Graph Invariants, Local Refinements, and the Expressive Power of Graph Neural Networks

Hyunje Lee

k-tuple and multiset

For a k-tuple x and an element y, by xy we denote the (k+1)-tuple (x1,...,xk,y). Moreover, for every i ∈ [k], by x[y/i] we denote the k-tuple (x1,...,xi−1,y,xi+1,...,xk) and by x[/i] the (k −1)-tuple (x1,...,xi−1,xi+1,...,xk)

A multiset is an unordered collection with repetition, which can formally be described as a function from a set to the positive integers. We use {{...}} to denote multisets.

Graphs

we assume graphs to be finite, undirected, simple, and vertex-labelled. Thus formally, a graph is a tuple G = (V(G),E(G),P1(G),...,Pℓ(G)) consisting of a finite vertex set V (G), a binary edge relation E(G) ⊆ V(G)2 that is symmetric and irreflexive, and unary relations P1(G),...,Pℓ(G) ⊆ V(G) representing ℓ vertex labels.

In other words $P_i(G)$ is a **set of vertices** that have label i.

We always use ℓ to denote the number of labels of a graph; if ℓ = 0, we speak of an unlabelled graph.

We sometimes refer to the vector col(G,v) := (c1, . . . , cℓ) ∈ {0,1}ℓ with ci = 1 if v ∈ Pi(G) and ci = 0 otherwise as the colour of vertex v ∈ V (G). Let me remark that col(G,v) is even defined for unlabelled graphs, where we simply have col(G,v) = () (the empty tuple) for all v. An isomorphism from a graph G to a graph G′ is a bijective mapping f : V(G) → V(G′) that preserves edges as well as labels, that is, vw ∈ E(G) ⇐⇒ f(v)f(w) ∈ E(G′) for all v, w ∈ V(G) and v ∈ Pi(G) ⇐⇒ f(v) ∈ Pi(G′) for all i ∈ [ℓ] and v ∈ V(G).

counting quantifiers $\exists^{\geq p}$

FO = First-Order Logic

a **language** used to write formulas about mathematical structures; syntax:symbols, semantics:meaning when interpreted on a structure

with symbols: variables x, y, … , equality x = y, adjacency E(x, y), predicates $P_i(x)$ (labels)<= These are called atomic formuas

C = FO + Counting Quantifiers

They take FO and **extend** it by using boolean connectives and new quantifiers of the form: $\exists^{\geq p} x\, \varphi(x)$

which means:“there exist at least p elements x such that φ(x) holds.”

This is called **C**, for “Counting logic”.

$\exists x$ as $\exists^{\geq 1} x$ and $\forall x$ as $\neg \exists^{\geq 1} x \neg$

$\exists^{\geq p} x\, \varphi(x)$ is equivalent to

$$\exists x_1 \ldots \exists x_p \Big(\bigwedge_{1 \leq i < j \leq p} x_i \neq x_j \wedge \bigwedge_{i=1}^{p} \varphi(x_i) \Big).$$

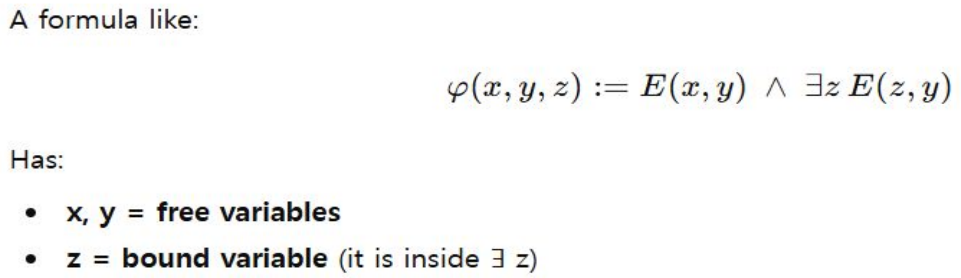
C-formula and free variables

C-formula: A first-order logic formula (FO formula) that is allowed to use counting quantifiers.

