Recurrent GNNs: The Power of Iteration

ESSAI 2026 course

Floris Geerts[1] Jonni Virtema[2]

[1] University of Antwerp, BE
[2] University of Glasgow, UK

Version of 10th July 2026

This course covers basics of graph learning, how theoretical limits are proven for (non-recurrent) GNNs, and what changes when recurrence is introduced.

Organisational information about the course

Material for the course:

- Lecture notes
- This slide set.
- Webpage: `http://www.virtema.fi/essai26.html`

About the lecturers

Floris Geerts (University of Antwerp)
Research Interests: Database Theory, Graph and Relational Learning, Quantum stuff.
`https://fgeerts.github.io/`

Jonni Virtema (University of Glasgow)
Research Interests: Logical Foundations of Neural Networks, Complexity Theory, Finite Model Theory, Temporal Logics for Hyperproperties.
`http://www.virtema.fi/`

Lecture 1: Basics of graph learning, GNNs, and notions of expressivity

Part I: Basics of Graph Learning

- What are we learning: Properties of graph structured data
- Constraints on learning: Invariants that any learned function must obey (e.g., permutation invariance).
- What is the model: Graph Neural Networks (GNNs) – built to satisfy these invariants.

Graphs

- We work with finite undirected graphs whose nodes carry labels.
- Let X be a set. An X-labeled graph is a triple

$$G = (N, E, g),$$

  where N is a finite set of nodes, $E \subseteq N \times N$ is the edge (symmetric, antireflexive) relation, and

$$g \colon N \to X$$

  assigns a label to each node.
- The map g is the node-labeling function. If $X = \mathbb{R}^d$, then g is also called a feature map.
- We write $\mathcal{G}[X]$ for the class of all finite X-labeled undirected graphs.

Graph data in the WILD


Event Graphs


Image credit: SalientNetworks
Computer Networks


Disease Pathways


Image credit: Wikipedia
Food Webs


Image credit: Pinterest
Particle Networks


Image credit: visitlondon.com
Underground Networks


Image credit: Medium
Social Networks

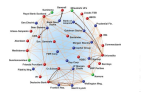


Image credit: Science
Economic Networks


Image credit: Lumen Learning
Communication Networks


Citation Networks


Image credit: Missoula Current News
Internet

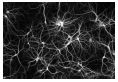


Image credit: The Conversation
Networks of Neurons

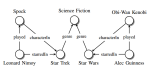


Image credit: Maximilian Nickel et al
Knowledge Graphs

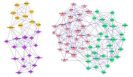


Image credit: eea.wustl.edu
Regulatory Networks


Image credit: math.hws.edu
Scene Graphs


Image credit: ResearchGate
Code Graphs

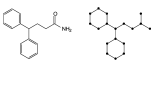


Image credit: MDPI
Molecules


Image credit: Wikipedia
3D Shapes

Basic graph concepts

Let $G = (N, E, g)$ be an X-labeled graph.

- For $n \in N$, the neighbourhood of n is

$$E(n) := \{m \in N \mid (n, m) \in E\}.$$

- The corresponding degree is

$$\deg(n) := |E(n)|.$$

- A pointed graph is a pair (G, n), where $n \in N$.

Graph learning tasks

- A node-level task predicts a property of each node:

$$n \mapsto \text{label of } n,$$

  e.g., classifying users in a social network.
- An edge-level task predicts a property of a pair of nodes:

$$(m, n) \mapsto \text{label of } (m, n),$$

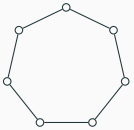
  such as whether an edge exists (*link prediction*) or its type (*edge classification*).
- A graph-level task predicts a property of the whole graph:

$$G \mapsto \text{label of } G,$$

  e.g., predicting a molecule's toxicity.
- In these lectures, we mainly focus on node classifiers mapping nodes to a finite set of labels.

Constraints on graph learning

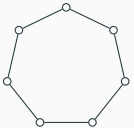
- In graph learning, we want to learn properties of graphs that should not depend on their representation.
- Intuitively: If two graphs look the same they should be treated the same!
- Example: Leader selection
  - In an odd linear path , which nodes could be selected as leaders?
  - In a cycle



which node set could be selected?

Constraints on graph learning

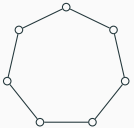
- In graph learning, we want to learn properties of graphs that should not depend on their representation.
- Intuitively: If two graphs look the same they should be treated the same!
- Example: Leader selection
  - In an odd linear path  , only the middle node can be distinguished from all others.
  - In a cycle

    which node set could be selected?

Constraints on graph learning

- In graph learning, we want to learn properties of graphs that should not depend on their representation.
- Intuitively: If two graphs look the same they should be treated the same!
- Example: Leader selection
  - In an odd linear path  ,
  - In a cycle



only two sets—empty set and all nodes—can be distinguished.

  - Why? Each node looks the identical from the node's perspective (automorphisms)!

Invariance: The formal notion capturing the constraints on graph learning

- Computations on graphs should not depend on arbitrary choices of representation.
  - In particular, node identifiers and the storage order of nodes or edges should not affect the result.
  - Graph isomorphisms formalize when two representations describe the same labeled graph.
- A graph-level function should assign the same value to isomorphic graphs.
  - This is called invariance.

Equivalence relations

- Graph learning deals with multiple notions of similarity captured by an equivalence relation.
- A binary relation $\sim \subseteq \mathcal{G}[X] \times \mathcal{G}[X]$ on labelled graphs is an equivalence relation, if it is
  - reflexive: $G \sim G$ for all labelled graphs $G \in \mathcal{G}[X]$,
  - symmetric: if $G \sim G'$ then $G' \sim G$, and
  - transitive: if $G_1 \sim G_2$ and $G_2 \sim G_3$ then $G_1 \sim G_3$.
- Examples of equivalence relations: equality, equicardinality, graph isomorphism.
- Equivalence relation for pointed graphs (or any base set) is defined analogously.

Graph isomorphisms

Let $G = (N, E, g)$ and $G' = (N', E', g')$ be X-labeled graphs.

- An isomorphism $f : G \to G'$ is a bijection $f : N \to N'$ such that, for all $m, n \in N$,

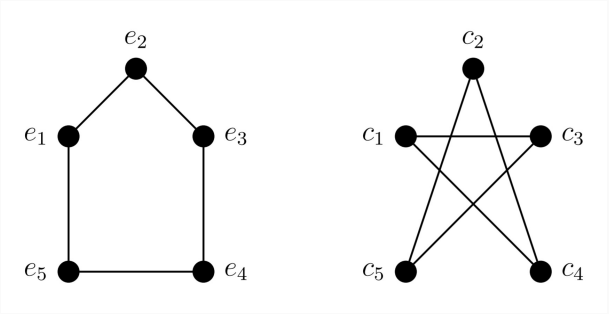
$$(m, n) \in E \iff (f(m), f(n)) \in E',$$

  and, for all $n \in N$,

$$g'(f(n)) = g(n).$$

- An isomorphism preserves both the edge structure and the node labels.
- We write $G \cong G'$ if there exists an isomorphism between G and G'.
- Analogously, $(G, n) \cong (G', n')$ for pointed graphs, for which $f(n) = n'$.
- Easy to check that $\cong$ is an equivalence relation on labelled graphs.

Graph isomorphism: Example



Graph invariants

Let Y be a set, $\sim_0$ be an equivalence relation on labelled graphs, and $\sim_1$ be an equivalence relation on labelled pointed graphs.

- A $\sim_0$-invariant is a function $\xi\colon \mathcal{G}[X] \to Y$ such that

  $$\xi(G) = \xi(G'),$$

  whenever $G \sim_0 G'$.
- A $\sim_1$-invariant assigns to every graph $G = (N, E, g)$ a function

  $$\xi_G\colon N \to Y$$

  that is invariant under $\sim_1$; that is

  $$\xi_G(n) = \xi_{G'}(n'),$$

  whenever $(G, n) \sim_1 (G', n')$.
- When $\sim_0$ and $\sim_1$ refer to graph isomorphism, we obtain the concepts of graph invariants and node invariants.

Graph invariants: examples

Let $G = (N, E, g)$ be an X-labeled graph.

- Examples of graph-level invariants are

$$G \mapsto |N|, \qquad G \mapsto |E|, \qquad G \mapsto \{\{g(n) \mid n \in N\}\}.$$

  where $\{\{\ \}\}$ denotes a multiset or bag.

- Examples of node-level invariants are

$$n \mapsto \deg(n) \qquad n \mapsto g(n).$$

- Non-examples are quantities depending on node identifiers or on the order in which nodes are stored.
- A cycle of length n has only two node-level invariants $\xi\colon N \to \mathbb{B}$: select all and select none.

The graph learning setting

- In graph learning, we are given training data consisting of graphs and target labels.
- For a graph-level task, the training data has the form

$$(G_1, y_1), \ldots, (G_r, y_r),$$

  where y_j is the target label for the graph G_j.
- For a node-level task, the training data has the form

$$((G_1, n_1), y_1), \ldots, ((G_r, n_r), y_r),$$

  where y_j is the target label for the graph G_j and node $n_j \in N_j$.
- The goal is to predict labels for new graphs or new nodes.

Why graph neural networks?

- Graph learning is not ordinary vector learning: the same graph can be represented using many different node orders.
- Learned functions should be invariant.
- Graph neural networks are designed so that these constraints are built into the architecture.
- They compute node representations by repeatedly combining

  the current representation of a node with representations of its neighbours.

- This gives a principled model for learning from node labels/features together with graph structure.

What is learned?

- As we will see graph neural networks (GNN) contain parameters, denoted by $\boldsymbol{\theta}$.
- These are typically weights and biases of neural networks used inside the GNN layers.
- The graph itself is not learned: the nodes, edges, and input features are the data.
- Training chooses $\boldsymbol{\theta}$ by minimizing a loss function on the training data.
- In expressivity theory, we ask a different question: which functions can be represented by some choice of parameters?

The basic idea of message passing

- Each node starts with an initial feature vector.
- If initial labels are not in $\mathbb{R}^d$, then an initial encoding is needed. For example, a one-hot encoding of discrete labels.
- In each layer, every node collects information from its neighbours.
- This neighbour information is combined with the node's current feature.
- After ℓ layers, the feature of a node depends on information up to distance ℓ.
- Thus, GNNs compute learned node representations, and possibly graph representations.

Graph Neural Networks: A single layer

- A GNN is built from layers.
- A layer $L_{\boldsymbol{\theta}}$ maps an $\mathbb{R}^p$-labeled graph

$$G = (N, E, g)$$

to an $\mathbb{R}^q$-labeled graph

$$L_{\boldsymbol{\theta}}(G) = (N, E, g').$$

- The nodes and edges stay fixed; only the node features are updated.
- For $n \in N$, the updated feature is

$$g'(n) \coloneqq \mathrm{Comb}_{\boldsymbol{\theta}}\Big(g(n), \mathrm{Agg}_{\boldsymbol{\theta}}\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big).$$

Here,

$$\mathrm{Agg}_{\boldsymbol{\theta}} \colon \mathcal{M}(\mathbb{R}^p) \to \mathbb{R}^h, \qquad \mathrm{Comb}_{\boldsymbol{\theta}} \colon \mathbb{R}^p \times \mathbb{R}^h \to \mathbb{R}^q.$$

$\mathcal{M}(\mathbb{R}^p)$ is the set of finite multisets of elements of the set $\mathbb{R}^p$.

Aggregate–Combine GNNs (AC-GNNs)

- The name AC refers to the two steps in one message-passing layer:

  Aggregate neighbour features $+$ Combine with own feature.

- For $n \in N$:

$$g'(n) := \underbrace{\mathrm{Comb}_{\boldsymbol{\theta}}\Big(g(n), \underbrace{\mathrm{Agg}_{\boldsymbol{\theta}}\big(\{\{g(m) \mid m \in E(n)\}\}\big)}_{\text{Aggregate}}\Big)}_{\text{Combine}}.$$

- Thus an AC layer first summarizes the neighbourhood of n, then updates the feature of n using this summary.

Graph Neural Networks: stacking layers

- A GNN consists of layers
$$L^{(1)}_{\boldsymbol{\theta}_1}, L^{(2)}_{\boldsymbol{\theta}_2}, \ldots, L^{(\ell)}_{\boldsymbol{\theta}_\ell}.$$
- Given an input graph
$$G_0 = (N, E, g_0), \qquad g_0 \colon N \to \mathbb{R}^{p_0},$$
the layers recursively define
$$G_i = (N, E, g_i) := L^{(i)}_{\boldsymbol{\theta}_i}(G_{i-1})$$
for $i = 1, \ldots, \ell$.
- The output dimension of $L^{(i)}_{\boldsymbol{\theta}_i}$ must match the input dimension of $L^{(i+1)}_{\boldsymbol{\theta}_{i+1}}$.
- The final node representation is
$$n \mapsto g_\ell(n).$$

Graph Neural Networks: readout

- For node-level tasks, the output is computed from the final node features:

$$n \mapsto g_\ell(n).$$

- For graph-level tasks, one adds a readout function

$$\mathrm{ReadOut}_{\boldsymbol{\theta}} \colon \mathcal{M}(\mathbb{R}^{p_\ell}) \to \mathbb{R}^q.$$

- The graph-level output is

$$\mathrm{ReadOut}_{\boldsymbol{\theta}}\big(\{\{g_\ell(n) \mid n \in N\}\}\big).$$

- For classification, the real-valued output is mapped to a finite label set $Y = \{1, \ldots, q\}$, for example by

$$\mathrm{Classify}(z) = \arg\max_{i \in Y} z_i, \qquad z \in \mathbb{R}^q.$$

Training a GNN

- A GNN with parameters $\boldsymbol{\theta}$ defines a prediction function $F_{\boldsymbol{\theta}}$.
- Given graph-level training data

$$(G_1, y_1), \ldots, (G_r, y_r),$$

the parameters are chosen by minimizing an empirical loss

$$\min_{\boldsymbol{\theta}} \frac{1}{r} \sum_{j=1}^{r} \ell\big(F_{\boldsymbol{\theta}}(G_j), y_j\big).$$

- The learned function should generalize to graphs not seen during training.
- In the rest of the lecture, we abstract away from training and focus on expressive power.

GNN layers are invariant

Let $G = (N, E, g)$ and $G' = (N', E', g')$ be isomorphic via $f : G \to G'$.

- Since f preserves edges,
$$f(E(n)) = E'(f(n)).$$
- Since f preserves labels,
$$g'(f(m)) = g(m).$$
- Therefore the multisets of neighbour features agree:
$$\{\{g(m) \mid m \in E(n)\}\} = \{\{g'(m') \mid m' \in E'(f(n))\}\}.$$
- Hence every message-passing layer is node-invariant.

Simple AC-GNN layers

A common special case:

- Uses sum aggregation for Agg;
- Use feedforward neural networks for Comb.
- At layer i, the update is

$$g_i(n) = f_i\left(g_{i-1}(n) \mid \sum_{m \in E(n)} g_{i-1}(m)\right).$$

- Here $x \mid y$ denotes vector concatenation and f_i is a feedforward neural network (FNN).
- This type of GNNs will be used later in these lectures.

Feedforward neural networks

- An FNN is a function $f : \mathrm{R}^k \to \mathrm{R}^l$ built by composing layers of the form

$$z \mapsto \sigma(Wz + b),$$

  where W is a weight matrix, b a bias vector, and σ a (non-linear) activation function applied component-wise.
- A network with L layers computes

$$f(x) = \sigma_L\big(W_L\, \sigma_{L-1}(\cdots \sigma_1(W_1 x + b_1)\cdots) + b_L\big).$$

- Typical choices for σ: ReLU, sigmoid, tanh; the output layer often has no activation (or a task-specific one).

Boolean graph properties

Let $\mathbb{B} := \{0, 1\}$.

- A Boolean graph property is a graph invariant

$$P \colon \mathcal{G}[X] \to \mathbb{B}.$$

- Examples:

$$\begin{aligned} P_{\mathrm{red}}(G) &= 1 && \text{iff } G \text{ contains a red node,} \\ P_{\mathrm{red}\to\mathrm{blue}}(G) &= 1 && \text{iff some red node has a blue neighbour.} \\ P_{\mathrm{conn}}(G) &= 1 && \text{iff } G \text{ is connected,} \\ P_{\Delta}(G) &= 1 && \text{iff } G \text{ contains a triangle.} \end{aligned}$$

Node classifiers

Let $\mathbb{B} := \{0, 1\}$.

- A node classifier is a label transformer

$$C \colon \mathcal{G}[X] \to \mathcal{G}[\mathbb{B}].$$

- Thus, for every input graph $G = (N, E, g)$,

$$C(G) = (N, E, c_G), \qquad c_G \colon N \to \mathbb{B}.$$

- Equivalently, C selects a set of nodes

$$C_G := \{n \in N \mid c_G(n) = 1\}.$$

Node classifiers: examples

$$
\begin{aligned}
c_{\text{red}}(G, n) = 1 &\iff g(n) = \text{red}, \\
c_{\text{blue-nbr}}(G, n) = 1 &\iff \exists m \in E(n) : g(m) = \text{blue}, \\
c_{\text{reach-blue}}(G, n) = 1 &\iff n \text{ can reach some blue node}, \\
c_{\text{cycle}}(G, n) = 1 &\iff n \text{ lies on a cycle}.
\end{aligned}
$$

Why Boolean properties?

- Boolean properties are a convenient way to study expressive power.
- Instead of asking whether a model computes a complicated output, we ask whether it can answer a yes/no question.
- This already captures many important tasks:

Does the graph contain a red node? Is this node on a cycle?

- Boolean graph properties correspond to graph-level tasks.
- Boolean node classifiers correspond to node-level tasks.
- Later, they will also match naturally with logical formulas.

From learning to expressivity

- We have seen that GNNs define functions that respect graph symmetries.
- Expressivity asks which invariant graph properties and node classifiers can be represented by such architectures.
- Boolean properties give a clean test language:

  can the model answer this yes/no question?

- Local properties, such as seeing a blue neighbour, are natural for message passing.
- Global properties, such as reachability and cycle membership, reveal its limitations.
- Part 2 makes this precise using colour refinement.

Part 2: Limits of Graph Neural Networks

- How capabilities of GNNs can be measured?
  - Distinguising power.
  - Uniform and non-uniform expressivity.
- To what can we compare GNNs?
  - Graph algorithms (colour refinement).
  - Later: Logics

Node-feature maps

Fix input labels X and output labels Y. Let $\mathcal{G}[X]$ denote the class of graphs whose nodes are labelled by X.

- A node-feature map is a graph transformation

$$M \colon \mathcal{G}[X] \to \mathcal{G}[Y]$$

  that changes only the node labels, not the underlying graph.
- Thus, for every input graph $G = (N, E, g)$,

$$M(G) = (N, E, h_G), \qquad h_G \colon N \to Y.$$

- In words: M assigns a new output label $h_G(n)$ to each node $n \in N$.
- A class $\mathcal{M}$ is a set of such maps.
- To study the expressivity of a GNN architecture, we study the class of node-feature maps it can produce.

What does it mean to be powerful?

- There are several ways to ask how powerful a class $\mathcal{M}$ of node-feature maps is.
- The weakest question is about distinguishing power:

  can some map in $\mathcal{M}$ tell these two nodes apart?

- The strongest question is about uniform expressivity:

  can one fixed map in $\mathcal{M}$ compute this classifier on all graphs?

- Between them sits non-uniform expressivity: the map may depend on the size of the input, or even on the particular comparison.
- The phrase expressive power is used for all of these in the literature, so we must say which question we are asking.

How to measure the power of classes of node feature maps?

- Many ways to classify the computing power of a class $\mathcal{M}$ of feature maps.
- Coarsest: distinguishability
- Finest: uniform expressiveness
- Middle ground: non-uniform expressiveness

How to measure the power of classes of node feature maps?

- Many ways to classify the computing power of a class $\mathcal{M}$ of feature maps.
- Coarsest: distinguishability
  - Given two pointed graphs (G, n) and (H, m), we ask whether there exists a node-feature map $M \in \mathcal{M}$ that separates the pointed graphs, i.e., such that

$$M(G)(n) \neq M(H)(m).$$

  - The map may depend on the size (or any other information) of the two pointed graphs.
  - Distinguishability does not answer to the question which classifiers can be expressed by a feature map.
  - It can be used to show which classifiers cannot be expressed!
- Finest: uniform expressiveness
- Middle ground: non-uniform expressiveness

How to measure the power of classes of node feature maps?

- Many ways to classify the computing power of a class $\mathcal{M}$ of feature maps.
- Coarsest: distinguishability
- Finest: uniform expressiveness
  - Which classifiers can be computed by using a single feature map from $\mathcal{M}$.
  - The same parameters, architecture, and number of layers must work for all input graphs.
  - Uniform expressivity answers to the question what classifiers belong to the class $\mathcal{M}$.
- Middle ground: non-uniform expressiveness

How to measure the power of classes of node feature maps?

- Many ways to classify the computing power of a class $\mathcal{M}$ of feature maps.
- Coarsest: distinguishability
- Finest: uniform expressiveness
- Middle ground: non-uniform expressiveness
  - Different maps can be used for different inputs; map selection may depend on graph size.
  - For example, one may allow a family of feature maps

    $$(M_s)_{s\in\mathbb{N}},$$

    where M_s only has to work on graphs of size s.
  - Then a node classifier C is non-uniformly expressible if

    $$M_{|N|}(G)(n) = C(G)(n)$$

    for every graph $G = (N, E, g)$ and every node $n \in N$.
  - This is weaker than uniform expressiveness, since the model may change with the graph size. In general $s \mapsto M_s$, need not be even computable!

How to measure the power of classes of node feature maps?

