RWTH AACHEN UNIVERSITY

Logic and the Power of Recurrent Graph Neural Networks

Martin Grohe

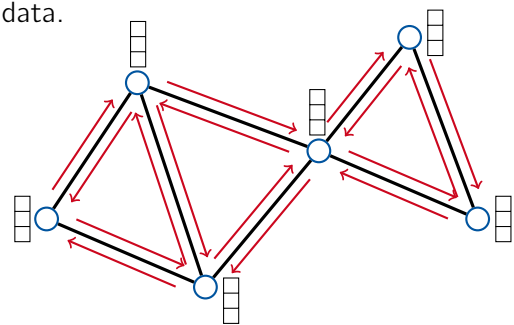
Graph Neural Networks

Graph Neural Networks

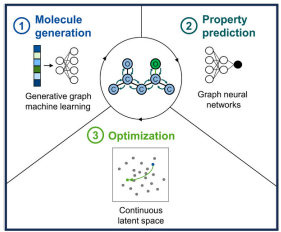
- Graph neural networks (GNNs) are deep learning architectures for machine learning problems on graphs.

Graph Neural Networks

- Graph neural networks (GNNs) are deep learning architectures for machine learning problems on graphs.
- A GNN may be viewed as distributed message passing algorithm operating on the vertices of its input graph. The algorithm is controlled by parameters typically learned from data.

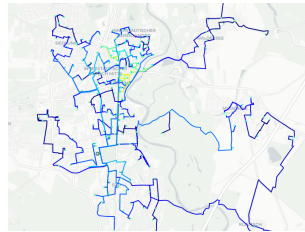


Example 1: Molecular design



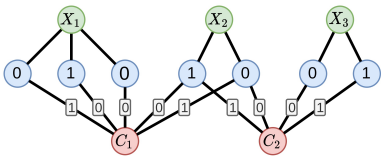
Rittig et al.: Graph machine learning for design of high-octane fuels. AIChe Journal, 2023. Doi: 10.1002/aic.17971

Example 2: Power flow in electrical grids

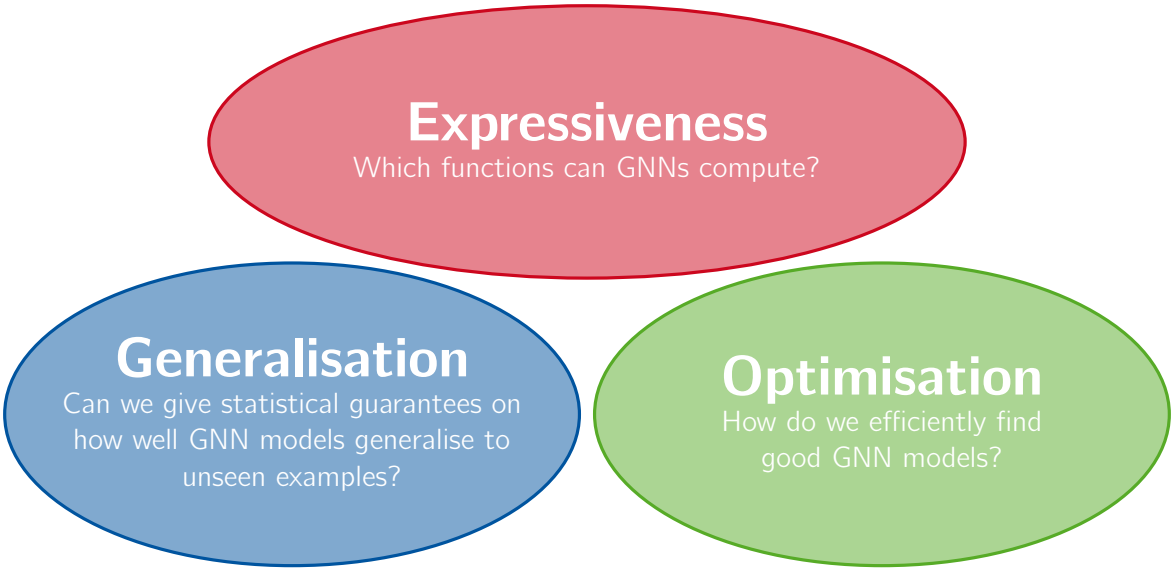


Böttcher et al.: Solving AC Power Flow with Graph Neural Networks under Realistic Constraints. Belgrade Power Tech, 2023. Doi: 10.1109/PowerTech55446.2023.10202246

Example 3: Constraint Satisfaction

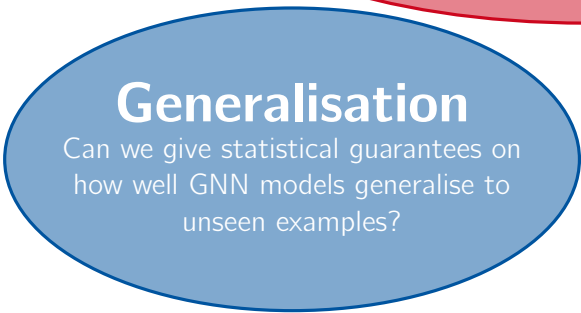


Tönshoff et al.: One Model, Any CSP: Graph Neural Networks as Fast Global Search Heuristics for Constraint Satisfaction. IJCAI, 2023. Doi: 10.24963/ijcai.2023/476


Expressiveness
Which functions can GNNs compute?
Generalisation
Can we give statistical guarantees on how well GNN models generalise to unseen examples?
Optimisation
How do we efficiently find good GNN models?

This talk


Expressiveness
Which functions can GNNs compute?

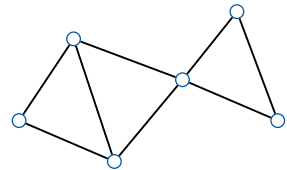

Generalisation
Can we give statistical guarantees on how well GNN models generalise to unseen examples?


Optimisation
How do we efficiently find good GNN models?