eg)There exist at least 5 vertices v such that v has color P₁ and v has degree ≥ 3 $\exists^{\geq 5} v \left(P_1(v) \ \wedge \ \exists^{\geq 3} u \, E(v,u)\right)$

Variables can take values from vertices(usually variables we used so far took values in R but here vertices)

ϕ(x1,...,xk) indicates that the free variables of a formula ϕ are among x1,...,xk



A formula like:

$$\varphi(x,y,z) := E(x,y) \ \wedge \ \exists z \, E(z,y)$$

Has:

- **x, y = free variables**
- **z = bound variable** (it is inside ∃ z)

ϕ(x1,...,xk) means x1,...,xk are all free variables other variables are bounded by quantifiers

$$G \models \varphi(v_1, \ldots, v_k)$$

We write G |= ϕ(v1,...,vk) to denote that G satisfies ϕ if the variables xi are interpreted by the vertices vi.

the meaning of the counting quantifier involving $G \models \varphi(v_1, \ldots, v_k)$: a labelled graph G together with vertices w1,...,wk ∈ V (G) satisfies a formula $\exists^{\geq p} x \, \varphi(x, y_1, \ldots, y_k)$ if there are at least p vertices v ∈ V (G) such that $G \models \varphi(v, w_1, \ldots, w_k)$

eg) if k=1, it’s the c formulas we just saw

k-ary invariant

A graph invariant (or 0-ary graph invariant) is a function ξ defined on graphs such that ξ(G) = ξ(G′) for isomorphic graphs G,G′.

For k ≥ 1, a k-ary graph invariant is a function ξ that associates with each graph G a function ξ(G) defined on $V(G)^k$ in such a way that for all graphs G,G′, all isomorphisms f from G to G′, and all tuples v ∈V(G)k it holds that

$$\xi(G)(\boldsymbol{v}) = \xi(f(G))(f(\boldsymbol{v})).$$

Formally, such a mapping ξ is called equivariant.

eg) the number of vertices is a graph(0-ary) invariant, the degree of a vertex is a vertex invariant(1-ary)

distinguishing and completeness

If $\xi(G, \boldsymbol{v}) \neq \xi(G', \boldsymbol{v}')$, then ξ distinguishes (G,v) and (G′,v′)

In other words, xi distinguishes the two k-tuples

equivariance condition implies that if ξ distinguishes (G,v) and (G′,v′) then there is no isomorphism from G to G′ that maps v to v′.(always true we can show this by contradiction)

If the converse also holds,(that is xi(G,v)=xi(G’,v’) implies that there exists an isomorphism) then ξ is a complete invariant.(generally not true)

An invariant ξ is **complete** if the reverse direction also holds:

$$\xi(G, v) = \xi(G', v') \iff \exists \text{ isomorphism } f : G \to G' \text{ with } f(v) = v'.$$

ξ captures *exactly* all isomorphism information.

$$\widehat{\xi}(G)$$

From a k-ary invariant ξ we can derive a graph invariant ξ^ by mapping each graph G to the multiset

$$\widehat{\xi}(G) := \{\!\{ \xi(G, \boldsymbol{v}) \mid \boldsymbol{v} \in V(G)^k \}\!\}$$

In other words, it is a multiset of all ξ-values over all k-tuples. It depends only on the isomorphism class of G because it is an equivariant.(if there is an isomorphism between G and G' then xi_hat(G)=xi_hat(G'))

ξ distinguishes two graphs G,G′ if $\widehat{\xi}(G) \neq \widehat{\xi}(G')$.

If ξ is a complete invariant then ξ^ is complete as well. (Why?)

if ξ is complete and $\hat{\xi}(G)=\hat{\xi}(G')$

G and G' have the same number of vertices.For all k tuples v of vertices in G there exists v' in G' $\xi(G,v)=\xi(G',v')$ and there is an isomorphism with v->v' for each pair(v,v') and these local isomorphisms agree on overlaps and glue together into a global isomorphism $\varphi:V(G)\to V(G')$ with $\varphi(v_i)=v'_i$ between G and G′. Hence $\hat{\xi}$ is complete.