- Many ways to classify the computing power of a class $\mathcal{M}$ of feature maps.
- Coarsest: distinguishability
- Finest: uniform expressiveness
- Middle ground: non-uniform expressiveness
- In the literature, expressive power may refer to any of the above notions.

Distinguishing power: what can never be separated?

- A natural first question is:

  are there theoretical limits to what GNNs can distinguish?

- The first limit comes from graph isomorphism.
- If two pointed graphs are isomorphic,

$$(G, n) \cong (H, m),$$

  then any isomorphism-invariant node-feature map must treat n and m the same.
- In particular, if $f : G \to G$ is an automorphism, then n and $f(n)$ cannot be separated.
- So the first wall is symmetry:

  GNNs cannot break graph symmetries.

- The more interesting wall is colour refinement. That is where we go next.

Colour refinement: what does a node see?

- Imagine that every node repeatedly asks:

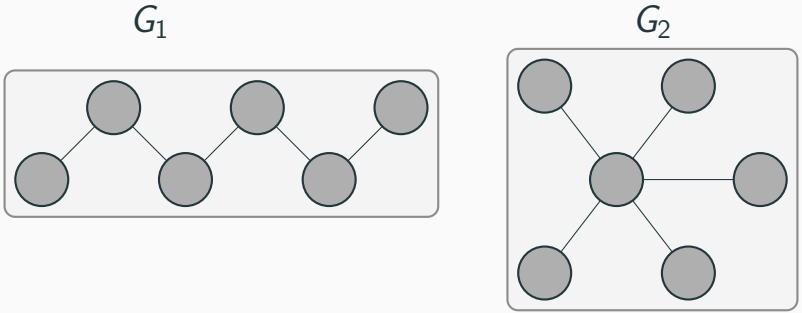
  what colour am I, and what colours do my neighbours have?

- Initially, a node only knows its own input label:

$$\chi_0(n) := g(n).$$

- After one round, it knows the multiset of labels around it.
- After two rounds, it knows how those neighbours themselves look.
- After t rounds, the colour $\chi_t(n)$ summarizes the depth-t view of n as seen by this message-passing process.

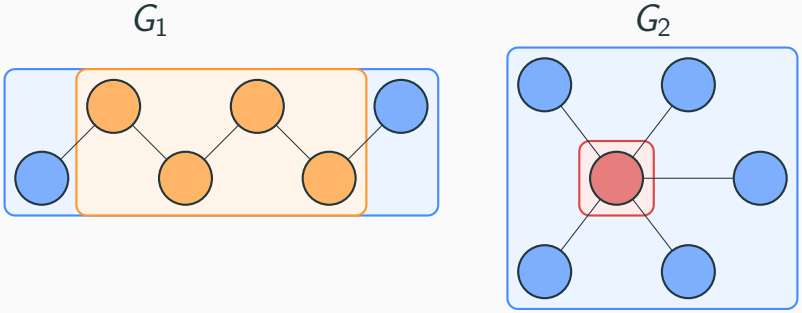
Colour Refinement (1-WL)



Round 0: one colour class

$$\chi_0(n) = \text{grey}$$

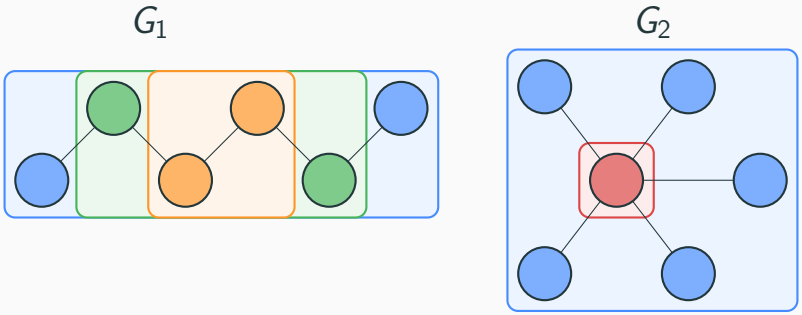
Colour Refinement (1-WL)



Round 1: split by degree

$$\chi_{t+1}(n) = \big(\chi_t(n), \{\!\{\chi_t(m) : m \in E(n)\}\!\}\big)$$

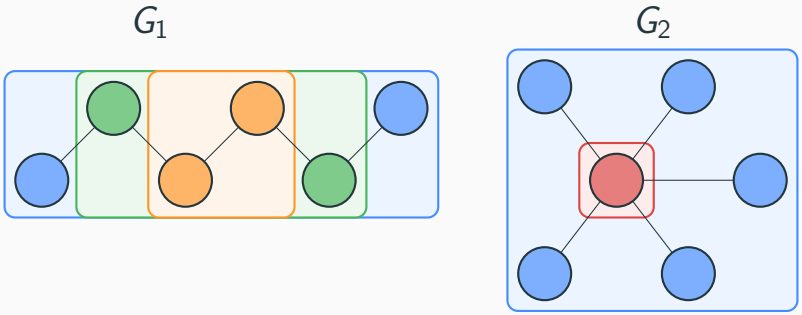
Colour Refinement (1-WL)



Round 2: colours propagate

$$\chi_{t+1}(n) = \big(\chi_t(n), \{\!\{\chi_t(m) : m \in E(n)\}\!\}\big)$$

Colour Refinement (1-WL)



Stable colouring

$$\chi_{t+1}(n) = \left(\chi_t(n), \{\!\{\chi_t(m) : m \in E(n)\}\!\}\right)$$

Colour refinement in one line

- The update rule is

$$\chi_{t+1}(n) \coloneqq \mathsf{hash}\Big(\chi_t(n), \{\{\chi_t(m) \mid m \in E(n)\}\}\Big).$$

- The hash is injective: two nodes get the same new colour exactly when they had
  - the same old colour, and
  - the same multiset of neighbour colours.
- So colour refinement never guesses. It separates nodes only when their local evidence is different.
- When no further split is possible, the colouring is stable.

Stable colouring

Let $\chi_t \colon N \to C_t$ be the colouring after round t.

- Each colouring χ_t induces a partition of N:

$$\Pi_t := \{\chi_t^{-1}(c) \mid c \in \chi_t(N)\}.$$

- Two nodes are in the same block of Π_t exactly when they have the same colour:

$$n \equiv_{\Pi_t} m \quad \iff \quad \chi_t(n) = \chi_t(m).$$

- Colour refinement stops once this partition no longer changes.
- Equivalently, it stops at the first T such that

$$\chi_T(n) = \chi_T(m) \quad \iff \quad \chi_{T+1}(n) = \chi_{T+1}(m)$$

  for all $n, m \in N$.
- The resulting stable colouring is denoted by χ_∞.

Colour histograms

Let $\chi\colon N \to C$ be a colouring of the nodes of a graph $G = (N, E, g)$.

- The colour histogram of χ records how often each colour occurs.
- Formally, it is the function

$$\mathrm{hist}_\chi\colon C \to \mathbb{N}$$

  defined by

$$\mathrm{hist}_\chi(c) := |\{n \in N \mid \chi(n) = c\}|.$$

- Equivalently, the colour histogram is the multiset of node colours

$$\{\{\chi(n) \mid n \in N\}\}.$$

- Thus two graphs have the same colour histogram at round t if they contain the same colours with the same multiplicities.

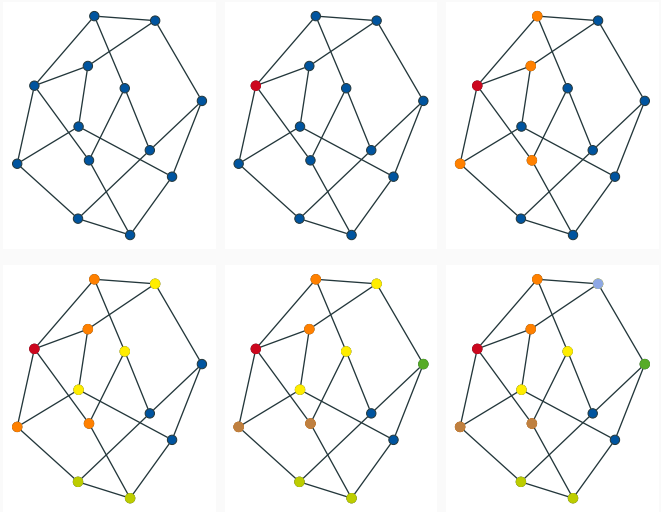
Graph isomorphism and colour refinement

- Colour refinement is invariant under graph isomorphism.
- If $G \cong G'$ via f, then for every round t,

$$\chi_t^G(n) = \chi_t^{G'}(f(n)).$$

- Hence, if colour refinement produces different colour histograms, then the graphs are not isomorphic.
- If the histograms agree, the test is inconclusive.
- Thus colour refinement is a useful but incomplete test for graph isomorphism.

Colour refinement: another example



GNNs and colour refinement

- Colour refinement and message-passing GNNs use the same basic pattern:

  old node information $+$ multiset of neighbour information.

- Colour refinement updates colours by

$$\chi_{t+1}(n) \coloneqq \mathsf{hash}\Big(\chi_t(n), \{\{\chi_t(m) \mid m \in E(n)\}\}\Big).$$

- A GNN layer updates features by

$$g_{t+1}(n) \coloneqq \mathsf{Comb}_t\Big(g_t(n), \mathsf{Agg}_t\{\{g_t(m) \mid m \in E(n)\}\}\Big).$$

- Thus colour refinement is a discrete analogue of message passing.

The punchline

A message-passing GNN cannot see more than colour refinement sees.

- This is not a statement about training.
- It is not a statement about the number of parameters.
- It is a structural statement about the form of message passing:

  a node repeatedly receives only a multiset summary of its neighbours.

Proposition 1 (Colour refinement and GNNs [Mor+19])

Let (G, n) and (H, m) be two pointed X-labeled graphs. Let χ_t denote the colouring after t rounds of colour refinement.

1. *For every t-layer message-passing GNN with node features g_t, we have*

$$\chi_t^G(n) = \chi_t^H(m) \quad \Longrightarrow \quad g_t^G(n) = g_t^H(m).$$

2. *Conversely, if colour refinement distinguishes the two pointed graphs after t rounds, that is,*

$$\chi_t^G(n) \neq \chi_t^H(m),$$

then there exists a t-layer message-passing GNN such that

$$g_t^G(n) \neq g_t^H(m).$$

Easy direction: GNNs are bounded by CR

- Let χ_t be the colour refinement colouring after t rounds.
- Let g_t be the node features computed by a t-layer GNN.
- The easy direction says:

$$\chi_t(n) = \chi_t(m) \quad \Longrightarrow \quad g_t(n) = g_t(m).$$

- Equivalently:

$$g_t(n) \neq g_t(m) \quad \Longrightarrow \quad \chi_t(n) \neq \chi_t(m).$$

- Hence a GNN cannot distinguish two nodes that colour refinement has not distinguished.

Proof of the easy direction

We prove by induction on t that

$$\chi_t(n) = \chi_t(m) \quad \Longrightarrow \quad g_t(n) = g_t(m).$$

- Base case $t = 0$: if $\chi_0(n) = \chi_0(m)$, then n and m have the same initial label, so

$$g_0(n) = g_0(m).$$

- Induction step: assume the claim holds for t.
- Suppose

$$\chi_{t+1}(n) = \chi_{t+1}(m).$$

- By definition of colour refinement,

$$\chi_t(n) = \chi_t(m)$$

and

$$\{\{\chi_t(u) \mid u \in E(n)\}\} = \{\{\chi_t(v) \mid v \in E(m)\}\}.$$

Proof of the easy direction continued

By the induction hypothesis, equal colours at round t imply equal GNN features.

- Therefore the neighbour-feature multisets agree:

$$\{\{g_t(u) \mid u \in E(n)\}\} = \{\{g_t(v) \mid v \in E(m)\}\}.$$

- Also,

$$g_t(n) = g_t(m).$$

- Since the same aggregation and combination functions are applied at every node,

$$g_{t+1}(n) = g_{t+1}(m).$$

- This proves the induction step.
- Hence every message-passing GNN is bounded by colour refinement.

Consequence

- If colour refinement gives two nodes the same stable colour, then no message-passing GNN can distinguish them:

$$\chi_\infty(n) = \chi_\infty(m) \quad \Longrightarrow \quad g_\ell(n) = g_\ell(m)$$

  for every number of layers ℓ.
- Therefore, message-passing GNNs are at most as expressive as colour refinement.
- In particular, every node classifier definable by such a GNN is constant on colour-refinement classes.

The converse direction

- The easy direction was:

$$\text{GNN distinguishes} \implies \text{CR distinguishes.}$$

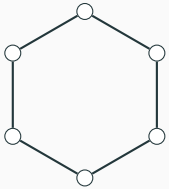
- The converse asks:

$$\text{CR distinguishes} \implies \text{some GNN distinguishes.}$$

- We show this later using logic!

Colour refinement cannot distinguish $2K_3$ from C_6

$G = 2K_3$



$H = C_6$

- Colour refinement does not distinguish $2K_3$ and C_6, even though they are not isomorphic.
- Neither can any GNN! Irregardless of number of layers or number of parameters or complexity of combine and aggregate functions..

Why colour refinement cannot see triangles

- Initially, all nodes have the same colour.
- In both graphs, every node has exactly two neighbours.
- Moreover, as long as all nodes have the same colour, every node sees the same multiset of neighbour colours:

$$\{\{\chi_t(m) \mid m \in E(n)\}\} = \{\{c_t, c_t\}\}.$$

- Hence all nodes receive the same colour at the next round:

$$\chi_{t+1}(n) = c_{t+1} \qquad \text{for every node } n.$$

- By induction, colour refinement never distinguishes nodes in $K_3 \sqcup K_3$ from nodes in C_6.

Consequence for GNNs

Define the node classifier

$$C_\Delta(G)(n) = 1 \quad \iff \quad n \text{ lies in a triangle.}$$

- In $K_3 \sqcup K_3$, every node satisfies $C_\Delta e(G)(n) = 1$.
- In C_6, every node satisfies $C_\Delta(H)(n) = 0$.
- But colour refinement assigns the same stable colour to all nodes in both graphs.
- Since message-passing GNNs are bounded by colour refinement, no standard message-passing GNN can express C_Δ.
- The problem is structural:

  message passing cannot tell whether two neighbours are connected to each other.

A different blind spot: information too far away

- The triangle example was not about distance. The triangle is right next to the node.
- Now consider a different limitation.
- After ℓ layers, a node can only receive information from nodes within distance ℓ.
- Therefore any fixed-depth GNN is local:

$$g_\ell(n) \text{ depends only on the labelled radius-}\ell \text{ neighbourhood of } n.$$

- So fixed-depth GNNs cannot express properties where the relevant witness may be arbitrarily far away.

Reachability is not fixed-depth

Fix a label p and define

$$C_{\mathrm{Reach}}(G)(n) = 1$$

iff there is a directed path from n to some node labelled p.

- For any fixed number of layers ℓ, choose two directed paths whose first ℓ steps from the start node look identical.
- In the first graph, a p-node occurs just beyond this visible horizon, at distance $\ell + 1$.
- In the second graph, no p-node is reachable.
- The start nodes have the same radius-ℓ view, so every ℓ-layer GNN gives them the same output.
- Hence no fixed-depth GNN can express general reachability.

Reachability needs iteration

Reachability is naturally computed by an iterative process.

- Start with the target nodes:

$$R_0 := \{n \mid p \in g(n)\}.$$

- Then repeatedly add predecessors:

$$R_{t+1} := R_t \cup \{n \mid E(n) \cap R_t \neq \emptyset\}.$$

- The process may require as many rounds as the length of the longest relevant path.
- Thus reachability is not a fixed-depth property.

From distinguishing power to expressive power (a)

- Distinguishing two fixed graphs can be easier than expressing one classifier uniformly on all graphs.
- For example, if G and G' are paths of different lengths, then some GNN can separate them.
- But as the paths become longer, the separating GNN may need more layers.
- This matters because uniform expressivity requires one fixed depth.
- Therefore: separating every finite example individually is not the same as computing one global classifier.

From distinguishing power to expressive power (b)

- Easy exercise: If G and G' are paths of different lengths n and m, there exists a GNN that separates G and G'.
  - Caveat: When n and m grow bigger, the separating GNN requires more layers.
- Separating power of l-layer GNNs is limited to that of l-round colour refinement.
  - Consequence: Any graph property that can be expressed with a GNN is invariant under l-round colour refinement, for large enough l!

From distinguishing power to expressive power (c)

- Fix l, no l-layer GNN can separate G, a path of length $3l + 2$, from G', a disjoint union of a path of $2l + 2$ and a cycle of length l.
  - Proof: The l-neighbourhoods of the l middle points of G are indistinguishable from the cycle points of G'. This can be formalised using l-round CR.
- Consequence: No fixed layer GNN can decide whether a graph has a cycle.

Two kinds of limitations

- Some properties fail because fixed-depth GNNs are local.
- Reachability is an example: the relevant information may be arbitrarily far away.
- Other properties fail because they are not visible to colour refinement, as we have seen before.

Reachability vs. cycle membership

- Reachability illustrates a locality limitation: fixed-depth GNNs cannot follow paths of unbounded length.
- Cycle membership illustrates a colour-refinement limitation.
- CR sees the unfolding of the graph around a node.
- It does not know whether a path eventually returns to the same node.
- Hence even very small cycles can be invisible to CR and to CR-bounded GNNs.

Takeaways

- Graphs provide a natural data model for relational data.
- GNNs learn node representations by message passing.
- Message-passing GNNs are invariant and define valid node classifiers.
- Their expressive power is closely tied to colour refinement.
  - They have the same distinguishing power!
  - Properties not invariant under colour refinement, such as Triangle, cannot be expressed by standard message-passing GNNs.
- Fixed-depth GNNs are local and cannot express general reachability.

Lecture 2: Graded modal logic and limits of GNN expressivity

Part 3: Graded modal logic and bisimulation

- How capabilities of GNNs be related to formal languages?
  - Formulas of graded modal logic define Boolean node properties of labelled graphs.
  - Compare properties of graphs that are definable by a formula vs computable by a GNN.
- What do we cover next?
  - Basics of graded modal logic.
  - Bisimulation, a central tool in modal logics, is essentially colour refinement.
  - We define characteristic formulae; these define a colour class of fixed round CR.
- Main sources:
  - Martin Otto: Graded modal logic and counting bisimulation. CoRR abs/1910.00039, 2019.

Why are we relating expressivity of GNNs to logics?

- Logic is a mature tool to study the expressivity of a model of computation.
- Automata theory: regular languages.
  - Languages accepted by finite automata and generated by regular expressions.
  - A first-order sentence interpreted over labelled linear orders

$$\forall x \forall y \big((x \leq y \wedge B(x)) \rightarrow B(y)\big) \wedge \forall z (A(z) \vee B(z))$$

  defines the language A^*B^*.
  - First-order logic has the same uniform expressivity than star-free languages.
  - Monadic second-order logic has the same uniform expressivity than regular languages.
- Complexity theory: many complexity classes have logical counterparts.

Why are we relating expressivity of GNNs to logics?

- Logic is a mature tool to study the expressivity of a model of computation.
- Automata theory: regular languages.
- Complexity theory: many complexity classes have logical counterparts.
  - A property of ordered graphs can be decided in polynomial time if and only if it can be described in first-order logic + recursion.
  - A property of graphs can be decided in NP if and only if it can be described in existential second-order logic.

Why are we relating expressivity of GNNs to logics?

- Logic is a mature tool to study the expressivity of a model of computation.
- Automata theory: regular languages.
- Complexity theory: many complexity classes have logical counterparts.
  - A property of ordered graphs can be decided in polynomial time if and only if it can be described in first-order logic + recursion.
  - A property of graphs can be decided in NP if and only if it can be described in existential second-order logic.
- Can we understand the limits of uniform expressivity of GNNs using the logical lens?
- Field: Descriptive complexity of GNNs!

Modal logic is logic for labelled graphs

- We consider input graphs with abstract labels from some finite set.
- Fix a finite set of atomic propositions

$$P = \{p_1, \ldots, p_s\}.$$

- We consider graphs labeled by subsets of P:

$$g \colon N \to \mathcal{P}(P).$$

- Intuitively,

$$p \in g(n)$$

means that proposition p is true at node n.

Graded modal logic: syntax

The formulas of graded modal logic are generated by

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Diamond_{\geq k}\varphi,$$

where $p \in P$ and $k \geq 1$.

- The modality

  $$\Diamond_{\geq k}\varphi$$

  means: at least k neighbours satisfy φ.
- Ordinary modal logic is the special case where only $k = 1$ is used.
  We write $\Diamond\varphi$ for $\Diamond_{\geq 1}\varphi$.
- The modal depth of a formula is the maximum nesting depth of modalities.

Graded modal logic: semantics

Let $G = (N, E, g)$ be a $\mathcal{P}(P)$-labeled graph, with $\mathcal{P}(P)$ denoting the powerset of P.

- Atomic propositions are evaluated by labels:

$$G, n \models p \quad \iff \quad p \in g(n).$$

- Boolean connectives are interpreted as usual.
- The graded modality is interpreted by counting out-neighbours:

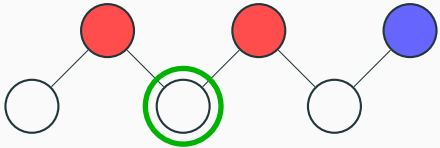
$$G, n \models \Diamond_{\geq k} \varphi$$

iff

$$\big|\{m \in E(n) \mid G, m \models \varphi\}\big| \geq k.$$

GML formulas define node properties of labelled graphs

$$\varphi := \Diamond_{\geq 2} \mathrm{red}$$

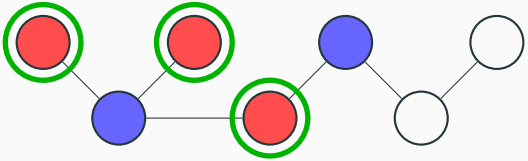


Green ring $= G, v \models \varphi$

"has at least two red neighbours"

GML formulas define node properties of labelled graphs

$$\psi := \Diamond_{\geq 1}\big(\text{purple} \wedge \Diamond_{\geq 2}\text{red}\big)$$



Green ring $= G, v \models \psi$

"has a purple neighbour that itself has at least two red neighbours"

GML formulas as node classifiers

Let φ be a GML formula.