Computation of GNNs

GNN N with d layers maps graph G to sequence of signals

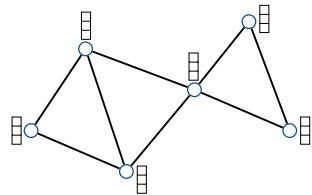
$$\mathcal{I}^{(t)} : V(G) \to \mathbb{R}^{p_t} \quad \text{for } t = 0, \ldots, d$$



Computation of GNNs

GNN N with d layers maps graph G to sequence of signals

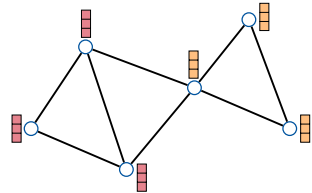
$$\mathcal{I}^{(t)} : V(G) \to \mathbb{R}^{p_t} \quad \text{for } t = 0, \ldots, d$$



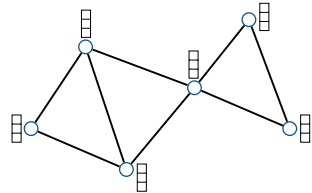
Computation of GNNs

GNN N with d layers maps graph G to sequence of signals

$$\mathcal{I}^{(t)} : V(G) \to \mathbb{R}^{p_t} \quad \text{for } t = 0, \ldots, d$$



Computation of GNNs

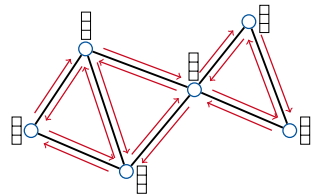


GNN N with d layers maps graph G to sequence of signals

$$\mathcal{I}^{(t)} : V(G) \to \mathbb{R}^{p_t} \quad \text{for } t = 0, \ldots, d$$

Initialisation: $\mathcal{I}^{(0)}(v) \in \mathbb{R}^{p_0}$ encodes node labels or initial signal on the graph.

Computation of GNNs



GNN N with d layers maps graph G to sequence of signals

$$\mathcal{I}^{(t)} : V(G) \to \mathbb{R}^{p_t} \quad \text{for } t = 0, \ldots, d$$

Initialisation: $\mathcal{I}^{(0)}(v) \in \mathbb{R}^{p_0}$ encodes node labels or initial signal on the graph.

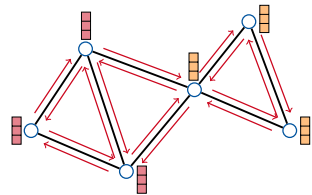
Aggregation: $\mathcal{A}^{(t)}(v) := \mathrm{agg}_t\Big(\big\{\!\!\big\{ \mathcal{I}^{(t-1)}(w) \;\big|\; w \in N_G(v) \big\}\!\!\big\}\Big)$

- agg_t aggregation function: coordinatewise sum, mean or max

Computation of GNNs

GNN N with d layers maps graph G to sequence of signals

$$\mathcal{I}^{(t)} : V(G) \to \mathbb{R}^{p_t} \quad \text{for } t = 0, \ldots, d$$



Initialisation: $\mathcal{I}^{(0)}(v) \in \mathbb{R}^{p_0}$ encodes node labels or initial signal on the graph.

Aggregation: $\mathcal{A}^{(t)}(v) := \mathrm{agg}_t\Big(\big\{\!\!\big\{ \mathcal{I}^{(t-1)}(w) \;\big|\; w \in N_G(v) \big\}\!\!\big\}\Big)$

- agg_t aggregation function: coordinatewise sum, mean or max

Combination: $\mathcal{I}^{(t)}(v) := \mathrm{comb}_t\big(\mathcal{I}^{(t-1)}(v), \mathcal{A}^{(t)}(v)\big)$

- $\mathrm{comb}_t : \mathbb{R}^{2p_{t-1}} \to \mathbb{R}^{p_t}$ combination function computed by a feedforward neural network (a.k.a. multilayer perceptron)

Global Aggregation (a.k.a. Virtual Node)

$$\mathscr{g}^{(t)} := \mathrm{agg}'_t\left(\{\!\{\mathscr{I}^{(t-1)}(w) \mid w \in V(G)\}\!\}\right),$$

where the aggregation function agg'_t is coordinatewise sum, mean or max.

Global Aggregation (a.k.a. Virtual Node)

$$\mathscr{g}^{(t)} \coloneqq \mathrm{agg}'_t\big(\{\!\{\mathscr{z}^{(t-1)}(w) \mid w \in V(G)\}\!\}\big),$$

where the aggregation function agg'_t is coordinatewise sum, mean or max.

Then combination becomes

$$\mathscr{z}^{(t)}(v) \coloneqq \mathrm{comb}_t\big(\mathscr{z}^{(t-1)}(v), \mathscr{a}^{(t)}(v), \mathscr{g}^{(t)}\big)$$

with $\mathrm{comb}_t : \mathbb{R}^{3p_{t-1}} \to \mathbb{R}^{p_t}$ computed by feedforward neural network.

Global Aggregation (a.k.a. Virtual Node)

$$\mathscr{g}^{(t)} := \mathrm{agg}'_t\big(\{\!\{\mathscr{l}^{(t-1)}(w) \mid w \in V(G)\}\!\}\big),$$

where the aggregation function agg'_t is coordinatewise sum, mean or max.

Then combination becomes

$$\mathscr{l}^{(t)}(v) := \mathrm{comb}_t\big(\mathscr{l}^{(t-1)}(v), \mathscr{a}^{(t)}(v), \mathscr{g}^{(t)}\big)$$

with $\mathrm{comb}_t : \mathbb{R}^{3p_{t-1}} \to \mathbb{R}^{p_t}$ computed by feedforward neural network.

Random Initialisation

$\mathscr{l}^{(0)}(v) = (s_1, \ldots, s_{p_0}, s_{p_0+1}) \in \mathbb{R}^{p_0+1}$ where $(s_1, \ldots, s_{p_0})$ endodes node labels or initial signal and s_{p_0+1} is chosen uniformly at random from $[0, 1]$.

Global Aggregation (a.k.a. Virtual Node)

$$\mathscr{g}^{(t)} \coloneqq \mathrm{agg}'_t\big(\{\!\{\mathscr{f}^{(t-1)}(w) \mid w \in V(G)\}\!\}\big),$$

where the aggregation function agg'_t is coordinatewise sum, mean or max.