an example that two graphs have the same multisets but they are not isomorphic:consider a k-ary invariant $\xi(G,v)=0$ for all v

and two graphs a path P_4 and a star $K_{1,3}$

obviously $\hat{\xi}(G)=\hat{\xi}(G')$ but there's no isomorphism.

atomic type $\mathrm{atp}_k(G, \boldsymbol{v})$

atp_k is a k-ary invariant such that

for a graph G with ℓ labels(think of it as the number of colorings) and a tuple v = (v1,...,vk) ∈ V(G),

$$\mathrm{atp}_k(G, \boldsymbol{v}) \in \{0,1\}^{2\binom{k}{2}+k\ell}$$

the vector which for 1 ≤ i < j ≤ k has two entries indicating whether vi = vj and whether vivj ∈ E(G) and for 1 ≤ i ≤ k has ℓ entries indicating whether vi is in P1(G),...,Pℓ(G).(it encodes all the structural information about the graph)

atp_k(G,v) = atp_k(G′,v′) if and only if the mapping vi → v′ i is an isomorphism from the induced subgraph G[{v1,...,vk}] to the induced subgraph G′[{v′ 1,...,v′ k}].(this is exactly the complete k-ary invariant)

colourings

atp1 = col

We often think of mappings χ defined on V(G) or V(G)^k as colourings of the vertices or k-tuples of vertices of a graph G

the range of χ is not necessarily{red,blue,green,etc…} it can be R or even R^n

in which case we call it n dimensional feature map.

elements of the image of χ: colors
pre-images χ−1(c):color classes

refinement

For colourings χ,χ′ : V (G)k → C, we say that

χ refines χ′ (we write $\chi \preceq \chi'$) for all v,w ∈ V(G)^->C if χ(v) = χ(w) then χ′(v) = χ′(w).

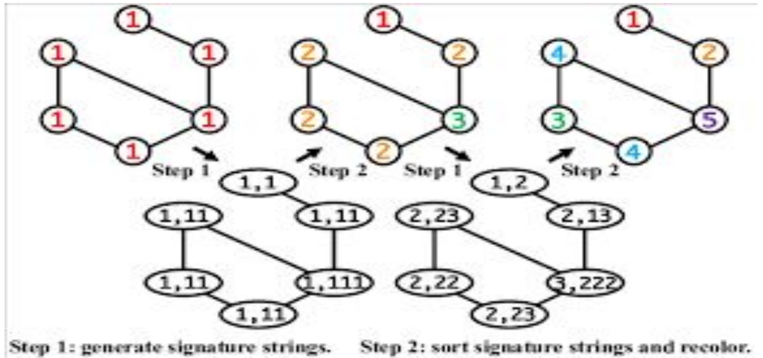
χ,χ′ equivalent (we write $\chi \equiv \chi'$) if $\chi \preceq \chi'$ and $\chi' \preceq \chi$

For k-ary invariants ξ,ξ′ we say that ξ refines ξ′ if ξ(G) refines ξ′(G) for every graph G, and similarly for equivalence.

THE WEISFEILER-LEMAN ALGORITHM

The basic idea is to label vertices of the graph with their iterated degree sequence. More precisely, the initial colouring given by the vertex labels in a graph is repeatedly refined by counting, for each colour, the number of neighbours of that colour.

In other words, we assign initial colours to vertices and we change the colours of each vertex each step based on the number of its colour, number of neighbours and neighbours' colours. And there is some number that after this number the vertex colours don't change anymore.


Step 1
Step 2
Step 1
Step 2
1,1
1,11
1,11
1,11
1,111
1,11
1,11
1,2
2,23
2,13
2,22
3,222
2,23
Step 1: generate signature strings.
Step 2: sort signature strings and recolor.

