Soft Computing (2024) 28:8527–8547
https://doi.org/10.1007/s00500-024-09676-1

On the approximation capability of GNNs in node classification/regression tasks

Giuseppe Alessio D'Inverno[1] · Monica Bianchini[1] · Maria Lucia Sampoli[1] · Franco Scarselli[1]

Accepted: 15 January 2024 / Published online: 5 July 2024

Abstract

Graph neural networks (GNNs) are a broad class of connectionist models for graph processing. Recent studies have shown that GNNs can approximate any function on graphs, modulo the equivalence relation on graphs defined by the Weisfeiler–Lehman (WL) test. However, these results suffer from some limitations, both because they were derived using the Stone–Weierstrass theorem—which is existential in nature—and because they assume that the target function to be approximated must be continuous. Furthermore, all current results are dedicated to graph classification/regression tasks, where the GNN must produce a single output for the whole graph, while also node classification/regression problems, in which an output is returned for each node, are very common. In this paper, we propose an alternative way to demonstrate the approximation capability of GNNs that overcomes these limitations. Indeed, we show that GNNs are universal approximators in probability for node classification/regression tasks, as they can approximate any measurable function that satisfies the 1-WL-equivalence on nodes. The proposed theoretical framework allows the approximation of generic discontinuous target functions and also suggests the GNN architecture that can reach a desired approximation. In addition, we provide a bound on the number of the GNN layers required to achieve the desired degree of approximation, namely $2r - 1$, where r is the maximum number of nodes for the graphs in the domain.

Keywords GNN · Approximation · Node-focused · 1-WL test · Unfolding trees

1 Introduction

Graph processing is becoming pervasive in many application domains, such as social networks, Web applications, biology, and finance. Intuitively, graphs allow to represent patterns along with their relationships. Indeed, graphs can naturally encode high-valued information that is hard to represent with vectors or sequences, the most common data structures used in Machine Learning (ML). Graph Neural Networks (GNNs) are a class of machine learning models that can process information represented in the form of graphs. In recent years, the interest in GNNs has grown rapidly and numerous new models and applications have emerged (Wu 2020). The first GNN model was introduced in Scarselli (2009b). Later, several other approaches have been proposed, including Spectral Networks (Bruna et al. 2014), Gated Graph Sequence Neural Networks (Li et al. 2015b), Graph Convolutional Neural Networks (Kipf and Welling 2017), GraphSAGE (Hamilton et al. 2017), Graph attention networks (Veličković 2018), and Graph Networks (Battaglia 2018). However, despite the differences among the various GNN models, most adopt the same computational scheme, based on a local aggregation mechanism. The information related to a node is stored into a feature vector, which is updated recursively by aggregating the feature vectors of neighboring nodes. After k iterations, the feature vector of a given node v captures both structural information and attributes of the nodes in the k-hop neighborhood of v. At the end of the learning process, the node feature vectors can be used to classify or to cluster the objects/concepts represented by a (some) node(s), or by the whole graph.

✉ Giuseppe Alessio D'Inverno
dinverno@diism.unisi.it

Monica Bianchini
monica.bianchini@unisi.it

Maria Lucia Sampoli
marialucia.sampoli@unisi.it

Franco Scarselli
franco.scarselli@unisi.it

[1] Department of Information Engineering and Mathematics, University of Siena, Via Roma 56, 53100 Siena (SI), Italy

Recently, a great effort has been devoted to study the expressive power of GNNs (Sato 2020). Such a theoretical property has an important impact in machine learning, since it defines what are the applications that can be faced by a neural network model, it can explain observed limitations in experiments and, finally, it can suggest novel advancements to improve the considered model. In GNNs, the capabilities and the limitations of the model primarily depend on the local computational framework, since GNNs can take into account both the connectivity and the features of the neighboring nodes, but they may not be able to distinguish between nodes having similar neighborhoods. Therefore, a fundamental question is to define which graphs (nodes) can be distinguished by a GNN, i.e., for which input graphs (nodes) the GNN produces different encodings. In Xu et al. (2018), GNNs are proved to be as powerful as the Weisfeiler–Lehman graph isomorphism test (1-WL) (Lehman and Weisfeiler 1968). Such an algorithm allows to test whether two graphs are isomorphic or not .[1] The 1-WL algorithm is based on a graph signature which is obtained by assigning a color to each node, where the graph coloring is achieved by iterating a local aggregation function. More generally, there exists a hierarchy of algorithms, called 1-WL, 2-WL, 3-WL, etc., which recognizes larger and larger classes of graphs. It has been shown that a GNN can simulate the 1-WL test, provided that a sufficiently general aggregation function is used, but the basic GNN model cannot implement higher order tests (Morris 2019). Consequently, the 1-WL test characterizes both the expressiveness and limitations of GNNs, defining the classes of graphs/nodes that GNNs can distinguish.

Another important aspect is the study of the approximation capability of GNNs. Formally, in node classification/regression tasks, a GNN implements a function $\varphi(\mathbf{G}, v) \to \mathbb{R}^o$ that takes in input a graph $\mathbf{G}$ and returns an output at each node. Similarly, in graph classification/regression tasks, a GNN implements a function $\varphi(\mathbf{G}) \to \mathbb{R}^o$. In both cases, the objective is to define which classes of functions can be approximated by a GNN.

In Scarselli (2009a), the approximation capability of the original GNN model (OGNN), namely the first GNN model to be proposed, has been studied using the concept of unfolding trees and unfolding equivalence. The unfolding tree $\mathbf{T}_v$, with root node v, is constructed by unrolling the graph starting from v (see Fig. 1). Intuitively, $\mathbf{T}_v$ exactly describes the information used by the GNN at node v and can be employed to study the expressive power of GNNs in node classification/regression tasks. The unfolding equivalence is, in turn, an equivalence relationship defined between nodes having the same unfolding tree. In Scarselli (2009a), it was proved that OGNNs can approximate in probability, up to any degree of precision, any measurable function $\tau(\mathbf{G}, v) \to \mathbb{R}^o$ that respects the unfolding equivalence, namely, that produces the same outputs on equivalent nodes. Currently, unfolding trees—also termed *computation graphs* (Garg et al. 2020)—are widely used to study the GNN expressiveness. Universal approximation results have been proved for Linear Graph Neural Networks (Azizian and Lelarge 2020; Maron et al. 2018), Folklore Graph Neural Networks (Maron et al. 2019), and, more generally, for a large class of GNNs (Xu et al. 2018; Azizian and Lelarge 2020) that includes most of the recent architectures, also considered in this paper.

Despite many advances in research on approximation theory for GNNs, there are still open problems to be investigated. First of all, the most general results available on modern GNNs are based on the Stone–Weierstrass theorem and state that the functions which can be approximated by GNNs are dense in the invariant continuous function space, modulo the 1-WL test (Azizian and Lelarge 2020). However, the Stone–Weierstrass theorem is existential in nature, so that, given a target function to be approximated, it does not allow to construct a GNN architecture that can reach the desired approximation—defining, for example, the number of its layers, and the feature dimension required to build the approximator. Moreover, the current results apply only to continuous functions on node/edge labels, which are defined on a compact subset of $\mathbb{R}^{\mathbf{q}}$, a fact that may not hold in practical application domains, since, for instance, the function to be approximated may show step-wise behavior with respect to some inputs. Finally, all the results on the expressive capacity of modern GNN models are dedicated to graph classification/regression tasks, but node classification/regression problems are also widely present in practical applications and it is important to generalize the theoretical results on expressivity to them as well. In addition, it is useful to study the relationships between unfolding trees and the 1-WL test in this context. Indeed, it can be observed that the Weisfeiler–Lehman test assigns a color to all the nodes of a graph to make them distinguishable, and it can be naturally expected that the equivalence classes defined by the colors are related to those defined by the unfolding trees. In fact, it has been proved that the two mechanisms, colors or unfolding trees, produce the same profiles for graphs (Krebs and Verbitsky 2015), namely the same number of nodes per equivalence class, but whether they produce exactly the same profiles with respect to single nodes, i.e., nodes get assigned the same equivalence class, is still an open problem. A formal and precise answer to this question will allow us to use the two frameworks in a targeted or exchangeable way in the context of node classification/regression tasks.

In this work, we present an alternative approach to study the approximation capability of recent GNNs that allows to answer to the above questions.

[1] It is worth noting that the 1-WL test is inconclusive, since there exist pairs of graphs that the test recognizes as isomorphic even if they are not.

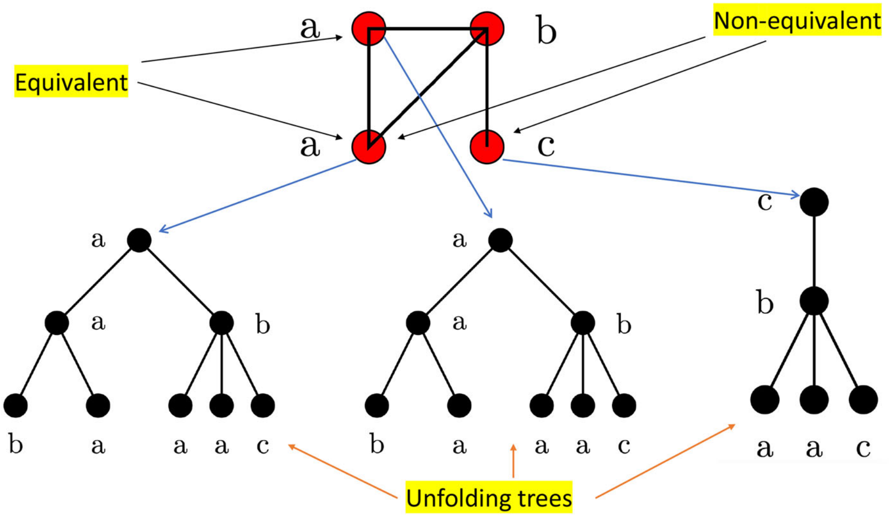


Fig. 1 An example of a graph with some unfolding trees. The symbols outside the nodes represent features. The two nodes on the left part of the graph are equivalent and have equivalent unfolding trees

The main contributions of this paper are listed below.