- The formula φ defines a (boolean) node classifier

$$C_\varphi \colon \mathcal{G}[\mathcal{P}(P)] \to \mathcal{G}[\mathbb{B}].$$

- For every graph $G = (N, E, g)$, we set

$$C_\varphi(G) = (N, E, c_G^\varphi),$$

  where

$$c_G^\varphi \colon N \to \mathbb{B}.$$

- A node is classified as positive exactly when it satisfies φ:

$$c_G^\varphi(n) = 1 \quad \iff \quad G, n \models \varphi.$$

From colour refinement to logic

- Colour refinement updates a node using

  its current colour and the multiset of colours of its neighbours.

- Graded modal logic describes the same kind of local counting information.
- A typical GML statement says:

  "there are at least k neighbours satisfying φ".

- Thus GML is a natural logical language for reasoning about colour refinement (CR)-types and message passing.

Logic and GNNs

- GML seems quite weak. How does it compare with GNNs?
- $\mathcal{P}(P)$-labeled graphs are transformed to $\mathbb{R}^k$ features using the one-hot encoding.
- What does your intuition say in regards to graphs with a finite set of labels?
  - Distinguishing power:
  - Uniform expressivity:
  - Non-uniform expressivity:

Logic and GNNs

- GML seems quite weak. How does it compare with GNNs?
- $\mathcal{P}(P)$-labeled graphs are transformed to $\mathbb{R}^k$ features using the one-hot encoding.
- What does your intuition say in regards to graphs with a finite set of labels?
  - Distinguishing power: GML $=$ GNN, we will prove this after the break!
  - Uniform expressivity:
  - Non-uniform expressivity:

Logic and GNNs

- GML seems quite weak. How does it compare with GNNs?
- $\mathcal{P}(P)$-labeled graphs are transformed to $\mathbb{R}^k$ features using the one-hot encoding.
- What does your intuition say in regards to graphs with a finite set of labels?
  - Distinguishing power: GML $=$ GNN, we will prove this after the break!
  - Uniform expressivity: GNNs are more expressive.
  - Non-uniform expressivity:

Logic and GNNs

- GML seems quite weak. How does it compare with GNNs?
- $\mathcal{P}(P)$-labeled graphs are transformed to $\mathbb{R}^k$ features using the one-hot encoding.
- What does your intuition say in regards to graphs with a finite set of labels?
  - Distinguishing power: GML = GNN, we will prove this after the break!
  - Uniform expressivity: GNNs are more expressive. E.g., having more red neighbours than blue neighbours cannot be expressed in GML.
  - Non-uniform expressivity:

Logic and GNNs

- GML seems quite weak. How does it compare with GNNs?
- $\mathcal{P}(P)$-labeled graphs are transformed to $\mathbb{R}^k$ features using the one-hot encoding.
- What does your intuition say in regards to graphs with a finite set of labels?
  - Distinguishing power: GML = GNN, we will prove this after the break!
  - Uniform expressivity: GNNs are more expressive. E.g., having more red neighbours than blue neighbours cannot be expressed in GML.
  - Non-uniform expressivity: GML = GNN, when graph size is fixed. We will prove this after the break.

From formulas to equivalence

- GML formulas describe what can be observed from a node by looking locally along edges.
- This induces a natural question regarding invariance:

When do two pointed graphs satisfy the same GML formulas?

- The answer is given by graded bisimulation.
- Intuitively, two nodes are graded bisimilar if they have the same label and the same number of neighbours of each relevant type.
- Graded bisimulation is an equivalence relation. Hence, we can talk about properties of (pointed) labelled graphs that are g-bisimulation invariant.
- This is the logical counterpart of colour refinement.

Why bisimulation?

- Bisimulation compares pointed graphs by matching their local behaviour.
- For ordinary modal logic, it is enough to match successors back and forth.
- For graded modal logic, this is not enough: we must also match how many successors of each kind there are.
- Therefore we use graded bisimulation, also called counting bisimulation.

Idea of graded bisimulation

- A procedure to check whether two nodes are in the same colour class of CR.

(u, v) are graded bisimilar $u \sim v$, if their labels are identical, and

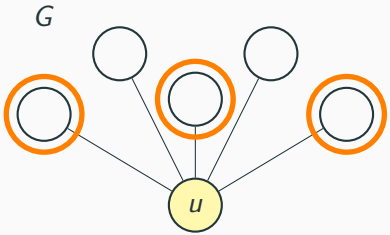
Idea of graded bisimulation

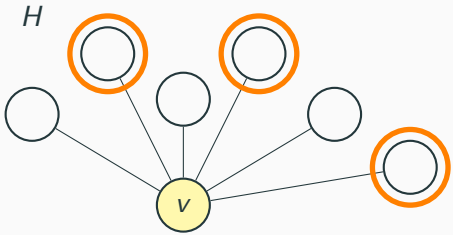
- A procedure to check whether two nodes are in the same colour class of CR.

for any sequence sequence (u_1, u_3, u_5) or successors of u

Idea of graded bisimulation

- A procedure to check whether two nodes are in the same colour class of CR.

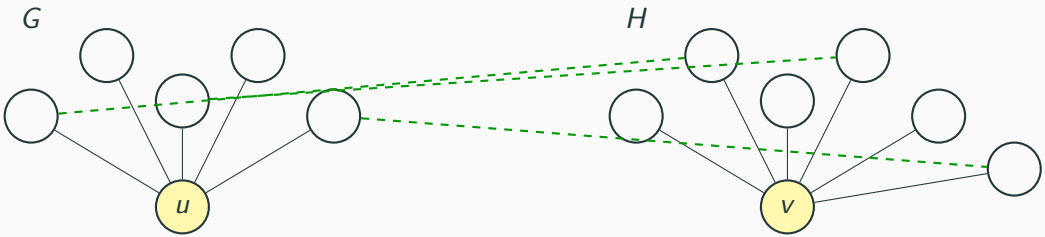




there exists a sequence (v_2, v_4, v_6) of successors of v such that

Idea of graded bisimulation

- A procedure to check whether two nodes are in the same colour class of CR.



the corresponding elements are bisimilar

$u_1 \sim v_2, u_3 \sim v_4, u_5 \sim v_6$

Idea of graded bisimulation

- A procedure to check whether two nodes are in the same colour class of CR.

Similarly, for any sequence or successors of v there must be a corresponding sequence or successors of u

r-round graded bisimulation

Let (G, n) and (H, m) be pointed $\mathcal{P}(P)$-labeled graphs.

- We define equivalence relations $\sim_r$ recursively.
- At depth 0,

$$(G, n) \sim_0 (H, m) \quad \iff \quad g_G(n) = g_H(m).$$

  - Two nodes are equivalent at depth 0 iff they satisfy the same atomic propositions.
- At depth $r + 1$, we require the same label and the same number of successors of each r-round equivalence class.

r-round graded bisimulation: recursive step

We say

$$(G, n) \sim_{r+1} (H, m)$$

iff

$$g_G(n) = g_H(m)$$

and, for every $\sim_r$-class C,

$$|\{u \in E_G(n) \mid (G, u) \in C\}| = |\{v \in E_H(m) \mid (H, v) \in C\}|.$$

- In words: n and m have the same number of neighbours of every previous type.
- The equivalence class of (G, n) under $\sim_r$ is its r-round graded bisimulation type.

Intuition

- Depth 0 only sees the node label.
- Depth 1 sees the node label and counts the labels of its neighbours.
- Depth 2 sees the node label and counts the depth-1 types of its neighbours.
- In general, depth r sees the rooted neighbourhood up to depth r, but only through counting types.
- This is exactly the kind of information used by graded modal logic.

GML is invariant under graded bisimulation

Proposition 2

Let φ be a GML formula of modal depth at most r. If

$$(G, n) \sim_r (H, m),$$

then

$$G, n \models \varphi \quad \Longleftrightarrow \quad H, m \models \varphi.$$

- Thus GML formulas of depth r cannot distinguish nodes with the same r-round graded bisimulation type.
- The proof is by induction on the structure of φ.

Connection to colour refinement

Proposition 3

Let χ_r be the colouring after r rounds of colour refinement. Then, for pointed graphs (G, n) and (H, m),

$$(G, n) \sim_r (H, m) \quad \iff \quad \chi_r^G(n) = \chi_r^H(m).$$

- Colour refinement computes graded bisimulation types.
- Two nodes have the same CR colour after r rounds exactly when they have the same r-round graded bisimulation type.
- The proof is by induction on r.

How expressive GML is?

- GML is expressive enough to define the r-round CR class of every pointed graph.
  - For every (G, n) there is a formula ψ s.t. $(H, m) \models \psi$ if and only if $\chi_r^G(n) = \chi_r^H(m)$.
  - We will prove this shortly.
- It does not follow that GML has the same uniform expressivity as GNNs. Why?

How expressive GML is?

- GML is expressive enough to define the r-round CR class of every pointed graph.
  - For every (G, n) there is a formula ψ s.t. $(H, m) \models \psi$ if and only if $\chi_r^G(n) = \chi_r^H(m)$.
  - We will prove this shortly.
- It does not follow that GML has the same uniform expressivity as GNNs. Why?
  - Let M_{GNN} be a node-feature map defined by an r-layer GNN.
  - Then, $\{(G, n) \mid M_{\mathsf{GNN}}(G)(n) = 1\}$ is invariant under r-round CR.
  - $\{(G, n) \mid M_{\mathsf{GNN}}(G)(n) = 1\}$ is a union of r-round C equivalence classes.
  - Each class is definable with a GML formula,

How expressive GML is?

- GML is expressive enough to define the r-round CR class of every pointed graph.
  - For every (G, n) there is a formula ψ s.t. $(H, m) \models \psi$ if and only if $\chi_r^G(n) = \chi_r^H(m)$.
  - We will prove this shortly.
- It does not follow that GML has the same uniform expressivity as GNNs. Why?
  - Let M_{GNN} be a node-feature map defined by an r-layer GNN.
  - Then, $\{(G, n) \mid M_{\mathsf{GNN}}(G)(n) = 1\}$ is invariant under r-round CR.
  - $\{(G, n) \mid M_{\mathsf{GNN}}(G)(n) = 1\}$ is a union of r-round C equivalence classes.
  - Each class is definable with a GML formula, but there can be infinitely many classes!

Characteristic formulae

- Fix a modal depth r.
- The r-type of a pointed graph (G, n) is its equivalence class under r-round graded bisimulation.
- A characteristic formula for this type is a GML formula

$$\vartheta^r_{G,n}$$

such that, for every pointed graph (H, m),

$$H, m \models \vartheta^r_{G,n} \quad \iff \quad (H, m) \sim_r (G, n).$$

- Thus $\vartheta^r_{G,n}$ describes exactly what can be seen from n up to depth r.

Characteristic formulae: depth 0

Let $G = (N, E, g)$ be a $\mathcal{P}(P)$-labeled graph.

- At depth 0, only the node label matters.
- For $n \in N$, define

$$\vartheta^0_{G,n} := \bigwedge_{p \in g(n)} p \ \wedge \bigwedge_{p \in P \setminus g(n)} \neg p.$$

- Then

$$H, m \models \vartheta^0_{G,n} \iff g_H(m) = g_G(n).$$

Characteristic formulae: induction step

Suppose $\vartheta^r_{G,u}$ has been defined for all neighbours $u \in E(n)$.

- Let $\tau_1, \ldots, \tau_s$ be the distinct r-types occurring among the neighbours of n.
- Let k_i be the number of neighbours of n of type τ_i.
- Choose a characteristic formula $\vartheta^r_{\tau_i}$ for each τ_i.
- Then $\vartheta^{r+1}_{G,n}$ says:
  - the current node has the same labels as n;
  - it has exactly k_i neighbours of type τ_i;
  - every neighbour has one of these types.

Characteristic formulae: recursive definition

We may define

$$\vartheta^{r+1}_{G,n}$$

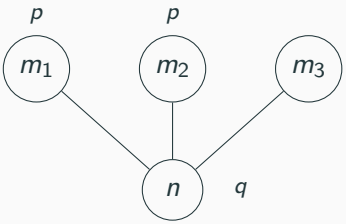
as

$$\vartheta^{0}_{G,n} \wedge \bigwedge_{i=1}^{s} \left(\Diamond_{\geq k_i} \vartheta^{r}_{\tau_i} \wedge \neg \Diamond_{\geq k_i+1} \vartheta^{r}_{\tau_i} \right)$$

$$\wedge \neg \Diamond \left(\bigwedge_{i=1}^{s} \neg \vartheta^{r}_{\tau_i} \right).$$

- The first conjunct fixes the node labels.
- The second part fixes the exact number of neighbours of each previous type.
- The last part says that there are no neighbours of any other type.

Example of a characteristic formula



- Take $r = 1$. Node n satisfies $q, \neg p$; among its neighbors, exactly two satisfy $p, \neg q$, and exactly one satisfies $\neg p, \neg q$.
- The 1-type of (G, n) is pinned down by counting, for each atomic type realized in $E(n)$, how many neighbors realize it.
- Characteristic formula:

$$\begin{aligned}\vartheta^1_{G,n} \;= (q \wedge \neg p) \\ \wedge \Diamond^{=2}(p \wedge \neg q) \wedge \Diamond^{=1}(\neg p \wedge \neg q) \\ \wedge \neg\Diamond(\neg(p \wedge \neg q) \wedge \neg(\neg p \wedge \neg q)).\end{aligned}$$

- Exact counting can be writting with a $\geq k \wedge \neg{\geq}k{+}1$ pair.
- Any $(H, m) \models \vartheta^1_{G,n}$ is 1-bisimilar to (G, n): same label at m, same neighbor counts for combinations of p and q.

GML characterises local bisimulation types

Proposition 4

For every $r \in \mathbb{N}$ and every pointed graph (G, n), the characteristic formula

$$\vartheta^r_{G,n}$$

is a GML-formula of modal depth r such that

$$H, m \models \vartheta^r_{G,n} \quad \iff \quad (H, m) \sim_r (G, n).$$

- The proof is by induction on r, which we essentially already covered.
- Hence GML can define every local graded bisimulation type of a pointed graph.
- Since r-round graded bisimulation coincides with r-round CR equivalence, GML can also define every r-round CR-type of pointed graphs.

Toolkit: GML, bisimulation, and colour refinement

- Graded modal logic is a language for CR-type message passing.
- Bisimulation is an invariance from logic that corresponds to colour refinement.
- Charcteristic formulae can define all l-round CR-types.
- Next we show how these concepts can be used to yield results about GNNs.

Part 4: Logical characterisations of GNNs

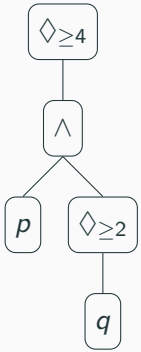
- Next we use tools from the previous lecture to study GNNs.
  - Distinguishability.
    When can two graphs be separated by a GNN?
  - Uniform expressivity:
    What are the properties that a single GNN can express over all graphs?
  - Non-uniform expressivity:
    What properties can be expressed if the used GNN can depend of size of the graph?
- What do we cover next?
  - We prove that GNNs are at least as expressive than GML.
    By showing how formulae of modal logic can be evaluated using a GNN.
  - We show that graded modal logic and GNNs have the same distinguishing power.
  - We prove that non-uniform expressivity of GNNs and GML coincide.
    Restricted to graphs of fixed size, GML can express all l-round CR colour classes.
- Main sources:
  - Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, Juan Pablo Silva: The Logical Expressiveness of Graph Neural Networks. ICLR 2020.

Evaluating GML with GNNs

- A GML formula can be evaluated by a GNN by induction on its syntax tree, exemplified for $\Diamond_{\geq 4}(p \wedge \Diamond_{\geq 2} q)$.
- The feature vector of a node can store truth values of relevant subformulae:
  $(\Diamond_{\geq 4}(p \wedge \Diamond_{\geq 2} q), (p \wedge \Diamond_{\geq 2} q), \Diamond_{\geq 2} q, p, q)$.
- Atomic propositions are read from the input label.
- Boolean connectives are computed by feedforward neural networks.
- The graded modality

  $$\Diamond_{\geq k} \varphi$$

  is evaluated by summing the truth values of φ over the neighbours and checking whether the sum is at least k.



Syntax tree

Evaluating graded modalities

Suppose the current feature of a node records whether φ is true.

- For example, φ expresses "red" node p_{red}.
- Assume $G = (N, E, g)$ with $g : N \to \{0,1\}^3$ hot-one encoding of colours red, green and blue (red: $(1,0,0)$, green: $(0,1,0)$, blue: $(0,0,1)$).
- Aggregate the truth values over the neighbours:

$$s(n) := \sum_{m \in E(n)} (1 \cdot g(m)_1 + 0 \cdot g(m)_1 + 0 \cdot g(m)_3) = \sum_{m \in E(n)} g(m)_1$$

- Then we can express "having no less than k red nodes":

$$G, n \models \Diamond_{\geq k} \varphi \quad \iff \quad \text{trReLU}(s(n) - k - 1) = 1.$$

- A simple feedforward network can implement such threshold tests.
- Thus one message-passing layer can evaluate one modal step.

Simple homogeneous GNNs

- In general, different layers of a GNN may use different Agg and Comb functions
  - In homogeneous GNNs each layer is identical.

Simple homogeneous GNNs

- In general, different layers of a GNN may use different Agg and Comb functions
  - In homogeneous GNNs each layer is identical.
- In general, Agg and Comb function can be any functions.
  - Simple GNNs use (weighted) sums passed through non-linear activation function as Comb and Agg.

$$\text{Agg}(X) := \sum_{\mathbf{x} \in \mathbf{X}} x$$

$$\text{Comb}(\mathbf{x}, \mathbf{y}) := \sigma(\mathbf{x}\mathbf{C} + \mathbf{y}\mathbf{B} + \mathbf{b}),$$

where $A, C \in \mathbb{R}^{l \times l}$, $b \in \mathbb{R}^{l}$, and σ is the truncated ReLU activation defined by

$$\sigma(x) := \min(\max(0, x), 1).$$

Simple homogeneous GNNs

- In general, different layers of a GNN may use different Agg and Comb functions
  - In homogeneous GNNs each layer is identical.
- In general, Agg and Comb function can be any functions.
  - Simple GNNs use (weighted) sums passed through non-linear activation function as Comb and Agg.

$$\text{Agg}(X) := \sum_{\mathbf{x} \in \mathbf{X}} x$$

$$\text{Comb}(\mathbf{x}, \mathbf{y}) := \sigma(\mathbf{xC} + \mathbf{yB} + \mathbf{b}),$$

where $A, C \in \mathbb{R}^{l \times l}$, $b \in \mathbb{R}^{l}$, and σ is the truncated ReLU activation defined by

$$\sigma(x) := \min(\max(0, x), 1).$$

- For any GML formula, there is a particular choice of B, C, and b that are used to simulate the logical formula.

From GML formulae to GNNs

Proposition 5

For every GML formula φ, there is a fixed-depth simple homogeneous message-passing GNN that defines the same node classifier:

$$c_G^{\varphi}(n) = 1 \quad \iff \quad G, n \models \varphi.$$

- The depth of the GNN is controlled by the modal depth of φ.
  - Direct implementation yields depth of the syntax tree, which can be optimised to modal depth.
- The construction is by structural induction on φ.
- This gives a logical way to build GNNs from GML formulae.

Back to CR and GNNs: Proof via logic

Proposition 1 (Colour refinement and GNNs [Mor+19])

Let (G, n) and (H, m) be two pointed X-labeled graphs. Let χ_t denote the colouring after t rounds of colour refinement.

1. For every t-layer message-passing GNN with node features g_t, we have

$$\chi_t^G(n) = \chi_t^H(m) \quad \Longrightarrow \quad g_t^G(n) = g_t^H(m).$$

2. Conversely, if colour refinement distinguishes the two pointed graphs after t rounds, that is,

$$\chi_t^G(n) \neq \chi_t^H(m),$$

then there exists a t-layer message-passing GNN such that

$$g_t^G(n) \neq g_t^H(m).$$

Proof of Proposition 1: harder direction

- No need to prove directly that GNNs can simulate CR, we did the required work already!

Proof.

Proof of Proposition 1: harder direction

- No need to prove directly that GNNs can simulate CR, we did the required work already!

Proof.

- Let (G, n) and (H, m) be such that $\chi_t^G(n) \neq \chi_t^H(m)$.
- By Propositions 3 and 4 the characteristic formula $\vartheta_{G,n}^t$ of modal depth t separates the pointed graphs

$$(G, n) \models \vartheta_{G,n}^t \quad \text{and} \quad (H, m) \not\models \vartheta_{G,n}^t.$$

Proof of Proposition 1: harder direction

- No need to prove directly that GNNs can simulate CR, we did the required work already!

Proof.

- Let (G, n) and (H, m) be such that $\chi_t^G(n) \neq \chi_t^H(m)$.
- By Propositions 3 and 4 the characteristic formula $\vartheta_{G,n}^t$ of modal depth t separates the pointed graphs

$$(G, n) \models \vartheta_{G,n}^t \quad \text{and} \quad (H, m) \not\models \vartheta_{G,n}^t.$$

- We can then find a simple homogeneous GNN with t layers that separates (G, n) and (H, m), by Proposition 5.

