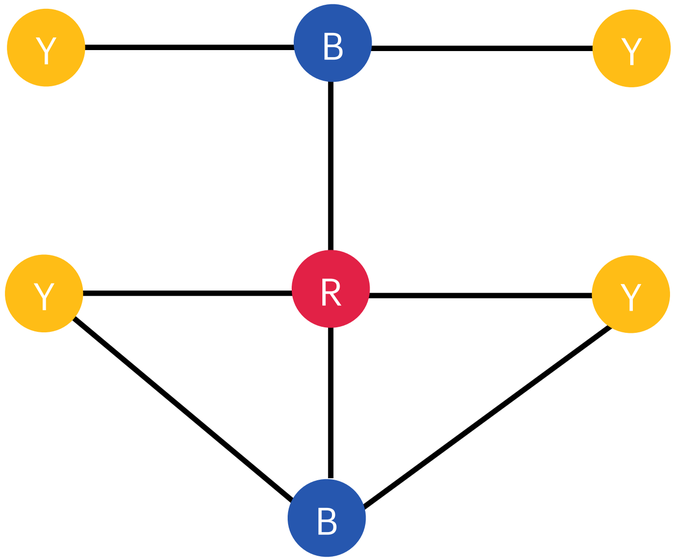


Colour Refinement: Example

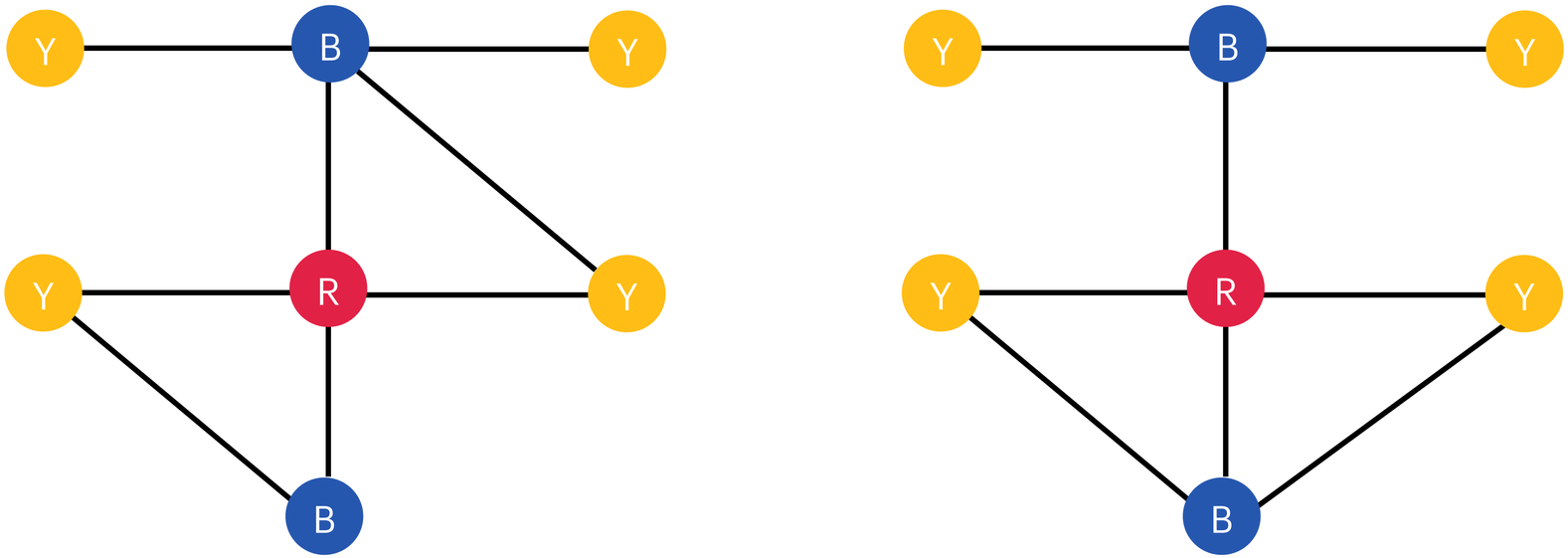


Colour Refinement: Example





Colour Refinement: Example



Vertex colour classes will be different for these two graphs, and so colour refinement can distinguish these non-isomorphic graphs.

Expressive Power of MPNNs

A Characterisation via Graph Isomorphism

A Characterisation via Graph Isomorphism

Observe that the 1-WL algorithm and the neural message passing are closely related:

A Characterisation via Graph Isomorphism

Observe that the 1-WL algorithm and the neural message passing are closely related:

Both iteratively aggregate information from local node neighbourhoods and use this aggregated information to update the representation of each node.

A Characterisation via Graph Isomorphism

Observe that the 1-WL algorithm and the neural message passing are closely related:

Both iteratively aggregate information from local node neighbourhoods and use this aggregated information to update the representation of each node.

Differently, the 1-WL algorithm aggregates and updates discrete labels while MPNNs aggregate and update node embeddings using neural networks.

A Characterisation via Graph Isomorphism

Observe that the 1-WL algorithm and the neural message passing are closely related:

Both iteratively aggregate information from local node neighbourhoods and use this aggregated information to update the representation of each node.

Differently, the 1-WL algorithm aggregates and updates discrete labels while MPNNs aggregate and update node embeddings using neural networks.

Can we view the rounds of the 1-WL algorithm as the layers of an MPNN?

A Characterisation via Graph Isomorphism

Observe that the 1-WL algorithm and the neural message passing are closely related:

Both iteratively aggregate information from local node neighbourhoods and use this aggregated information to update the representation of each node.

Differently, the 1-WL algorithm aggregates and updates discrete labels while MPNNs aggregate and update node embeddings using neural networks.

Can we view the rounds of the 1-WL algorithm as the layers of an MPNN?

Are MPNNs (at most) as powerful as 1-WL?

An Upper Bound for Expressiveness of MPNNs

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 of the following form:

$$\mathbf{h}_u^{(t)} = combine^{(t)}\Big(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\big(\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\}\big)\Big),$$

where $aggregate^{(t)}$ is a permutation-invariant differentiable function and $combine^{(t)}$ a differentiable function. Assuming 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 have different labels after k iterations of the 1-WL algorithm.

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 of the following form:

$$\mathbf{h}_u^{(t)} = combine^{(t)}\Big(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\big(\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\}\big)\Big),$$

where $aggregate^{(t)}$ is a permutation-invariant differentiable function and $combine^{(t)}$ a differentiable function. Assuming 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 have different labels after k iterations of the 1-WL algorithm.

Intuitively, this means that MPNNs can never contradict the 1-WL test:

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 of the following form:

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

where $aggregate^{(t)}$ is a permutation-invariant differentiable function and $combine^{(t)}$ a differentiable function. Assuming 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 have different labels after k iterations of the 1-WL algorithm.

Intuitively, this means that MPNNs can never contradict 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. Similarly, if the 1-WL test cannot distinguish between two graphs, then an MPNN is also incapable of distinguishing between these two graphs.

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 of the following form:

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

where $aggregate^{(t)}$ is a permutation-invariant differentiable function and $combine^{(t)}$ a differentiable function. Assuming 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 have different labels after k iterations of the 1-WL algorithm.

Intuitively, this means that MPNNs can never contradict 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. Similarly, if the 1-WL test cannot distinguish between two graphs, then an MPNN is also incapable of distinguishing between these two graphs.