- We prove that, on connected graphs, modern GNNs, realizing node-focused functions, are capable of approximating, in probability and up to any precision, any measurable function that respects the 1-WL-equivalence. Intuitively, this means that GNNs are a kind of universal approximators for functions on the nodes of the graph, modulo the limits enforced by the 1-WL test. Such a result describes the GNN capability for node classification/regression tasks.
- The presented proof is the most general on the GNN approximation capability that we are aware of, since it holds for generic graphs with real feature vectors and for a broad class of GNNs, which includes most of the current models. Moreover, it is assumed that the target function is measurable, which permits the approximation of discontinuous and more complex functions w.r.t. existing results, e.g., Jegelka (2022). Finally, the proof is based on a technique that allows us to deduce information on the architecture of the GNN that can reach the desired approximation. Such an information cannot be derived with the Stone–Weierstrass theorem and includes, for instance, hints on the number of iterations, the number of layers, the dimension of hidden features, and the type of the network to be used to implement the aggregation function.
- It is shown that, to reach any desired approximation accuracy, a single real hidden feature is sufficient, the aggregation network must contain at least one hidden layer, and the GNN must adopt at least $2r-1$ iterations, namely the GNN must include $2r-1$ layers, where r is the maximum number of nodes of any graph in the domain. The latter bound on GNN iterations/layers can be surprising, because we may expect that r iterations are sufficient to diffuse the information on the whole graph. We will clarify that such a bound is due to the nature of node classification/regression tasks. Actually, r iterations are sufficient for graph classification/regression tasks, but they are not enough for node-focused tasks, which are more expensive from a computational point of view.
- A set of experiments has been carried out to show that GNNs, if their architectures are sufficiently general, can approximate any function, modulo the unfolding equivalence/1-WL test, up to a desired degree of precision, so as suggested by the proposed theoretical results.

We remark that understanding the approximation power of GNNs is fundamental to explain GNN limitations and capabilities in practical applications and to have suggestions for designing novel advanced models. The present contribution aims to fill the gap left in literature on the characterization of the expressive power of modern GNNs under some crucial aspects, such as the universality on real-attributed graphs, the approximation capabilities on node-focused tasks, and the number of required layers. This finally contributes to a more thorough comprehension of the GNN machine learning framework.

The rest of the paper is organized as follows. In Sect. 2, some related work is described. Notation and basic concepts are introduced in Sect. 3, while Sect. 4 presents the main contribution of this paper. In Sect. 5, we present the experiments conducted to validate our theoretical results. Finally, Sect. 6 gives some conclusive remarks and presents future perspectives. To make the reading more fluid, the proofs are collected in the Appendix.

2 Related work

Great attention has recently been paid to the Weisfeiler–Lehman test and its correlation with the expressiveness of GNNs. Xu et al. (2018) have shown that message passing GNNs are at most as powerful as the 1-WL test; this upper bound could be overcome by injecting the node identity in the message passing procedure, as implemented in You et al. (2021). Morris (2019) have gone beyond the 1-WL test, implementing k-order WL tests as message passing mechanisms into GNNs. In Sato (2020), the WL test mechanism applied to GNNs is studied within the paradigm of unfolding trees (also called *computational graphs*), without really establishing an equivalence between the two concepts, so as in Zhang and Li (2021) (where the unfolding trees are called *rooted subgraphs*). In Alon and Yahav (2020), it is shown that the Weisfeiler–Lehman test tends to oversquash the information coming from the neighbors; moreover, it is claimed that GNNs with at least K layers, where K is the diameter of the graphs in the dataset, do not suffer from under-reaching, which means that the information cannot travel farther than K edges along the graph. Nevertheless, a theoretical proof that GNNs succeed in overcoming the under-reaching behavior is not provided.

Universal approximation properties have been demonstrated for several GNN settings. The OGNN (Scarselli 2009b) model was proved to be a universal approximator on graphs preserving the unfolding equivalence in Scarselli (2009a). Universal approximation is shown for GNNs with random node initialization in Abboud et al. (2020) while, in Xu et al. (2018), and they are proved to be able to encode any graph with countable input features. The universal approximation property has been extended to Folklore Graph Neural Networks in Maron et al. (2019), to Linear Graph Neural Networks and general GNNs in Azizian and Lelarge (2020) and Maron et al. (2018), both in the invariant and equivariant case, but without any reference to the required number of layers. A relation between the graph diameter and the computational power of GNNs has been established in Loukas (2019), where the GNNs are assimilated to the so-called LOCAL models (Angluin 1980; Linial 1992; Naor and Stockmeyer 1993) and it is proved that a GNN with a number of layers larger than the diameter of the graph can compute any Turing function of the graph. Nevertheless, no information on the aggregation function characterization is given. The generalization capability of GNNs has been also studied using different approaches, which include the Vapnik–Chervonenkis dimension for OGNNs (Scarselli et al. 2018), and the uniform stability (Zhou and Wang 2021) and Rademacher complexity (Garg et al. 2020) for modern GNNs. Designing GNN architectures that provide good generalization along with good expressive power is a hot research topic (see, e.g., Puny et al. 2020). Moreover, an extensive survey on the theory of Graph Neural Networks can be found in Jegelka (2022).

The results presented in this work differ from what can be found in literature mainly because we prove the GNN ability to approximate measurable functions based on a proof which is constructive, i.e., capable of suggesting the network architecture that will guarantee a given approximation.

3 Preliminaries

In this section, we introduce the required notation and the basic definitions used throughout the manuscript.

3.1 Graphs

A graph $\mathbf{G}$ is a pair $(\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the set of *vertices* or *nodes* and $\mathbf{E}$ is the set of *edges* between nodes in $\mathbf{V}$. Graphs are *directed* or *undirected*, according to whether the edge (v, u) is different from the edge (u, v) or not. Moreover, a graph is *connected* if there is a path from any node to any other node in the graph. In the following, we assume that graphs are undirected and connected.

The set $ne[v]$ is the *neighborhood* of v, i.e., the set of nodes connected to v by an edge, while $ne_i(v)$ denote the i-th neighbor of v—the set of all nodes connected to v with a path of length i. Finally, $|\mathbf{G}|$ defines the cardinality of the set of vertices in $\mathbf{G}$. From now on, we will always consider graphs with finite cardinality, i.e., $|\mathbf{G}| = r < \infty$.

Nodes may have attached features, collected into vectors called *labels*, identified with $\ell_v \in \mathbb{R}^q$.

3.2 Graph neural networks

Graph Neural Networks adopt a local computational mechanism to process graphs. The information related to a node v is stored into a feature vector $\mathbf{h}_v \in \mathbb{R}^m$, which is updated recursively by combining the feature vectors of neighboring nodes. After k iterations, the feature vector $\mathbf{h}_v^k$ is supposed to contain a representation of both the structural information and the node information within a k-hop neighborhood. After processing is complete, the node feature vectors can be used to classify the nodes or the entire graph.

More rigorously, in this paper, we consider GNNs that use the following general updating scheme:

$$\begin{aligned}\mathbf{h}_v^k = \ & \mathrm{COMBINE}^{(k)}\big(\mathbf{h}_v^{k-1}, \\ & \mathrm{AGGREGATE}^{(k)}\big(\{\{\mathbf{h}_u^{k-1},\ u \in ne[v]\}\}\big)\big), \qquad (1)\end{aligned}$$

where the node feature vectors are initialized with the node labels, i.e., $\mathbf{h}_v^0 = \ell_v \in \mathbb{R}^q$ for each v. Here, differently from other approaches, we assume that labels can contain real numbers. Moreover, AGGREGATE$^{(k)}$ is a function which aggregates the node features obtained in the $(k-1)$-th iteration, and COMBINE$^{(k)}$ is a function that combines the aggregation of the neighborhood of a node with its feature at the $(k-1)$-th iteration. In graph classification/regression tasks, the GNN is provided with a final READOUT layer that produces the output combining all the feature vectors at the last iteration K

$$\mathbf{o} = \text{READOUT}(\{\{\mathbf{h}_v^K,\ v \in \mathbf{V}\}\}) \tag{2}$$

whereas, in node classification/regression tasks, the READOUT layer produces an output for each node, based on its features:

$$\mathbf{o}_v = \text{READOUT}(\mathbf{h}_v^K). \tag{3}$$

In this paper, we will focus mainly on node classification/regression tasks. The learning domain of the GNN will be denoted by the graph–node pair $\mathcal{D} = \mathcal{G} \times \mathcal{V}$, where $\mathcal{G}$ is a set of graphs and $\mathcal{V}$ is a subset of their nodes. Therefore, the function φ, implemented by the GNN, takes in input a graph $\mathbf{G}$ and one of its nodes v, and returns an output $\varphi(\mathbf{G}, v) \in \mathbb{R}^o$, where o is the output dimension.

The framework described by Eqs. (1)–(3) is commonly used to study theoretical properties of modern GNNs (see, e.g., Xu et al. 2018). The class of models covered by such a framework is rather wide and includes, for example, GraphSAGE (Hamilton et al. 2017), GCN (Kipf and Welling 2017), GATs (Veličković 2018), GIN (Xu et al. 2018), ID-GNN (You et al. 2021), and GSN (Bouritsas et al. 2020).

It is worth mentioning that the OGNN model is not formally covered, both because in OGNNs, the input of AGGREGATE$^{(k)}$ and COMBINE$^{(k)}$ contains the node labels ℓ_v and possibly also the edge features, and because the node features are not initialized to ℓ_v. Other models, such as MPNN (Gilmer et al. 2017), NN4G (Micheli 2009), and GN (Battaglia 2018), are not included as well for similar reasons. Of course, Eq. (1) could easily be extended to include also OGNNs and the models mentioned above, but here we prefer not to complicate the proposed framework to keep the notation and proofs simple.

3.3 Unfolding trees and unfolding equivalence

Unfolding trees[2] and *unfolding equivalence* are two concepts that have been introduced in Scarselli (2009a) with the aim of capturing the expressive power of the OGNN model. Intuitively, an *unfolding tree* $\mathbf{T}_v^d$ is the tree obtained by unfolding the graph up to the depth d, using the node v as its root. Figure 1 shows some examples of unfolding trees. In the following, a formal recursive definition is provided.

Definition 3.1 The unfolding tree $\mathbf{T}_v^d$ of a node v up to depth d is

$$\mathbf{T}_v^d = \begin{cases} \text{Tree}(\ell_v) & \text{if} \quad d = 0 \\ \text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{d-1}) & \text{if} \quad d > 0, \end{cases}$$

where $\text{Tree}(\ell_v)$ is a tree constituted of a single node with label ℓ_v and $\text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{d-1})$ is the tree with the root node labeled with ℓ_v and having sub-trees $\mathbf{T}_{ne[v]}^{d-1}$. The set $\mathbf{T}_{ne[v]}^{d-1} = \{\mathbf{T}_{u_1}^{d-1}, \mathbf{T}_{u_2}^{d-1}, \ldots\}$ collects all unfolding trees having depth $d-1$, with $u_i \in ne[v]$, $\forall i$.