□

Logical equivalence and distinguishability

We are ready to show that the distinguising powers of GML and GNNs coincide.

Proposition 6 (GNNs and logical equivalence)

Let (G, n) and (H, m) be two pointed X-labeled graphs. TFAE

1. *There exists t-layer message-passing GNN that separates (G, n) and (H, m).*
2. $(H, m) \models \neg\vartheta^t_{G,n}$.
3. *There exists a GML-formula φ of modal depth at most t s.t $(G, n) \models \varphi$ and $(H, m) \not\models \varphi$.*

Logical equivalence and distinguishability

We are ready to show that the distinguising powers of GML and GNNs coincide.

Proposition 6 (GNNs and logical equivalence)

Let (G, n) and (H, m) be two pointed X-labeled graphs. TFAE

1. *There exists t-layer message-passing GNN that separates (G, n) and (H, m).*
2. $(H, m) \models \neg\vartheta^t_{G,n}$.
3. *There exists a GML-formula φ of modal depth at most t s.t $(G, n) \models \varphi$ and $(H, m) \not\models \varphi$.*

Proof.

$(1 \Leftrightarrow 2)$ By Proposition 1, 1. is equivalent to colour refinement yielding $\chi^G_t(n) \neq \chi^H_t(m)$, which is equivalent to $(H, m) \models \neg\vartheta^r_{G,n}$ by Propositions 3 and 4. $(2 \Rightarrow 3)$ Trivial. $(3 \Rightarrow 2)$ Follows from Propositions 2 and 4. □

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m).

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m). By Prop. 1, M is invariant under t-layer CR and hence $\chi_t^G(m) \neq \chi_t^H(m)$.

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m). By Prop. 1, M is invariant under t-layer CR and hence $\chi_t^G(m) \neq \chi_t^H(m)$. By Prop. 3, $(G, n) \not\sim_t (H, m)$ and hence by Prop 4. $(H, m) \not\models \vartheta^t_{G,n}$.

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m). By Prop. 1, M is invariant under t-layer CR and hence $\chi_t^G(m) \neq \chi_t^H(m)$. By Prop. 3, $(G, n) \not\sim_t (H, m)$ and hence by Prop 4. $(H, m) \not\models \vartheta^t_{G,n}$.

$(2 \Rightarrow 1)$ All the steps of the proof go both directions.

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m). By Prop. 1, M is invariant under t-layer CR and hence $\chi_t^G(m) \neq \chi_t^H(m)$. By Prop. 3, $(G, n) \not\sim_t (H, m)$ and hence by Prop 4. $(H, m) \not\models \vartheta^t_{G,n}$.

$(2 \Rightarrow 1)$ All the steps of the proof go both directions.

$(2 \Rightarrow 3)$ Assume $(H, m) \models \neg\vartheta^t_{G,n}$ and notice that $(G, n) \models \vartheta^t_{G,n}$.

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m). By Prop. 1, M is invariant under t-layer CR and hence $\chi_t^G(m) \neq \chi_t^H(m)$. By Prop. 3, $(G, n) \not\sim_t (H, m)$ and hence by Prop 4. $(H, m) \not\models \vartheta^t_{G,n}$.

$(2 \Rightarrow 1)$ All the steps of the proof go both directions.

$(2 \Rightarrow 3)$ Assume $(H, m) \models \neg\vartheta^t_{G,n}$ and notice that $(G, n) \models \vartheta^t_{G,n}$.

$(3 \Rightarrow 2)$ Let φ be a GML-formula of modal depth at most t s.t $(G, n) \models \varphi$ and $(H, m) \not\models \varphi$.

A more detailed proof

$(1 \Rightarrow 2)$ Assume that there is a node-feature map M computed by a t-layer GNN that separates (G, m) and (H, m). By Prop. 1, M is invariant under t-layer CR and hence $\chi_t^G(m) \neq \chi_t^H(m)$. By Prop. 3, $(G, n) \not\sim_t (H, m)$ and hence by Prop 4. $(H, m) \not\models \vartheta^t_{G,n}$.

$(2 \Rightarrow 1)$ All the steps of the proof go both directions.

$(2 \Rightarrow 3)$ Assume $(H, m) \models \neg\vartheta^t_{G,n}$ and notice that $(G, n) \models \vartheta^t_{G,n}$.

$(3 \Rightarrow 2)$ Let φ be a GML-formula of modal depth at most t s.t $(G, n) \models \varphi$ and $(H, m) \not\models \varphi$. By Proposition 2 it follows that $(G, n) \not\sim_t (H, m)$. Proposition 4 then yields $(H, m) \models \neg\vartheta^t_{G,n}$.

Implications to GNN-architecture selection

- GNNs have the same distinguishing power than GML.
- GML formulas can be simulated by simple homogeneous GNNs
- Simple homogeneous GNNs have the same distinguishing power as all GNNs!

GNNs and GML have the same non-uniform expressivity

Let's consider the non-uniform case, where GNNs may depend on the the graph size:

Proposition 7

For every sequence of fixed depth message-passing GNNs $(M_s)_{s\in\mathbb{N}}$ that compute Boolean node classifiers there exists a sequence of GML formulae $(\varphi_s)_{s\in\mathbb{N}}$ such that for every labelled pointed graph (G, n)

$$M_s(G, n) = 1 \text{ if and only if } G, n \models \varphi_s,$$

where $s = |G|$, and vice versa.

Proof.

Fix $s \in \mathbb{N}$ and t be the number of layers of M_s. By Props. 1 and 3, the model of M_s is invariant under t-round graded bisimulation. Set φ_s to be the disjunction of those $\psi^t_{G,n}$ s.t. $|G| = s$ and $M_s(G, n) = 1$. By Prop. 4, the disjuncts of φ_s characterise the bisimulation equivalence classes where $M_s(G, n) = 1$. Vice versa by Prop. 5. □

Proof of Proposition 7 in more detail

- Let M_s be a t-layer GNN that is used to decide G, n of size s.
- We have shown that M_s is invariant under t-round CR.
- Restricted to graphs of size s, there are only finitely many t-round CR types. Easy to proof by induction on t.
- We have shown that each of those t-round CR types are infact, t-round graded bisimulation types.
- For each of those finite types, put a representative pair (G', m') to a set Φ, if $M_s(G', m') = 1$.
- Claim:

$$M_s(G, n) = 1 \text{ if and only if } G, n \models \bigvee_{(G', n') \in \Phi} \vartheta^t_{G', n'},$$

  where $\vartheta^r_{G', m'}$ is the t-characteristic formula of G, n.
- Proposition 6 shows that vice versa holds even without restricting the graph size.

Uniform expressivity

- GNNs can simulate logical formulae
  ($\Rightarrow$) GNNs are at least as expressive than GML.
- GML is invariant under so-called bounded colour refinement
  - Consider *CR* that, for some fixed k, does not differentiate multiplicities larger than k
  - $\chi_{t+1}(n) = \chi_{t+1}(m)$, if $\chi_t(n) = \chi_t(m)$ and, for any colour, m and n have the same number of neighbours of that colour, or both have at least k neighbours of that colour.

Uniform expressivity

- GNNs can simulate logical formulae
  ($\Rightarrow$) GNNs are at least as expressive than GML.
- GML is invariant under so-called bounded colour refinement
  - Consider *CR* that, for some fixed k, does not differentiate multiplicities larger than k
  - $\chi_{t+1}(n) = \chi_{t+1}(m)$, if $\chi_t(n) = \chi_t(m)$ and, for any colour, m and n have the same number of neighbours of that colour, or both have at least k neighbours of that colour.
  - If node property is definable by a GML formula with modal depth d and graded modalities $\Diamond_{\geq i}$ where $i \leq k$, then the node property is invariant under d-round k-bounded CR.
- Node property: more neighbours satisfying p than neighbours satisfying q is not invariant under bounded CR, and thus not in GML.

Uniform expressivity

- GNNs can simulate logical formulae
  ($\Rightarrow$) GNNs are at least as expressive than GML.
- GML is invariant under so-called bounded colour refinement
  - Consider *CR* that, for some fixed k, does not differentiate multiplicities larger than k
  - $\chi_{t+1}(n) = \chi_{t+1}(m)$, if $\chi_t(n) = \chi_t(m)$ and, for any colour, m and n have the same number of neighbours of that colour, or both have at least k neighbours of that colour.
  - If node property is definable by a GML formula with modal depth d and graded modalities $\Diamond_{\geq i}$ where $i \leq k$, then the node property is invariant under d-round k-bounded CR.
- Node property: more neighbours satisfying p than neighbours satisfying q is not invariant under bounded CR, and thus not in GML.
- Easy to design a GNN for this property.
- GNNs are at strictly more expressive than GML.

Overview of the non-recursive case

- GNNs and GML have the same distinguishing power. This is captured by CR, or equivalently by graded bisimulation.
- When parameterised with the size of the input graph, GNNs and GML have the same non-uniform expressivity.
- In the uniform case, GNNs are strictly more expressive than GML.
- Literature gives some cases where the uniform expressivity of GNNs can be exactly characterised with a logic.
  - In these logics are enriched with some mild form of unbounded counting such as counting terms t and comparisons between those terms $t \leq t'$:

    $$t ::= \Diamond_{=x} \varphi,$$

    returns the number of neighbours satisfying φ.
- There are simple properties that cannot be uniformly expressed with GNNs
  - Not-invariant under CR (triangle detection) or fixed round CR (reachability).

Lecture 3: Basics of recurrent GNNs, μ-calculus, and their uniform expressivity

Roadmap

- We now move from fixed-depth GNNs to recurrent GNNs.
- We revise the difference between:

  distinguishing power and uniform expressivity.

- We show that recurrent GNNs do not add distinguishing power.
- But they do add uniform expressive power! yes!
- Example: reachability.

Part 5: GNNs and recursion

- What do we cover:
  - Different approaches to add recursion to GNNs.
  - Expressivity of these recurrent GNN alternatives.
  - Two extension of GML to define recursive classifiers.
- Main sources
  - Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical Characterizations of Recurrent Graph Neural Networks with Reals and Floats. Neurips 2024.
  - Jeroen Bollen, Jan Van den Bussche, Stijn Vansummeren, Jonni Virtema. Halting Recurrent GNNs and the Graded μ-Calculus. KR 2025.
  - Eran Rosenbluth, Martin Grohe. Repetition Makes Perfect: Recurrent Graph Neural Networks Match Message-Passing Limit. AAAI 2026.

Recurrent GNNs

Recurrent = Keep repeating GNN layers.

- In fact, the original GNNs in the paper by Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini (*IEEE TNN* 2009) were recurrent!
- They define a vertex feature update via a global transition function $F_{\boldsymbol{\theta}}$:

$$g_{t+1}(n) = F_{\boldsymbol{\theta}}\big(g_t(n)\big).$$

Recurrent GNNs

How are these trained?

- Banach fixed-point theorem: if $F_{\boldsymbol{\theta}}$ is a contraction, i.e. $\exists\, \mu < 1$ with $\|F_{\boldsymbol{\theta}}(x) - F_{\boldsymbol{\theta}}(y)\| \leq \mu \|x - y\|$ (for all x, y) then a unique fixed point x^* exists, and the iteration above converges to it exponentially fast from any x_0.

- **Training** (gradient descent):
  1. iterate $g_{t+1} = F_{\boldsymbol{\theta}}(g_t)$ until (approximately) converged to g^*;
  2. compute the loss and its gradient w.r.t. $\boldsymbol{\theta}$;
  3. update $\boldsymbol{\theta}$.

Recurrent GNNs: Challenging to model

Theoretically, it is hard to work with fixpoints of such contractions.

Also, how is convergence decided? How many steps.

If only up to finite iteration, why not use simple non-recurrent GNNs?

- There is no unique recurrent GNN model.
- Commonality: repeated message passing
- Differences: When the recurrent GNN is "done" with a node or graph.

Recurrent GNNs

Shared recurrence

$$g_t^0(n) = \mathsf{Init}(g(n)), \qquad g_t^{t+1}(n) = \mathsf{Comb}(g_t(n), \mathsf{Agg}\{\{g_t(m) \mid m \in E(n)\}\}) .$$

We denote by $\mathcal{L} = (\mathsf{Ini}, L)$ and $L = (\mathsf{Comb}, \mathsf{Agg})$ the recurrent layer.

Architectural choices

- **Domain:** reals, floats, rationals, bitstrings.
- **Aggregation:** sum, ordered float-sum, arbitrary multiset map.
- **Extra's:** graph size, global aggregation, random initialization.

Semantic choices

- **Stopping:** accepting vector, global halt, finished flag, fixed point.
- **Output:** node classifier vs. feature transformer.

Fixed-depth versus recurrent GNNs

- A fixed-depth GNN applies a fixed sequence of layers

$$L^{(1)}, \ldots, L^{(\ell)}.$$

- The number of message-passing rounds is fixed in advance.
- A recurrent GNN applies the same layer repeatedly:

$$G_0, \; G_1 = L(G_0), \; G_2 = L(G_1), \; \ldots$$

- The layer L may at times be composition of more single layers.
- The number of rounds may depend on the input graph.
- Thus recurrent GNNs are designed to express iterative or recursive computations.

Challenge of recursion

- From one layer to another recurrent GNNs behave as their non-recurrent counterparts.
- Given an labelled input graph, each layer computes a feature transformation

$$G[X] \to \mathcal{G}[Y]$$

via the standard equation

$$g'(n) \coloneqq \mathrm{Comb}_{\boldsymbol{\theta}}\Big(g(n), \mathrm{Agg}_{\boldsymbol{\theta}}\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big).$$

- Challenge: When do we read the output of the computation?
- We cover two alternative approaches.

Two alternative approaches

- Accepting state model
- Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical Characterizations of Recurrent Graph Neural Networks with Reals and Floats. Neurips 2024.
- Halting model
- Jeroen Bollen, Jan Van den Bussche, Stijn Vansummeren, Jonni Virtema. Halting Recurrent GNNs and the Graded μ-Calculus. KR 2025.

Part 5 (a): GNNs and recursion: Accepting state model

- Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical Characterizations of Recurrent Graph Neural Networks with Reals and Floats.

Accepting state model

Let X be a finite input label set.

- A recurrent GNN is a tuple
$$\mathcal{N} = (\mathsf{In}, L, \mathsf{Hlt}).$$
- The map
$$\mathsf{In} : X \to \mathbb{R}^d$$
initializes node features.
- The layer L is an aggregate-combine layer
$$L : \mathcal{G}[\mathbb{R}^d] \to \mathcal{G}[\mathbb{R}^d].$$
- The function
$$\mathsf{Hlt} : \mathbb{R}^d \to \mathbb{B}$$
decides whether a node is ready to halt.

Basic graph concepts: pointwise lifting

- Let
$$\alpha \colon X \to Y$$
be a function between label sets. We write
$$\alpha^{\uparrow} \colon \mathcal{G}[X] \to \mathcal{G}[Y]$$
for the pointwise lifting of α to labeled graphs.
- For an X-labeled graph
$$G = (N, E, g),$$
it is defined by
$$\alpha^{\uparrow}(G) := (N, E, \alpha \circ g).$$

Runs in the accepting state model

Let $G = (N, E, g)$ be an X-labeled graph.

- The initial graph is
$$G_0 := \mathsf{In}^{\uparrow}(G).$$
- The recurrent layer is applied repeatedly:
$$G_{i+1} := L(G_i).$$
- A run accepts (G, n) iff
$$\mathsf{Hlt}(g_k(n)) = 1 \text{ for some } k \in \mathbb{N},$$
with $G_k = (N, E, g_k)$.

Acceptance by halting at the node level.

Acceptance: the GNN decides itself

Acceptance feature vector f such that $\mathsf{Hlt}(f) = 1$.

Acceptance condition

$\mathcal{G}$ **accepts** (G, v) iff n visits an accepting feature vector at least once.

- Termination is *signaled by halting function*. There is no externally fixed number of rounds.

Reachability of a label p

Given $\mathcal{P}(P)$-labeled graph $G = (N, E, g)$.

Node Property: from n there is a path to some node labeled p.

$$g_0(n) = \begin{cases} 1 & p \in g(n) \\ 0 & \text{else} \end{cases} \qquad g_t(n) = \mathrm{trReLU}\Big(g_{t-1}(n) + \textstyle\sum_{(n,m)\in E} g_{t-1}(m)\Big)$$

with $\mathrm{trReLU}(x) = \min(\max(0, x), 1)$ and $f = 1$ accepting feature vector (scalar)

- After t rounds: $g_t(n) = 1$ iff some node labeled p is reachable from n in $\leq t$ steps.
- n is accepted iff it *ever* reaches 1: exactly reachability!
- Note: no constant-depth GNN (and no GML-formula) can do this; recurrence is essential.

Centre-point property

Node Property: There exists a $k \in \mathbb{N}$ such that every directed path from n reaches an end node in exactly k steps.

$$h_t(n) = (p_t(n), b_t(n)) \in \mathbb{R}^2,$$

Initialize

$$h_0(n) = (0, 0) \qquad \text{for all } n.$$

Aggregate out-neighbours E^+ by

$$\mathrm{Agg}\{\{h_t(m) \mid m \in E^+(n)\}\} = (d_t(n), s_t(n)),$$

where

$$d_t(n) = |E^+(n)|, \qquad s_t(n) = \sum_{m \in E^+(n)} b_t(m).$$

Centre-point property: combine

The aggregate gives

$$(d, s) = \left(|E^+(n)|, \sum_{m \in E^+(n)} b_t(m) \right).$$

Thus d is the number of out-neighbours, and $s = d$ means that all out-neighbours currently have value 1.

The combine map is

$$\mathrm{Comb}\big((p, b), (d, s)\big) = (1, B),$$

where

$$B = \begin{cases} 1 & \text{if } p = 0 \text{ and } d = 0, \\ 1 & \text{if } p = 1 \text{ and } d > 0 \text{ and } s = d, \\ 0 & \text{otherwise.} \end{cases}$$

Centre-point property: what the combine does

- The first round is a warm-up round:

$$b_1(n) = 1 \quad \Longleftrightarrow \quad n \text{ is an end node.}$$

- From then on, the same recurrent layer propagates the value backwards:

$$b_{t+1}(n) = 1$$

iff

$$n \text{ is not an end node and all out-neighbours } m \text{ satisfy } b_t(m) = 1.$$

- Therefore $b_{t+1}(n) = 1$ says: every directed path from n reaches an end node in exactly $t + 1$ steps.

The phase bit p separates the base case from the recursive case.

Directed to undirected

Encoding idea

Replace every directed edge $u \to v$ by a small undirected labelled gadget:

$$u \to v \qquad \rightsquigarrow \qquad u \;-\; e^{\mathrm{out}}_{u,v} \;-\; e^{\mathrm{in}}_{u,v} \;-\; v.$$

- The original nodes u, v keep their old labels.
- The two new nodes carry fresh labels

  $$\text{out}, \qquad \text{in}.$$

- To follow a directed edge, move through the pattern

  $$\text{out-node then in-node}.$$

Expressive power?

- As before, how to assess the expressive power of recurrent GNNs in the accepting state mode?
- Can we find logical characterization?

The answer is yes!... using extension of GML.

Syntax of ω-GML

Fix a finite set of propositions P

ω-**GML** extends graded modal logic with infinitary disjunctions:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Diamond_{\geq k}\varphi$$

where $p \in P$, $k \geq 1$ and

$$\psi := \varphi \mid \bigvee_{i \in I} \varphi_i$$

and I ranges over possibly countable index sets.

ω-GML: semantics

Let $G = (N, E, g)$ be a $\mathcal{P}(P)$-labeled graph.

- $$G, n \models p \quad \iff \quad p \in g(n).$$
- Boolean connectives are interpreted as usual.
- $$G, n \models \Diamond_{\geq k} \varphi \text{ iff } \big|\{m \in E(n) \mid G, m \models \varphi\}\big| \geq k.$$
- $$G, n \models \bigvee_{i \in I} \varphi_i \text{ iff there exists an } i \text{ such that } G, n \models \varphi_i.$$

So: one (possibly infinite) disjunction over ordinary GML-formulae: no infinite conjunctions, no nesting of infinite disjunctions.

Reachability in ω-GML

Node property

From the current node n, there is a path to some node labelled p.

Define finite-depth GML formulae R_t by

$$R_0 \coloneqq p, \qquad R_{t+1} \coloneqq R_t \vee \Diamond R_t.$$

- R_t says: a p-node is reachable in at most t steps.
- Hence unbounded reachability is the ω-GML formula

$$\mathrm{Reach}_p \coloneqq \bigvee_{t \in \mathbb{N}_0} R_t.$$

- Equivalently,

$$\mathrm{Reach}_p \equiv \bigvee_{t \in \mathbb{N}_0} \Diamond^t p, \qquad \Diamond^0 p \coloneqq p, \quad \Diamond^{t+1} p \coloneqq \Diamond(\Diamond^t p).$$

Centre-point property in ω-GML

Node property

There exists a $k \in \mathbb{N}_0$ such that every directed path from n reaches an end node in exactly k steps.

Here $\Diamond$ range over outgoing neighbours.
Define finite-depth GML formulae C_t by

$$C_0 \coloneqq \Diamond_{=0} \top, \qquad C_{t+1} \coloneqq \Diamond C_t \wedge \neg \Diamond \neg C_t.$$

- C_0 says: n is already an end node.
- C_{t+1} says: n has at least one outgoing neighbour, and all outgoing neighbours satisfy C_t.
- Thus C_t says: every directed path from n reaches an end node in exactly t steps.

$$\text{Centre} \coloneqq \bigvee_{t \in \mathbb{N}_0} C_t.$$

Main theorem

Theorem

Recurrent GNNs in the accepting-state model and ω-GML have the same expressive power.