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

A Lower Bound for Expressiveness of MPNNs

A Lower Bound for Expressiveness of MPNNs

Theorem ([Morris et al., 2019, Xu et al., 2019]). There exists an MPNN such that $\mathbf{h}_u^{(k)} = \mathbf{h}_v^{(k)}$ if and only if the two nodes u and v have the same label after k iterations of the 1-WL algorithm.

For example, the basic MPNN model:

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

has been shown to be as powerful as 1-WL.

A Lower Bound for Expressiveness of MPNNs

Theorem ([Morris et al., 2019, Xu et al., 2019]). There exists an MPNN such that $\mathbf{h}_u^{(k)} = \mathbf{h}_v^{(k)}$ if and only if the two nodes u and v have the same label after k iterations of the 1-WL algorithm.

For example, the basic MPNN model:

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

has been shown to be as powerful as 1-WL.

Interestingly, however, most of the popular MPNN models, such as GCNs, are not even as expressive as 1-WL.

A Lower Bound for Expressiveness of MPNNs

Theorem ([Morris et al., 2019, Xu et al., 2019]). There exists an MPNN such that $\mathbf{h}_u^{(k)} = \mathbf{h}_v^{(k)}$ if and only if the two nodes u and v have the same label after k iterations of the 1-WL algorithm.

For example, the basic MPNN model:

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

has been shown to be as powerful as 1-WL.

Interestingly, however, most of the popular MPNN models, such as GCNs, are not even as expressive as 1-WL.

Key ingredient: The functions $aggregate^{(t)}$ and $combine^{(t)}$ need to be injective to achieve maximal expressivity (Xu et al., 2019).

A Lower Bound for Expressiveness of MPNNs

Theorem ([Morris et al., 2019, Xu et al., 2019]). There exists an MPNN such that $\mathbf{h}_u^{(k)} = \mathbf{h}_v^{(k)}$ if and only if the two nodes u and v have the same label after k iterations of the 1-WL algorithm.

For example, the basic MPNN model:

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

has been shown to be as powerful as 1-WL.

Interestingly, however, most of the popular MPNN models, such as GCNs, are not even as expressive as 1-WL.

Key ingredient: The functions $aggregate^{(t)}$ and $combine^{(t)}$ need to be injective to achieve maximal expressivity (Xu et al., 2019).

Indeed, we can view the rounds of the 1-WL algorithm as the layers of an MPNN with injective combine and aggregate functions!

A Lower Bound for Expressiveness of MPNNs

Theorem ([Morris et al., 2019, Xu et al., 2019]). There exists an MPNN such that $\mathbf{h}_u^{(k)} = \mathbf{h}_v^{(k)}$ if and only if the two nodes u and v have the same label after k iterations of the 1-WL algorithm.

For example, the basic MPNN model:

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

has been shown to be as powerful as 1-WL.

Interestingly, however, most of the popular MPNN models, such as GCNs, are not even as expressive as 1-WL.

Key ingredient: The functions $aggregate^{(t)}$ and $combine^{(t)}$ need to be injective to achieve maximal expressivity (Xu et al., 2019).

Indeed, we can view the rounds of the 1-WL algorithm as the layers of an MPNN with injective combine and aggregate functions!

MPNNs are as powerful as 1-WL test under mild assumptions.

A Descriptive Complexity Perspective

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

WL is a class of algorithms and forms an hierarchy, i.e., 1-WL, 2-WL, etc.. as we shall see in the Lecture 6.

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

WL is a class of algorithms and forms an hierarchy, i.e., 1-WL, 2-WL, etc.. as we shall see in the Lecture 6.

There is an interesting connection between the WL hierarchy and the extension of first order logic with counting quantifiers, given by a classical result:

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

WL is a class of algorithms and forms an hierarchy, i.e., 1-WL, 2-WL, etc.. as we shall see in the Lecture 6.

There is an interesting connection between the WL hierarchy and the extension of first order logic with counting quantifiers, given by a classical result:

Theorem (Cai et al., 1992). 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.

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

WL is a class of algorithms and forms an hierarchy, i.e., 1-WL, 2-WL, etc.. as we shall see in the Lecture 6.

There is an interesting connection between the WL hierarchy and the extension of first order logic with counting quantifiers, given by a classical result:

Theorem (Cai et al., 1992). 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.

Together with the results of (Morris et al., 2019; Xu et al., 2019), this implies the following:

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

WL is a class of algorithms and forms an hierarchy, i.e., 1-WL, 2-WL, etc.. as we shall see in the Lecture 6.

There is an interesting connection between the WL hierarchy and the extension of first order logic with counting quantifiers, given by a classical result:

Theorem (Cai et al., 1992). 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.

Together with the results of (Morris et al., 2019; Xu et al., 2019), this implies the following:

Proposition (Morris et al., 2019; Xu et al., 2019). Two graphs G and H are indistinguishable by all MPNNs if and only if they satisfy the same C^2 -sentences.

A Descriptive Complexity Perspective

MPNNs can be viewed as an extension of the 1-WL algorithm which have the same power but are more flexible in their ability to adapt to the learning task at hand and are able to handle continuous node features.

WL is a class of algorithms and forms an hierarchy, i.e., 1-WL, 2-WL, etc.. as we shall see in the Lecture 6.

There is an interesting connection between the WL hierarchy and the extension of first order logic with counting quantifiers, given by a classical result:

Theorem (Cai et al., 1992). 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.

Together with the results of (Morris et al., 2019; Xu et al., 2019), this implies the following:

Proposition (Morris et al., 2019; Xu et al., 2019). Two graphs G and H are indistinguishable by all MPNNs if and only if they satisfy the same C^2 -sentences.

This is the territory of descriptive complexity — a branch of complexity theory, where the goal is to characterise complexity classes in terms of the logics that can capture the complexity classes (Immerman, 1995).

Logic of Graphs

First-Order Logic: Syntax

First-Order Logic: Syntax

Basics: A (first-order) relational vocabulary denoted by σ, consists of sets $\mathbf{R}$ of relation, $\mathbf{C}$ of constant, and $\mathbf{V}$ of variable names. A term is either a constant or a variable. An atom is of the form $P(s_1, \ldots, s_n)$, where P is an n-ary relation, and $s_1, \ldots, s_n$ are terms. A ground atom is an atom without variables.

First-Order Logic: Syntax

Basics: A (first-order) relational vocabulary denoted by σ, consists of sets $\mathbf{R}$ of relation, $\mathbf{C}$ of constant, and $\mathbf{V}$ of variable names. A term is either a constant or a variable. An atom is of the form $P(s_1, \ldots, s_n)$, where P is an n-ary relation, and $s_1, \ldots, s_n$ are terms. A ground atom is an atom without variables.

Logical connectives and quantifiers: The logical connectives are negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$), and quantifiers are existential quantifier ($\exists$) and universal quantifier ($\forall$).

First-Order Logic: Syntax

Basics: A (first-order) relational vocabulary denoted by σ, consists of sets $\mathbf{R}$ of relation, $\mathbf{C}$ of constant, and $\mathbf{V}$ of variable names. A term is either a constant or a variable. An atom is of the form $P(s_1, \ldots, s_n)$, where P is an n-ary relation, and $s_1, \ldots, s_n$ are terms. A ground atom is an atom without variables.

Logical connectives and quantifiers: The logical connectives are negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$), and quantifiers are existential quantifier ($\exists$) and universal quantifier ($\forall$).