Moreover, the *unfolding tree of* v, $\mathbf{T}_v = \lim_{d \to \infty} \mathbf{T}_v^d$, is obtained by merging all unfolding trees $\mathbf{T}_v^d$ for any d. □

Note that, since a GNN adopts a local computation framework, its knowledge about the graph is updated step by step; every time Eq. (1) is applied. Actually, at the first step, $k = 0$, the feature vectors $\mathbf{h}_v^0$ depends only on the local label. Then, at step k, the GNN updates the feature vector $\mathbf{h}_v^k$ using the neighbor data, with the node feature vector that depends on the k-distant neighborhood of v. Thus, intuitively, the unfolding tree $\mathbf{T}_v^k$ describes the information that is theoretically available to the GNN at node v and step k. Such an observation has been used in Scarselli (2009a) to study the expressive power of the OGNN model and will be used also in this paper for the same purpose.

In this context, two questions have been studied.

(1) Can GNNs compute and store into the node features a coding of the unfolding trees, namely can GNNs store all the theoretically available information?
(2) Since unfolding trees are different from the input graphs, how does this affect the GNN expressive power?

Regarding the first question, it has been shown that indeed both OGNNs and modern GNNs can compute and store in the node features a coding of the unfolding trees, provided that the appropriate network architectures are used in COMBINE$^{(k)}$ and AGGREGATE$^{(k)}$ (Sato 2020; Scarselli 2009a; Xu et al. 2018). Regarding question (2), we can easily argue that if two nodes have the same unfolding tree, then GNNs produce the same encoding on those nodes. Such a fact highlights an evident limitation of the expressive power of GNNs. The unfolding equivalence is a formal tool designed to capture such a limit: it is an equivalence relation that brings together nodes with the same unfolding tree, namely it groups nodes that cannot be distinguished by GNNs.

Definition 3.2 Two nodes u, v are said to be *unfolding equivalent* $u \backsim_{ue} v$, if $\mathbf{T}_u = \mathbf{T}_v$. Analogously, two graphs $\mathbf{G}_1, \mathbf{G}_2$

[2] Unfolding trees are also referred to as *computational graphs* (Garg et al. 2020) or *search trees* (Sato 2020; Xu et al. 2018).

are said to be *unfolding equivalent* $\mathbf{G}_1 \backsim_{ue} \mathbf{G}_2$, if there exists a bijection between the nodes of the graphs that respects the partition induced by the unfolding equivalence on the nodes .[3] □

Since GNNs have to fulfill the unfolding equivalence, also the functions on graphs that they can realize share this limit. In our results on the approximation capability of GNNs, our focus is on functions that preserve the unfolding equivalence. Those functions are general enough except that they produce the same output on equivalent nodes.

3.4 The color refinement algorithm and the Weisfeiler–Lehman test

The *first-order Weisfeiler–Lehman test* (*1-WL test* in short) (Lehman and Weisfeiler 1968) is a method to test whether two graphs are isomorphic, based on a graph coloring algorithm, called *color refinement*. The coloring algorithm is applied in parallel on the two graphs. Each node keeps a state (or color) that gets refined in each iteration by aggregating information from its neighbors' state. The refinement stabilizes after a few iterations and it outputs a representation of the graph. Two graphs with different representations, i.e., with a different number of nodes for each color, are not isomorphic. Conversely, if the numbers match, then the graphs are *possibly* isomorphic. Note that the test is not conclusive in the case of a positive answer, as the graphs may still be non-isomorphic. Actually, the algorithm just provides an approximate solution to the problem of graph isomorphism.

There exist different versions of the coloring algorithm: in this paper, we adopt a coloring scheme in which also the node labels are considered. Since GNNs process both the structure and the labels of the graphs, it is useful to consider both these sources of information, to analyze the GNN expressive power. Such an approach has been used, for example, in Sato (2020). More precisely, the coloring is carried out by an iterative algorithm which, at each iteration, computes a *node coloring* $c_l^{(t)} \in \Sigma$, being Σ a subset of values representing the colors. The node colors are initialized on the basis of the node features, and then, they are updated using the coloring from the previous iteration. The algorithm is sketched in the following:

1. At iteration 0, we set

   $$c_v^{(0)} = \mathrm{HASH}_0(\ell_v),$$

   where $\mathrm{HASH}_0 : \mathbb{R}^q \to \Sigma$ is a function that bijectively encodes real features using colors. In case of unattributed graphs, we assume $q = 1$ and $\ell_v = 1$, $\forall v \in V$, $\forall \mathbf{G} = (\mathbf{V}, \mathbf{E}) \in \mathcal{G}$.
2. For any iteration $t > 0$, we set

   $$c_v^{(t)} = \mathrm{HASH}(c_v^{(t-1)}, \{\{c_n^{(t-1)} | n \in ne[v]\}\}),$$

   where $\mathrm{HASH} : \Sigma \times \Sigma^* \to \Sigma$ is a function that bijectively maps the input pairs to a unique value in Σ. The notation $\{\{\cdot\}\}$ represents *multisets*, which can be formulated, in our setting, without loss of generality, as ordered sequences of elements in Σ, i.e., they belong to $\Sigma^* = \bigcup_{n \geq 1} \Sigma^n$. Moreover, we assume that the same HASH function is used for all the iterations[4].

To compare two graphs $\mathbf{G}' = (\mathbf{V}', \mathbf{E}')$, $\mathbf{G}'' = (\mathbf{V}'', \mathbf{E}'')$, the coloring refinement is applied in parallel on $\mathbf{G}'$, $\mathbf{G}'$, and, at each step, the color profiles generated on each graph are compared, namely, $\{\{c_n^{(t)} | n \in \mathbf{V}'\}\} = \{\{c_m^{(t)} | m \in \mathbf{V}''\}\}$ is evaluated. If, at any iteration, the colors of the graphs are different, then the 1-WL test fails and we can conclude that the graphs are not isomorphic; otherwise, the test succeeds. The 1-WL test allows to distinguish most non-isomorphic graphs, but may succeed on some rare examples.

In this paper, we use the color refinement also to compare nodes. Thus, given two nodes u, v, which in the most general case can belong to different graphs, we compare their colors at each iteration, i.e., $c_u^t = c_v^t$. If, at any iteration, the node colors are different, then the 1-WL node test fails; otherwise, it succeeds. Notice that the color of a node n at iteration t depends on the sub-graph $\mathbf{G}_n^t$, defined by the t-hop neighborhood of n. Thus, intuitively, the 1-WL node test allows to check the isomorphism of the neighborhoods of two nodes, $\mathbf{G}_u^t \backsim \mathbf{G}_v^t$.

By the mentioned algorithms, we can easily produce a definition of WL-equivalence for graphs and nodes.

Definition 3.3 (*WL-equivalence*) Two graphs, $\mathbf{G}' = (\mathbf{V}', \mathbf{E}')$ and $\mathbf{G}'' = (\mathbf{V}'', \mathbf{E}'')$, are said to be **WL-equivalent**, if they have the same multisets of colors at each iteration of the color refinement algorithm, i.e., $\{\{c_n^{(t)} | n \in \mathbf{V}'\}\} = \{\{c_m^{(t)} | m \in \mathbf{V}''\}\}$ for any t. Analogously, two nodes, u and v, are said to be **WL-equivalent**, $u \backsim_{WL} v$, if they have the same colors at each step of the color refinement algorithm, i.e., $c_u^{(t)} = c_v^{(t)}$ for any t. □

[3] For the sake of simplicity, and with notation overloading, we adopt the same symbol $\backsim_{ue}$ both for the equivalence between graphs and the equivalence between nodes.

[4] In Kiefer (2020), it is assumed that the HASH functions are different at each step, so that the algorithm can reuse the same finite set of colors, e.g., denoted by the integer numbers 1 to r, where r is the number of nodes in the graph. This can be achieved by bijectively re-mapping the colors after each refinement step. The two algorithms are equivalent w.r.t. the goal of isomorphism testing. Here, we prefer to assume that a unique HASH function is adopted, because such an assumption will simplify our discussion about the properties of the algorithm.

It is interesting to observe that the color refinement procedure must be iterated until a difference in colors is detected between the compared items, either graphs or nodes, or until a maximum number of iterations is reached. It is well known that the color refinement of the common Weisfeiler–Lehman test, defined for graph comparison, can be halted when the node partition defined by colors become stable: if the two graphs share the colors when the stability is reached, then the equality will last forever. More precisely, let $\pi_t(\mathbf{G})$ be the partition of the nodes of $\mathbf{G}$ constructed by collecting in the same class the nodes that have the same color at iteration t. It is not difficult to prove that the partitions become finer at each iteration, $\pi_{t-1}(\mathbf{G}) \succeq \pi_t(\mathbf{G})$, and that there exists an iteration T at which they become stable, $\pi_{T-1}(\mathbf{G}) \equiv \pi_T(\mathbf{G})$, Moreover, it can be proved that $r-1$, where r is the number of nodes in $\mathbf{G}$, is both an upper bound and a lower bound for the number T of iterations required to reach the stability (Kiefer and McKay 2020).

Note that the stability of the node partition does not imply that the colors do not change. Actually, if the colors are not reused, as in our definition, except in the case where the graph is free of connections, new colors appear at each iteration. Intuitively, this happens, because the use, at a node u, of a new color, which has not been considered in the past, causes the algorithm to create new colors for the neighbors of u as well: thus, new colors will be generated forever. This observation can be used to explain why the upper bound on the iterations of the color refining procedure is different in the case of node or graph equivalence. We will see that we must wait for $2r-1$ iterations before halting the procedure in the former case, whereas, as mentioned above, $r-1$ iterations are sufficient in the latter.

4 Main results

In this section, the main results of the paper are presented and discussed. For ease of reading, the proofs of the theorems are given in the Appendix.

4.1 Unfolding and Weisfeiler–Lehman equivalence

The first proposed result regards the relationship between the unfolding and the Weisfeiler–Lehman equivalence. The following two theorems clarify that the two equivalence relations produce the same partitions of nodes and graphs. Moreover, the correspondence exists also between the intermediate equivalences defined by, respectively, the colors at each iteration of the WL algorithm and the unfolding trees having a corresponding depth. Formally, let us denote by $\backsim_{ue_t}$ the unfolding equivalences, at depth t, between nodes and graphs that are defined as in 3.2 but considering unfolding trees of depth t in place of infinite trees. Similarly, let us denote by $\backsim_{WL_t}$ the WL-equivalences, at iteration t, that are defined as in 3.3, where only the colors of the refinement procedure up to the tth iteration are considered.