Proof strategy. Neither direction is proved directly. Both go through an intermediate *automaton* model that strips away the real numbers but keeps the message-passing structure:

$$\omega\text{-GML} \longleftrightarrow \text{CMPA} \longleftrightarrow \text{recurrent GNNs}$$

Why automata? Counting message passing automata

A recurrent GNN in the accepting-state model is just a message-passing machine whose "states" happen to be vectors in $\mathbb{R}^d$. Abstracting the state space gives:

Definition (CMPA)

A counting message passing automaton over X is $\mathcal{A} = (Q, \pi, \delta, F)$:

- Q: an at most **countable** set of states,
- $\pi \colon \mathcal{P}(X) \to Q, \quad \delta \colon Q \times \mathcal{M}(Q) \to Q, \quad F \subseteq Q.$

Computation and acceptance exactly as for GNNs: each node updates from its own state and the *multiset* of out-neighbour states; accept if an accepting state is ever visited.

- Same skeleton as a recurrent GNNs: only $\mathbb{R}^d$ is replaced by countable Q.

The key tool: r-types

- The r-type of a pointed graph (G, w) is its equivalence class under r-round *graded bisimulation* $\sim_r$: same labels at the points, and for r rounds, matching out-neighbours with exact counts.
- A characteristic formula for this type is a GML-formula $\vartheta^r_{G,w}$ of modal depth r such that, for every pointed graph (H, v),

$$H, v \models \vartheta^r_{G,w} \quad \iff \quad (H, v) \sim_r (G, w).$$

- Thus $\vartheta^r_{G,w}$ describes exactly what can be seen from w up to depth r.

- Each pointed graph has **exactly one** r-type for each r; there are countably many r-types overall.
- **Type decomposition:** every GML-formula of modal depth r is logically equivalent to a countable disjunction of formulae $\vartheta^r_{G,w}$. Hence so is every ω-GML-formula (disjunct by disjunct).

The counting type automaton

There is one "universal" CMPA whose run computes r-types:

Counting type automaton

- States Q = all characteristic formulae $\vartheta^r_{G,w}$ (all r, all (G, w); countable).
- π: maps a label set P to the 0-type $\bigwedge_{p\in P} p \wedge \bigwedge_{p\notin P} \neg p$.
- $\delta(\vartheta, N)$: from own r-type ϑ and the multiset N of the neighbours' r-types, assemble the $(r+1)$-type:

$$\delta(\vartheta, N) = \vartheta_0 \wedge \bigwedge_{\ell\geq 1} \{\Diamond_{=\ell}\, \sigma \mid N(\sigma) = \ell\} \wedge \Diamond_{=|N|} \top$$

where ϑ_0 is the 0-type part (the labels) of ϑ.

Invariant: after round r, node w of G is in state $\vartheta^r_{G,w}$: the automaton *is* the type construction, run distributedly. Only the accepting set F remains to be chosen.

ω-GML $\Rightarrow$ CMPA

Given $\chi = \bigvee_{\varphi \in S} \varphi$:

- rewrite each φ (of depth r) as a disjunction of characteristic formulae $\vartheta^r_{G,w}$ (type decomposition);
- take the counting type automaton with $F = \{\text{all } \vartheta^r_{G,w} \text{ appearing in any disjunct}\}$.

Then $(G, w) \models \chi$ iff w visits an accepting state.

CMPA $\Rightarrow \omega$-GML

The crucial lemma:

Lemma (r-types determine runs)

$(G, w) \sim_r (H, v) \iff$ (G, w) and (H, v) are in the same state at every round $\leq r$, *in every* CMPA

(proof: induction on r; multisets of neighbour states $\leftrightarrow$ exact counts of neighbour r-types).

So an automaton's behaviour up to round r depends only on the r-type. Take

$$\Phi = \{\vartheta^r_{G,w} \mid \mathcal{A} \text{ accepts } (G, w) \text{ in round } r\}, \qquad \chi_{\mathcal{A}} := \bigvee_{\vartheta \in \Phi} \vartheta \ \in \ \omega\text{-GML}.$$

Closing the loop with recurrent GNNs

Recurrent GNNs $\Rightarrow$ ω-GML.

The lemma also holds verbatim for recurrent GNNs:
$(G, w) \sim_r (H, v) \Rightarrow$ same feature vector up to round r in every recurrent GNN (a GNN is just a message passer).

So again $\bigvee\{\vartheta^r_{G,w} \mid \mathcal{G} \text{ accepts } (G, w) \text{ in round } r\}$ works.

Closing the loop with recurrent GNNs

ω-GML $\Rightarrow$ Recurrent GNNs

Translate the formula to a counting type automaton, then simulate it by a GNN over using *natural numbers* as features:

- Each r-type $\leftrightarrow$ its minimal tree model $\leftrightarrow$ a binary string $\leftrightarrow$ an integer.
- AGG: decode the neighbour multiset's trees, hang them under a fresh root, re-encode.
- COM: fix the root's labels using the node's own code.
- F: the integers coding accepting r-types.

Recurrent GNNs evaluate ω-GML

For every ω-GML-formula χ, we can thus construct one recurrent GNN $\mathcal{N}_\chi$ such that, for all pointed graphs (G, w),

$$(G, w) \models \chi \quad \iff \quad \mathcal{N}_\chi \text{ accepts } w \text{ on } G.$$

What about distinguishing power

Proposition 8

Recurrent GNNs in the accepting state model coincide with non-recurrent GNNs when it comes to distinguishability.

Using logic.

$(G, m) \models \phi$ and $(H, n) \not\models \phi$ for ω-GML formula $\phi = \bigvee \psi_i$ iff exist i such that $(H, n) \not\models \psi_i$ and $(G, m) \models \psi_i$ for GML ψ.

Recall, for distinguishability $GML = GNN$ and recurrent GNNs $= \omega$-GNN.

Non-reachability is not expressible in ω-GML

Claim

No ω-GML formula φ satisfies, for every graph G and node m:

$$G, m \models \varphi \quad \iff \quad m \text{ does not reach } p.$$

Proof.

1. Suppose φ exists. Fix G, m with $m \not\to p$. Then $G, m \models \varphi$.

□

Non-reachability is not expressible in ω-GML

Claim

No ω-GML formula φ satisfies, for every graph G and node m:

$$G, m \models \varphi \quad \iff \quad m \text{ does not reach } p.$$

Proof.

1. Suppose φ exists. Fix G, m with $m \not\to p$. Then $G, m \models \varphi$.
2. φ is a (possibly infinite) disjunction $\bigvee_i \varphi_i$, so $G, m \models \varphi_i$ for some i; let k be the modal depth of φ_i.

□

Non-reachability is not expressible in ω-GML

Claim

No ω-GML formula φ satisfies, for every graph G and node m:

$$G, m \models \varphi \quad \iff \quad m \text{ does not reach } p.$$

Proof.

1. Suppose φ exists. Fix G, m with $m \not\to p$. Then $G, m \models \varphi$.
2. φ is a (possibly infinite) disjunction $\bigvee_i \varphi_i$, so $G, m \models \varphi_i$ for some i; let k be the modal depth of φ_i.
3. Build G', m' agreeing with m on its k-neighborhood, but where p is reachable from m' in exactly $k+1$ steps.

□

Non-reachability is not expressible in ω-GML

Claim

No ω-GML formula φ satisfies, for every graph G and node m:

$$G, m \models \varphi \quad \iff \quad m \text{ does not reach } p.$$

Proof.

1. Suppose φ exists. Fix G, m with $m \not\to p$. Then $G, m \models \varphi$.
2. φ is a (possibly infinite) disjunction $\bigvee_i \varphi_i$, so $G, m \models \varphi_i$ for some i; let k be the modal depth of φ_i.
3. Build G', m' agreeing with m on its k-neighborhood, but where p is reachable from m' in exactly $k+1$ steps.
4. Depth-k formulas cannot distinguish m from m', so $G', m' \models \varphi_i$, hence $G', m' \models \varphi$ — but m' reaches p. Contradiction.

□

Conclusion: Accepting state model

- Non-reachability and any other property not in ω-GML cannot be compiled into the recurrent network model.
- Everything in ω-GML can be compiled, however.

Good and bad: accepting-state model

Good

- Very simple semantics:
- Natural for eventual or recursive properties.
- Captures reachability-like computations directly:

$$p \vee \Diamond p \vee \Diamond^2 p \vee \cdots$$

- Clean logical characterisation:

accepting-state RGNNs $\equiv \omega$-GML.

Bad

- Acceptance is existential in time: accept if an accepting state is reached at some round.
- Negative instances do not produce a final rejecting state; they simply never accept.
- Thus the model is closer to a semi-decision procedure than to a terminating classifier.
- Complements need not be expressible: reachability is expressible, but non-reachability is not.

Part 5 (b): Basics of halting recurrent GNNs

- What is covered next:
  - Introduction of halting-classifier-based recurrent GNNs.
  - We prove that on connected graph these GNNs have the same distinguishing power as fixed-depth GNNs.
  - The model can express reachability (shown here) and non-reachability (shown later).
  - Most likely, the model cannot express centre-point property.
- Reference:
  - Jeroen Bollen, Jan Van den Bussche, Stijn Vansummeren, Jonni Virtema. Halting Recurrent GNNs and the Graded μ-Calculus. KR 2025.

Halting model: Intuition

- AC-GNN with no limit on the number of layers.
- Each node internally indicates when their feature vector is ready to be read.
- Global read-out is used only to check when all nodes are ready to halt.
  - This indicates the final layer of the GNN!

Halting model: Formally

Let X be a finite input label set.

- A halting-classifier-based recurrent GNN is a tuple

$$\mathcal{N} = (\mathsf{In}, L, \mathsf{Hlt})$$

- The map

$$\mathsf{In} : X \to \mathbb{R}^d$$

initializes node features of an input graph $\mathcal{G}(X)$.

- The layer L is an aggregate-combine layer

$$L : \mathcal{G}[\mathbb{R}^d] \to \mathcal{G}[\mathbb{R}^d].$$

- The function

$$\mathsf{Hlt} : \mathbb{R}^d \to \mathbb{B}$$

decides whether a node is ready to halt.

Runs in the halting model

Let $G = (N, E, g)$ be an X-labeled graph.

- The initial graph is
$$H_0 \coloneqq \mathsf{In}^{\uparrow}(G).$$
- The recurrent layer is applied repeatedly:
$$H_{i+1} \coloneqq L(H_i).$$
- A run is complete at time k if every node halts:
$$\mathsf{Hlt}(h_k(n)) = 1 \qquad \text{for all } n \in N,$$
and k is the smallest number where this holds.
- The output of graph is then $\mathcal{N}(G) = H_k$.

Runs in the halting model

Let $G = (N, E, g)$ be an X-labeled graph.

- The initial graph is
$$H_0 := \mathsf{In}^{\uparrow}(G).$$
- The recurrent layer is applied repeatedly:
$$H_{i+1} := L(H_i).$$
- A run is complete at time k if every node halts:
$$\mathsf{Hlt}(h_k(n)) = 1 \qquad \text{for all } n \in N,$$
and k is the smallest number where this holds.
- The output of graph is then $\mathcal{N}(G) = H_k$.
- Difference to the accepting state model:
  - Halting relies on a global read-out. Layer-uniform output-layer is H_k.

Halting model

- A recurrent GNN is halting if every finite input graph has a complete run.
- Let $\mathcal{N}$ be a halting GNN with with Agg-Comb layer L.
  - For every input graph G there exists k such that $\mathcal{N}(G) = H_k$.
  - If $\mathcal{N}_k$ is the (non-recursive) k-layer GNN whose layers compute L, then $\mathcal{N}_k(G) = H_k$

Halting model

- A recurrent GNN is halting if every finite input graph has a complete run.
- Let $\mathcal{N}$ be a halting GNN with with Agg-Comb layer L.
  - For every input graph G there exists k such that $\mathcal{N}(G) = H_k$.
  - If $\mathcal{N}_k$ is the (non-recursive) k-layer GNN whose layers compute L, then $\mathcal{N}_k(G) = H_k$
- Caviats:
  - Halting function does not indicate when computation halts locally.
  - Output layer of the halting model is the first layer where all nodes indicate halting.
  - In general, a node may alternate between being ready or not to halt.

Halting model

- A recurrent GNN is halting if every finite input graph has a complete run.
- Let $\mathcal{N}$ be a halting GNN with with Agg-Comb layer L.
  - For every input graph G there exists k such that $\mathcal{N}(G) = H_k$.
  - If $\mathcal{N}_k$ is the (non-recursive) k-layer GNN whose layers compute L, then $\mathcal{N}_k(G) = H_k$
- Caviats:
  - Halting function does not indicate when computation halts locally.
  - Output layer of the halting model is the first layer where all nodes indicate halting.
  - In general, a node may alternate between being ready or not to halt.
  - Recurrent GNNs may define partial node-feature maps. All GNNs are not Halting!
  - Solution: Only consider GNNs that are halting.

Example: Reachability in a given bound

- The halting mechanism allows simulation of various counters.
  (HALT when counter reaches 0).
- Consider the following node property for pointed graphs (G, n):
  Does there exists path from n to a node satisfying p in at most $\deg(n)$ many steps.

Example: Reachability in a given bound

- The halting mechanism allows simulation of various counters.
  (HALT when counter reaches 0).
- Consider the following node property for pointed graphs (G, n):
  Does there exists path from n to a node satisfying p in at most $\deg(n)$ many steps.
- It is easy to design a halting GNN that
  - sets the value of one feature to $\deg(n)$,
  - in each successive layer, decreases this counter by one,
  - HALTS when this feature reaches 0,
  - in a second feature, the GNN computes the truth value of a GML formula $\bigvee_{j \leq i} \Diamond^j p$, for increasing values of i.

Example: Reachability in a given bound

- The halting mechanism allows simulation of various counters.
  (HALT when counter reaches 0).
- Consider the following node property for pointed graphs (G, n):
  Does there exists path from n to a node satisfying p in at most $\deg(n)$ many steps.
- It is easy to design a halting GNN that
  - sets the value of one feature to $\deg(n)$,
  - in each successive layer, decreases this counter by one,
  - HALTS when this feature reaches 0,
  - in a second feature, the GNN computes the truth value of a GML formula $\bigvee_{j \leq i} \Diamond^j p$, for increasing values of i.
- The implementation of L is the same as in the accepting state model.

Example: Reachability in a given bound

- The halting mechanism allows simulation of various counters.
  (HALT when counter reaches 0).
- Consider the following node property for pointed graphs (G, n):
  Does there exists path from n to a node satisfying p in at most $\deg(n)$ many steps.
- It is easy to design a halting GNN that
  - sets the value of one feature to $\deg(n)$,
  - in each successive layer, decreases this counter by one,
  - HALTS when this feature reaches 0,
  - in a second feature, the GNN computes the truth value of a GML formula $\bigvee_{j \leq i} \Diamond^j p$, for increasing values of i.
- The implementation of L is the same as in the accepting state model.
- When this GNN halts, each node n has computed the truth value of the formula

$$\bigvee_{j \leq \deg n} \Diamond^j p.$$

Example: General Reachability

- It is not immediately clear how to design a halting GNN for checking whether there exist a path to a node satisfying p.

Example: General Reachability

- It is not immediately clear how to design a halting GNN for checking whether there exist a path to a node satisfying p.
- Easy part: Compute the truth value of $\bigvee_{j \leq i} \Diamond^j p$ for increasing values of i, and HALT and return 1 if p is reached. As in the accepting state model.
- Tricky part: How does the GNN know to HALT, if there is no p reachable?

Example: General Reachability

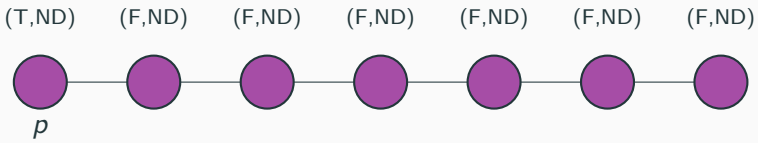
- Easy part: Compute the truth value of $\bigvee_{j \leq i} \Diamond^j p$ for increasing values of i, and HALT and return 1 if p is reached. As in the accepting state model.
- Tricky part: How does the GNN know to HALT, if there is no p reachable?
- The GNN will use two features
  - one to compute the truth value of $\bigvee_{j \leq i} \Diamond^j p$ for increasing values of i, and
  - another to compute the truth value of $\bigvee_{j \leq i-1} \Diamond^j p$ for increasing values of i.
  - HALT will output 1 iff the value of these features are equal (and $i \geq 1$).

Example: General Reachability

- Easy part: Compute the truth value of $\bigvee_{j \leq i} \Diamond^j p$ for increasing values of i, and HALT and return 1 if p is reached. As in the accepting state model.
- Tricky part: How does the GNN know to HALT, if there is no p reachable?
- The GNN will use two features
  - one to compute the truth value of $\bigvee_{j \leq i} \Diamond^j p$ for increasing values of i, and
  - another to compute the truth value of $\bigvee_{j \leq i-1} \Diamond^j p$ for increasing values of i.
  - HALT will output 1 iff the value of these features are equal (and $i \geq 1$).
- After halting, each n has computed the value of $\bigvee_{j \leq i} \Diamond^j p$, for some value of i.
- Why does this correctly indicate whether there is a path from n to a p node?

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

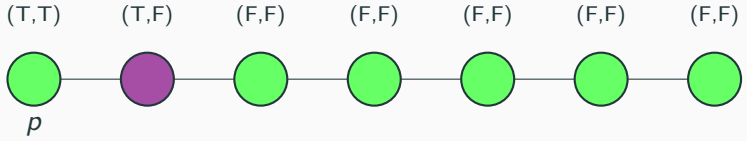


$\varphi_0 = p$

φ_{-1} undefined

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

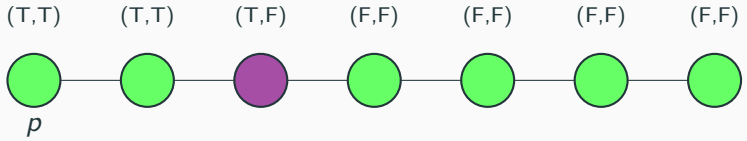


$$\varphi_1 = p \vee \Diamond p$$

$$\varphi_0 = p$$

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

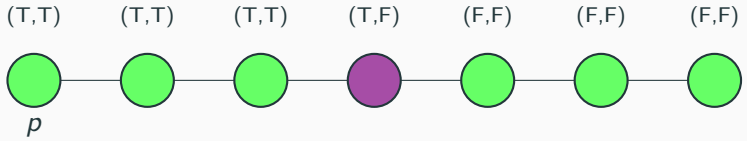


$$\varphi_2 = p \vee \Diamond(p \vee \Diamond p)$$

$$\varphi_1 = p \vee \Diamond p$$

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

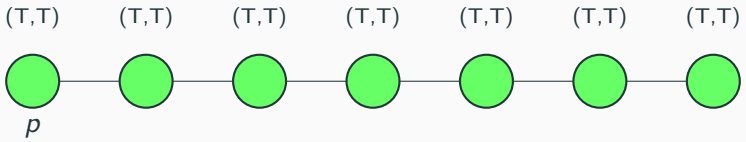


$$\varphi_3 = p \vee \Diamond \varphi_2$$

$$\varphi_2 = p \vee \Diamond \varphi_1$$

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.



$$\varphi_7 = p \vee \Diamond \varphi_6$$

$$\varphi_6 = p \vee \Diamond \varphi_5$$

Every node agree on the truth value of φ_7 and φ_6

Hence the GNN halts

Each node outputs 1, the truth value of φ_6

Proposition 9

Let (G, n) and (H, m) be two pointed X-labeled graphs. Let χ_∞^G denote the stable colouring of colour refinement of G.

1. *For every halting GNN $\mathcal{N}$ with node features g, we have*

$$\chi_\infty^G(n) = \chi_\infty^H(m) \text{ and } \mathrm{set}(\mathrm{hist}_{\chi_\infty^G}) = \mathrm{set}(\mathrm{hist}_{\chi_\infty^H}) \implies g_{h_G}^G(n) = g_{h_H}^H(m),$$

   where h_H and h_G are the halting layers of $\mathcal{N}(G)$ and $\mathcal{N}(H)$. Moreover, $h_G = h_H$.

2. *Conversely, if colour refinement distinguishes the two pointed graphs, that is,*

$$\chi_\infty^G(n) \neq \chi_\infty^H(m),$$

   then there exists a halting GNN such that

$$g_k^G(n) \neq g_k^H(m).$$

Proof of Proposition 9: First direction

Proof.

- Assume $\chi_\infty^G(n) = \chi_\infty^H(m)$. In particular $\chi_l^G(n) = \chi_l^H(m)$, for every $l \in \mathbb{N}$.

Proof of Proposition 9: First direction

Proof.

- Assume $\chi^G_\infty(n) = \chi^H_\infty(m)$. In particular $\chi^G_l(n) = \chi^H_l(m)$, for every $l \in \mathbb{N}$.
- For any fixed number of layers l, the Halting GNN computes graph transformations identical to an l-layer GNN.
  - $g^G_t(n) = g^H_t(m)$ follows from Proposition 1 (this proposition for non-rec GNNs).

Proof of Proposition 9: First direction

Proof.

- Assume $\chi^G_\infty(n) = \chi^H_\infty(m)$. In particular $\chi^G_l(n) = \chi^H_l(m)$, for every $l \in \mathbb{N}$.
- For any fixed number of layers l, the Halting GNN computes graph transformations identical to an l-layer GNN.
  - $g^G_t(n) = g^H_t(m)$ follows from Proposition 1 (this proposition for non-rec GNNs).
- Since $\mathrm{set}(\mathrm{hist}_{\chi^G_\infty}) = \mathrm{set}(\mathrm{hist}_{\chi^H_\infty})$, in particular $\mathrm{set}(\mathrm{hist}_{\chi^G_l}) = \mathrm{set}(\mathrm{hist}_{\chi^H_l})$ for every l. Hence H and G halt at the same layer.