Formulas: First-order logic (FO) formulas are inductively built from atomic formulas using the logical constructors and quantifiers based on the grammar rule:

$$\Phi = \; P(s_1, \ldots, s_n) \;\mid\; \neg\Phi \;\mid\; \Phi \wedge \Phi \;\mid\; \Phi \vee \Phi \;\mid\; \exists x \,.\, \Phi \;\mid\; \forall x \,.\, \Phi,$$

where P is an n-ary relation, $s_1, \ldots, s_n$ are terms, and x is a variable.

Note: We are using upper-case letters to denote relation names, and lower case letters to denote variables/ constants — In Lecture 1 & 2, we used lower case for everything to align with conventions in node embeddings.

First-Order Logic: Syntax

First-Order Logic: Syntax

A variable x in a formula Φ is quantified, or bound if it is in the scope of a quantifier; otherwise, it is free.

First-Order Logic: Syntax

A variable x in a formula Φ is quantified, or bound if it is in the scope of a quantifier; otherwise, it is free.

A (first-order) sentence is a (first-order) formula without any free variables, also called a Boolean formula.

First-Order Logic: Syntax

A variable x in a formula Φ is quantified, or bound if it is in the scope of a quantifier; otherwise, it is free.

A (first-order) sentence is a (first-order) formula without any free variables, also called a Boolean formula.

In the sequel, we write, e.g., Φ to denote Boolean formulas, and $\Phi(x_1, \ldots, x_k)$ to denote formulas with free variables $x_1, \ldots, x_k$.

First-Order Logic: Syntax

A variable x in a formula Φ is quantified, or bound if it is in the scope of a quantifier; otherwise, it is free.

A (first-order) sentence is a (first-order) formula without any free variables, also called a Boolean formula.

In the sequel, we write, e.g., Φ to denote Boolean formulas, and $\Phi(x_1, \ldots, x_k)$ to denote formulas with free variables $x_1, \ldots, x_k$.

As usual, some constructors are only syntactic sugar, i.e., we use usual abbreviations:

$$\forall x \,.\, \Phi \equiv \neg\, \exists x \,.\, \neg \Phi,$$

$$\Phi \vee \Psi \equiv \neg(\neg \Phi \wedge \neg \Psi),$$

$$\Phi \rightarrow \Psi \equiv \neg \Phi \vee \Psi,$$

and so we define the semantics based on the constructors $\neg$, $\wedge$, $\exists$.

First-Order Logic: Semantics

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

A variable assignment is a function $\mu : \mathbf{V} \mapsto \Delta^I$ that maps variables to domain elements.

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

A variable assignment is a function $\mu : \mathbf{V} \mapsto \Delta^I$ that maps variables to domain elements.

Given an element $e \in \Delta^I$ and a variable $x \in \mathbf{V}$, we write $\mu[x \mapsto e]$ to denote the variable assignment that maps x to e, and that agrees with μ on all other variables.

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

A variable assignment is a function $\mu : \mathbf{V} \mapsto \Delta^I$ that maps variables to domain elements.

Given an element $e \in \Delta^I$ and a variable $x \in \mathbf{V}$, we write $\mu[x \mapsto e]$ to denote the variable assignment that maps x to e, and that agrees with μ on all other variables.

For an interpretation I and a variable assignment μ, we define:

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

A variable assignment is a function $\mu : \mathbf{V} \mapsto \Delta^I$ that maps variables to domain elements.

Given an element $e \in \Delta^I$ and a variable $x \in \mathbf{V}$, we write $\mu[x \mapsto e]$ to denote the variable assignment that maps x to e, and that agrees with μ on all other variables.

For an interpretation I and a variable assignment μ, we define:

- $a^{I,\mu} = a^I$ for all constant names $a \in \mathbf{C}$,

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

A variable assignment is a function $\mu : \mathbf{V} \mapsto \Delta^I$ that maps variables to domain elements.

Given an element $e \in \Delta^I$ and a variable $x \in \mathbf{V}$, we write $\mu[x \mapsto e]$ to denote the variable assignment that maps x to e, and that agrees with μ on all other variables.

For an interpretation I and a variable assignment μ, we define:

- $a^{I,\mu} = a^I$ for all constant names $a \in \mathbf{C}$,
- $x^{I,\mu} = \mu(x)$ for all variable names $x \in \mathbf{V}$,

First-Order Logic: Semantics

A first-order interpretation is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a non-empty domain, and $\cdot^I$ is an interpretation function.

The interpretation function $\cdot^I$ maps every constant name a to an element $a^I \in \Delta^I$ of the domain, and every predicate name P with arity n to a subset $P^I \subseteq (\Delta^I)^n$ of the domain.

A variable assignment is a function $\mu : \mathbf{V} \mapsto \Delta^I$ that maps variables to domain elements.

Given an element $e \in \Delta^I$ and a variable $x \in \mathbf{V}$, we write $\mu[x \mapsto e]$ to denote the variable assignment that maps x to e, and that agrees with μ on all other variables.

For an interpretation I and a variable assignment μ, we define:

- $a^{I,\mu} = a^I$ for all constant names $a \in \mathbf{C}$,
- $x^{I,\mu} = \mu(x)$ for all variable names $x \in \mathbf{V}$,
- $P^{I,\mu} = P^I$ for all relation names $P \in \mathbf{R}$.

First-Order Logic: Semantics

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,
- $I, \mu \models \neg\Phi(x_1, \ldots, x_n)$ if $I, \mu \not\models \Phi(x_1, \ldots, x_n)$,

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,
- $I, \mu \models \neg\Phi(x_1, \ldots, x_n)$ if $I, \mu \not\models \Phi(x_1, \ldots, x_n)$,
- $I, \mu \models \Phi(x_1, \ldots, x_n) \wedge \Psi(y_1, \ldots, y_m)$ if $I, \mu \models \Phi(x_1, \ldots, x_n)$ and $I, \mu \models \Psi(y_1, \ldots, y_n)$,

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,
- $I, \mu \models \neg\Phi(x_1, \ldots, x_n)$ if $I, \mu \not\models \Phi(x_1, \ldots, x_n)$,
- $I, \mu \models \Phi(x_1, \ldots, x_n) \wedge \Psi(y_1, \ldots, y_m)$ if $I, \mu \models \Phi(x_1, \ldots, x_n)$ and $I, \mu \models \Psi(y_1, \ldots, y_n)$,
- $I, \mu \models \exists x \,.\, \Phi(y_1, \ldots, y_n)$ if there exists $e \in \Delta^I$ such that $I, \mu[x \mapsto e] \models \Phi(y_1, \ldots, y_n)$,

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,
- $I, \mu \models \neg\Phi(x_1, \ldots, x_n)$ if $I, \mu \not\models \Phi(x_1, \ldots, x_n)$,
- $I, \mu \models \Phi(x_1, \ldots, x_n) \wedge \Psi(y_1, \ldots, y_m)$ if $I, \mu \models \Phi(x_1, \ldots, x_n)$ and $I, \mu \models \Psi(y_1, \ldots, y_n)$,
- $I, \mu \models \exists x \,.\, \Phi(y_1, \ldots, y_n)$ if there exists $e \in \Delta^I$ such that $I, \mu[x \mapsto e] \models \Phi(y_1, \ldots, y_n)$,

The truth value of sentences does not depend on any variable assignment; so, assignments are omitted in this case. We say that an interpretation I is a model of a sentence Φ if $I \models \Phi$.