Theorem 4.1 *Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a labeled graph. Then, for each $u, v \in \mathbf{V}$, $u \backsim_{ue} v$ holds if and only if $u \backsim_{WL} v$ holds. Moreover, for each integer $t \geq 0$, $u \backsim_{ue_t} v$ if and only if $u \backsim_{WL_t} v$.* □

Theorem 4.2 *Let $\mathbf{G}_1, \mathbf{G}_2$ be two graphs. Then, $\mathbf{G}_1 \backsim_{ue} \mathbf{G}_2$ if and only if $\mathbf{G}_1 \backsim_{WL} \mathbf{G}_2$. Moreover, for each integer $t \geq 0$, $\mathbf{G}_1 \backsim_{ue_t} \mathbf{G}_2$ if and only if $\mathbf{G}_1 \backsim_{WL_t} \mathbf{G}_2$.* □

Both the unfolding equivalence and the WL-equivalence have been described using a recursive definition local to nodes. Figure 2 shows an example in which the unfolding trees and the colors of two nodes are iteratively computed: in the example, the colors of the nodes become different when also the unfolding trees become different.

Indeed, the existence of a relationship between the equivalences appears to be a natural consequence of their definition. In fact, it is sometimes assumed in the literature (f.i., in Maron et al. 2018) that the two tools can be used interchangeably but, as far as we know, there is no formal demonstration of their effective equivalence. More precisely, in Krebs and Verbitsky (2015), Angluin (1980), and Dell et al. (2018), it has been proved that the 1-WL test and unfolding trees produce the same profile on graph without attributes. Therefore, Theorem 4.2 is just an extension of those results to the case of graph with attributes. On the other hand, Theorem 4.1, focused on nodes, is completely novel.

Theorems 4.1 and 4.2 are interesting, since they formally confirm that the two equivalences are exactly interchangeable and can be used together to study GNNs. While the Weisfeiler–Lehman test has been often adopted to analyze the expressive power of GNNs in terms of their capability of recognizing different graphs, the unfolding equivalence and, more precisely, unfolding trees, can provide a tool to understand the information that a GNN can use at each node to implement its function.

For example, it is well known that GNNs cannot distinguish regular graphs where nodes have the same features (see, e.g., Sato 2020). Of course, in this case, a GNN is not able to distinguish any node, since all the unfolding trees are equal (see Fig. 3a). On the one hand, when a target node has different features with respect to the others, also the unfolding trees incorporate such a difference and the nodes at different distances from this target node belong to different equivalence classes (see Fig. 3b). On the other hand, if all the labels are different, then each node belongs to a different class, since all unfolding trees are different (see Fig. 3c).

We observe that, in principle, by adding random features to the node labels, we could make all the nodes distinguishable and improve the GNN expressive power. This fact was

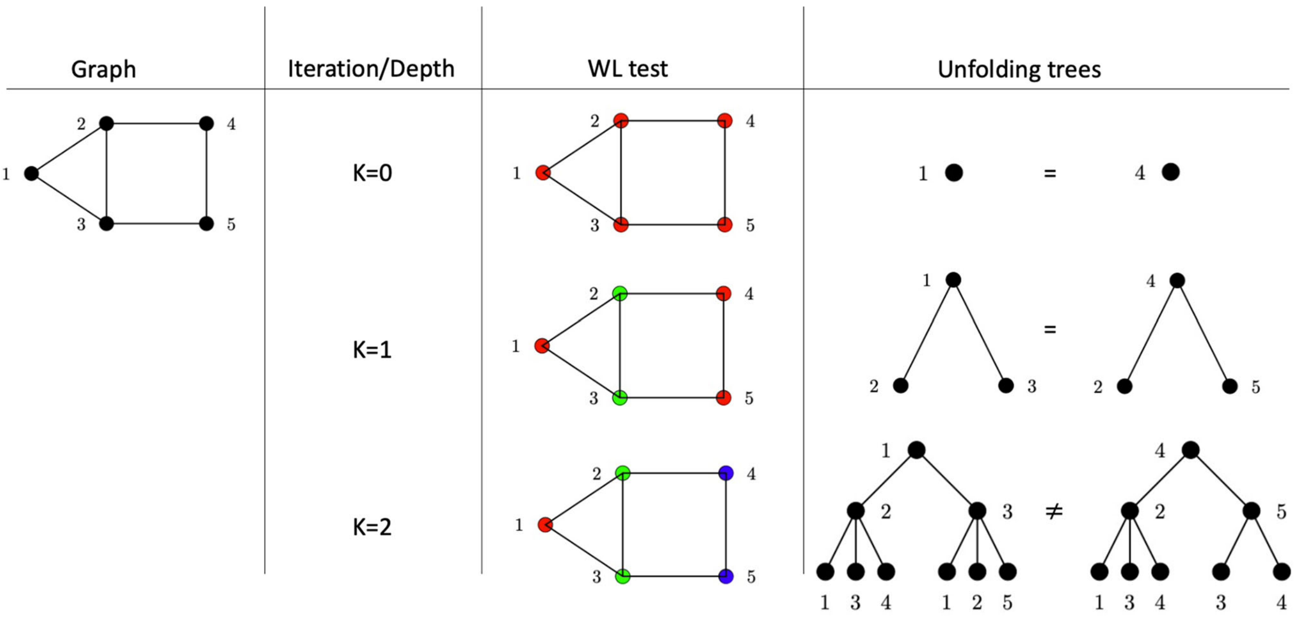
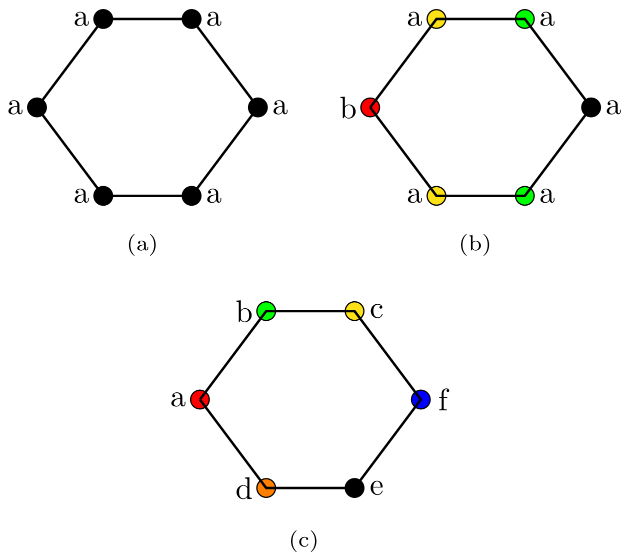


Fig. 2 A graphical representation of the relationship between the color refinement and the unfolding equivalence, applied on nodes 1 and 4 of the given graph



(a)

(b)

(c)

Fig. 3 **a** A regular graph where all nodes have the same features. All unfolding trees are equal. **b** The equivalence classes when only one node has different features. **c** The equivalence classes when all nodes has different features

already mentioned for OGNNs (Scarselli 2009a) and has been recently observed also for modern GNN models (Sato et al. 2021). Obviously, this is true only in theory, as the introduction of random features usually produces overfitting. However, some particular tasks exist where random features do not cause any overfitting, for example, if these features are not related to the node content (see Scarselli 2009a, Section IV.A), while, in other cases, it is the particular model which is able to efficiently use random labels (Sato 2020).

A further important argument of our analysis regards how much deep must be unfolding trees, i.e., how many iterations of color refinement are needed, to make the equivalence stable. Actually, Theorems 4.1 and 4.2 suggest that the unfolding and Weisfeiler–Lehman equivalences remain paired up to any depth/iteration t. Those equivalences naturally become finer and finer as the iterations proceed, i.e., $\backsim_{ue_{t-1}} \succ \backsim_{ue_t}$ and $\backsim_{WL_{t-1}} \succ \backsim_{WL_t}$, until T, when they become stable and equal to the corresponding infinite equivalences, namely $\backsim_{ue_{T-1}} \equiv \backsim_{ue_T} \equiv \backsim_{ue}$ and $\backsim_{WL_{T-1}} \equiv \backsim_{WL_T} \equiv \backsim_{WL}$. As already mentioned in Sect. 3, according to the literature (Kiefer and McKay 2020), it is known that, for the WL-equivalence on graphs, $r-1$ is both an upper and lower bound on T, where r is the maximum number of nodes in the graphs. The following theorem, which takes inspiration from the results in Kiefer and McKay (2020) about covering trees, shows that, for equivalences *on nodes*, the bounds are different and we must wait up to $2r-1$ iterations, i.e., trees of depth of $2r-1$, until the equivalences become stable.

Theorem 4.3 *The following statements hold for graphs with at most r nodes.*

1. *Let* $\mathbf{G}$ *and* $\mathbf{H}$ *be connected graphs and* x, y *be nodes of* $\mathbf{G}$ *and* $\mathbf{H}$, *respectively. The infinite unfolding trees* $\mathbf{T}_x$, $\mathbf{T}_y$ *are equal if and only if they are equal up to depth* $2r-1$, *i.e.,* $\mathbf{T}_x = \mathbf{T}_y$ *iff* $\mathbf{T}_x^{2r-1} = \mathbf{T}_y^{2r-1}$.