More formally…

For every graph G, we define a sequence of vertex colourings $\mathsf{cr}^{(t)}(G)$ as follows: for every $v \in V(G)$, we let $\mathsf{cr}^{(0)}(G, v) := \mathsf{col}(G, v)$ and

$$\mathsf{cr}^{(t+1)}(G, v) := \Big(\mathsf{cr}^{(t)}(G, v), \{\!\{\mathsf{cr}^{(t)}(G, w) \mid w \in N(v)\}\!\}\Big).$$

this means that the colour of v at step t+1 depends on color at t of v and the multiset of its neighbours colours

And two vertices have the same colour only when they had same colour before and they had the same number of neighbours with the same colours

At each step the coloring becomes finer and finer $\mathsf{cr}^{(t+1)}(G) \preceq \mathsf{cr}^{(t)}(G)$

and there is some number the coloring doesn't become strictly finer anymore

$$\mathsf{cr}^{(t_\infty+1)}(G) \equiv \mathsf{cr}^{(t_\infty)}(G)$$

we call this stable colouring and denote it by $\mathsf{cr}^{(\infty)}(G)$. $\mathsf{cr}^{(t)}$ is a vertex invariant.

k-dimensional Weisfeiler-Leman algorithm (k-WL)

The essential idea is the same as 1dimensional case. each vertex is replaced k tuple of vertices and the neigbours are replaced with v[w/i] = (v1,...,vi−1,w,vi+1,...vk)

we let $\mathsf{wl}_k^{(0)}(G, \boldsymbol{v}) := \mathsf{atp}_k(G, \boldsymbol{v})$ and

$$\mathsf{wl}_k^{(t+1)}(G, \boldsymbol{v}) := \left(\mathsf{wl}_k^{(t)}(G, \boldsymbol{v}), M\right)$$

with

$$\begin{aligned} M = \Big\{\!\!\Big\{ \Big(\mathsf{atp}_{k+1}(G, \boldsymbol{v}w), \mathsf{wl}_k^{(t)}&(G, \boldsymbol{v}[w/1]), \\ \mathsf{wl}_k^{(t)}&(G, \boldsymbol{v}[w/2]), \\ &\vdots \\ \mathsf{wl}_k^{(t)}&(G, \boldsymbol{v}[w/k]) \Big) \;\Big|\; w \in V(G) \Big\}\!\!\Big\}. \end{aligned}$$

machine learning scenarios on graphs

There are two machine learning scenarios on graphs.

(1)supervised learning: we want to learn an unknown function defined on graphs,nodes or tuples (just like in statistics)

(2) In the representation-learning case, we are given graphs as inputs and we want to compute vector representations that summarize their structure (similar in spirit to atomic). atp_k **)**

and we want the functions we learn to be invariant. FNNs do not ensure the invariance but GNN architectures were designed so that invariance holds automatically.

GNN(1)

aggregation function: (written as “agg”) mappingfinite multisets of vectors in R^p to vectors in $R^{p'}$

combination function(written as “comb”) : $R^{(p+p')} \to R^q$

A GNN is a tuple $(L^{(1)},...,L^{(d)})$ of GNN layers, where the output dimension $q(i)$ of $L^{(i)}$ matches the input dimension $p^{(i+1)}$ of $L^{(i+1)}$. We call $q^{(0)} := p^{(1)}$ the input dimension of the GNN and $q^{(d)}$ the output dimension.

(recap) p-dimensional feature maps: $\zeta : V(G) \to R^p$

GNN(2)

Let L be a GNN layer of input dimension p and output dimension q

Applied to a graph G, it transforms a p-dimensional feature map ζ to a q-dimensional feature map η defined by

$$\eta(v) := \mathsf{comb}\Big(\zeta(v), \mathsf{agg}\big(\{\!\{\zeta(w) \mid w \in N^G(v)\}\!\}\big)\Big)$$

this formula describes the transformation of p dimensional featuremap zeta into q dimensional feature map eta

zeta(v) is the current feature vector. {{ζ(w)∣w∈N^G(v)}} is the multi set of feature vectors of all neighbours of v and we aggregate them(eg.sum,mean) and we combine the node’s feature and the aggregated neighbour features.

GNN(3)

Think of the features $\zeta(v)$ of the vertices as colouring. Then the feature transformation computed by the GNN layer is based on a simple distributed message passing protocol(Each node sends its feature vector to its neighbors, and then each node updates its own feature using what it received.) whose computation nodes are the vertices of our graph: each node v sends its colour $\zeta(v)$ to all its neighbours. Upon receiving the colour $\zeta(w)$ of its neighbours, node v applies the aggregation function agg to the multiset of these states, yielding a vector $\alpha(v)$ and then uses the combination function comb to compute the new state $\eta(v)$ from $\zeta(v)$ and $\alpha(v)$.