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,
- $I, \mu \models \neg\Phi(x_1, \ldots, x_n)$ if $I, \mu \not\models \Phi(x_1, \ldots, x_n)$,
- $I, \mu \models \Phi(x_1, \ldots, x_n) \wedge \Psi(y_1, \ldots, y_m)$ if $I, \mu \models \Phi(x_1, \ldots, x_n)$ and $I, \mu \models \Psi(y_1, \ldots, y_n)$,
- $I, \mu \models \exists x \,.\, \Phi(y_1, \ldots, y_n)$ if there exists $e \in \Delta^I$ such that $I, \mu[x \mapsto e] \models \Phi(y_1, \ldots, y_n)$,

The truth value of sentences does not depend on any variable assignment; so, assignments are omitted in this case. We say that an interpretation I is a model of a sentence Φ if $I \models \Phi$.

An interpretation, or a model, is finite if its domain (or, universe) is finite. Our focus is on first-order logic over finite models/structures — we assume finite domains and we are in the context of finite model theory.

First-Order Logic: Semantics

Given an interpretation I and a variable assignment μ, the entailment relation ($\models$) is inductively defined as

- $I, \mu \models P(s_1, \ldots, s_n)$ if $(s_1^{I,\mu}, \ldots, s_n^{I,\mu}) \in P^{I,\mu}$,
- $I, \mu \models \neg\Phi(x_1, \ldots, x_n)$ if $I, \mu \not\models \Phi(x_1, \ldots, x_n)$,
- $I, \mu \models \Phi(x_1, \ldots, x_n) \wedge \Psi(y_1, \ldots, y_m)$ if $I, \mu \models \Phi(x_1, \ldots, x_n)$ and $I, \mu \models \Psi(y_1, \ldots, y_n)$,
- $I, \mu \models \exists x \,.\, \Phi(y_1, \ldots, y_n)$ if there exists $e \in \Delta^I$ such that $I, \mu[x \mapsto e] \models \Phi(y_1, \ldots, y_n)$,

The truth value of sentences does not depend on any variable assignment; so, assignments are omitted in this case. We say that an interpretation I is a model of a sentence Φ if $I \models \Phi$.

An interpretation, or a model, is finite if its domain (or, universe) is finite. Our focus is on first-order logic over finite models/structures — we assume finite domains and we are in the context of finite model theory.

We assume that constants are mapped to themselves by any interpretation (i.e., unique name assumption).

First-Order Logic of Graphs

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

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

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

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

Consider a simple set $\{u, v, w\}$ as a domain, and define an interpretation I which interprets the relation E as $E^I = \{(u, v), (v, w), (u, w)\}$.

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

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

Consider a simple set $\{u, v, w\}$ as a domain, and define an interpretation I which interprets the relation E as $E^I = \{(u, v), (v, w), (u, w)\}$.

It is easy to see that $I \vDash \Phi(u)$, i.e., I is a model of $\Phi(x)$ when interpreting the free variable x as u.

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

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

Consider a simple set $\{u, v, w\}$ as a domain, and define an interpretation I which interprets the relation E as $E^I = \{(u, v), (v, w), (u, w)\}$.

It is easy to see that $I \vDash \Phi(u)$, i.e., I is a model of $\Phi(x)$ when interpreting the free variable x as u.

Observe that this interpretation is a graph — domain elements are vertices and relations are edges!

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

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

Consider a simple set $\{u, v, w\}$ as a domain, and define an interpretation I which interprets the relation E as $E^I = \{(u, v), (v, w), (u, w)\}$.

It is easy to see that $I \vDash \Phi(u)$, i.e., I is a model of $\Phi(x)$ when interpreting the free variable x as u.

Observe that this interpretation is a graph — domain elements are vertices and relations are edges!

The formula $\Phi(x)$ specifies a graph property over some input graph and relative to some vertex interpreting x!

First-Order Logic of Graphs

Consider the FO formula with one free variable x:

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

Consider a simple set $\{u, v, w\}$ as a domain, and define an interpretation I which interprets the relation E as $E^I = \{(u, v), (v, w), (u, w)\}$.

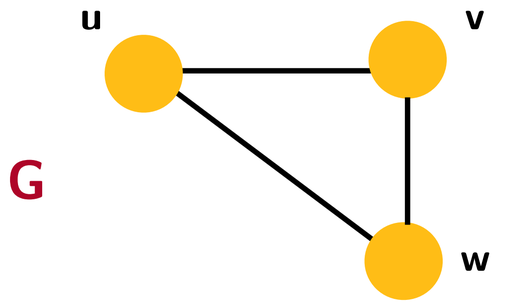
It is easy to see that $I \vDash \Phi(u)$, i.e., I is a model of $\Phi(x)$ when interpreting the free variable x as u.

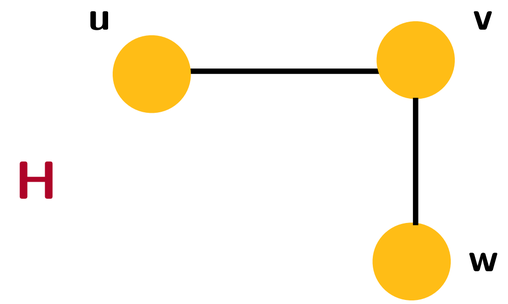
Observe that this interpretation is a graph — domain elements are vertices and relations are edges!

The formula $\Phi(x)$ specifies a graph property over some input graph and relative to some vertex interpreting x!

Logic can be used to characterise graph properties and we can view graphs as interpretations, where the domain is simply the set of vertices!

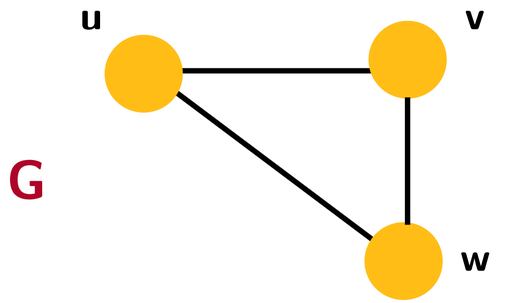
First-Order Logic of Graphs

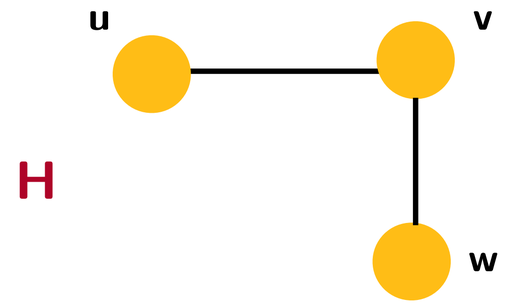




First-Order Logic of Graphs

The formula $\Phi(x) = \exists y, z \; E(x, y) \wedge E(y, z) \wedge E(x, z)$ is in the language of graphs, where $E(x, y)$ means that there is an edge between the nodes interpreting x and y.

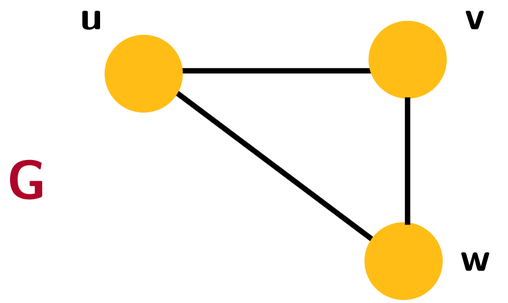


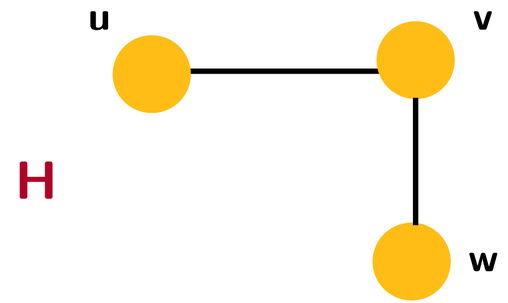


First-Order Logic of Graphs

The formula $\Phi(x) = \exists y, z\ E(x, y) \wedge E(y, z) \wedge E(x, z)$ is in the language of graphs, where $E(x, y)$ means that there is an edge between the nodes interpreting x and y.