2. *For any r, there exist two graphs* $\mathbf{G}$ *and* $\mathbf{H}$ *with nodes x, y, respectively, such that the infinite unfolding trees* $\mathbf{T}_x$, $\mathbf{T}_y$ *are different, but they are equal up to depth $2r - 16\sqrt{r}$, i.e.,* $\mathbf{T}_x \neq \mathbf{T}_y$ *and* $\mathbf{T}_x^i = \mathbf{T}_y^i$ *for $i \leq 2r - 16\sqrt{r}$.*

□

To get an intuitive explanation of the reason why the bounds are different for graph and node equivalences, let us consider the case of two graphs $\mathbf{G}$ and $\mathbf{H}$ that are not equivalent, i.e., $\mathbf{G} \not\equiv_{WL} \mathbf{H}$ holds. Moreover, let us assume that the parallel application of the refinement algorithm detects the difference in colors at iteration $\bar{T}$, namely $\mathbf{G} \not\equiv_{WL_{\bar{T}}} \mathbf{H}$, for example, because a new color is generated for graph $\mathbf{G}$ that is not present in $\mathbf{H}$. At this iteration, the WL algorithm is halted, since we detected at least a node u in $\mathbf{G}$ that is different from all the nodes in $\mathbf{H}$. Conversely, if we continue the color refinement, the new color of u will generate other new colors, which are not present in $\mathbf{H}$, also for the neighbors of u. After at most r iterations, the difference spreads throughout the graph, so that, finally, all the nodes in $\mathbf{G}$ are different from those in $\mathbf{H}$. This is intuitively correct, since all the nodes in $\mathbf{G}$ are connected to a node that does not exist in $\mathbf{H}$. Therefore, we can observe that, while the first difference between the nodes of the two graphs arises after $r - 1$ iterations, the diffusion of such information to all the nodes takes additional r steps. Obviously, a similar conclusion can be derived also considering the unfolding equivalence and the depths of the unfolding trees.

An example that illustrates this situation is depicted in Fig. 4. The two graphs in (a) and (b) have been proposed in Krebs and Verbitsky (2015) and satisfy the lower bound of point 2 of Theorem 4.3. In the example, we assume that all the nodes have the same attributes, even if, for the sake of clarity, they are displayed with different symbols in terms of their“role”in the coloring scheme. The graphs in (a) and (b) are constructed using copies of the sub-graph modules in (c), (d) and (e), which are merged in a sequence; (a) and (b) are equal except at the top: in (a), at the end of the sequence, there is a copy of (d), while in (b), there is a copy of (e). The interesting case happens when the sequence is long enough, so that $2r - 16\sqrt{r} > r$ holds. In this case, we have the following situation: graphs (a) and (b) are distinguishable by the 1-WL test in less than r steps; nevertheless, a number of steps $t > 2r - 16\sqrt{r} > r$ is needed to distinguish the nodes u and v. Thus, intuitively, color refinement can recognize that (a) and (b) are not isomorphic, but the detection of the difference occurs only when the information about the asymmetry—which is on one side of the sequence—arrives to the other side of the sequence, where the different modules have been placed. After that, the different modules have been detected and the information on their difference is propagated to the rest of the graphs in a number of iterations proportional to the length of the sequences to arrive back to nodes u and v.

To formally link the concept of unfolding trees to the computational capability of GNNs, let us now recall the definition of unfolding equivalence.

Definition 4.4 A function $f : \mathcal{D} \rightarrow \mathbb{R}^o$ is said to preserve the unfolding equivalence on $\mathcal{D}$ if $v \sim u$ implies $f(\mathbf{G}, v) = f(\mathbf{G}, u)$. □

The class of functions that preserve the unfolding equivalence on $\mathcal{D}$ will be denoted with $\mathcal{F}(\mathcal{D})$. A characterization of $\mathcal{F}(\mathcal{D})$ is given by the following result.

Theorem 4.5 (Functions of unfolding trees) *A function f belongs to $\mathcal{F}(\mathcal{D})$ if and only if there exists a function κ, defined on trees, such that $f(\mathbf{G}, v) = \kappa(\mathbf{T}_v^{2r-1})$, for any node $v \in \mathcal{D}$.* □

A short, formal proof can be found in Appendix A.

Theorem 4.5 represents an improvement of the results reported in Scarselli and Chung Tsoi (1998); our contribution here is to show that, considering the unfolding tree down to the depth $2r - 1$, we can provide the complete information on a graph to a function f belonging to $\mathcal{F}(\mathcal{D})$.

Note that Theorem 4.5 suggests not only that the functions that compute the output on a node using unfolding trees preserve the unfolding equivalence, but also that the converse holds, namely all the functions that preserve the unfolding equivalence can be computed as functions of the unfolding trees. Since GNNs can implement only functions of the unfolding trees, we may expect that there is a tight relationship between what GNNs can do and the class $\mathcal{F}(\mathcal{D})$. Actually, in Scarselli (2009a), it has been shown that the OGNN model can approximate in probability, up to any degree of precision, any function in $\mathcal{F}(\mathcal{D})$ and a similar result will be derived for modern GNNs in this manuscript.

4.2 Approximation capability

The above discussion is about what GNNs cannot do, since we have proved that they are unable to distinguish nodes that originate equal unfolding trees. Another obvious limit is that, at each node v, a GNN considers only the part of the graph that is reachable from v and cannot implement any function depending on the information inaccessible from that node. For this reason, for simplicity, we have decided to consider only connected graphs. In this section, we pose our attention on two further questions that are related to each other, namely which functions can be approximated by GNNs and if there are any limitations other than that relating to the unfolding equivalence.

To address these issues, we consider the class of functions that preserve the unfolding equivalence (see Definition 4.4).

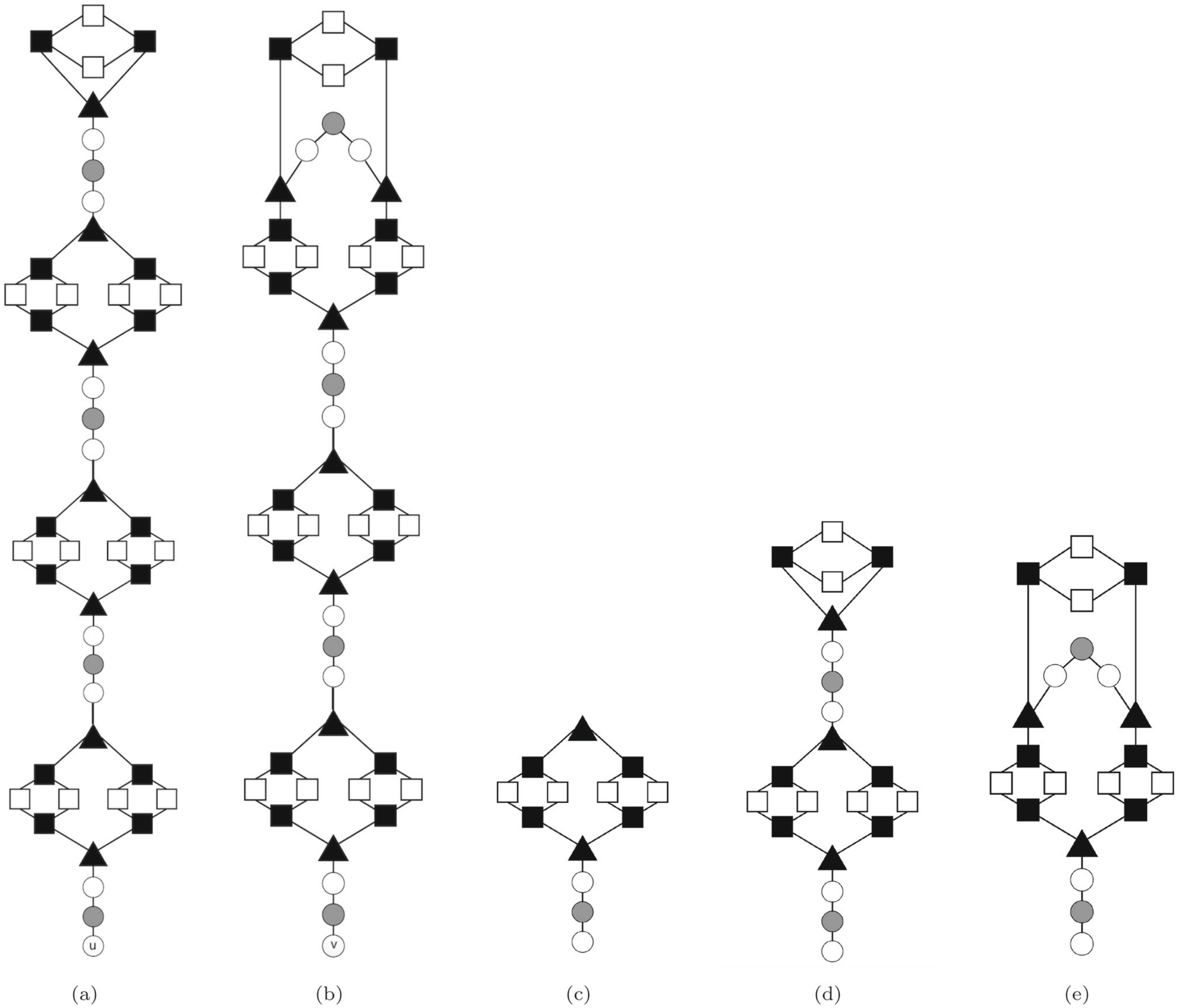


Fig. 4 In **a** and **b**, two graphs **G**, **H** are depicted that satisfy the lower bound of point 2 of of Theorem 4.3. We assume that all the nodes have the same attributes even if they are displayed with different symbols in terms of their "role" in the coloring scheme. Graphs in **a** and **b** are constructed by aggregating in a sequence two copies of the same subgraph **c**; then, module **d** is added at the top of graph **a**, while module **e** is added at the top of graph **b**. It is worth noting that **a** and **b** *do not* satisfy the relation $2r - 16\sqrt{r} > r$; nevertheless, adding multiple times module **c** to the tail of both **a** and **b**, we can find two graphs satisfying the requested relation

The following theorem proves that GNNs can approximate in probability, up to any precision, any function of this class, which means that GNNs are a sort of universal approximators on graphs, modulo the limitations due to the unfolding equivalence.

Theorem 4.6 (Approximation by GNNs) *Let $\mathcal{D}$ be a domain containing connected graphs with at most r nodes. For any measurable function $\tau \in \mathcal{F}(\mathcal{D})$ preserving the unfolding equivalence, any norm $\|\cdot\|$ on $\mathbb{R}$, and any probability measure P on $\mathcal{D}$, there exists a GNN defined by the continuously differentiable functions COMBINE$^{(k)}$, AGGREGATE$^{(k)}$, $\forall k \leq r-1$, and by the function READOUT, with feature dimension $m = 1$ (i.e, $h_v^k \in \mathbb{R}$), such that the function φ (realized by the GNN) computed after $2r - 1$ steps satisfies the condition*

$$P(\|\tau(\mathbf{G}, v) - \varphi(\mathbf{G}, v)\| \leq \varepsilon) \geq 1 - \lambda$$

for any reals ϵ, λ, where $\epsilon > 0$, $0 < \lambda < 1$. □

Theorem 4.6 intuitively states that, given a function τ, there exists a GNN that can approximate it. COMBINE$^{(k)}$ and AGGREGATE$^{(k)}$ can be any continuously differentiable function, while no assumptions are made on READOUT. This situation does not correspond to practical cases, where the GNN adopts particular architectures and those functions are realized by neural networks or, more generally, parametric models—for example made of layers of sums, max, average, etc. Therefore, it is of fundamental interest to clarify whether the theorem still holds when the components $\text{COMBINE}^{(k)}$, $\text{AGGREGATE}^{(k)}$ and READOUT are parametric models.