□

Proof of Proposition 9: Second direction.

- For any two pointed graphs $\chi^G_\infty(n) \neq \chi^H_\infty(m)$ iff $\chi^G_l(n) \neq \chi^H_l(m)$, for some $l \in \mathbb{N}$.

Proof of Proposition 9: Second direction.

- For any two pointed graphs $\chi^G_\infty(n) \neq \chi^H_\infty(m)$ iff $\chi^G_l(n) \neq \chi^H_l(m)$, for some $l \in \mathbb{N}$.
- We will show next that the distinguishing power of halting recurrent GNNs is at least that of non-rec GNNs.
- The result then follows from Proposition 1 (this proposition for non-recurrent GNNs).

Distinguishing power: fixed-depth versus recurrent

Proposition 10

- *If two finite pointed graphs can be distinguished by a fixed-depth GNN they can be distinguished by a halting recurrent GNN.*
- *If two connected finite pointed graphs can be distinguished by a halting recurrent GNN then they can be distinguished by a fixed-depth GNN.*

- Thus recurrence does not add distinguishing power for connected graphs.
- This is a non-uniform statement.
- The fixed-depth GNN obtained from a recurrent GNN may depend on the two pointed graphs being separated.

Proof: fixed-depth implies recurrent

- Suppose a fixed-depth GNN $\mathcal{N}$ with ℓ layers distinguishes (G, n) and (H, m).
- We have proven that $\mathcal{N}$ can be assumed to be homogeneous.
- A recurrent GNN can simulate this by storing a counter from 0 to ℓ.
- It applies the unique layer of $\mathcal{N}$ at each step.
- It halts after exactly ℓ rounds.
- Therefore every fixed-depth distinction can also be made by a recurrent GNN.

Proof: recurrent implies fixed-depth

Proof by contraposition:

- Two connected graphs cannot be distinguished by a fixed-depth GNN
  $\Rightarrow$ the graphs cannot be distinguished by a halting recurrent GNN.

1. Let (G, n) and (H, m) be two connected finite pointed graphs that cannot be distinguished by any fixed-depth GNN.
2. By Proposition 1, (G, n) and (H, m) cannot be separated by any finite-round CR, i.e., $\chi_t^G(n) = \chi_t^H(m)$, for each $t \in \mathbb{N}$. Hence $\chi_\infty^G(n) = \chi_\infty^H(m)$.
3. For connected graphs this implies that $\mathrm{set}(\mathrm{hist}_{\chi_\infty^G}) = \mathrm{set}(\mathrm{hist}_{\chi_\infty^H})$.
4. Proposition 9 (claim 1) then implies that no recurrent GNN can separate (G, n) and (H, m).

Logic...strikes back

- With respect to connected graphs, the distinguishing power of halting RGNNs coincide with fixed-layer GNNs and GML.
- Uniform expressivity of halting RGNNs goes past GML (e.g., reachability).
  - Is there a logical counterpart for halting RGNNs?

Part 5 (c): Halting recurrent GNNs and modal μ-calculus

From GML to graded modal μ-calculus

- GML describes bounded-depth local properties.
- A formula of modal depth r can only look r steps away.
- To express iterative properties such as reachability, we proceed in two ways:
  - add infinitary disjunctions, or
  - add fixpoint operators.
- This gives us ω-GML and graded modal μ-calculus, denoted μGML, respectively.

Syntax of μGML

Fix a finite set of propositions P and a set of variables Var.

- Formulae of graded modal μ-calculus are generated by

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Diamond_{\geq k} \varphi \mid \mu X.\varphi \mid \nu X.\varphi,$$

  where $p \in P$, $X \in \mathrm{Var}$, and $k \geq 1$.

- The modality $\Diamond_{\geq k} \varphi$ means:

$$\text{at least } k \text{ neighbours satisfy } \varphi.$$

- The formula $\mu X.\varphi$ denotes a least fixpoint.
- The formula $\nu X.\varphi$ denotes a greatest fixpoint ($\nu X.\varphi \equiv \neg \mu X.\neg \varphi$).
- Variables must occur positively in fixpoint bodies, so that the induced operator is monotone.

Intuition for fixed points

- We write $\varphi[\psi/X]$ for the formula obtained from φ by substituting X with ψ.
- Let $\varphi := \Diamond X \vee p$,
- Define $\varphi_0 := \varphi[\bot/X] = \Diamond\bot \vee p$, which is equivalent to p.
- Define $\varphi_1 := \varphi[\varphi_0/X] = \Diamond(\Diamond\bot \vee p) \vee p$, which is equivalent to $\Diamond p \vee p$.
- Define $\varphi_2 := \varphi[\varphi_1/X] = \Diamond(\Diamond(\Diamond\bot \vee p) \vee p) \vee p$, is equivalent to $\Diamond\Diamond p \vee \Diamond p \vee p$.

Intuition for fixed points

- We write $\varphi[\psi/X]$ for the formula obtained from φ by substituting X with ψ.
- Let $\varphi := \Diamond X \vee p$,
- Define $\varphi_0 := \varphi[\bot/X] = \Diamond\bot \vee p$, which is equivalent to p.
- Define $\varphi_1 := \varphi[\varphi_0/X] = \Diamond(\Diamond\bot \vee p) \vee p$, which is equivalent to $\Diamond p \vee p$.
- Define $\varphi_2 := \varphi[\varphi_1/X] = \Diamond(\Diamond(\Diamond\bot \vee p) \vee p) \vee p$, is equivalent to $\Diamond\Diamond p \vee \Diamond p \vee p$.
- Define $\varphi_{i+1} := \varphi[\varphi_i/X]$ is equivalent to $\bigvee_{j \leq i} \Diamond^i p$.
- Hence φ_i is ϕ unravelled recursively within itself i times.

Intuition for fixed points

- We write $\varphi[\psi/X]$ for the formula obtained from φ by substituting X with ψ.
- Let $\varphi := \Diamond X \vee p$,
- Define $\varphi_0 := \varphi[\bot/X] = \Diamond\bot \vee p$, which is equivalent to p.
- Define $\varphi_1 := \varphi[\varphi_0/X] = \Diamond(\Diamond\bot \vee p) \vee p$, which is equivalent to $\Diamond p \vee p$.
- Define $\varphi_2 := \varphi[\varphi_1/X] = \Diamond(\Diamond(\Diamond\bot \vee p) \vee p) \vee p$, is equivalent to $\Diamond\Diamond p \vee \Diamond p \vee p$.
- Define $\varphi_{i+1} := \varphi[\varphi_i/X]$ is equivalent to $\bigvee_{j\leq i} \Diamond^j p$.
- Hence φ_i is ϕ unravelled recursively within itself i times.
- $(G, m) \models \mu X.\varphi(X)$ iff $(G, m) \models \varphi_i$, for some i.
- Intuition: $\mu X.\varphi(X)$ is true if the recursive procedure defined by φ holds after finite number of recursion.

Semantics: environments

Let $G = (N, E, g)$ be a $\mathcal{P}(P)$-labeled graph.

- To interpret variables, we use an environment

$$\eta : \mathsf{Var} \to \mathcal{P}(N).$$

- Thus $\eta(X)$ is the set of nodes currently assigned to the variable X.
- For variables,

$$G, n \models_\eta X \quad \iff \quad n \in \eta(X).$$

- Atomic propositions and graded modalities are interpreted as before:

$$G, n \models_\eta \Diamond_{\geq k} \varphi$$

iff

$$|\{m \in E(n) \mid G, m \models_\eta \varphi\}| \geq k.$$

Least fixpoints

Consider a formula $\mu X.\varphi(X)$.

- The body $\varphi(X)$ defines a monotone operator

$$F : \mathcal{P}(N) \to \mathcal{P}(N),$$

  where

$$F(S) := \{n \in N \mid G, n \models_{\eta[X \mapsto S]} \varphi\}.$$

- The formula $\mu X.\varphi(X)$ denotes the least fixpoint of F:

$$G, n \models_\eta \mu X.\varphi \quad \iff \quad n \in \mathrm{lfp}(F).$$

- On finite graphs, the least fixpoint is obtained by iteration:

$$S_0 := \emptyset, \qquad S_{i+1} := F(S_i).$$

Example: reachability

Fix an atomic proposition p.

- The formula
$$\mathrm{Reach}_p := \mu X.\,(p \vee \Diamond X)$$
says that a node can reach a p-node.
- The least-fixpoint iteration is
$$R_0 := \emptyset,$$
$$R_{i+1} := \{n \mid p \in g(n)\} \cup \{n \mid E(n) \cap R_i \neq \emptyset\}.$$
- At the fixpoint, R_i is exactly the set of nodes from which some p-node is reachable.

μGML formulae as node classifiers

Let φ be a closed μGML formula.

- The formula φ defines a node classifier

$$C_\varphi : \mathcal{G}[\mathcal{P}(P)] \to \mathcal{G}[\mathbb{B}].$$

- For each graph $G = (N, E, g)$,

$$C_\varphi(G) = (N, E, c_G^\varphi),$$

where

$$c_G^\varphi(n) = 1 \quad \iff \quad G, n \models \varphi.$$

- Thus μGML extends GML from bounded-depth classifiers to iterative classifiers.

Simple recurrent GNNs

- We focus on simple recurrent GNNs.
- The aggregation is sum aggregation:

$$\mathrm{Agg}(M) = \sum_{x \in \mathrm{supp}(M)} M(x) \cdot x.$$

- The combination function has the form

$$\mathrm{Comb}(x, y) = f(x \mid y),$$

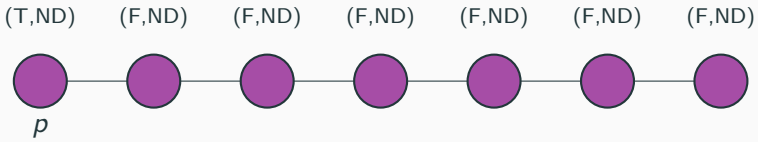
  where f is a ReLU feedforward neural network.
- The halting and output functions are simple threshold tests on coordinates.
- Thus the recurrence comes from repeatedly applying the same ordinary message-passing layer.

Evaluating μGML with GNNs

- Typical approach: a feature vector is used to recursively compute the truth values of subformulas of φ.
  - The initialisation map uses one-hot encoding for proposition symbols.
  - GNN layers are used to compute recursively truth values of composite formulae.
- Challenge: If we compute fixed points recursively, the number of subformulas is not fixed.
- Solution: Develop approximation semantics for μ-calculus.
- Recall how reachability was expressed!

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

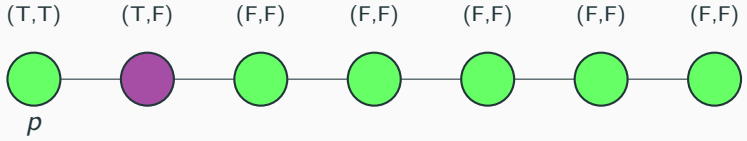


$\varphi_0 = p$

φ_{-1} undefined

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

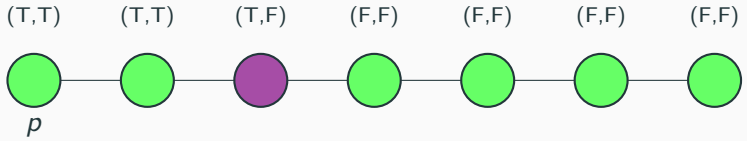


$$\varphi_1 = p \vee \Diamond p$$

$$\varphi_0 = p$$

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

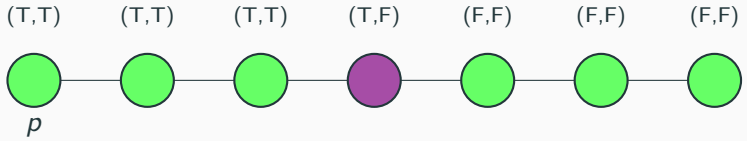


$$\varphi_2 = p \vee \Diamond(p \vee \Diamond p)$$

$$\varphi_1 = p \vee \Diamond p$$

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.

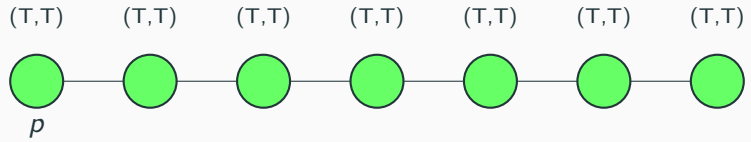


$$\varphi_3 = p \vee \Diamond \varphi_2$$

$$\varphi_2 = p \vee \Diamond \varphi_1$$

Illustration of the GNN computation

- Computation of reachability to p using the method of the previous page.
- After i layers, the figure shows the truth value of formulas $(\bigvee_{j \leq i} \Diamond^j p, \bigvee_{j \leq i-1} \Diamond^j p)$
- A node is green, if that node is ready to halt.



$$\varphi_7 = p \vee \Diamond \varphi_6$$

$$\varphi_6 = p \vee \Diamond \varphi_5$$

Every node agree on the truth value of φ_7 and φ_6

Hence the GNN halts

Each node outputs 1, the truth value of φ_6

Approximate semantics of μ-calculus

- Recall: $G, m \models \mu X.\varphi(X)$ iff $(G, m) \models \varphi_i$, for some i, where φ_i is the ith unravelling of $\varphi(X)$.
- Monotonicity: If $G, m \models \varphi_i$ then $(G, m) \models \varphi_{i+1}$.
- Idea: Each node computes the truth values of $(\varphi_{i+1}, \varphi_i)$.
  - When all nodes agree on those values, their truth value is $\mu X.\varphi(X)$.
- Complex formula $G, m \models \mu X \mu Y.\varphi(X, Y)$ uses more complex counters $(\varphi_{(i,j)}, \varphi_{(i,j+1)})$.
  - Here $(3, 2)$ that we are in the third unravelling of X inside of which Y is unravelled twice.

Uniform expressivity of recurrent GNNs

- Node properties are invariant under (total) surjective g-bisimulations.
  If $(G, n) \in P$ and $f : G \to G'$ is a g-bisimulation, total and surjective function, then $(G', f(n)) \in P$
  - If two points are g-bisimilar, after any finite iterations, they are labelled with the same feature vector.
  - Totality and surjectivity imply that the two computations halt at the same iteration.
- On connected graphs, g-bisimilarity and total surjective g-bimilarity coincide.
- Every μGML-definable node property can be decided with a recurrent GNN.

Part 6: Further directions

- What have we covered:
  - non-recurrent and two models of recurrent GNNs;
  - connection of graded model logic and extensions thereof.
- What further directions could be considered
  - Going beyond boolean logic by considering real-valued logics.
  - More fine-grained analysis relating impact of specific choices of aggregation functions, non-linear activation functions to expressive power.
  - Study impact of computational power of Comb and Agg. For example, restrict those functions to come from some function complexity class (non-recurrent vs recurrent arithmetic circuits).
  - Taking training process into account. If it can be expressed, can it also be learned? That is, can one construct training data when used to train GNNs expresses the desired property?

Lecture 4: Real computation and arithmetic circuits

Our focus so far

- Indistinguishability of GNNs:
  - When two input graphs can be separated by a GNN?
  - Input graphs can have numerical features in $\mathbb{R}^p$ or abstract features.
- Expressivity characterisations:
  - What graph/node properties can be expressed with GNNs?
  - Input: A graph with features in some finite set. A graph embedding is used to give features in $\mathbb{R}^p$.
  - Output: A classification of (pointed) graphs to Boolean (or finitely many) classes.
  - Type of characterisation: $M_s(G, n) = 1$ if and only if $G, n \models \varphi_s$.

Part 7: GNNs and arithmetic circuits

- What do we cover next?
  - Consider GNNs as devices computing real labelled graph transformations.
    - Meaning, functions of type $\mathcal{G}[\mathbb{R}^{P_1}] \to \mathcal{G}[\mathbb{R}^{P_2}]$.
    - We omit the embedding part and classification part.
    - This gives a cleaner picture of the computation power of GNNs.
    - We relate their power to a mature field of study; circuit complexity theory.
  - Short introduction to arithmetic circuits and circuit complexity classes.
  - Circuit Graph Neural Networks $\approx$ arithmetic circuit families
  - Recursive Circuit Graph Neural Networks $\approx$ recursive arithmetic circuit families
- Main sources:
  - Timon Barlag, Vivian Holzapfel, Laura Strieker, Jonni Virtema and Heribert Vollmer. Graph Neural Networks and Arithmetic Circuits. NeurIPS 2024.
  - Timon Barlag, Vivian Holzapfel, Laura Strieker, Jonni Virtema and Heribert Vollmer. Recurrent Graph Neural Networks and Arithmetic Circuits. KR 2026 (in two weeks).
  - Many things below from slides by Laura and Vivian.

What GNNs compute?

- Recall that a GNN-layer updates feature vectors via

$$g'(n) \coloneqq \mathrm{Comb}\Big(g(n), \mathrm{Agg}\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big).$$

  where Comb and Agg are simple (often piecewise linear) functions.
- We stipulate that Comb and Agg can be written using $+$, $\times$, and some activation functions σ_i.
- Given graph (G, v) and an l-layer GNN, we can write an expression using $+$, $\times$, σ that computes the final feature vector of v after l-layers.
- In this level of abstraction, we can relate GNNs with arithmetic circuit complexity classes! If we stipulate that Comb and Agg come from circuit complexity classes!

What GNNs compute?

- Recall that a GNN-layer updates feature vectors via

$$g'(n) \coloneqq \mathrm{Comb}\Big(g(n), \mathrm{Agg}\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big).$$

  where Comb and Agg are simple (often piecewise linear) functions.
- We stipulate that Comb and Agg can be written using $+$, $\times$, and some activation functions σ_i.
- Given graph (G, v) and an l-layer GNN, we can write an expression using $+$, $\times$, σ that computes the final feature vector of v after l-layers.
- In this level of abstraction, we can relate GNNs with arithmetic circuit complexity classes! If we stipulate that Comb and Agg come from circuit complexity classes!
- We simplify presentation and write feature vectors in $\mathbb{R}$ instead of $\mathbb{R}^*$.

Gates make a circuit

- A gate: An entity that takes a finite sequence of input values returns an output:
  - Fan-in: How many inputs the gate takes.
  - Sum gate: $f_+(x, y) = x + y$ (fan-in 2), $f_+(x, y, y) = x + y + z$ (fan-in 3).
  - Product gate: $f(x, y) = x \times y$ (fan-in 2), $f(x_1, \ldots, x_n) = x_1 \times \cdots \times x_n$ (fan-in n).
  - In general, any n-ary function can be made a gate.

Input gates and output gates

- Put together a bunch of gates and you have a circuit.
- More gates:
  - Constant gate: 7 (fan-in 0).
  - Input gate: x (fan-in 0, labelled with a variable).
  - Output gate: Fan-in 1, receives the output of the circuit.

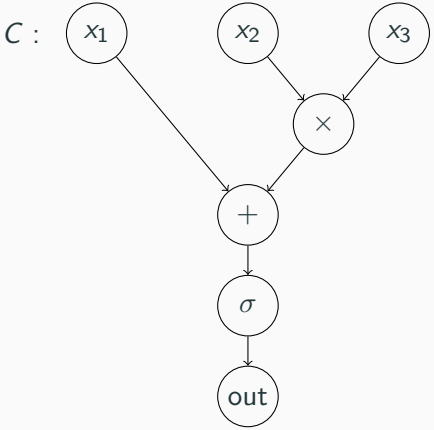
Input gates and output gates

- Put together a bunch of gates and you have a circuit.
- More gates:
  - Constant gate: 7 (fan-in 0).
  - Input gate: x (fan-in 0, labelled with a variable).
  - Output gate: Fan-in 1, receives the output of the circuit.
- A circuit computes a function:
  - Input is placed to the input gates.
  - Output is read from the output gate.