Then combination becomes

$$\mathscr{f}^{(t)}(v) \coloneqq \mathrm{comb}_t\big(\mathscr{f}^{(t-1)}(v), \mathscr{a}^{(t)}(v), \mathscr{g}^{(t)}\big)$$

with $\mathrm{comb}_t : \mathbb{R}^{3p_{t-1}} \to \mathbb{R}^{p_t}$ computed by feedforward neural network.

Random Initialisation

$\mathscr{f}^{(0)}(v) = (s_1, \ldots, s_{p_0}, s_{p_0+1}) \in \mathbb{R}^{p_0+1}$ where $(s_1, \ldots, s_{p_0})$ endodes node labels or initial signal and s_{p_0+1} is chosen uniformly at random from $[0, 1]$.

In this talk, we always consider GNNs with global aggregation. We sometimes consider random initialisation, but mention it explicitly when we do.

Node-Level Functions

GNNs compute a function $\mathscr{f}$ that maps each graph G (possibly with node labels or an initial signal) to a signal $\mathscr{f}(G, \cdot) : V(G) \to \mathbb{R}^q$ defined by

$$\mathscr{f}(G, v) = \mathscr{z}^{(d)}(v).$$

Functions Computed by GNNs

Node-Level Functions

GNNs compute a function $\mathscr{f}$ that maps each graph G (possibly with node labels or an initial signal) to a signal $\mathscr{f}(G, \cdot) : V(G) \to \mathbb{R}^q$ defined by

$$\mathscr{f}(G, v) = \mathscr{z}^{(d)}(v).$$

Graph-Level Functions

To compute a function $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ mapping graphs to vectors of real numbers,

Functions Computed by GNNs

Node-Level Functions

GNNs compute a function $\mathscr{f}$ that maps each graph G (possibly with node labels or an initial signal) to a signal $\mathscr{f}(G, \cdot) : V(G) \to \mathbb{R}^q$ defined by

$$\mathscr{f}(G, v) = \mathscr{z}^{(d)}(v).$$

class of all graphs

Graph-Level Functions

To compute a function $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ mapping graphs to vectors of real numbers,

Functions Computed by GNNs

Node-Level Functions

GNNs compute a function $\mathscr{f}$ that maps each graph G (possibly with node labels or an initial signal) to a signal $\mathscr{f}(G, \cdot) : V(G) \to \mathbb{R}^q$ defined by

$$\mathscr{f}(G, v) = \mathscr{z}^{(d)}(v).$$

Graph-Level Functions

To compute a function $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ mapping graphs to vectors of real numbers, we aggregate the values and then apply a readout function:

$$\mathscr{F}(G) = \mathrm{ro}\Big(\mathrm{agg}\Big(\Big\{\!\!\Big\{ \mathscr{z}^{(d)}(v) \;\Big|\; v \in V(G) \Big\}\!\!\Big\}\Big)\Big).$$

- agg aggregation function: sum, mean or max
- $\mathrm{ro} : \mathbb{R}^p \to \mathbb{R}^q$ readout function computed by a feedforward neural network

Invariance

In this talk, we focus on graph-level functions.

In this talk, we focus on graph-level functions.

Invariance

Let $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ be computed by a GNN. Then for all isomorphic graphs G, H,

$$\mathscr{F}(G) = \mathscr{F}(H).$$

In this talk, we focus on graph-level functions.

Invariance

Let $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ be computed by a GNN. Then for all isomorphic graphs G, H,

$$\mathscr{F}(G) = \mathscr{F}(H).$$

Graph Properties

When comparing GNNs with logic and circuit complexity, we often look at (isomorphism invariant) Boolean functions $\mathscr{P} : \mathscr{G} \to \{0, 1\}$, that is, properties of graphs.

In this talk, we focus on graph-level functions.

Invariance

Let $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ be computed by a GNN. Then for all isomorphic graphs G, H,

$$\mathscr{F}(G) = \mathscr{F}(H).$$

Graph Properties

When comparing GNNs with logic and circuit complexity, we often look at (isomorphism invariant) Boolean functions $\mathscr{P} : \mathscr{G} \to \{0, 1\}$, that is, properties of graphs.

Remark

GNNs with random initialisation compute an isomorphism invariant random variable.

In this talk, we focus on graph-level functions.

Invariance

Let $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ be computed by a GNN. Then for all isomorphic graphs G, H,

$$\mathscr{F}(G) = \mathscr{F}(H).$$

Graph Properties

When comparing GNNs with logic and circuit complexity, we often look at (isomorphism invariant) Boolean functions $\mathscr{P} : \mathscr{G} \to \{0, 1\}$, that is, properties of graphs.

Remark

GNNs with random initialisation compute an isomorphism invariant random variable.

To compute a function, we can run the GNN several times with independent random initialisations and then take the average value, or for Boolean functions, the majority.

Digression: Colour Refinement and Weisfeiler-Leman

A Simple Combinatorial Algorithm

The colour refinement (CR) algorithm iteratively computes a colouring of the vertices of graph G.

A Simple Combinatorial Algorithm

The colour refinement (CR) algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

A Simple Combinatorial Algorithm

The colour refinement (CR) algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

Refinement Step Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.

A Simple Combinatorial Algorithm

The colour refinement (CR) algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

Refinement Step Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.

Refinement is repeated until colouring stays stable.

A Simple Combinatorial Algorithm

The colour refinement (CR) algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

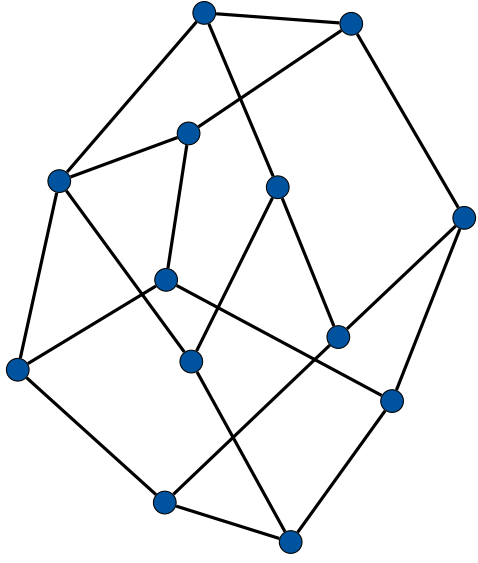
Refinement Step Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.