Let us now study the case when the employed components are sufficiently general to be able to approximate any function. We call $\mathcal{Q}$ this class of networks, which corresponds to GNN models with universal components. To simplify our discussion, we introduce the transition function $f^{(k)}$ to indicate the stacking of the $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$, that is

$$f^{(k)}(\mathbf{h}_v^{k-1}, \{\{\mathbf{h}_u^{k-1},\ u \in ne[v]\}\}) = \text{COMBINE}^{(k)}\big(\mathbf{h}_v^{k-1},\\ \text{AGGREGATE}^{(k)}\big(\{\{\mathbf{h}_u^{k-1},\ u \in ne[v]\}\}\big)\big).$$

Then, we can formally define the class $\mathcal{Q}$.

Definition 4.7 A class $\mathcal{Q}$ of GNN models is said to have *universal components* if, for any any $\epsilon > 0$ and any continuous target functions $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\overline{\text{READOUT}}$, there exists a GNN belonging to $\mathcal{Q}$, with functions $\text{COMBINE}_w^{(k)}$, $\text{AGGREGATE}_w^{(k)}$, READOUT_w and parameters w, such that

$$\left\| \bar{f}^{(k)}(\mathbf{h}, \{\mathbf{h}_1, \ldots, \mathbf{h}_s\}) - f_w^{(k)}(\mathbf{h}, \{\mathbf{h}_1, \ldots, \mathbf{h}_s\}) \right\|_\infty \leq \epsilon$$

$$\left\| \overline{\text{READOUT}}(\mathbf{q}) - \text{READOUT}_w(\mathbf{q}) \right\|_\infty \leq \epsilon$$

holds, for any input values $\mathbf{h}, \mathbf{h}_1, \ldots, \mathbf{h}_s, \mathbf{q}$. Here, the transition functions $\bar{f}^{(k)}$ and $f_w^{(k)}$ are defined using the target functions $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, and the GNN functions $\text{COMBINE}_w^{(k)}$, $\text{AGGREGATE}_w^{(k)}$, respectively, and $\| \cdot \|_\infty$ is the infinity norm. □

The following result shows that Theorem 4.6 still holds even for GNNs with universal components.

Theorem 4.8 (Approximation by neural networks) *Let us assume that the hypotheses of Theorem 4.6 are fulfilled and $\mathcal{Q}$ is a class of GNNs with universal components. Then, there exist a parameter set w and some functions $\text{COMBINE}_w^{(k)}$, $\text{AGGREGATE}_w^{(k)}$, READOUT_w, implemented by neural networks in $\mathcal{Q}$, such that the thesis of Theorem 4.6 holds.* □

The proof of Theorem 4.8 is included in the Appendix. However, some related topics are discussed below, to better understand some properties of GNNs.

- In the proof of Theorem 4.6, we first define an encoding function $\triangledown$ (see the Appendix) that maps trees to real numbers. The functions $\text{COMBINE}^{(k)}$ and $\text{AGGREGATE}^{(k)}$ are designed, so that, at each step, the node feature vector approximates a coding of the unfolding function $\mathbf{h}_v^k = \triangledown(\mathbf{T}_v^k)$. The function READOUT decodes the unfolding and produces the desired outputs.
- In the proof of Theorem 4.8, it is shown that Theorem 4.6 still holds even when the transition and READOUT functions are approximated. Thus, we can use any parametric model to implement those functions. We can expect that, also for the GNNs of Theorem 4.8, the transition function stores into the feature vector an approximate coding of the unfolding tree, while READOUT decodes such a coding and gives the desired outputs. Obviously, in a practical case, a GNN can store only useful information, required to produce the output, and not just all the informative content of the unfolding trees.

The following remarks may further help to understand our results.

- *GNNs with universal components*. Intuitively, the universality condition means that the architectures used to implement $f_w^{(k)}$ and READOUT_w must be sufficiently general to be able to approximate any possible target function. From the theory of standard neural networks, those architectures must have at least two layers (one hidden and one output layer) (Scarselli and Chung Tsoi 1998). Such a conclusion is similar to the one reported in Xu et al. (2018), where a related result is described and where it is suggested that, to be able to implement the 1-WL test, the GNN must use a two-layer transition function. Indeed, in this way, the GNN can implement an injective encoding of the input graph into the node features. Nonetheless, the proposed result is slightly different with respect to the one reported in Xu et al. (2018) as, in theory, the encoding may fail to be injective, provided that the approximation remains sufficiently good in probability. However, the conclusion about the architecture still holds.

  GNNs with transition functions $f_w^{(k)}$ exploiting two-layer architectures include Graph Isomorphism Networks (GINs) (Xu et al. 2018), which were claimed to realize an injective encoding. Similarly, also the OGNN model, for which a result similar to Theorem 4.6 was proved, adopts a two-layer architecture for the transition function: in this case, $\text{AGGREGATE}_w^{(k)}$ consists of a MultiLayer Perceptron (MLP) with a hidden layer and $\text{COMBINE}_w^{(k)}$ was implemented by a sum. Similar results have been devised also in Azizian and Lelarge (2020), where a different version of the $\text{COMBINE}_w^{(k)}$ function has been modeled as a sum of MLPs.
- READOUT *universality.* The condition on the universality of the READOUT function can be relaxed, provided that a higher dimension for the feature vector is used, namely $m > 1$. READOUT_w can indeed cooperate with

the transition function to produce the output. In the limit case, the output can be completely prepared by the transition function and stored in some components of $\mathbf{h}_v^K$ so that READOUT$_w$ is just a projection function.

- *GNN architectures that are not universal approximators*. Most of GNN models, e.g., Graph Convolutional Neural Networks, GraphSAGE, and so on, use a single layer architecture to implement the transition function. Thus, even if they do employ universal components, such as those specified by Definition 4.7, they have a limited computational power with respect to two-layer architectures and this is supported by theoretical results. In Xu et al. (2018), Lemma 7, it is shown that, if the transition function is made up by a single layer with ReLU activation functions, the encoding function cannot be injective. A similar result was obtained for linear recursive neural networks [5] in Bianchini and Gori (2001). However, in general, it is not correct to assert that GNNs with single layer transition functions cannot be universal approximators for functions on graphs, as this property depends on the used GNN model and on other architectural/training details. For example, a GNN model with a single layer transition component can use several iterations of Eq. (1) to emulate a GNN with a deeper transition component. In the former model, the node features emulate the transition network hidden layers and COMBINE must contain a self-loop, namely must have access to the previous features of each node.
- *Feature dimension.* Surprisingly, Theorems 4.6 and 4.8 suggest that a feature vector of dimension $m = 1$ is enough to establish the universal approximation capability of GNNs. It is obvious, however, that the dimension of the feature vector plays an important role in determining the complexity of the coding function for a given domain. We expect that the larger the dimension, the smaller the complexity of the coding. This complexity, in turn, affects the complexity of the transition function, the difficulty in learning such a function, the number of patterns required for training the GNN, and so on.
- *Number of steps.* Theorems 4.6 and 4.8 suggest that $2r-1$ steps are enough to approximate any function. Such a result is a consequence of Theorems 4.3 and 4.5. Intuitively, this bound can be explained reusing the discussion on Theorem 4.3. A GNN can employ up to $r-1$ iterations/layers to diffuse all the information from one node to any other node with the message passing mechanism. After $r-1$ iterations, the information stored in a node provides a sort of signature for that node, which may allow to distinguish some nodes from others. Yet, such a signature is not complete, since the first time a node "communicate" with another has no information about itself. Adding r iterations/layers allows nodes to communicate again and exchange their current signatures to produce more accurate signatures. It is worth noting that this reasoning provides also an intuitive explanation about why graph regression/classification tasks differ from node tasks. In graph tasks, the GNN uses a READOUT function that aggregates the features of all the nodes in the graph, and possibly can do the work required by the second diffusion phase. In node tasks, READOUT operates only on a single node, so that the second diffusion phase is mandatory.
- The same COMBINE and AGGREGATE can be used for all the layers. Even if, for clarity, in our theoretical analysis, we focus on the GNN model that is the most used and exploits different functions in each layer k, our proofs do not exploit such a characteristic. Therefore, all the results hold also for those GNNs—sometimes called *recursive*—using the same COMBINE and AGGREGATE functions on each layer.

Note that, throughout the manuscript, we have used the idea that the unfolding tree represents the information available to a GNN to compute its output, and we have mentioned that a similar approach has been applied by other authors as well. From a formal point of view, Theorem 4.6 defines a method by which a GNN can actually encode an unfolding tree into the node features, so that it has been proved that all the information collected into the unfolding trees can be used by GNNs. However, also the reverse implication holds true, that is, a GNN cannot encode more information into features than that contained into the unfolding trees. Indeed, this is a consequence of the fact that GNNs have no greater discriminatory capability than the 1-WL test (see Morris 2019, Theorem 1). Therefore, the unfolding trees totally collect the information used by a GNN.

Finally, the following corollary provides an alternative way to describe the approximation ability of GNNs as a function of their unfolding trees.

Corollary 4.9 *The class of functions implemented by a GNN with universal components is dense in probability in the $\mathcal{F}(\mathcal{D})$ class of functions that preserve the unfolding equivalence in the domain $\mathcal{D}$ of connected graphs.* □

5 Experimental validation

In this section, we support our theoretical findings with a set of experiments. For this purpose, we show that a GNN can approximate a function $F_{WL} : \mathcal{G} \to \mathbb{N}$ that models the 1-WL test. Indeed, the function F_{WL} assigns to each graph a target label that represents the class of equivalence of the

[5] Recursive neural networks (Sperduti and Starita 1997) are the ancestors of GNNs and assume that the input graph is acyclic.

1-WL. For simplicity, we only focus on the ability of the GNN to approximate this function, so that only the training performance is considered, i.e., we do not investigate its generalization capability over a test set. Since the 1-WL test provides the finest partition of graphs reachable by a GNN, the mentioned task experimentally establishes the expressive power of GNNs.