Arithmetic Circuits

Definition 11

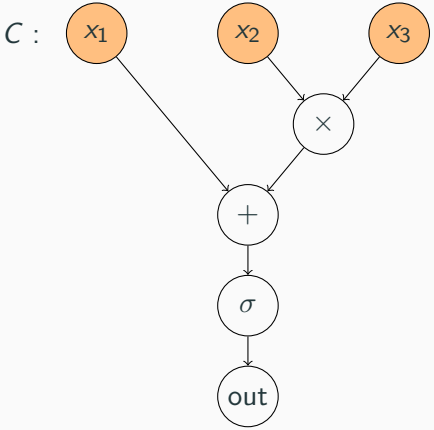
Directed acyclic graph with gates that perform arithmetic operations



Arithmetic Circuits

Definition 11

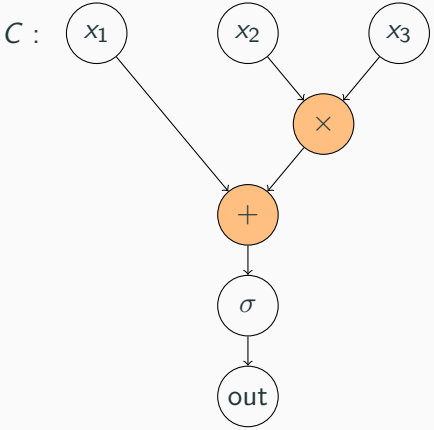
Directed acyclic graph with gates that perform arithmetic operations



Arithmetic Circuits

Definition 11

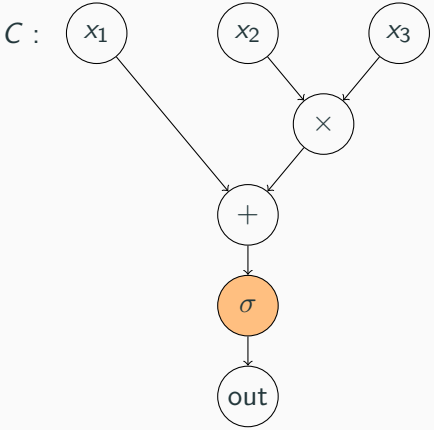
Directed acyclic graph with gates that perform arithmetic operations



Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations

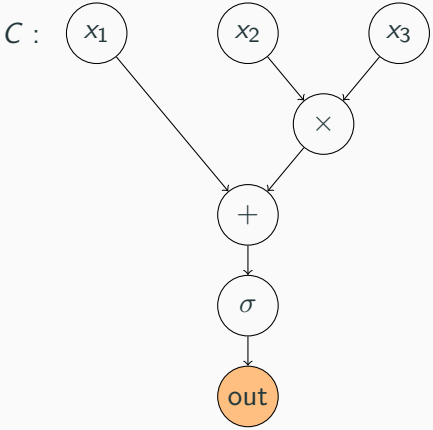


- additional function gates for activation functions

Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations

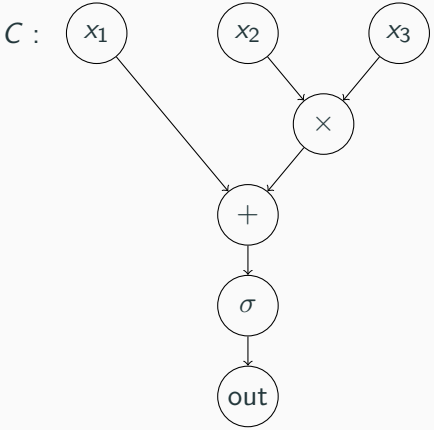


- $f_C(x_1, x_2, x_3) = \sigma(x_1 + x_2 x_3)$

Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations

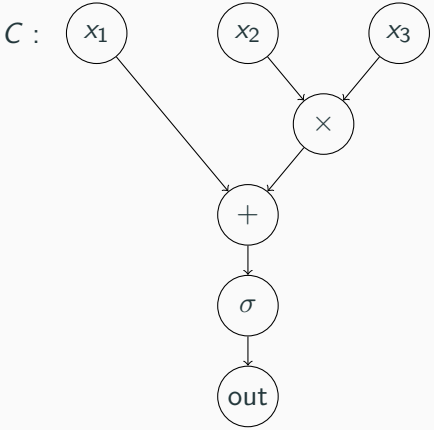


- size: number of gates
- depth: length of longest path from input to output

Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations

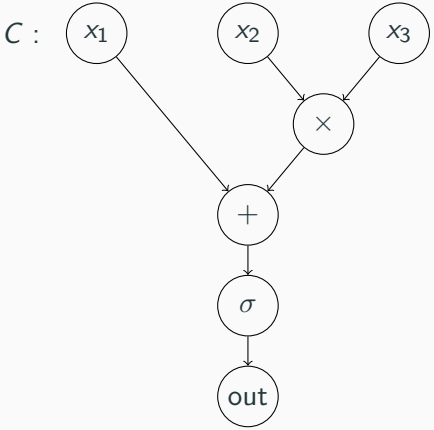


- size: number of gates
- depth: length of longest path from input to output
- A circuit has fixed number of input gates and computes $f : \mathbb{R}^p \to \mathbb{R}$.

Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations

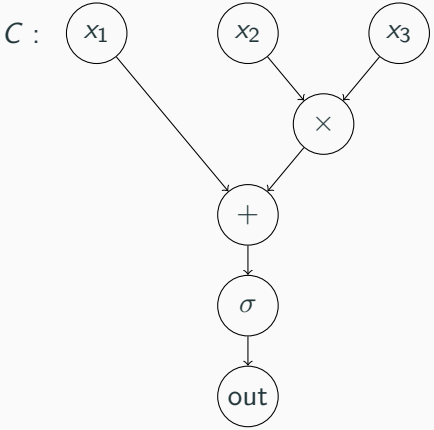


- size: number of gates
- depth: length of longest path from input to output
- A circuit has fixed number of input gates and computes $f : \mathbb{R}^p \to \mathbb{R}$.
- A circuit family $\{C_i\}_{i \in \mathbb{N}}$ has one circuit for each input length, and computes a function $f : \mathbb{R}^* \to \mathbb{R}$.

$\mathbb{R}^k$- Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations over $\mathbb{R}^k$.

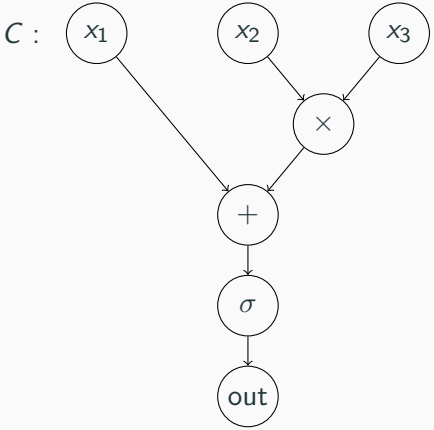


- size: number of gates
- depth: length of longest path from input to output
- A circuit has fixed number of input gates and computes $f : \mathbb{R}^p \to \mathbb{R}$.
- A circuit family $\{C_i\}_{i \in \mathbb{N}}$ has one circuit for each input length, and computes a function $f : \mathbb{R}^* \to \mathbb{R}$.
- Circuit function complexity classes obtained by restriction the size of C_i.

Arithmetic Circuits

Definition 11

Directed acyclic graph with gates that perform arithmetic operations over $\mathbb{R}^k$.



- size: number of gates
- depth: length of longest path from input to output
- A circuit family $\{C_i\}_{i\in\mathbb{N}}$ has one circuit for each input length, and computes a function $f : \mathbb{R}^* \to \mathbb{R}$.
- Circuit function complexity classes obtained by restriction the size of C_i.
  - $\mathrm{FAC}^0_{\mathbb{R}^k}$: constant depth/polynomial size
  - $\mathrm{FAC}^0_{\mathbb{R}^k}[\mathcal{A}]$: additional function gates

Boolean Circuit complexity classes

- A Boolean circuit family $\{C_i\}_{i \in \mathbb{N}}$ computes a function $f : \mathbb{B}^* \to \mathbb{B}*$ such that

$$f(\vec{x}) \coloneqq C_{|\vec{x}|}(\vec{x}).$$

- Boolean circuits: strong connections to complexity theory
  - $+$ is an or-gate, $\times$ is an and-gate.
  - (uniform)-AC^0 is a well understood tractable complexity class.
    Problems that can be decided with constant depth polynomial size circuit families.
    Cannot express parity or majority.

Arithmetic Circuit complexity classes

- An arithmetic circuit family $\{C_i\}_{i \in \mathbb{N}}$ computes a function $f : \mathbb{R}^* \to \mathbb{R}^*$ such that

$$f(\vec{x}) := C_{|\vec{x}|}(\vec{x}).$$

- Arithmetic circuits: function complexity classes
  - Characterise: Which functions $f : \mathbb{R}^* \to \mathbb{R}^*$ can be computed with circuit families.
  - $\mathrm{FAC}^0_{\mathbb{R}}$: functions by circuit families of constant depth and polynomial size.
  - Less studied than Boolean circuits, but many results known:
    - a polynomial of degree 2^{2^n} require a circuit of size 2^n.
    - determinant of an $n \times n$ matrix: a circuit of polynomial size and $\mathcal{O}(\log^2(n))$ depth.

Arithmetic Circuit complexity classes

- An arithmetic circuit family $\{C_i\}_{i \in \mathbb{N}}$ computes a function $f : \mathbb{R}^* \to \mathbb{R}^*$ such that

$$f(\vec{x}) := C_{|\vec{x}|}(\vec{x}).$$

- Arithmetic circuits: function complexity classes
  - Characterise: Which functions $f : \mathbb{R}^* \to \mathbb{R}^*$ can be computed with circuit families.
  - $\mathrm{FAC}^0_{\mathbb{R}}$: functions by circuit families of constant depth and polynomial size.
  - Less studied than Boolean circuits, but many results known:
    - a polynomial of degree 2^{2^n} require a circuit of size 2^n.
    - determinant of an $n \times n$ matrix: a circuit of polynomial size and $\mathcal{O}(\log^2(n))$ depth.
- Question: Can we find precise correspondence between expressivity of GNNs and circuit complexity classes?

Examples of circuit families

- The sum aggregation $\sum_{x \in E(n)} g(x)$ is a very simple circuit family of depth 1.
- Given a specification of aggregation and combination function in terms of $+$, x, and activation functions σ_i, it is easy to define the corresponding circuit class with gates for $+$, x, and σ_i.
- Most common aggregation and combination function used in practice can be computed with constant depth polynomial size circuit families.

Circuits, circuit families, and GNNs

- A feedforward neural network is an arithmetic circuit of very regular form!
- Gates for scalar multiplication, addition, and for the activation function.

Circuits, circuit families, and GNNs

- A feedforward neural network is an arithmetic circuit of very regular form!
- Gates for scalar multiplication, addition, and for the activation function.
- A GNN-layer is a very regular circuit family!
- A node of degree i uses the circuit C_{i+1}.

Circuits, circuit families, and GNNs

- A feedforward neural network is an arithmetic circuit of very regular form!
- Gates for scalar multiplication, addition, and for the activation function.
- A GNN-layer is a very regular circuit family!
- A node of degree i uses the circuit C_{i+1}.
- A graph transformation that a GNN computes is a concatanation of it's layers $\Rightarrow$ a GNN is just a special kind of circuit family.

Circuits, circuit families, and GNNs

- A feedforward neural network is an arithmetic circuit of very regular form!
- Gates for scalar multiplication, addition, and for the activation function.
- A GNN-layer is a very regular circuit family!
- A node of degree i uses the circuit C_{i+1}.
- A graph transformation that a GNN computes is a concatanation of it's layers $\Rightarrow$ a GNN is just a special kind of circuit family.
- Why is this interesting?
  - Relating GNNs with artihmetic circuit complexity classes, gives a mature way to understand capabilites of GNNs.

Arithmetic circuits and GNNs

1st Goal: Show that GNNs can be simulated by circuit families

Let N be a GNN. Then there exists a circuit family $\mathcal{C}$, such that for all labeled graphs the circuit family computes the same feature vectors as N.

2nd Goal: Show that circuit families can be simulated by GNNs

Let $\mathcal{C}$ be a circuit family. Show there exists a GNN N that has the output of the circuit family among its feature vectors.

Goal 1: Simulate GNNs with circuits

1st Goal: Show that GNNs can be simulated by circuit families

Let N be a GNN. Then there exists a circuit family $\mathcal{C}$, such that for all labeled graphs the circuit family computes the same feature vectors as N.

- Input graphs G for GNNs are encoded as as strings of real numbers by listing its adjacency matrix and feature vectors $\langle G \rangle$.
- Circuit family simulates the GNN such that $N(G) = G'$ iff $\mathcal{C}(\langle G \rangle) = \langle G' \rangle$.

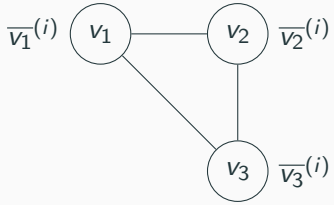
Goal 2: Simulate circuits with GNNs

2nd Goal: Show that circuit families can be simulated by GNNs

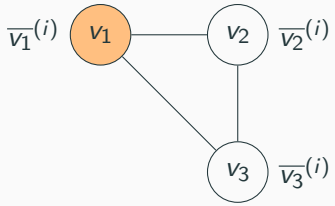
Let $\mathcal{C}$ be a circuit family. Show there exists a GNN N that has the output of the circuit family among its feature vectors.

- GNN receives the graph structure and input of the circuit as the input graph

Example: AC-GNN

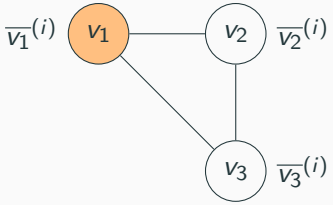


Example: AC-GNN



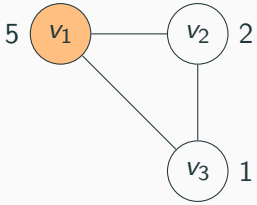
$$\overline{v_1}^{(i+1)} = \mathsf{COM}^{(i)}(\overline{v_1}^{(i)}, \mathsf{AGG}^{(i)}(\overline{v_2}^{(i)}, \overline{v_3}^{(i)}))$$

Example: AC-GNN



$$\overline{v_1}^{(i+1)} = \mathrm{sigmoid}(\mathrm{avg}(\overline{v_1}^{(i)}, (\overline{v_2}^{(i)} + \overline{v_3}^{(i)})))$$

Example: AC-GNN

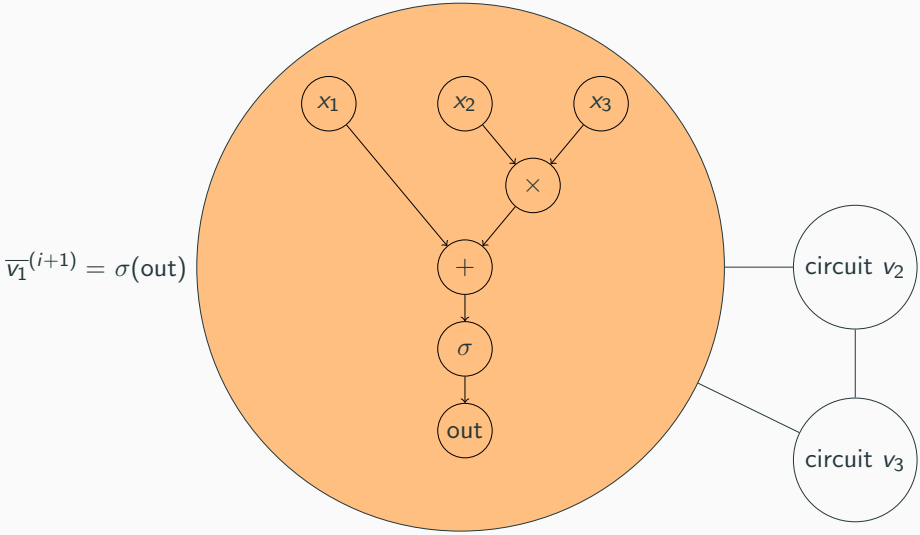


$$\overline{v_1}^{(i+1)} = \mathrm{sigmoid}(\mathrm{avg}(5, (2+1)))$$

Circuit GNNs: An abstraction of AC-GNNs

- Goal: Abstract concrete aggregation-combination functions to ones that come from a circuit function class.
- Why? Facilitate clean and general statements for capabilities for range of GNNs.

Example: Circuit-GNN



Building blocks of Circuit-GNNs

- Goal: Define GNNs in a level of abstraction that makes relating to circuits easier.
- Aggregate-combine layer will be defined using circuits.

Building blocks of Circuit-GNNs

- Goal: Define GNNs in a level of abstraction that makes relating to circuits easier.
- Aggregate-combine layer will be defined using circuits.
- Agg-Comb basis of a circuit GNN:
  - Fix $\mathcal{S}$, a set of functions $\mathbb{R}^* \to \mathbb{R}$ (aggregate-combine layer).
  - Fix $\mathcal{A}$, a set of activation functions $\mathbb{R} \to \mathbb{R}$ (activation function of a layer).

Building blocks of Circuit-GNNs

- Goal: Define GNNs in a level of abstraction that makes relating to circuits easier.
- Aggregate-combine layer will be defined using circuits.
- Agg-Comb basis of a circuit GNN:
  - Fix $\mathcal{S}$, a set of functions $\mathbb{R}^* \to \mathbb{R}$ (aggregate-combine layer).
  - Fix $\mathcal{A}$, a set of activation functions $\mathbb{R} \to \mathbb{R}$ (activation function of a layer).
- $(\mathcal{S}, \mathcal{A})$-GNN of depth d is a function $[d] \to \mathcal{S} \times \mathcal{A}$.

Building blocks of Circuit-GNNs

- Goal: Define GNNs in a level of abstraction that makes relating to circuits easier.
- Aggregate-combine layer will be defined using circuits.
- Agg-Comb basis of a circuit GNN:
  - Fix $\mathcal{S}$, a set of functions $\mathbb{R}^* \to \mathbb{R}$ (aggregate-combine layer).
  - Fix $\mathcal{A}$, a set of activation functions $\mathbb{R} \to \mathbb{R}$ (activation function of a layer).
- $(\mathcal{S}, \mathcal{A})$-GNN of depth d is a function $[d] \to \mathcal{S} \times \mathcal{A}$.
  - Standard $\mathrm{Comb}^i\Big(g(n), \mathrm{Agg}^i\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big)$ is replaced with applying $(f_i, \sigma_i) \in \mathcal{S} \times \mathcal{A}$ given by the function d.

Building blocks of Circuit-GNNs

- Goal: Define GNNs in a level of abstraction that makes relating to circuits easier.
- Aggregate-combine layer will be defined using circuits.
- Agg-Comb basis of a circuit GNN:
  - Fix $\mathcal{S}$, a set of functions $\mathbb{R}^* \to \mathbb{R}$ (aggregate-combine layer).
  - Fix $\mathcal{A}$, a set of activation functions $\mathbb{R} \to \mathbb{R}$ (activation function of a layer).
- $(\mathcal{S}, \mathcal{A})$-GNN of depth d is a function $[d] \to \mathcal{S} \times \mathcal{A}$.
  - Standard $\mathrm{Comb}^i\Big(g(n), \mathrm{Agg}^i\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big)$ is replaced with applying $(f_i, \sigma_i) \in \mathcal{S} \times \mathcal{A}$ given by the function d.
- if all functions in $\mathcal{S}$ belong to circuit function class $\mathfrak{F}$: $(\mathfrak{F}, \mathcal{A})$-GNN, e.g. $(\mathrm{FAC}^0_{\mathbb{R}}, \{\mathrm{id}\})$-GNN

Building blocks of Circuit-GNNs

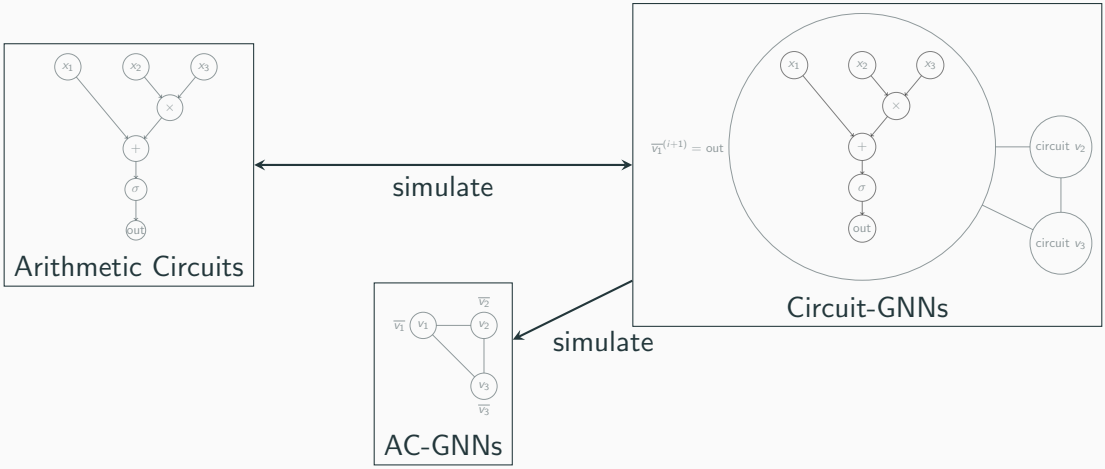
- Goal: Define GNNs in a level of abstraction that makes relating to circuits easier.
- Aggregate-combine layer will be defined using circuits.
- Agg-Comb basis of a circuit GNN:
  - Fix $\mathcal{S}$, a set of functions $\mathbb{R}^* \to \mathbb{R}$ (aggregate-combine layer).
  - Fix $\mathcal{A}$, a set of activation functions $\mathbb{R} \to \mathbb{R}$ (activation function of a layer).
- $(\mathcal{S}, \mathcal{A})$-GNN of depth d is a function $[d] \to \mathcal{S} \times \mathcal{A}$.
  - Standard $\mathrm{Comb}^i\Big(g(n), \mathrm{Agg}^i\big(\{\{g(m) \mid m \in E(n)\}\}\big)\Big)$ is replaced with applying $(f_i, \sigma_i) \in \mathcal{S} \times \mathcal{A}$ given by the function d.
- if all functions in $\mathcal{S}$ belong to circuit function class $\mathfrak{F}$: $(\mathfrak{F}, \mathcal{A})$-GNN, e.g. $(\mathrm{FAC}^0_{\mathbb{R}}, \{\mathrm{id}\})$-GNN

Intuition: a GNN with circuits in its nodes

Circuit GNNs

- Circuit GNNs are AC-GNNs where complexity of layer computation can be specified using circuit complexity theory!

Overview of Connections



AC-GNNs as C-GNNs

- For any AC-GNN with activation functions $\mathcal{A}$ and where the aggregation functions are computable by $\mathrm{FAC}^0_{\mathbb{R}}$-circuit families, there exists a $\left(\mathrm{FAC}^0_{\mathbb{R}}, \mathcal{A}\right)$-GNN with a constant number of layers which computes the same function, omitting only the functionality of the classification function.
- This holds for the common aggregation functions like the sum, product or mean.

Simulating Circuit-GNNs with Arithmetic Circuits

A function is tail symmetric, if $f(x_1, y_1, \ldots, y_n) = f(x_1, p(y_1), \ldots, p(y_n))$, for all variable permutations p.