Refinement is repeated until colouring stays stable.

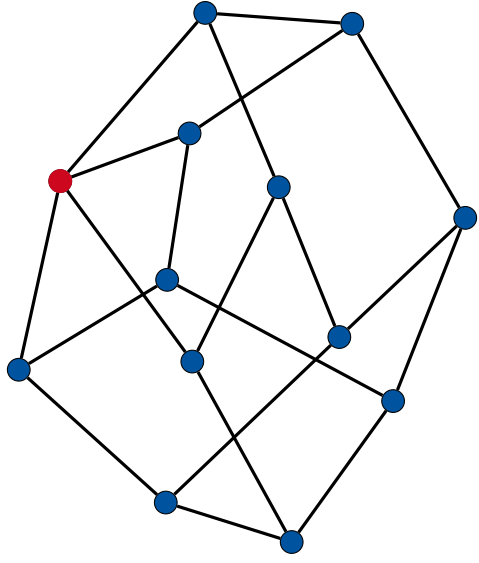
Remark

The algorithm is also referred to as naive vertex classification and as 1-dimensional Weisfeiler-Leman algorithm. One needs to be careful though, because there are two slightly different versions of the algorithm, and this difference sometimes plays a role.

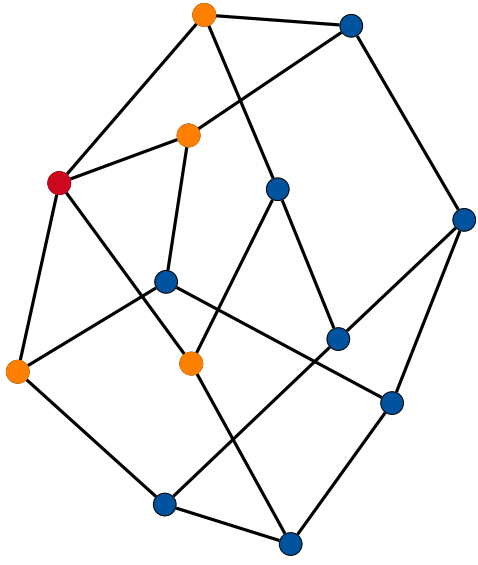
Example



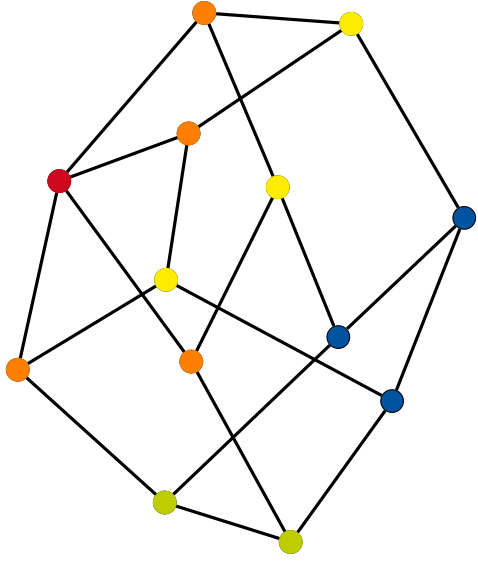
Example



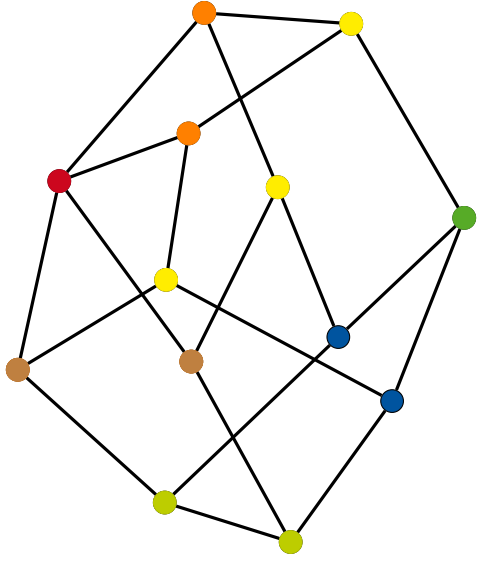
Example



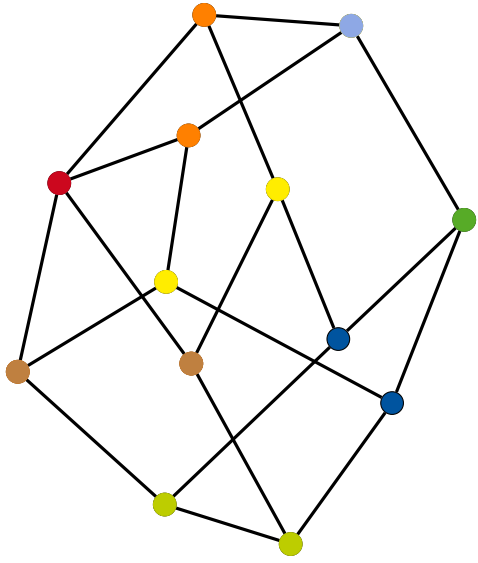
Example



Example



Example



CR as an Isomorphism Test

CR distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

CR as an Isomorphism Test

CR distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Thus CR can be used as an incomplete isomorphism test.

CR as an Isomorphism Test

CR distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Thus CR can be used as an incomplete isomorphism test.

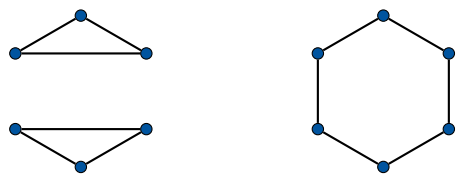
- works on almost all graphs (Babai, Erdös, Selkow 1980)

CR as an Isomorphism Test

CR distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Thus CR can be used as an incomplete isomorphism test.