Dataset The graph datasets used for the experiments are derived from the QM9 molecules dataset (Ruddigkeit et al. 2012; Ramakrishnan et al. 2014). Specifically, the subsets of molecules that compose our dataset are selected as follows:

- Homogeneous features are assigned to all nodes of all graphs in QM9, as we are interested in evaluating the approximation ability of GNNs based only the graph topology;
- The 1-WL test is run all over the entire QM9 dataset for $k = 4$ iterations, and for each graph, the target is the corresponding 1-WL output, represented as a natural number;
- We select the color classes containing more than T graphs, where T is a fixed threshold.

For training purposes, the targets are normalized between 0 and 1 and spaced uniformly in the range [0, 1]. Therefore, the distance between each class label is $d = \frac{1}{\text{num\_classes-1}}$. A graph G with target y_G will be said to be correctly classified if, given $\mathsf{out} = \text{GNN}(G)$, we have $|\mathsf{out} - y_G| < d/2$.

Experimental setup The GNN used in the experiments is the Graph Isomorphism Network (GIN) (Xu et al. 2018). A GIN computes

$$\mathbf{h}_v^{(t)} = \text{MLP}\left((1+\epsilon)\mathbf{h}_v^{(t-1)} + \sum_{u \in ne_v} \mathbf{h}_u^{(t-1)}\right), \quad (4)$$

where the attention parameter ϵ is either a trainable parameter or a fixed scalar. In our setting, we fix $\epsilon = 0$. It has been proven that GINs can implement 1-WL test and produce a different representation for each graph that can be distinguished by 1-WL test (Xu et al. 2018). Thus, GINs, with an appropriate READOUT, can approximate any function on graphs preserving the unfolding equivalence. The MLP in a GIN layer has one hidden layer with h_{gin} neurons; the dimension of the GIN features is h_{gin} as well. The MLP in a GIN layer implements the hyperbolic tangent as activation function, and batch normalization. The GIN includes $n_{\max} = k$ layers, so as k is the number of iterations performed by 1-WL to generate the targets. After the last GIN layer, the READOUT function is implemented performing a global aggregation, after which a linear layer $W_{\mathsf{gin\_out}}$ of size $h_{\mathsf{gin}} \times 1$ is added; eventually, a sigmoid activation function is applied. The model is trained over 500 epochs using the Adam optimizer with an initial learning rate $\lambda = 10^{-3}$. We carried out the experiments as follows.

- In the first experimental setting, we evaluate the GNN performance for different values of the threshold T, which affects the cardinality of the training set and its 1-WL color classes. The values of the threshold T are taken in the integer interval [30, 45], the hidden layer of the MLP has dimension $h_{\mathsf{gin}} = 64$.
- In the second experimental setting, we evaluate the GNN expressive power varying both the number of neurons in the GIN MLP and the size of the hidden features, which, as specified above, are kept equal. In these experiments, the threshold is fixed as $T = 35$, and the hidden layer sizes h_{gin} are taken from the list [4, 8, 16, 32, 64].

Each experiment is statistically evaluated over 15 runs. The overall training is performed on an Intel(R) Core(TM) i7-9800X processor running at 3.80 GHz, using 31 GB of RAM and a GeForce GTX 1080 Ti GPU unit.

The code developed to run the experiments can be found at https://github.com/AleDinve/static-gnn.

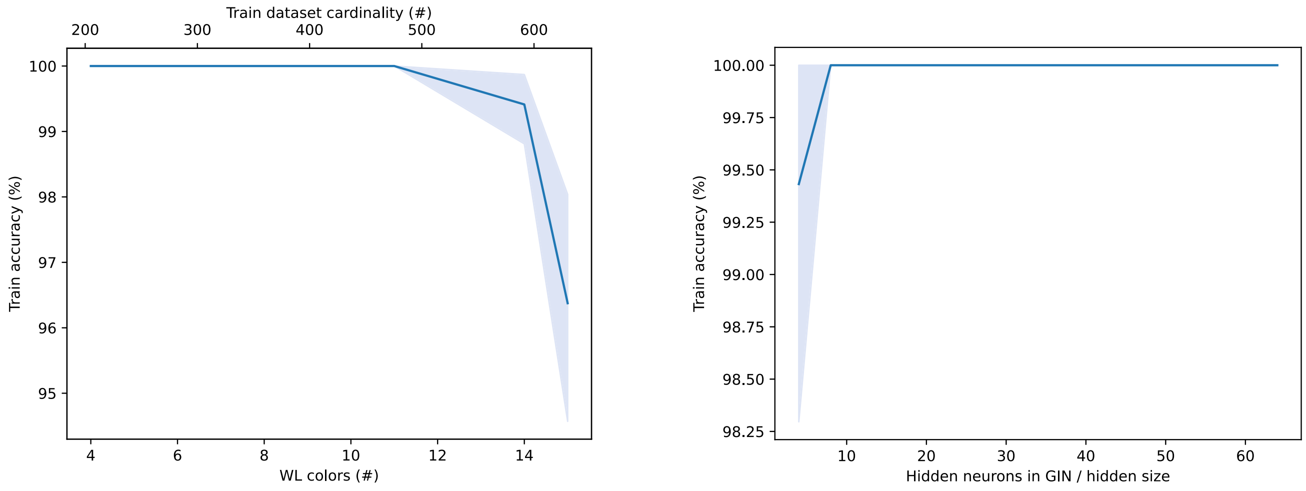
Results Our experimental results are summarized in Fig. 5. Figure 5a shows the evolution of the training accuracy for different numbers of WL colors; Fig. 5b displays the evolution of the training accuracy for increasing numbers of hidden neurons in the GIN MLP.

In both experiments, the average training accuracy is never less than 96%. Moreover, in at least one of the 15 runs per value, 100% training accuracy is reached. These results confirm the approximation power of GNNs equipped with a sufficiently general components.

6 Conclusion

In this paper, we have shown that GNNs can approximate, in probability, any function that preserves the unfolding equivalence (i.e., that passes the 1-WL test). Our proof improves on existing results both because it applies to node classification/regression tasks and because it is more general, since it holds for measurable functions. Moreover, using our theoretical framework, we have provided details on the GNN architectures that can reach a given approximation, including the number of iteration/layers, the state dimension, and the architecture of AGGREGATE$^{(k)}$, COMBINE$^{(k)}$ and READOUT networks.

Future developments may include further extensions of our results beyond the traditional 1-WL domain and covering GNN models not considered by the framework used in this paper. For instance, it would be interesting to characterize the class of *node-focused* functions learnable by a specific GNN model in terms of the isomorphism-wise test paradigm



(a)

(b)

Fig. 5 Training accuracy on subsampled QM9 datasets, increasing number of WL colors (**a**), and increasing hidden layer size (**b**). The solid line represents the average over 15 runs; the shaded area represents the confidence interval

on which it has been built (see Bodnar et al. 2021a, b for examples of GNNs built following isomorphism test mechanisms different from the 1-WL test). Moreover, the proposed results are mainly focused on the expressive power of GNNs, but GNNs with the same expressive power may differ for other fundamental properties, e.g., the computational and memory requirements and the generalization capability (that can be measured through well-established metrics, such as Rademacher complexity and VC-dimension, as pointed out in Sect. 2, or evaluated in terms of neurocognitive task learning (Brugiapaglia et al. 2020, 2022; D’Inverno et al. 2023)). Understanding how the architecture of AGGREGATE$^{(k)}$, COMBINE$^{(k)}$ and READOUT impacts on those properties is of fundamental importance for practical applications of GNNs.

Funding Open access funding provided by Università degli Studi di Siena within the CRUI-CARE Agreement. Giuseppe Alessio D’Inverno and Maria Lucia Sampoli are partially supported by INdAM GNCS group. Monica Bianchini and Maria Lucia Sampoli are partially supported by the PNRR Project “THE—Tuscany Health Ecosystem”, CUP: B83C22003920001.

Data availability Synthetic data have been downloaded from the Pytorch Geometric repo available at https://deepchemdata.s3-us-west-1.amazonaws.com/datasets/’’molnet_publish/qm9.zip.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Code availability Code has been made available in the GitHub repo https://github.com/AleDinve/static-gnn.

Appendix

Proof of Theorems 4.1 and 4.2

Since both unfolding equivalence and color equivalence have been described using a node-localized recursive definition, it is natural to investigate the possible connections between these two equivalence relations. Indeed, in the following, we show that they are equivalent on a domain of graphs with node features, i.e., that define the same relationship between nodes.

To prove Theorems 4.1 and 4.2, the following lemma is required.

Lemma A.1 *Let* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *be a graph and let* $u, v \in \mathbf{V}$, *with features* ℓ_u, ℓ_v. *Then,* $\forall t \in \mathbb{N}$

$$\mathbf{T}_u^t = \mathbf{T}_v^t \quad \textit{iff} \quad c_u^{(t)} = c_v^{(t)}, \tag{5}$$

where $c_u^{(t)}$ and $c_v^{(t)}$ represent the node coloring of u and v at time t, respectively.

Proof The proof is carried out by induction on t, which represents both the depth of the unfolding trees and the iteration step in the WL coloring.

For $t = 0$, $\mathbf{T}_u^0 = \text{Tree}(\ell_u) = \text{Tree}(\ell_v) = \mathbf{T}_v^0$ if and only if $\ell_u = \ell_v$ and $c_u^{(0)} = \text{HASH}_0(\ell_u) = \text{HASH}_0(\ell_v) = c_v^{(0)}$. Let us suppose that Eq. (5) holds for $t-1$, and prove that it holds also for t.

($\rightarrow$) Assuming that $\mathbf{T}_u^t = \mathbf{T}_v^t$, we have

$$\mathbf{T}_u^{t-1} = \mathbf{T}_v^{t-1} \tag{6}$$

and

$$\text{Tree}(\ell_u, \mathbf{T}_{ne[u]}^{t-1}) = \text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{t-1}). \tag{7}$$

By induction, Eq. (6) is true if and only if

$$c_u^{(t-1)} = c_v^{(t-1)}. \tag{8}$$

Equation (7) implies that $\ell_u = \ell_v$ and $\mathbf{T}_{ne[u]}^{t-1} = \mathbf{T}_{ne[v]}^{t-1}$, which means that an ordering on $ne[u]$ and $ne[v]$ exists s.t.

$$T_{ne_i(u)}^{t-1} = T_{ne_i(v)}^{t-1} \ \forall\, i = 1, \dots, |ne[u]|. \tag{9}$$

Hence, Eq. (9) holds if and only if an ordering on $ne[u]$ and $ne[v]$ exists s.t.

$$c_{ne(u)_i}^{t-1} = c_{ne(v)_i}^{t-1} \ \forall i = 1, \dots, |ne[u]|,$$

that is

$$\{\{c_m^{(t-1)} | m \in ne[u]\} = \{c_n^{(t-1)} | n \in ne[v]\}\}. \tag{10}$$

Putting together Eqs. (8) and (10), we obtain

$$\begin{aligned}&\text{HASH}(c_u^{(t-1)}, \{\{c_m^{(t-1)} | m \in ne[u]\}\})\\ &\quad = \text{HASH}(c_v^{(t-1)}, \{\{c_n^{(t-1)} | n \in ne[v]\}\}),\end{aligned}$$

which implies that $c_u^{(t)} = c_v^{(t)}$.