Theorem 12

Let N be a $\left(t\mathrm{FAC}^0_{\mathbb{R}}, \mathcal{A}\right)$-GNN. Then there exists an $\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}]$-circuit family $\mathcal{C}$, such that for all labeled graphs G the circuit family computes the same feature vectors as N.

Simulating Circuit-GNNs with Arithmetic Circuits

A function is tail symmetric, if $f(x_1, y_1, \ldots, y_n) = f(x_1, p(y_1), \ldots, p(y_n))$, for all variable permutations p.

Theorem 12

Let N be a $\left(t\mathrm{FAC}^0_{\mathbb{R}}, \mathcal{A}\right)$-GNN. Then there exists an $\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}]$-circuit family $\mathcal{C}$, such that for all labeled graphs G the circuit family computes the same feature vectors as N.

- Each GNN-layer is computed by a $\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}]$-circuit family
- For any input graph G, a node $v \in V$, and layer i, the feature vector $v^{(i+1)}$ is computed via a suitable circuit from the fixed circuit families.
- Essentially $N(G)$ computation unrolled is a big $\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}]$-circuit.
- Difficulty: Construct a circuit family $\mathcal{C}$ such that $C_{|\langle G \rangle|} = \langle N(G) \rangle$ for all G. (output the graph G as adjacency matrix and corresponding feature vectors)
  - The idea is to construct the circuits so that it uses the input adjacency matrix to select suitable circuits from the layer families to compute with.
  - C_k contains all layer circuits needed for degree k graphs.

Simulating Arithmetic Circuits with Circuit-GNNs

Theorem

For every $\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}]$-circuit family $\mathcal{C}$ there exists a $\left(t\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}], \{\mathrm{id}\}\right)$-GNN N such that for all $n \in \mathbb{N}$ and $\vec{x} \in \mathbb{R}^n$ $N(G_{C_n,\vec{x}})$ has $C_n(\vec{x})$ as feature vectors.

- the graph structure of the circuit is given as the input graph of the Circuit-GNN
- Pre-process the circuit to path-length normal form.
  - All paths from an input to an output gate has the same length.
  - Every non-input and non-output gate has exactly one successor.
- simulate circuit computation layerwise and store intermediate results in features
- Difficulty: A circuit computes a gate when inputs to it are computed, but a GNN re-computes all feature values on every layer.
  - Solution: Add an additional feature value that is used to guide the GNN computation by resetting feature values so that circuit computation can be simulated.

Simulating Arithmetic Circuits with Circuit-GNNs

A circuit is in function layer form, if it is in path-length normal form, and on each depth the type of gate is uniform.

Theorem

For every $f\mathrm{FAC}^0_{\mathbb{R}}[\mathcal{A}]$-circuit family $\mathcal{C}$ there exists a $\left(t\mathrm{FAC}^0_{\mathbb{R}}, \mathcal{A} \cup \{\mathrm{id}\}\right)$-GNN N such that for all $n \in \mathbb{N}$ and $\vec{x} \in \mathbb{R}^n$ $N(G_{C_n,\vec{x}})$ has $C_n(\vec{x})$ as feature vectors.

- the graph structure of the circuit is given as the input graph of the Circuit-GNN
- Pre-process the circuit to path-length normal form.
  - All paths from an input to an output gate has the same length.
  - Every non-input and non-output gate has exactly one successor.
- simulate circuit computation layerwise and store intermediate results in features
- Difficulty: A circuit computes a gate when inputs to it are computed, but a GNN re-computes all feature values on every layer.
  - Solution: Add an additional feature value that is used to guide the GNN computation by resetting feature values so that circuit computation can be simulated.

Can we make this recursive?

Definition 13

An L layer *aggregate combine graph neural network* (AC-GNN) is a tuple $\mathcal{D} = (\{\mathsf{AGG}^{(i)}\}_{i=1}^{L}, \{\mathsf{COM}^{(i)}\}_{i=1}^{L}, \{\sigma^{(i)}\}_{i=1}^{L}, \mathsf{CLS})$, where

- $\{\mathsf{AGG}^{(i)}\}_{i=1}^{L}$: aggregate functions
- $\{\mathsf{COM}^{(i)}\}_{i=1}^{L}$: combine functions
- $\{\sigma^{(i)}\}_{i=1}^{L}$: activation functions
- CLS: $\mathbb{R} \to \{0, 1\}$: classification function

Updating vectors in every layer $1 \leq i \leq L$ as follows:

- initial feature vector of v: $\overline{v}^{(0)} = g_V(v)$
- for $i > 1$:
  - $\overline{v}^{(i)} = \sigma^{(i)}\left(\mathsf{COM}^{(i)}\left(\overline{v}^{(i-1)}, \overline{y}\right)\right)$, where $\overline{y} = \mathsf{AGG}^{(i)}\left(\{\{\overline{u}^{(i-1)} \mid u \in N_G(v)\}\}\right)$

Definition 14

A period length d *recurrent aggregate combine graph neural network* (AC-GNN) is a tuple $\mathcal{D} = (\{\mathrm{AGG}^{(i)}\}_{i=1}^{d}, \{\mathrm{COM}^{(i)}\}_{i=1}^{d}, \{\sigma^{(i)}\}_{i=1}^{d}, \mathit{HLT}, \mathrm{CLS})$, where

- $\{\mathrm{AGG}^{(i)}\}_{i=1}^{d}$: aggregate functions
- $\{\mathrm{COM}^{(i)}\}_{i=1}^{d}$: combine functions
- $\{\sigma^{(i)}\}_{i=1}^{d}$: activation functions
- $\mathrm{HLT}\colon \mathbb{N}_{>0} \times \mathbb{R}^{*} \to \{0,1\}$
- $\mathrm{CLS}\colon \mathbb{R} \to \{0,1\}$: classification function

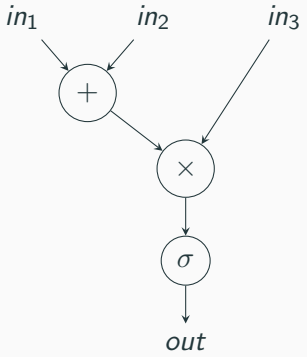
Updating vectors in every layer $1 \leq i \leq d$ as follows:

- initial feature vector of v: $\overline{v}^{(0)} = g_V(v)$
- for $i > 1$:
  - $\overline{v}^{(i)} = \sigma^{(i)}\left(\mathrm{COM}^{(i)}\left(\overline{v}^{(i-1)}, \overline{y}\right)\right)$, where $\overline{y} = \mathrm{AGG}^{(i)}\left(\{\{\overline{u}^{(i-1)} \mid u \in N_G(v)\}\}\right)$
  - $\mathrm{HLT}\left(i, \left(x_v^{(i)} \mid v \in V\right)\right)$

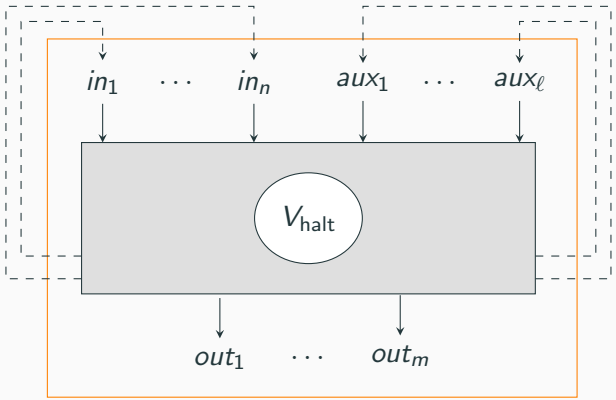
Arithmetic Circuits

Definition 15

Directed acyclic graph with gates that perform arithmetic operations over $\mathbb{R}$.



Recurrent Arithmetic Circuit



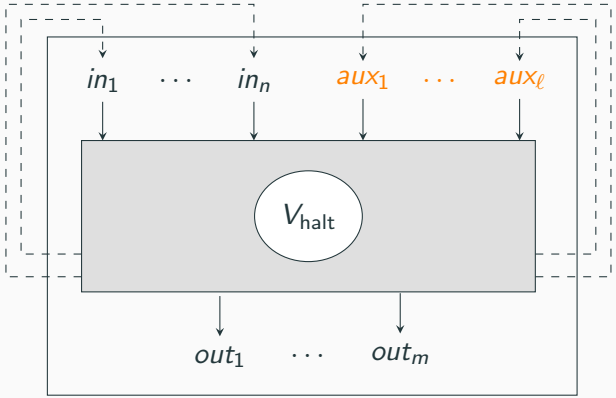
underlying extended arithmetic circuit C

Definition 16

rec-$C = (C, \bar{a}, E_{\text{rec}}, \text{HLT}, V_{\text{halt}})$

- $\text{HLT}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$

Recurrent Arithmetic Circuit



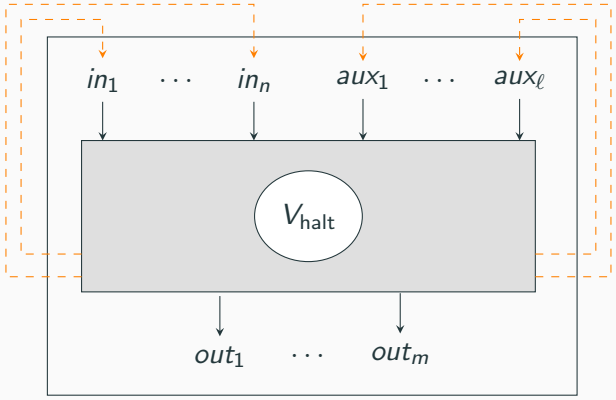
underlying extended arithmetic circuit C

Definition 16

rec-$C = (C, \overline{a}, E_{\text{rec}}, \text{HLT}, V_{\text{halt}})$

- $\text{HLT}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$

Recurrent Arithmetic Circuit



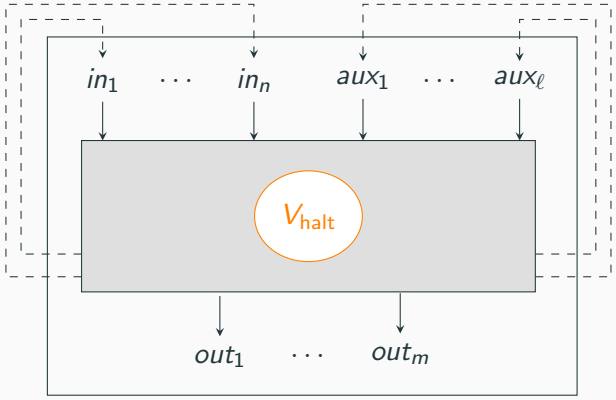
underlying extended arithmetic circuit C

Definition 16

rec-$C = (C, \bar{a}, E_{\text{rec}}, \text{HLT}, V_{\text{halt}})$

- $\text{HLT}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$

Recurrent Arithmetic Circuit



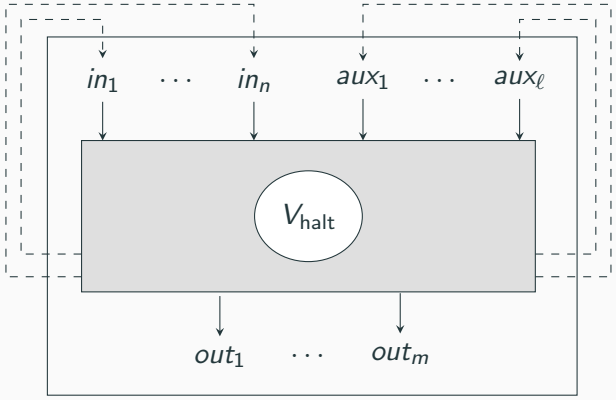
underlying extended arithmetic circuit C

Definition 16

rec-$C = (C, \bar{a}, E_{\text{rec}}, \text{HLT}, V_{\text{halt}})$

- $\text{HLT}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$

Recurrent Arithmetic Circuit



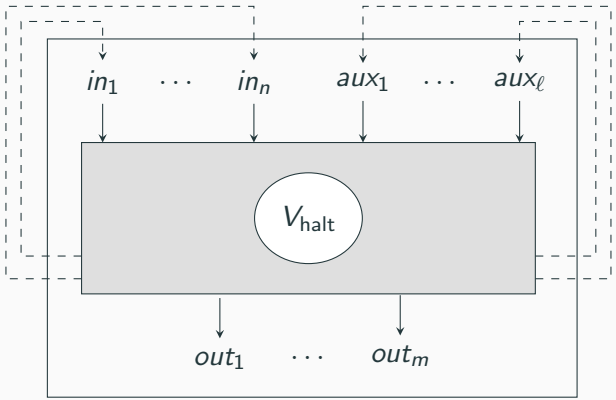
underlying extended arithmetic circuit C

Definition 16

rec-$C = (C, \bar{a}, E_{\text{rec}}, \text{HLT}, V_{\text{halt}})$

- $\text{HLT} \colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$

Recurrent Arithmetic Circuit



underlying extended arithmetic circuit C

Definition 16

rec-$C = (C, \overline{a}, E_{\text{rec}}, \text{HLT}, V_{\text{halt}})$

- $\text{HLT}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$

Recurrent Circuit Function Class

$f_{\text{rec-}\mathcal{C}}\colon \mathbb{R}^* \to \mathbb{R}^* \in \text{rec}[\mathfrak{F}_{\text{halt}}]\text{-}\mathfrak{F}_C$ if:

- function of underlying arithmetic circuit family $\in \mathfrak{F}_C$
- halting functions $\in \mathfrak{F}_{\text{halt}}$

Example: Fibonacci sequence: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$

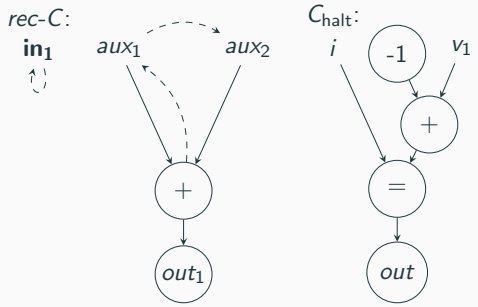
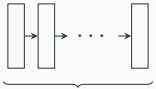


Figure 1: Recurrent circuit *rec-C* with halting circuit C_{halt} computing the x_1-th Fibonacci number for $1 < x_1 \in \mathbb{N}$. The dashed arrows mark recurrent edges and the bold gate in_1 is the single halting gate, i. e. the gate whose value forms the input for C_{halt} along with the iteration number i. aux_1 and aux_2 originally store F_0 and F_1.

Recursive GNNs and Recursive circuits

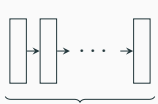
- As we saw: Arithmetic circuits characterise non-recursive GNNs.
- Recursive circuits do the same for recursive GNNs.
- For this we introduce recursive circuit GNNs.
- rec$[t\mathfrak{F}_{\text{halt}}]$-$(t\mathfrak{F}, \mathcal{A})$-GNN:
  - Halting Complexity in $t\mathfrak{F}_{\text{halt}}$
  - Layer Computation Complexity in $t\mathfrak{F}$

Models of Recurrent C-GNNs

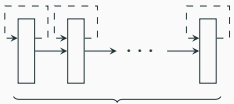


(a) C-GNN without recurrence and a fixed depth $d \in \mathbb{N}$

Models of Recurrent C-GNNs

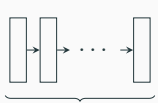


(a) C-GNN without recurrence and a fixed depth $d \in \mathbb{N}$

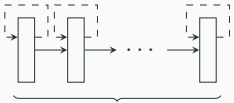


(b) C-GNN with only inner recurrence and a fixed depth $d \in \mathbb{N}$

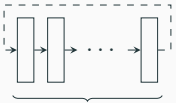
Models of Recurrent C-GNNs



(a) C-GNN without recurrence and a fixed depth $d \in \mathbb{N}$

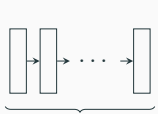


(b) C-GNN with only inner recurrence and a fixed depth $d \in \mathbb{N}$

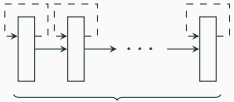


(c) C-GNN with only outer recurrence

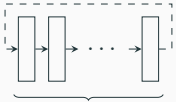
Models of Recurrent C-GNNs



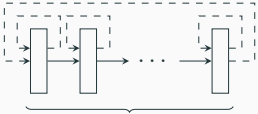
(a) C-GNN without recurrence and a fixed depth $d \in \mathbb{N}$



(b) C-GNN with only inner recurrence and a fixed depth $d \in \mathbb{N}$

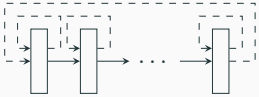


(c) C-GNN with only outer recurrence

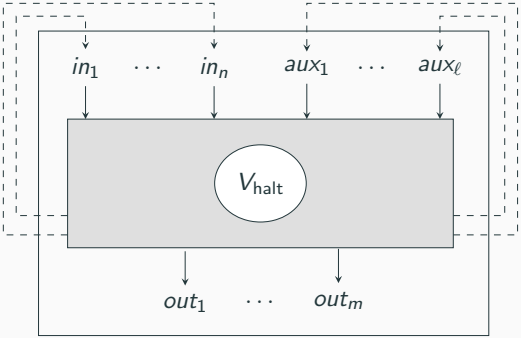


(d) C-GNNs with inner and outer recurrence

Simulating Recurrent C-GNNs With Recurrent Arithmetic Circuits



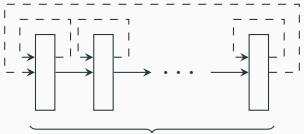
Recurrent C-GNN



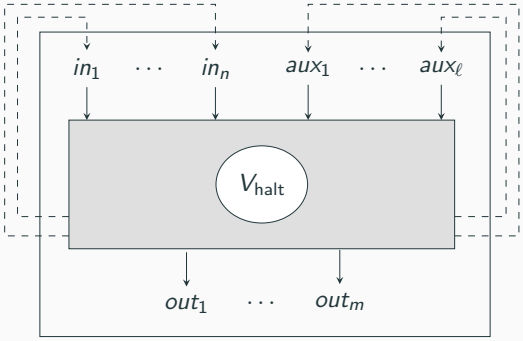
Recurrent Arithmetic Circuit

Simulating Recurrent C-GNNs With Recurrent Arithmetic Circuits

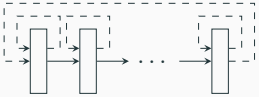
- Encode labelled graphs and use them as input to the circuit as before
- Simulate the layers of the recurrent C-GNNs as before
- New: Use the recurrence of the circuit to simulate recurrence of the C-GNN



Simulating Recurrent Arithmetic Circuits with Recurrent C-GNNs

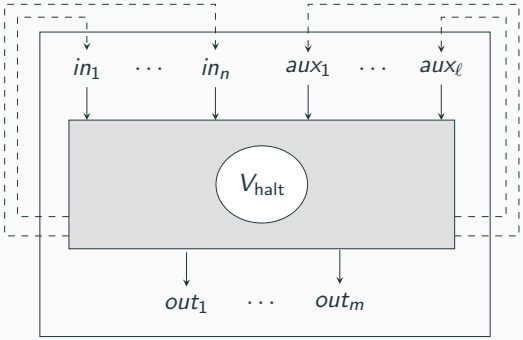


Recurrent Arithmetic Circuit

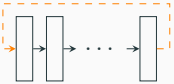


Recurrent C-GNN

Simulating Recurrent Arithmetic Circuits with Outer Recurrent C-GNNs



Recurrent Arithmetic Circuit



Outer Recurrent C-GNN

Simulating Recurrent Arithmetic Circuits with Outer Recurrent C-GNNs

Idea: Encode a recurrent arithmetic circuit as a labelled graph and define a recurrent C-GNN that works on this labelled graph as an input

- Fix a notion of logspace labelled graph encoding of a recurrent circuit
- Simulate the computations of the recurrent circuit
- Halting function of the circuit needs to be partially simulated partly as well
- Challenge: GNN feature vectors need to resetted to simulate recursive computation of the circuit.

Theorem 17

Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a rec$[\mathfrak{F}]$-$\mathfrak{F}[\mathcal{A}]$-circuit family. Then there exists a rec$[\mathrm{t}\mathfrak{F}]$-$(\mathrm{t}\mathfrak{F}[\mathcal{A}], \{id\})$-GNN, such that such that for all $n \in \mathbb{N}$ and $\vec{x} \in \mathbb{R}^n$ $N(G_{C_n,\vec{x}})$ has $C_n(\vec{x})$ as feature vectors.

Simulating Recurrent Arithmetic Circuits with Outer Recurrent C-GNNs

Idea: Encode a recurrent arithmetic circuit as a labelled graph and define a recurrent C-GNN that works on this labelled graph as an input

- Fix a notion of logspace labelled graph encoding of a recurrent circuit
- Simulate the computations of the recurrent circuit
- Halting function of the circuit needs to be partially simulated partly as well
- Challenge: GNN feature vectors need to resetted to simulate recursive computation of the circuit.

As before, a stronger result holds for circuits in function layer form.

Theorem 17

Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a rec$[\mathfrak{F}]$-$f\mathfrak{F}[\mathcal{A}]$-circuit family. Then there exists a rec$[\mathfrak{tF}]$-$(\mathfrak{tF}, \mathcal{A}\cup\{id\})$-GNN, such that such that for all $n \in \mathbb{N}$ and $\vec{x} \in \mathbb{R}^n$ $N(G_{C_n,\vec{x}})$ has $C_n(\vec{x})$ as feature vectors.

Conclusion

- Expressivity of GNNs can be rigorously studied via arithmetic circuits.
- Expressivity of recursive GNNs can be rigorously studied via recursive arithmetic circuits.
- Deep connections between circuit complexity classes and problems decidable with GNNs.

Bibliography i

[Mor+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan and Martin Grohe. 'Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks'. In: *AAAI*. 2019.