- works on almost all graphs (Babai, Erdös, Selkow 1980)
- fails on some very simple graphs:



Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

For all graphs G, H, the following are equivalent:

1. *CR does not distinguish G and H;*

Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

For all graphs G, H, the following are equivalent:

1. *CR does not distinguish G and H;*
2. *G and H satisfy the same sentences of the logic C^2, the 2-variable fragment of first-order logic with counting quantifiers $\exists^{\geq n} x$.*

Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

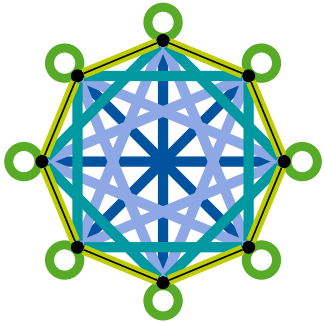
For all graphs G, H, the following are equivalent:

1. *CR does not distinguish G and H;*
2. *G and H satisfy the same sentences of the logic C^2, the 2-variable fragment of first-order logic with counting quantifiers $\exists^{\geq n} x$.*
3. *G and H are fractionally isomorphic, that is, there is a doubly stochastic matrix X such that $A_G X = X A_H$.*

Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

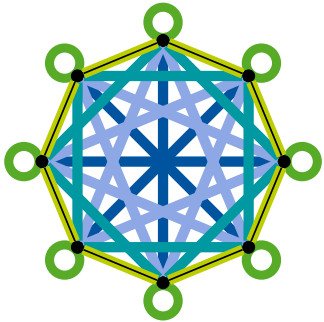
For all graphs G, H, the following are equivalent:

1. *CR does not distinguish G and H;*
2. *G and H satisfy the same sentences of the logic C^2, the 2-variable fragment of first-order logic with counting quantifiers $\exists^{\geq n} x$.*
3. *G and H are fractionally isomorphic, that is, there is a doubly stochastic matrix X such that $A_G X = X A_H$.*
4. *For all trees T, the number of homomorphisms from T to G equals the number of homomorphisms from T to H.*



Higher-Dimensional Weisfeiler-Leman

The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

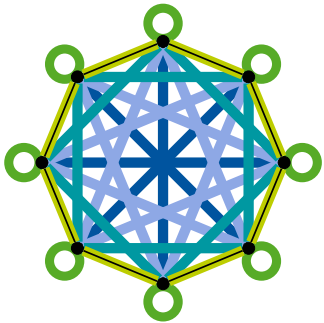


Higher-Dimensional Weisfeiler-Leman

The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

Running time: $O(n^{k+1} \log n)$

Higher-Dimensional Weisfeiler-Leman

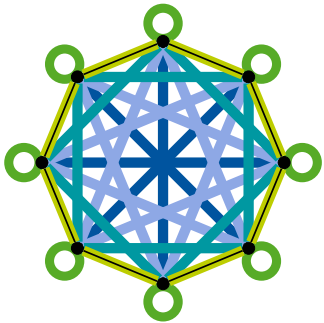


The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

Running time: $O(n^{k+1} \log n)$

- 1-WL is almost the same as Colour Refinement.

Higher-Dimensional Weisfeiler-Leman

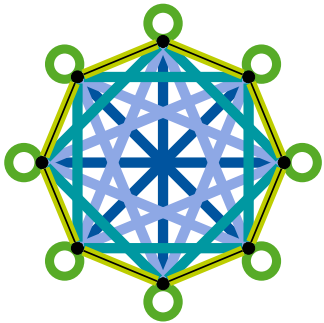


The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai ∼1980)

Running time: $O(n^{k+1} \log n)$

- 1-WL is almost the same as Colour Refinement.
- k-WL for $k \geq 2$ is much more powerful than 1-WL, but still not a complete isomorphism test: *for every k there are non-isomorphic graphs G_k, H_k of size $O(k)$ not distinguished by k-WL* (Cai, Fürer, Immerman 1991).

Higher-Dimensional Weisfeiler-Leman



The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai ∼1980)

Running time: $O(n^{k+1} \log n)$

- 1-WL is almost the same as Colour Refinement.
- k-WL for $k \geq 2$ is much more powerful than 1-WL, but still not a complete isomorphism test: *for every k there are non-isomorphic graphs G_k, H_k of size $O(k)$ not distinguished by k-WL* (Cai, Fürer, Immerman 1991).
- The characterisations of CR in terms of logic, linear (in)equalities, and homomorphism counts can be generalised to k-WL.

The Logic of GNNs

The Distinguishing Power of GNNs

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019, Xu, Hu, Leskovec, Jegelka 2019)

For all graphs G, H, the following are equivalent:

1. *G and H are distinguishable by a GNN, that is, there is a GNN computing a function $\mathscr{F}$ such that $\mathscr{F}(G) \neq \mathscr{F}(H)$;*
2. *CR distinguishes G and H.*

The Distinguishing Power of GNNs

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019, Xu, Hu, Leskovec, Jegelka 2019)

For all graphs G, H, the following are equivalent:

1. *G and H are distinguishable by a GNN, that is, there is a GNN computing a function $\mathscr{F}$ such that $\mathscr{F}(G) \neq \mathscr{F}(H)$;*
2. *CR distinguishes G and H.*

Remark

(Morris et al. 2019) ensure that graphs G, H that can be distinguished by CR can be distinguished by a GNN N of size polynomial in $|G|, |H|$.

The Distinguishing Power of GNNs

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019, Xu, Hu, Leskovec, Jegelka 2019)

For all graphs G, H, the following are equivalent:

1. *G and H are distinguishable by a GNN, that is, there is a GNN computing a function $\mathscr{F}$ such that $\mathscr{F}(G) \neq \mathscr{F}(H)$;*
2. *CR distinguishes G and H.*

Remark

(Morris et al. 2019) ensure that graphs G, H that can be distinguished by CR can be distinguished by a GNN N of size polynomial in $|G|, |H|$.

Corollary

Every function $\mathscr{F} : \mathscr{G} \to \mathbb{R}^q$ computable by a GNN is CR-invariant, that is, if CR does not distinguish G and H then $\mathscr{F}(G) = \mathscr{F}(H)$.

Expressing Graph Properties

Theorem (Barceló, Kostylev, Monet, Pérez, Reutter, Silva 2019)

Let $\mathscr{P}$ be a graph property expressible in the logic C^2. Then there is a GNN computing $\mathscr{P}$.

Theorem (Barceló, Kostylev, Monet, Pérez, Reutter, Silva 2019)

Let $\mathscr{P}$ be a graph property expressible in the logic C^2. Then there is a GNN computing $\mathscr{P}$.