Graphs as interpretations: View the graphs G and H as interpretations over a domain $\{u, v, w\}$. Then:



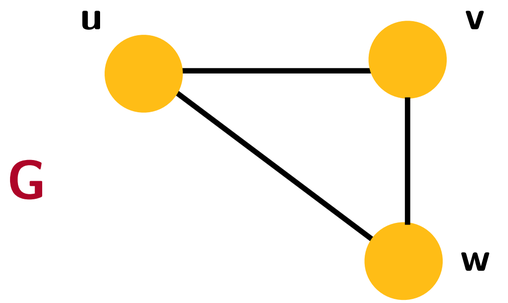


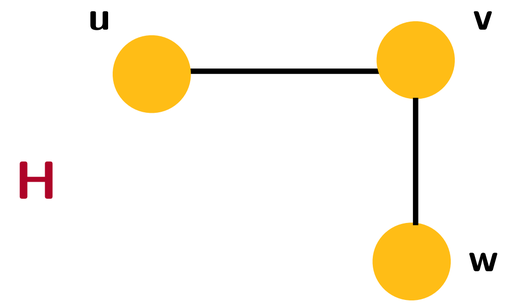
First-Order Logic of Graphs

The formula $\Phi(x) = \exists y, z\ E(x, y) \wedge E(y, z) \wedge E(x, z)$ is in the language of graphs, where $E(x, y)$ means that there is an edge between the nodes interpreting x and y.

Graphs as interpretations: View the graphs G and H as interpretations over a domain $\{u, v, w\}$. Then:

- $G \models \Phi(u)$



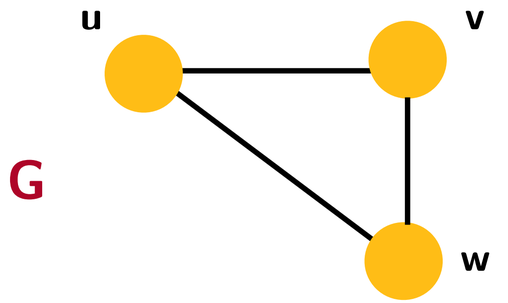


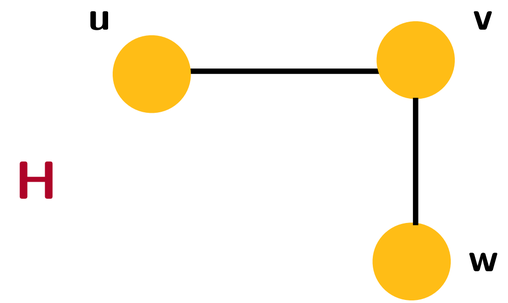
First-Order Logic of Graphs

The formula $\Phi(x) = \exists y, z\ E(x, y) \wedge E(y, z) \wedge E(x, z)$ is in the language of graphs, where $E(x, y)$ means that there is an edge between the nodes interpreting x and y.

Graphs as interpretations: View the graphs G and H as interpretations over a domain $\{u, v, w\}$. Then:

- $G \models \Phi(u)$
- $H \not\models \Phi(u)$





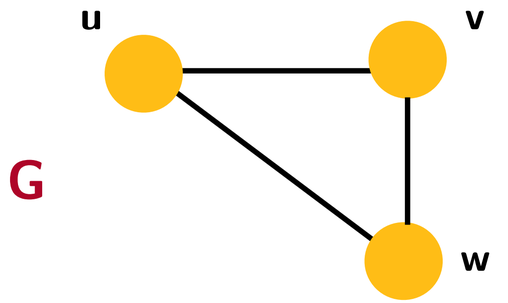
First-Order Logic of Graphs

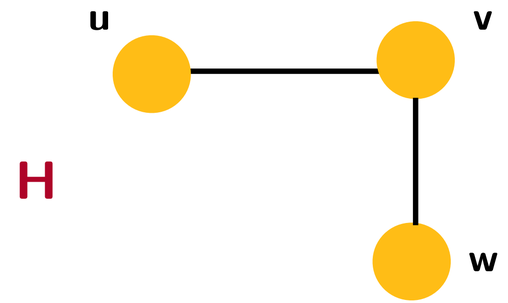
The formula $\Phi(x) = \exists y, z\ E(x, y) \wedge E(y, z) \wedge E(x, z)$ is in the language of graphs, where $E(x, y)$ means that there is an edge between the nodes interpreting x and y.

Graphs as interpretations: View the graphs G and H as interpretations over a domain $\{u, v, w\}$. Then:

- $G \models \Phi(u)$
- $H \not\models \Phi(u)$

The graph G is a model of $\Phi(x)$ when x is interpreted as u!





First-Order Logic of Graphs

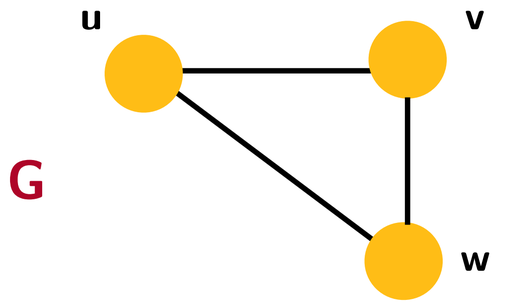
The formula $\Phi(x) = \exists y, z\ E(x, y) \wedge E(y, z) \wedge E(x, z)$ is in the language of graphs, where $E(x, y)$ means that there is an edge between the nodes interpreting x and y.

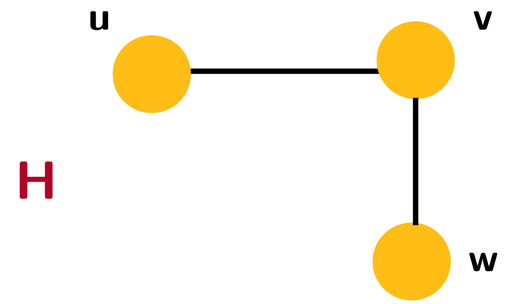
Graphs as interpretations: View the graphs G and H as interpretations over a domain $\{u, v, w\}$. Then:

- $G \vDash \Phi(u)$
- $H \nvDash \Phi(u)$

The graph G is a model of $\Phi(x)$ when x is interpreted as u!

Intuitively, any graph relative to a node a which takes part in a triangle is a model of $\Phi(a)$.





Logic of Coloured Graphs

Logic of Coloured Graphs

Let us focus on formulas of the form $\Phi(x)$ in the language of graphs, i.e., formulas with one free variable, where each free variable will be instantiated with a node in the graph.

Logic of Coloured Graphs

Let us focus on formulas of the form $\Phi(x)$ in the language of graphs, i.e., formulas with one free variable, where each free variable will be instantiated with a node in the graph.

Given a graph $G = (V, E)$, we write $G \vDash \Phi(u)$ for some $u \in V_G$ to mean that the graph G satisfies $\Phi(x)$ when interpreting the free variable x as the node u.