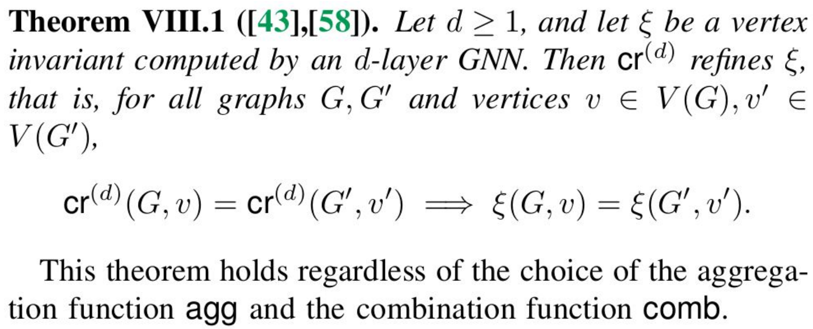
Additionally, this transformation is isomorphism invariant: for graphs G,G' and feature maps $\zeta \in Fp(G),\zeta' \in Fp(G')$ transformed by L to feature maps η,η', if f is an isomorphism from G to G' satisfying $\zeta(v) = \zeta'(f(v))$ for all $v \in V(G)$, then $\eta(v) = \eta'(f(v))$ for all $v \in V (G)$.

recurrent GNN

There is an alternative mode of operation of GNNs where we do not have a fixed number of layers but repeatedly apply the same layer. A recurrent GNN consists of a single GNN layer with the same input and output dimension q. Applied to an initial q-dimensional feature map $\zeta(0)$, it yields an infinite sequence $\zeta(1),\zeta(2),...$ of q-dimensional feature maps, where $\zeta(t+1)$ is obtained by applying the GNN layer to $(G,\zeta(t))$.

In the end, we usually do not want to compute feature trans formations but functions defined on graphs (graph invariants) or on the vertices of graphs (vertex invariants). The features are only supposed to be internal representations. Towards this end, we choose the initial feature map $\zeta(0)$ to represent the labelling of the input graph G. That is, we let $\zeta(0) := \mathrm{col}(G)$ possibly padded by 0s to reach the input dimension of our GNN. We always assume that the input dimension $q(0)$ of the GNN is at least the label number ℓ of the input graphs.

1-WL is as expressive as GNN



Theorem VIII.1 ([43],[58]). *Let $d \geq 1$, and let ξ be a vertex invariant computed by an d-layer GNN. Then $\mathsf{cr}^{(d)}$ refines ξ, that is, for all graphs G, G' and vertices $v \in V(G), v' \in V(G')$,*

$$\mathsf{cr}^{(d)}(G, v) = \mathsf{cr}^{(d)}(G', v') \implies \xi(G, v) = \xi(G', v').$$

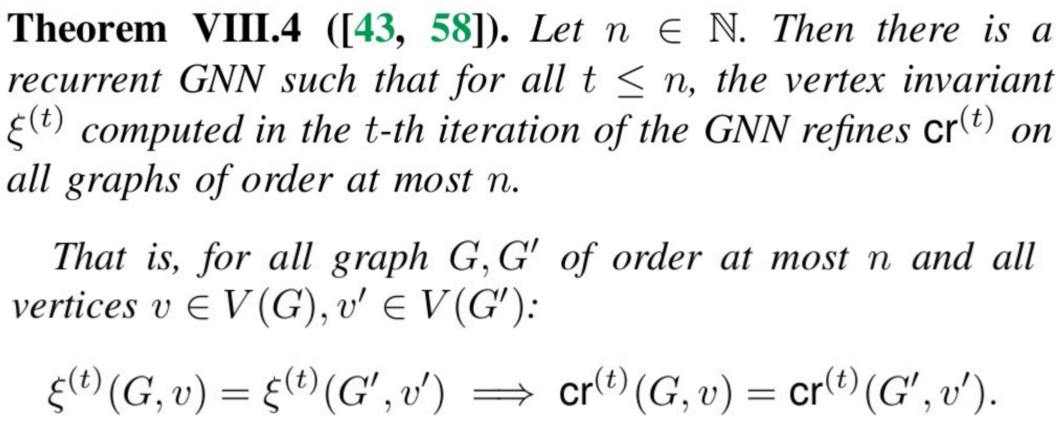
This theorem holds regardless of the choice of the aggregation function agg and the combination function comb.