Remarks

- This is a uniform expressibility result: the property can be expressed by a single GNN across input graphs of all sizes.

Expressing Graph Properties

Theorem (Barceló, Kostylev, Monet, Pérez, Reutter, Silva 2019)

Let $\mathscr{P}$ be a graph property expressible in the logic C^2. Then there is a GNN computing $\mathscr{P}$.

Remarks

- This is a uniform expressibility result: the property can be expressed by a single GNN across input graphs of all sizes.
- The converse of the theorem does not hold, not even for properties $\mathscr{P}$ that are expressible in first-order logic (Funk and Kojelis 2026).

Expressing Graph Properties

Theorem (Barceló, Kostylev, Monet, Pérez, Reutter, Silva 2019)

Let $\mathscr{P}$ be a graph property expressible in the logic C^2. Then there is a GNN computing $\mathscr{P}$.

Remarks

- This is a uniform expressibility result: the property can be expressed by a single GNN across input graphs of all sizes.
- The converse of the theorem does not hold, not even for properties $\mathscr{P}$ that are expressible in first-order logic (Funk and Kojelis 2026).
- The proof assumes piecewise linear activation functions like ReLU. It is open whether the theorem also holds for GNNs with logistic (sigmoid) or tanh activations.

Upper Bound for the Logical Expressivity of GNNs

FO^2+C: 2-variable first-order logic with counting and arithmetic on numbers

Upper Bound for the Logical Expressivity of GNNs

FO^2+C: 2-variable first-order logic with counting and arithmetic on numbers

C^2: fragment of FO^2+C with only number constants

Upper Bound for the Logical Expressivity of GNNs

FO^2+C: 2-variable first-order logic with counting and arithmetic on numbers

C^2: fragment of FO^2+C with only number constants

Theorem (G. 2023)

Let $\mathscr{P}$ be a graph property computable by a GNN with rational weights and piecewise linear activations. Then $\mathscr{P}$ is expressible in in FO^2+C.

Upper Bound for the Logical Expressivity of GNNs

FO^2+C: 2-variable first-order logic with counting and arithmetic on numbers

C^2: fragment of FO^2+C with only number constants

Theorem (G. 2023)

Let $\mathscr{P}$ be a graph property computable by a GNN with rational weights and piecewise linear activations. Then $\mathscr{P}$ is expressible in in FO^2+C.

Corollary

Combined with Barcelo et al. (2019), we get

$$\mathrm{C}^2 \quad \subset \quad \mathrm{GNN} \quad \subset \quad \mathrm{FO}^2\text{+C}.$$

Both inclusions are strict.

Theorem (G. 2023)

For all graph properties $\mathscr{P}$, the following are equivalent.

1. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with arbitrary real weights (that may use activations like sigmoid or tanh besides ReLU).*

Theorem (G. 2023)

For all graph properties $\mathscr{P}$, the following are equivalent.

1. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with arbitrary real weights (that may use activations like sigmoid or tanh besides ReLU).*
2. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with rational weights, using only sum aggregation.*

Theorem (G. 2023)

For all graph properties $\mathscr{P}$, the following are equivalent.

1. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with arbitrary real weights (that may use activations like sigmoid or tanh besides ReLU).*
2. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with rational weights, using only sum aggregation.*
3. *$\mathscr{P}$ is expressible in FO^2+C with built-in relations.*

Theorem (G. 2023)

For all graph properties $\mathscr{P}$, the following are equivalent.

1. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with arbitrary real weights (that may use activations like sigmoid or tanh besides ReLU).*
2. *$\mathscr{P}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with rational weights, using only sum aggregation.*
3. *$\mathscr{P}$ is expressible in $\mathsf{FO}^2{+}\mathsf{C}$ with built-in relations.*
4. *$\mathscr{P}$ is in the complexity class TC^0.*

Recurrent GNNs

Recurrent GNNs

So far, GNNs have a fixed number d of layers, and each layer t has its own functions agg_t, agg'_t, comb_t.

Recurrent GNNs

So far, GNNs have a fixed number d of layers, and each layer t has its own functions agg_t, agg'_t, comb_t.

Recurrent GNN Computation

A recurrent GNN has a single layer and applies it repeatedly. That is, we have aggregation functions agg, agg' and a combination function comb and for $t \geq 1$ let:

$$\mathcal{J}^{(t)}(v) = \text{comb}\Big(\mathcal{J}^{(t-1)}(v), \text{agg}\big(\{\!\{\mathcal{J}^{(t-1)}(w) \mid w \in N_G(v)\}\!\}\big), \text{agg}'\big(\{\!\{\mathcal{J}^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

So far, GNNs have a fixed number d of layers, and each layer t has its own functions agg_t, agg'_t, comb_t.

Recurrent GNN Computation

A recurrent GNN has a single layer and applies it repeatedly. That is, we have aggregation functions agg, agg′ and a combination function comb and for $t \geq 1$ let:

$$\mathcal{J}^{(t)}(v) = \text{comb}\Big(\mathcal{J}^{(t-1)}(v), \text{agg}\big(\{\!\{\mathcal{J}^{(t-1)}(w) \mid w \in N_G(v)\}\!\}\big), \text{agg}'\big(\{\!\{\mathcal{J}^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Termination and Readout

- Each node has flag (say, the last coordinate in $\mathcal{J}^{(t)}(v)$) indicating whether the local computation at the node is complete.

So far, GNNs have a fixed number d of layers, and each layer t has its own functions agg_t, agg'_t, comb_t.

Recurrent GNN Computation

A recurrent GNN has a single layer and applies it repeatedly. That is, we have aggregation functions $\mathrm{agg}, \mathrm{agg}'$ and a combination function comb and for $t \geq 1$ let:

$$\mathfrak{z}^{(t)}(v) = \mathrm{comb}\Big(\mathfrak{z}^{(t-1)}(v), \mathrm{agg}\big(\{\!\{\mathfrak{z}^{(t-1)}(w) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'\big(\{\!\{\mathfrak{z}^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Termination and Readout

- Each node has flag (say, the last coordinate in $\mathfrak{z}^{(t)}(v)$) indicating whether the local computation at the node is complete.
- The global computation halts once every node has set its flag.

So far, GNNs have a fixed number d of layers, and each layer t has its own functions agg_t, agg'_t, comb_t.