Logic of Coloured Graphs

Let us focus on formulas of the form $\Phi(x)$ in the language of graphs, i.e., formulas with one free variable, where each free variable will be instantiated with a node in the graph.

Given a graph $G = (V, E)$, we write $G \models \Phi(u)$ for some $u \in V_G$ to mean that the graph G satisfies $\Phi(x)$ when interpreting the free variable x as the node u.

We will also consider formulas in the language of coloured graphs, where in addition to the binary edge relation we also have unary relations, that is, sets of nodes, which we may view as colours of the nodes.

Logic of Coloured Graphs

Let us focus on formulas of the form $\Phi(x)$ in the language of graphs, i.e., formulas with one free variable, where each free variable will be instantiated with a node in the graph.

Given a graph $G = (V, E)$, we write $G \vDash \Phi(u)$ for some $u \in V_G$ to mean that the graph G satisfies $\Phi(x)$ when interpreting the free variable x as the node u.

We will also consider formulas in the language of coloured graphs, where in addition to the binary edge relation we also have unary relations, that is, sets of nodes, which we may view as colours of the nodes.

Consider, for example, the formula:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y \Big(E(x, y) \wedge \mathit{Blue}(y) \wedge \exists z \Big(E(x, z) \wedge \mathit{Green}(z) \Big) \Big).$$

This formula is satisfied by nodes nodes $u \in V_G$ such that $G \vDash \Psi(u)$, i.e., red nodes that are connected to a blue and a green node.

Two-Variable Fragment of First-Order Logic

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

For example the formula from earlier is in FO^3:

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

For example the formula from earlier is in FO^3:

$$\Psi(x) = Red(x) \wedge \exists y\Big(E(x,y) \wedge Blue(y) \wedge \exists z\Big(E(x,z) \wedge Green(z)\Big)\Big)$$

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

For example the formula from earlier is in FO^3:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y\Big(E(x,y) \wedge \mathit{Blue}(y) \wedge \exists z\Big(E(x,z) \wedge \mathit{Green}(z)\Big)\Big)$$

Observe that reducing the number of variables used in formulas can severely reduce their expressive power. However, such fragments are still quite expressive, as we can re-use variables in different quantifier scopes!

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

For example the formula from earlier is in FO^3:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y\Big(E(x,y) \wedge \mathit{Blue}(y) \wedge \exists z\Big(E(x,z) \wedge \mathit{Green}(z)\Big)\Big)$$

Observe that reducing the number of variables used in formulas can severely reduce their expressive power. However, such fragments are still quite expressive, as we can re-use variables in different quantifier scopes!

For example, $\Psi(x)$ can be equivalently written (by re-using the variable y in place of z) as an FO^2 formula:

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

For example the formula from earlier is in FO^3:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y\Big(E(x,y) \wedge \mathit{Blue}(y) \wedge \exists z\Big(E(x,z) \wedge \mathit{Green}(z)\Big)\Big)$$

Observe that reducing the number of variables used in formulas can severely reduce their expressive power. However, such fragments are still quite expressive, as we can re-use variables in different quantifier scopes!

For example, $\Psi(x)$ can be equivalently written (by re-using the variable y in place of z) as an FO^2 formula:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y\Big(E(x,y) \wedge \mathit{Blue}(y) \wedge \exists y\Big(E(x,y) \wedge \mathit{Green}(y)\Big)\Big)$$

Two-Variable Fragment of First-Order Logic

We write FO^k to denote FO with at most k variables, i.e., the k-variable fragment of first-order logic.

For example the formula from earlier is in FO^3:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y\Big(E(x,y) \wedge \mathit{Blue}(y) \wedge \exists z\Big(E(x,z) \wedge \mathit{Green}(z)\Big)\Big)$$

Observe that reducing the number of variables used in formulas can severely reduce their expressive power. However, such fragments are still quite expressive, as we can re-use variables in different quantifier scopes!

For example, $\Psi(x)$ can be equivalently written (by re-using the variable y in place of z) as an FO^2 formula:

$$\Psi(x) = \mathit{Red}(x) \wedge \exists y\Big(E(x,y) \wedge \mathit{Blue}(y) \wedge \exists y\Big(E(x,y) \wedge \mathit{Green}(y)\Big)\Big)$$

This works, intuitively, because the variables refer to different things in the scope of different quantifiers. This trick is not always possible: Indeed FO^2 is strictly contained in FO, i.e, there are formulas in FO that cannot be expressed in FO^2.

Two-Variable Fragment with Counting Quantifier

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^{k} denotes the k-variable fragment of C.

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^k denotes the k-variable fragment of C.

For example, the following formula is in C^3 and so in C:

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^k denotes the k-variable fragment of C.

For example, the following formula is in C^3 and so in C:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} z E(y, z) \big)$$

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^k denotes the k-variable fragment of C.

For example, the following formula is in C^3 and so in C:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} z E(y, z) \big)$$

This formula can also be expressed in C^2:

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^k denotes the k-variable fragment of C.

For example, the following formula is in C^3 and so in C:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} z E(y, z) \big)$$

This formula can also be expressed in C^2:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} x E(y, x) \big).$$

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^k denotes the k-variable fragment of C.

For example, the following formula is in C^3 and so in C:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} z E(y, z) \big)$$

This formula can also be expressed in C^2:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} x E(y, x) \big).$$

Intuitively, a graph G satisfies this formula with a node v if and only if v has at most 2 red neighbours in G that have degree at least 5.

Two-Variable Fragment with Counting Quantifier

C extends first-order logic with counting quantifiers of the form $\exists^{\geq k} x$ for $k \geq 0$, where $\exists^{\geq k} x \Phi(x)$ means that there are at least k elements x satisfying Φ. As before, C^k denotes the k-variable fragment of C.

For example, the following formula is in C^3 and so in C:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} z E(y, z) \big)$$

This formula can also be expressed in C^2:

$$\Theta(x) = \neg \exists^{\geq 3} y \big(Red(y) \wedge E(x, y) \wedge \exists^{\geq 5} x E(y, x) \big).$$

Intuitively, a graph G satisfies this formula with a node v if and only if v has at most 2 red neighbours in G that have degree at least 5.

It is well-known that C is only a syntactic extension of FO, as counting quantifiers of the form $\exists^{\geq k} x$ can be simulated with standard existential quantifiers, and using k variables. However, counting quantifiers add expressiveness if we restrict the number of variables.

Logical Characterisation of MPNNs

A Logical Characterisation for MPNNs

A Logical Characterisation for MPNNs

We are interested in the following question:

A Logical Characterisation for MPNNs

We are interested in the following question:

What is the class of functions that is captured by MPNNs and can we logically characterise these?

A Logical Characterisation for MPNNs

We are interested in the following question:

What is the class of functions that is captured by MPNNs and can we logically characterise these?

Let us focus on node classification and on Boolean functions.

A Logical Characterisation for MPNNs

We are interested in the following question:

What is the class of functions that is captured by MPNNs and can we logically characterise these?

Let us focus on node classification and on Boolean functions.

Formally, a logical node classifier is given by a formula $\Phi(x)$ in C^2 with exactly one free variable. Indeed, such a formula can be viewed as a Boolean function for each particular choice of node $u \in V_G$, that is, $\Phi(u) : V_G \mapsto \mathbb{B}$ mapping the node to **true** or **false**.