ξ is the vertex invariant computed by GNN, in other words the final feature vector GNN assigns to node v and cr^(d) is the colouring at step d of the 1 dimensional WL algorithm.

The theorem states that if WL assigns the same colour to two nodes after d steps then a d-layer GNN assigns the same feature vector to those two nodes.

Remark: cr^(d) is at least as strong(expressive) as d-layer GNN

GNN is as expressive as 1-WL node wise



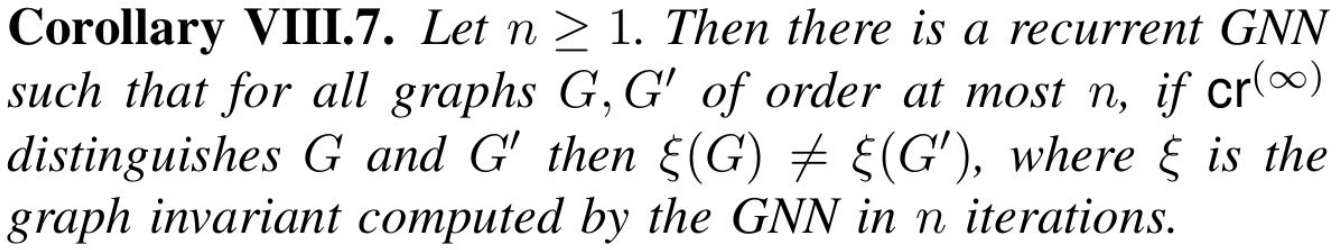
Theorem VIII.4 ([43, 58]). *Let $n \in \mathbb{N}$. Then there is a recurrent GNN such that for all $t \leq n$, the vertex invariant $\xi^{(t)}$ computed in the t-th iteration of the GNN refines $\mathsf{cr}^{(t)}$ on all graphs of order at most n.*

That is, for all graph G, G' of order at most n and all vertices $v \in V(G), v' \in V(G')$:

$$\xi^{(t)}(G, v) = \xi^{(t)}(G', v') \implies \mathsf{cr}^{(t)}(G, v) = \mathsf{cr}^{(t)}(G', v').$$

xi(t) is the vertex feature map produced by GNN after t steps and cr(t) is the colouring produced by 1-WL algorithm after t steps. xi(t) refines cr(t) means xi(t) can distinguish at least as much as cr(t)

GNN is as expressive as 1-WL graph wise



Corollary VIII.7. *Let $n \geq 1$. Then there is a recurrent GNN such that for all graphs G, G' of order at most n, if* $\mathsf{cr}^{(\infty)}$ *distinguishes G and G' then $\xi(G) \neq \xi(G')$, where ξ is the graph invariant computed by the GNN in n iterations.*

This corollary is slightly different from the previous theorem, xi was vertex invariant before and it's now graph invariant and it compares GNN in n iterations to cr^(∞), the stable colouring.

$$cr^{(\infty)}(G) \neq cr^{(\infty)}(G') \quad \Rightarrow \quad \xi(G) \neq \xi(G').$$

It states that if cr(∞) distinguishes G and G', so does xi.

In other words, a recurrent GNN can match the full distinguishing power of 1-WL

Conclusion

From theorems VIII.1, VIII.4, Corollary VIII.7, we could conclude that on graphs of bounded size(eg.finite number of vertices), GNNs and 1-WL have the same expressive power.

Reference

Grohe, M. (2021). *The Logic of Graph Neural Networks*.

arXiv:2104.14624.