Recurrent GNN Computation

A recurrent GNN has a single layer and applies it repeatedly. That is, we have aggregation functions $\mathrm{agg}, \mathrm{agg}'$ and a combination function comb and for $t \geq 1$ let:

$$\jmath^{(t)}(v) = \mathrm{comb}\Big(\jmath^{(t-1)}(v), \mathrm{agg}\big(\{\!\{\jmath^{(t-1)}(w) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'\big(\{\!\{\jmath^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Termination and Readout

- Each node has flag (say, the last coordinate in $\jmath^{(t)}(v)$) indicating whether the local computation at the node is complete.
- The global computation halts once every node has set its flag.
- At that point, the signal is aggregated and a readout functions is applied.

So far, GNNs have a fixed number d of layers, and each layer t has its own functions agg_t, agg'_t, comb_t.

Recurrent GNN Computation

A recurrent GNN has a single layer and applies it repeatedly. That is, we have aggregation functions $\mathrm{agg}, \mathrm{agg}'$ and a combination function comb and for $t \geq 1$ let:

$$\jmath^{(t)}(v) = \mathrm{comb}\Big(\jmath^{(t-1)}(v), \mathrm{agg}\big(\{\!\{\jmath^{(t-1)}(w) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'\big(\{\!\{\jmath^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Termination and Readout

- Each node has flag (say, the last coordinate in $\jmath^{(t)}(v)$) indicating whether the local computation at the node is complete.
- The global computation halts once every node has set its flag.
- At that point, the signal is aggregated and a readout functions is applied.

We do not require convergence of the signals.

Universality of Recurrent GNNs

Theorem (Rosenbluth and G. 2026)

For every computable $\mathscr{F} : \mathscr{G} \to \mathbb{Q}^n$, the following are equivalent.

(i) *$\mathscr{F}$ is computable by a recurrent GNN.*

(ii) *$\mathscr{F}$ is CR-invariant.*

Eran Rosenbluth

Universality of Recurrent GNNs

Theorem (Rosenbluth and G. 2026)

For every computable $\mathscr{F} : \mathscr{G} \to \mathbb{Q}^n$, the following are equivalent.

(i) *$\mathscr{F}$ is computable by a recurrent GNN.*

(ii) *$\mathscr{F}$ is CR-invariant.*

Eran Rosenbluth

Furthermore,

- *we may choose the GNN in (i) to have rational weights and to only use SUM aggregation;*
- *if $\mathscr{F}$ is computable in polynomial time then the GNN stops after polynomially many iterations and only uses internal states of polynomial bitlength.*

Corollary

Every computable function $\mathscr{F} : \mathscr{G} \to \mathbb{Q}^n$ is computable by a recurrent GNN with random initialisation only using rational weights and SUM aggregation.

Corollary

Every computable function $\mathscr{F} : \mathscr{G} \to \mathbb{Q}^n$ is computable by a recurrent GNN with random initialisation only using rational weights and SUM aggregation.

Furthermore, if $\mathscr{F}$ is computable in polynomial time then the GNN runs in polynomial time and only uses internal states of polynomial bitlength.

Corollary

Every computable function $\mathscr{F} : \mathscr{G} \to \mathbb{Q}^n$ is computable by a recurrent GNN with random initialisation only using rational weights and SUM aggregation.

Furthermore, if $\mathscr{F}$ is computable in polynomial time then the GNN runs in polynomial time and only uses internal states of polynomial bitlength.

Remarks

- Global aggregation simplifies the theorem and its proof. We also have a version without global aggregation, but it requires to pass the size of the graph as part of the initial signal, and it only applies to connected graphs.

Corollary

Every computable function $\mathscr{F} : \mathscr{G} \to \mathbb{Q}^n$ is computable by a recurrent GNN with random initialisation only using rational weights and SUM aggregation.

Furthermore, if $\mathscr{F}$ is computable in polynomial time then the GNN runs in polynomial time and only uses internal states of polynomial bitlength.

Remarks

- Global aggregation simplifies the theorem and its proof. We also have a version without global aggregation, but it requires to pass the size of the graph as part of the initial signal, and it only applies to connected graphs.
- There are also a versions of the results for node-level functions (actually, we prove these first and derive the graph-level results as a consequence).

Proof of the Theorem (Outline)

Let $\mathscr{F}$ be a computable CR-invariant function.
We want to compute $\mathscr{F}(G)$ for some graph G by a recurrent GNN.

Proof of the Theorem (Outline)

Let $\mathscr{F}$ be a computable CR-invariant function.
We want to compute $\mathscr{F}(G)$ for some graph G by a recurrent GNN.

Phase 1: Computing the Colours

We simulate colour refinement. After Phase 1, the signal at every node v holds a representation D_v of v's its colour.

Proof of the Theorem (Outline)

Let $\mathscr{F}$ be a computable CR-invariant function.
We want to compute $\mathscr{F}(G)$ for some graph G by a recurrent GNN.

Phase 1: Computing the Colours

We simulate colour refinement. After Phase 1, the signal at every node v holds a representation D_v of v's its colour.

Phase 2: Inversion

Locally at every node v, from the colour D_v we compute a graph G_v that is C^2-equivalent to the input graph G.

Proof of the Theorem (Outline)

Let $\mathscr{F}$ be a computable CR-invariant function.
We want to compute $\mathscr{F}(G)$ for some graph G by a recurrent GNN.

Phase 1: Computing the Colours

We simulate colour refinement. After Phase 1, the signal at every node v holds a representation D_v of v's its colour.

Phase 2: Inversion

Locally at every node v, from the colour D_v we compute a graph G_v that is C^2-equivalent to the input graph G.

Phase 3: Computation of $\mathscr{F}$

Locally at every node v, we compute $\mathscr{F}(G_v)$. Since $\mathscr{F}$ is C^2-invariant, we have $\mathscr{F}(G_v) = \mathscr{F}(G)$.

Proof of the Theorem (Outline)

Let $\mathscr{F}$ be a computable CR-invariant function.
We want to compute $\mathscr{F}(G)$ for some graph G by a recurrent GNN.

Phase 1: Computing the Colours

We simulate colour refinement. After Phase 1, the signal at every node v holds a representation D_v of v's its colour.