A Logical Characterisation for MPNNs

We are interested in the following question:

What is the class of functions that is captured by MPNNs and can we logically characterise these?

Let us focus on node classification and on Boolean functions.

Formally, a logical node classifier is given by a formula $\Phi(x)$ in C^2 with exactly one free variable. Indeed, such a formula can be viewed as a Boolean function for each particular choice of node $u \in V_G$, that is, $\Phi(u) : V_G \mapsto \mathbb{B}$ mapping the node to **true** or **false**.

Following (Barcelo et al, 2020), we say that an MPNN classifier captures a logical classifier when both classifiers coincide over every node in every possible input graph.

A Logical Characterisation for MPNNs

We are interested in the following question:

What is the class of functions that is captured by MPNNs and can we logically characterise these?

Let us focus on node classification and on Boolean functions.

Formally, a logical node classifier is given by a formula $\Phi(x)$ in C^2 with exactly one free variable. Indeed, such a formula can be viewed as a Boolean function for each particular choice of node $u \in V_G$, that is, $\Phi(u) : V_G \mapsto \mathbb{B}$ mapping the node to **true** or **false**.

Following (Barcelo et al, 2020), we say that an MPNN classifier captures a logical classifier when both classifiers coincide over every node in every possible input graph.

Formally, an MPNN classifier M captures a logical classifier $\Phi(x)$ if for every graph G and node u in G, it holds that $M(G, v)$ evaluates to true if and only if $G \vDash \Phi(u)$.

A Logical Characterisation for MPNNs

We are interested in the following question:

What is the class of functions that is captured by MPNNs and can we logically characterise these?

Let us focus on node classification and on Boolean functions.

Formally, a logical node classifier is given by a formula $\Phi(x)$ in C^2 with exactly one free variable. Indeed, such a formula can be viewed as a Boolean function for each particular choice of node $u \in V_G$, that is, $\Phi(u) : V_G \mapsto \mathbb{B}$ mapping the node to **true** or **false**.

Following (Barcelo et al, 2020), we say that an MPNN classifier captures a logical classifier when both classifiers coincide over every node in every possible input graph.

Formally, an MPNN classifier M captures a logical classifier $\Phi(x)$ if for every graph G and node u in G, it holds that $M(G, v)$ evaluates to true if and only if $G \vDash \Phi(u)$.

Our goal is to identify a logic that is captured by MPNNs — identifying the expressive power of MPNNs.

A Logical Characterisation for MPNNs

A Logical Characterisation for MPNNs

Proposition (Morris et al., 2019; Xu et al., 2019). Two graphs G and H are indistinguishable by all MPNNs if and only if they satisfy the same C^2 -sentences.

A Logical Characterisation for MPNNs

Proposition (Morris et al., 2019; Xu et al., 2019). Two graphs G and H are indistinguishable by all MPNNs if and only if they satisfy the same C^2 -sentences.

One may be tempted to think that this result already entails that MPNNs can capture C^2. The subtlety is that this result focuses on graph/node distinguishability, which is crucial to identify the class of functions that are captured by MPNNs, but it is not sufficient to characterise the class of functions that are captured.

A Logical Characterisation for MPNNs

Proposition (Morris et al., 2019; Xu et al., 2019). Two graphs G and H are indistinguishable by all MPNNs if and only if they satisfy the same C^2 -sentences.

One may be tempted to think that this result already entails that MPNNs can capture C^2. The subtlety is that this result focuses on graph/node distinguishability, which is crucial to identify the class of functions that are captured by MPNNs, but it is not sufficient to characterise the class of functions that are captured.

Recall that the above result holds already for MPNNs without any readouts. There are, however, many C^2 node classifiers that cannot be expressed by MPNNs without any readouts — called aggregate-combine GNN (AC-GNN) in the following:

A Logical Characterisation for MPNNs

Proposition (Morris et al., 2019; Xu et al., 2019). Two graphs G and H are indistinguishable by all MPNNs if and only if they satisfy the same C^2 -sentences.

One may be tempted to think that this result already entails that MPNNs can capture C^2. The subtlety is that this result focuses on graph/node distinguishability, which is crucial to identify the class of functions that are captured by MPNNs, but it is not sufficient to characterise the class of functions that are captured.

Recall that the above result holds already for MPNNs without any readouts. There are, however, many C^2 node classifiers that cannot be expressed by MPNNs without any readouts — called aggregate-combine GNN (AC-GNN) in the following:

"...there are AC-GNNs that can reproduce the WL labelling, and hence AC-GNNs can be as powerful as the WL test for distinguishing nodes. This does not imply, however, that AC-GNNs can capture every node classifier—that is, a function assigning true or false to every node — that is refined by the WL test. In fact, it is not difficult to see that there are many such classifiers that cannot be captured by AC-GNNs; one simple example is a classifier assigning true to every node if and only if the graph has an isolated node."

(Barcelo et al., 2020)

A Logical Characterisation for MPNNs

A Logical Characterisation for MPNNs

For example, MPNNs without any readouts cannot capture the function described by the following formula (Barcelo et al., 2020):

$$\gamma(x) = Red(x) \wedge \exists y \Big(\neg E(x, y) \wedge \exists^{\geq 2} x \Big(E(y, x) \wedge Blue(x) \Big) \Big),$$

since, e.g., the red and blue nodes may be in disjoint subgraphs and never communicate.

A Logical Characterisation for MPNNs

For example, MPNNs without any readouts cannot capture the function described by the following formula (Barcelo et al., 2020):

$$\gamma(x) = Red(x) \wedge \exists y \Big(\neg E(x,y) \wedge \exists^{\geq 2} x \Big(E(y,x) \wedge Blue(x) \Big) \Big),$$

since, e.g., the red and blue nodes may be in disjoint subgraphs and never communicate.

It turns out that MPNNs without any readouts can capture graded modal logic, a strict subset of C^2.

A Logical Characterisation for MPNNs

For example, MPNNs without any readouts cannot capture the function described by the following formula (Barcelo et al., 2020):

$$\gamma(x) = Red(x) \wedge \exists y \Big(\neg E(x,y) \wedge \exists^{\geq 2} x \Big(E(y,x) \wedge Blue(x) \Big) \Big),$$

since, e.g., the red and blue nodes may be in disjoint subgraphs and never communicate.

It turns out that MPNNs without any readouts can capture graded modal logic, a strict subset of C^2.

This brings up a natural question: Is there a class of MPNNs that can capture C^2?

A Logical Characterisation for MPNNs

For example, MPNNs without any readouts cannot capture the function described by the following formula (Barcelo et al., 2020):

$$\gamma(x) = Red(x) \wedge \exists y \Big(\neg E(x,y) \wedge \exists^{\geq 2} x \Big(E(y,x) \wedge Blue(x) \Big) \Big),$$

since, e.g., the red and blue nodes may be in disjoint subgraphs and never communicate.

It turns out that MPNNs without any readouts can capture graded modal logic, a strict subset of C^2.

This brings up a natural question: Is there a class of MPNNs that can capture C^2?

An obvious candidate is MPNNs with global readout in the sense we defined earlier, i.e., there is a global feature computation happening at every layer.

A Logical Characterisation for MPNNs

A Logical Characterisation for MPNNs

Theorem (Barcelo et al., 2020). Each C^2 classifier can be captured by an MPNN with global readout.

A Logical Characterisation for MPNNs

Theorem (Barcelo et al., 2020). Each C^2 classifier can be captured by an MPNN with global readout.