Phase 2: Inversion

Locally at every node v, from the colour D_v we compute a graph G_v that is C^2-equivalent to the input graph G.

Phase 3: Computation of $\mathscr{F}$

Locally at every node v, we compute $\mathscr{F}(G_v)$. Since $\mathscr{F}$ is C^2-invariant, we have $\mathscr{F}(G_v) = \mathscr{F}(G)$.

Thus after Phase 3, every node holds the correct function value, and readout becomes trivial.

Phase 3: Computation of $\mathscr{F}$

Locally at every node v, we need to compute $\mathscr{F}(G_v)$ from the graph G_v that is encoded in the state of node v after Phase 2.

Phase 3: Computation of $\mathscr{F}$

Locally at every node v, we need to compute $\mathscr{F}(G_v)$ from the graph G_v that is encoded in the state of node v after Phase 2.

Local computation of a GNN at a node that ignores the message passing is essentially the computation of a recurrent feedforward neural network.

Phase 3: Computation of $\mathscr{F}$

Locally at every node v, we need to compute $\mathscr{F}(G_v)$ from the graph G_v that is encoded in the state of node v after Phase 2.

Local computation of a GNN at a node that ignores the message passing is essentially the computation of a recurrent feedforward neural network.

Thus we can use the following result

Theorem (Siegelmann and Sontag 1992)

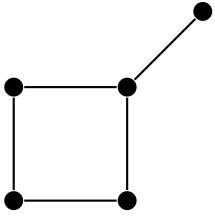
Every computable function $f : \mathbb{Q}^m \to \mathbb{Q}^n$ is computable by a recurrent feedforward neural network.

Phase 1: Computing the Colours

What are the “colours”?

Phase 1: Computing the Colours

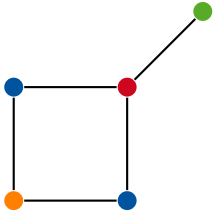
What are the “colours”?



Graph G

Phase 1: Computing the Colours

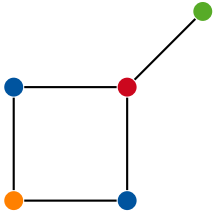
What are the “colours”?



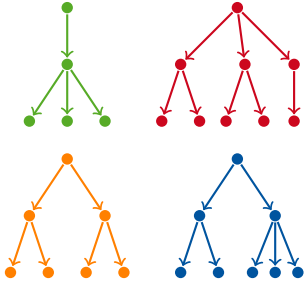
Graph G
coloured

Phase 1: Computing the Colours

What are the “colours”?



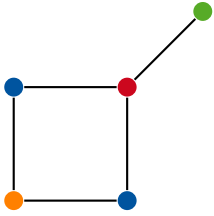
Graph G
coloured



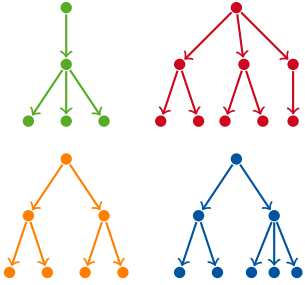
Tree representation

Phase 1: Computing the Colours

What are the "colours"?



Graph G coloured

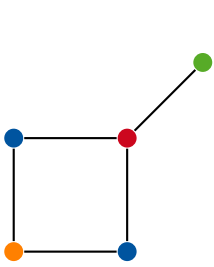


Tree representation

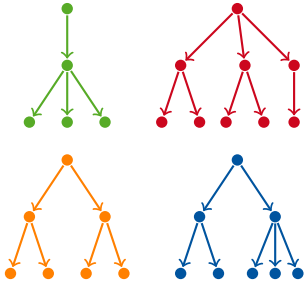
Compact dag representation

Phase 1: Computing the Colours

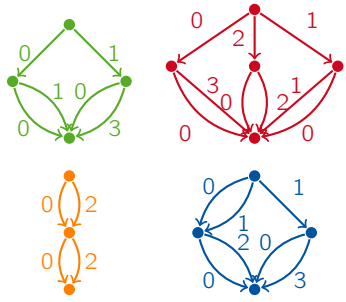
What are the "colours"?



Graph G coloured



Tree representation



Compact dag representation

To simulate colour refinement by a recurrent GNN, we encode the compact dag representation of colours by rational numbers and in several rounds by decreasing c for each node v count the number of neighbours of v of colour c.

The following lemma is an easy adaptation of a result on "inverting C^2-invariants" due to Martin Otto.

The following lemma is an easy adaptation of a result on "inverting C^2-invariants" due to Martin Otto.

Lemma (Otto 1997)

There is a polynomial time algorithm that, given the dag representation D_v of the colour of a node v in a graph G, computes a graph G_v that is C^2-equivalent to G.

Concluding Remarks

Concluding Remarks

- Our results on recurrent GNNs necessarily require rationals of unbounded precision in the (internal) signals computed by the GNN, even if we are only interested in computing a Boolean function.

Concluding Remarks

- Our results on recurrent GNNs necessarily require rationals of unbounded precision in the (internal) signals computed by the GNN, even if we are only interested in computing a Boolean function.

  It would be interesting, to understand the precise bitlength required. For example, which functions can we compute with logarithmic bitlength (in the size of the input graph)?

Concluding Remarks

- Our results on recurrent GNNs necessarily require rationals of unbounded precision in the (internal) signals computed by the GNN, even if we are only interested in computing a Boolean function.

  It would be interesting, to understand the precise bitlength required. For example, which functions can we compute with logarithmic bitlength (in the size of the input graph)?
- We have a good understanding of the expressiveness of GNNs. But expressiveness results only tell half the story, because they ignore learning. At least, many of the results presented here have good experimental support.

A Few References

Grohe, M. (2021). The Logic of Graph Neural Networks.
In: Proc. LICS 2021.
arXiv:2104.14624

Grohe, M. (2024). The Descriptive Complexity of Graph Neural Networks.
In: TheoretiCS 3(25), 2024.
arXiv:2303.04613

Rosenbluth, E. and Grohe, M. (2026) Repetition Makes Perfect: Recurrent Sum-GNNs Match Message Passing Limit.
In: Proc. AAAI 2026.
arXiv:2505.00291