This theorem is further strengthened, as this result holds even if we assume that the MPNN is homogeneous.

A Logical Characterisation for MPNNs

Theorem (Barcelo et al., 2020). Each C^2 classifier can be captured by an MPNN with global readout.

This theorem is further strengthened, as this result holds even if we assume that the MPNN is homogeneous.

The result has implications on the size of the MPNN:

A Logical Characterisation for MPNNs

Theorem (Barcelo et al., 2020). Each C^2 classifier can be captured by an MPNN with global readout.

This theorem is further strengthened, as this result holds even if we assume that the MPNN is homogeneous.

The result has implications on the size of the MPNN:

The depth of the MPNN is bounded by the quantifier depth of the formula that corresponds to the target function, where quantifier depth is measured in the size of quantifiers as well as constructors!

A Logical Characterisation for MPNNs

Theorem (Barcelo et al., 2020). Each C^2 classifier can be captured by an MPNN with global readout.

This theorem is further strengthened, as this result holds even if we assume that the MPNN is homogeneous.

The result has implications on the size of the MPNN:

The depth of the MPNN is bounded by the quantifier depth of the formula that corresponds to the target function, where quantifier depth is measured in the size of quantifiers as well as constructors!

This opens up new perspectives, as we can formally study ,e.g., the class of functions can be captured by MPNNs with restrictions on their size.

A Logical Characterisation for MPNNs

Theorem (Barcelo et al., 2020). Each C^2 classifier can be captured by an MPNN with global readout.

This theorem is further strengthened, as this result holds even if we assume that the MPNN is homogeneous.

The result has implications on the size of the MPNN:

The depth of the MPNN is bounded by the quantifier depth of the formula that corresponds to the target function, where quantifier depth is measured in the size of quantifiers as well as constructors!

This opens up new perspectives, as we can formally study ,e.g., the class of functions can be captured by MPNNs with restrictions on their size.

This result is also strengthened in another direction: It holds also for MPNNs with a single (final) global readout, but in this case we require MPNN to be non-homogeneous.

A Logical Characterisation for MPNNs

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

The proof shows how to simulate a C^2 sentence with MPNNs intuitively following the roadmap:

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

The proof shows how to simulate a C^2 sentence with MPNNs intuitively following the roadmap:

- Enumerate all sub-formulas $(\phi_1, \ldots, \phi_L)$ of a given formula Φ, such that $\Phi = \phi_L$

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

The proof shows how to simulate a C^2 sentence with MPNNs intuitively following the roadmap:

- Enumerate all sub-formulas $(\phi_1, \ldots, \phi_L)$ of a given formula Φ, such that $\Phi = \phi_L$
- Define an MPNN M_Φ with feature vectors in $\mathbb{R}^L$ such that every component of those vectors represents a different sub-formula.

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

The proof shows how to simulate a C^2 sentence with MPNNs intuitively following the roadmap:

- Enumerate all sub-formulas $(\phi_1, \ldots, \phi_L)$ of a given formula Φ, such that $\Phi = \phi_L$
- Define an MPNN M_Φ with feature vectors in $\mathbb{R}^L$ such that every component of those vectors represents a different sub-formula.
- M_Φ updates the feature vector $\mathbf{x}_u$ of node u ensuring that its component corresponding to the sub-formula ϕ_i gets a value 1 if and only if the sub-formula ϕ_i is satisfied in node u.

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

The proof shows how to simulate a C^2 sentence with MPNNs intuitively following the roadmap:

- Enumerate all sub-formulas $(\phi_1, \ldots, \phi_L)$ of a given formula Φ, such that $\Phi = \phi_L$
- Define an MPNN M_Φ with feature vectors in $\mathbb{R}^L$ such that every component of those vectors represents a different sub-formula.
- M_Φ updates the feature vector $\mathbf{x}_u$ of node u ensuring that its component corresponding to the sub-formula ϕ_i gets a value 1 if and only if the sub-formula ϕ_i is satisfied in node u.

The precise construction establishes the described correspondences, from which the result can be derived.

A Logical Characterisation for MPNNs

Overall, (Barcelo et al., 2020) strengthened the earlier results of (Morris et al., 2019; Xu et al., 2019) and showed that every C^2-sentence can be simulated by an MPNN with a global readout.

The proof shows how to simulate a C^2 sentence with MPNNs intuitively following the roadmap:

- Enumerate all sub-formulas $(\phi_1, \ldots, \phi_L)$ of a given formula Φ, such that $\Phi = \phi_L$
- Define an MPNN M_Φ with feature vectors in $\mathbb{R}^L$ such that every component of those vectors represents a different sub-formula.
- M_Φ updates the feature vector $\mathbf{x}_u$ of node u ensuring that its component corresponding to the sub-formula ϕ_i gets a value 1 if and only if the sub-formula ϕ_i is satisfied in node u.

The precise construction establishes the described correspondences, from which the result can be derived.

This result is not complete: MPNNs with global readout can capture C^2, but is this all what MPNNs can capture? Is there a logic which MPNNs with global readout can capture, but cannot capture anything beyond?

Summary

Summary

- Model representation capacity & expressive power

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test
- MPNNs with injective aggregation and combine functions are as powerful as 1-WL test.

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test
- MPNNs with injective aggregation and combine functions are as powerful as 1-WL test.
- The logic of graphs: FO, C, FO^2, C^2 — an interesting connection to descriptive complexity!

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test
- MPNNs with injective aggregation and combine functions are as powerful as 1-WL test.
- The logic of graphs: FO, C, FO^2, C^2 — an interesting connection to descriptive complexity!
- Logical characterisation of MPNNs

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test
- MPNNs with injective aggregation and combine functions are as powerful as 1-WL test.
- The logic of graphs: FO, C, FO^2, C^2 — an interesting connection to descriptive complexity!
- Logical characterisation of MPNNs
  - Each C^2 classifier can be captured by an MPNNs with global readout (even with a final readout)!

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test
- MPNNs with injective aggregation and combine functions are as powerful as 1-WL test.
- The logic of graphs: FO, C, FO^2, C^2 — an interesting connection to descriptive complexity!
- Logical characterisation of MPNNs
  - Each C^2 classifier can be captured by an MPNNs with global readout (even with a final readout)!
  - MPNNs without global readout cannot capture C^2, but can capture graded model logic.

Summary

- Model representation capacity & expressive power
- Graph isomorphism, colour refinement, 1-WL
- MPNNs are at most as powerful as 1-WL test
- MPNNs with injective aggregation and combine functions are as powerful as 1-WL test.
- The logic of graphs: FO, C, FO^2, C^2 — an interesting connection to descriptive complexity!
- Logical characterisation of MPNNs
  - Each C^2 classifier can be captured by an MPNNs with global readout (even with a final readout)!
  - MPNNs without global readout cannot capture C^2, but can capture graded model logic.
- We have not discussed the practical implications of the limitations in expressive power, and neither the proposed tools to address such limitations — Lecture 6 & 7.

References

- 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.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.
- P. Barcelo, E. Kostylev, M. Monet, J. Perez, J. Reutter, and J. Silva. The logical expressiveness of graph neural networks. *ICLR*, 2020.
- G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- K. Hornik, M. Stinchcombe, H. White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989
- Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989

References

- L. Babai, Graph isomorphism in quasipolynomial time, arXiv:1512.03547, 2016.
- N. Immermann, Descriptive Complexity. 1999.
- J. Cai, M. Furer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. Combinatorica, 12(4):389–410, 1992.