($\leftarrow$) The proof of the converse implication follows a similar reasoning, but some different steps are required to reconstruct the unfolding equivalence from the equivalence based on the 1-WL test.

Let us assume that $c_u^{(t)} = c_v^{(t)}$; by definition

$$\begin{aligned}&\text{HASH}(c_u^{(t-1)}, \{\{c_m^{(t-1)} | m \in ne[u]\}\})\\ &\quad = \text{HASH}(c_v^{(t-1)}, \{\{c_n^{(t-1)} | n \in ne[v]\}\}).\end{aligned} \tag{11}$$

Being the HASH function bijective, Eq. (11) implies that

$$c_u^{(t-1)} = c_v^{(t-1)} \tag{12}$$

and

$$\{\{c_m^{(t-1)} | m \in ne[u]\} = \{c_n^{(t-1)} | n \in ne[v]\}\}. \tag{13}$$

Equation (12) is true if and only if by induction

$$\mathbf{T}_u^{t-1} = \mathbf{T}_v^{t-1} \tag{14}$$

which implies that

$$\ell_u = \ell_v. \tag{15}$$

Moreover, Eq. (13) means that an ordering on $ne[u]$ and $ne[v]$ exists, such that

$$c_{ne(u)_i}^{t-1} = c_{ne(v)_i}^{t-1} \ \forall i = 1, \dots, |ne[u]|. \tag{16}$$

Instead, by induction, Eq. (16) holds if and only if an ordering on $ne[u]$ and $ne[v]$ exists so as $T_{ne_i(u)}^{t-1} = T_{ne_i(v)}^{t-1} \ \forall\, i = 1, \dots, |ne[u]|$, that is

$$\mathbf{T}_{ne[u]}^{t-1} = \mathbf{T}_{ne[v]}^{t-1}. \tag{17}$$

Finally, putting together Eqs. (15) and (17), we obtain

$$\text{Tree}(\ell_u, \mathbf{T}_{ne[u]}^{t-1}) = \text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{t-1})$$

that means $\mathbf{T}_u^t = \mathbf{T}_v^t$.

□

Theorem 4.1 is therefore proven, as its statement just rephrases the statement of Lemma A.1 in terms of the equivalence notation. Theorem 4.2 is the natural extension of Theorem 4.1 to graphs.

Proof of Theorem 4.3

To prove the theorem, we introduce the concept of *universal covering*, first presented in Krebs and Verbitsky (2015), which allows us to derive useful properties on the unfolding trees (see Krebs and Verbitsky 2015 for more details).

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. Given a graph $\mathbf{H} = (\mathbf{V}', \mathbf{E}')$ and a homomorphism α from $\mathbf{H}$ to $\mathbf{G}$, if:

- α is a bijection from $ne(v)$ onto $ne(\alpha(v))$
- $f_v(v) = f_v(\alpha(v))$
- $f_v(u) = f_v(\alpha(u))\ \forall u \in ne(v)$

for all $v \in \mathbf{V}'$, then α is called an *attributed covering map* and $\mathbf{H}$ is called a *covering graph*. Given a connected graph $\mathbf{G}$ and a vertex $x \in \mathbf{V}$, let us define a graph $\mathbf{U}_x(\mathbf{G})$ as follows. The vertex set of $\mathbf{U}_x(\mathbf{G})$ consists of all non-backtracking walks in $\mathbf{G}$ starting at x, that is, of sequences $(x_0, x_1, \ldots, x_k)$ such that $x_0 = x$, x_i and x_{i+1} are adjacent, and $x_{i+1} \neq x_{i-1}$. Two such walks are *adjacent* in $\mathbf{U}_x(\mathbf{G})$ if one of them extends the other by one component, that is, one is $(x_0, \ldots, x_k, x_{k+1})$ and the other is $(x_0, \ldots, x_k)$. $\mathbf{U}_x(\mathbf{G})$ is a tree and γ_G defined as $\gamma_G(x_0, \ldots, x_k, x_{k+1}) = x_k$ is a covering map from $\mathbf{U}_x(\mathbf{G})$ to $\mathbf{G}$. We call $\mathbf{U}$ an *attributed universal cover* of $\mathbf{G}$ if $\mathbf{U}$ covers any covering graph of $\mathbf{G}$. Therefore, $\mathbf{U}_x(\mathbf{G})$ is an attributed universal cover of $\mathbf{G}$.

Remark A.2 Given that we are dealing with attributed graphs, we will drop the "attributed"adjective from now on, to make the notation lighter.

The next lemma, which is proved in Krebs and Verbitsky (2015), shows the bijective correspondence between universal coverings and colors up to a certain depth/iteration.

Lemma A.3 (Krebs and Verbitsky 2015) *Let* $\mathbf{U}$ *and* $\mathbf{W}$ *be universal covers of graphs* $\mathbf{G}$ *and* $\mathbf{H}$*, respectively. Furthermore, let* α *be a covering map from* $\mathbf{U}$ *to* $\mathbf{G}$ *and* β *be a covering map from* $\mathbf{W}$ *to* $\mathbf{H}$*. Let* $x \in \mathbf{V}(\mathbf{U})$ *and* $y \in \mathbf{V}(\mathbf{W})$*, and let* $u = \alpha(x)$ *and* $v = \beta(y)$*. Then, for any* t*,* $\mathbf{U}_x^t \cong \mathbf{W}_y^t$ *if and only if* $c^{(t)}(u) = c^{(t)}(v)$*.*

Remark A.4 We can always identify the node x from a covering $\mathbf{W}_x$ of a graph $\mathbf{H}$ with its mapping u via α; i.e., $x = u$. This allows us to restate the previous bijection as: $\mathbf{U}_u^t \cong \mathbf{W}_v^t$ if and only if $c^{(i)}(u) = c^{(i)}(v)$.

We will now bridge the concepts of universal coverings and unfolding trees, passing through the color refinement algorithm.

Lemma A.5 *Let* $\mathbf{G}$ *and* $\mathbf{H}$ *be connected graphs and* x, y *be nodes of* $\mathbf{G}$ *and* $\mathbf{H}$*, respectively. Then,* $\mathbf{T}_x^t \cong \mathbf{T}_y^t$ *if and only if* $\mathbf{U}_x^t \cong \mathbf{W}_y^t$ *for all* i*.*

Proof Coupling Lemmas A.1 and A.3, we straightforwardly obtain the result. □

The established bijection leads us directly to the proof of Theorem 4.3.

Proof of Theorem 4.3. The proof is based on the reasoning adopted for Lemma 2.4 and Theorem 3.2 in Krebs and Verbitsky (2015). Actually, such a lemma and theorem are similar to points (1) and (2) of Theorem 4.3 and differ only, because the results in Krebs and Verbitsky (2015) are about universal covers, whereas our points are about unfolding trees. However, Lemma A.5 shows that universal covers and unfolding trees produce the same isomorphism on nodes. □

Proof of Theorem 4.5

Proof It follows directly from the combination of Theorem 1 in Scarselli (2009a) and Theorem 4.3. □

Proof of Theorem 4.6 (approximation by GNNs)

First, we need a preliminary lemma, for the the proof of which we refer to Scarselli (2009a). Intuitively, this lemma suggests that a graph domain with continuous features can be partitioned into small subsets, so that the features of the graphs are almost constant in each partition. Moreover, a finite number of partitions are sufficient, in probability, to cover a large part of the domain.

Lemma A.6 (Lemma 1 in Scarselli 2009a) *For any probability measure* P *on* $\mathcal{D}$*, and any reals* λ*,* δ*, where* $0 < \lambda \leq 1$*,* $\delta \geq 0$*, there exist a real* $\bar{b} > 0$*, which is independent of* δ*, a set* $\bar{\mathcal{D}} \subseteq \mathcal{D}$*, and a finite number of partitions* $\bar{\mathcal{D}}_1, \ldots, \bar{\mathcal{D}}_l$ *of* $\bar{\mathcal{D}}$*, where* $\bar{\mathcal{D}} = \mathcal{G}_i \times \{v_i\}$*, with* $\mathcal{G}_i \subseteq \mathcal{G}$ *and* $v_i \in \mathcal{G}_i$*, such that:*

1. $P(\bar{\mathcal{D}}) \geq 1 - \lambda$ *holds;*
2. *for each* i*, all the graphs in* $\mathcal{G}_i$ *have the same structure, i.e., they differ only for the values of their labels;*
3. *for each set* $\bar{\mathcal{D}}_i$*, there exists a hypercube* $\mathcal{H}_i \in \mathbb{R}^a$*, such that* $\ell_{\mathbf{G}} \in \mathcal{H}_i$ *holds for any graph* $\mathbf{G} \in \mathcal{G}_i$*, where* $\ell_{\mathbf{G}}$ *denotes the vector obtained by stacking all the feature vectors of* $\mathbf{G}$*;*
4. *for any two different sets* $\mathcal{G}_i, \mathcal{G}_j$*,* $i \neq j$*, their graphs have different structures or their hypercubes* $\mathcal{H}_i$*,* $\mathcal{H}_j$ *have a null intersection, i.e.,* $\mathcal{H}_i \bigcap \mathcal{H}_j = \emptyset$*;*
5. *for each* i *and each pair of graphs* $\mathbf{G}_1$*,* $\mathbf{G}_2 \in \mathcal{G}_i$*, the inequality* $\|\ell_{\mathbf{G}_1} - \ell_{\mathbf{G}_2}\|_\infty \leq \delta$ *holds;*
6. *for each graph* $\mathbf{G} \in \bar{\mathcal{D}}$*, the inequality* $\|\ell_{\mathbf{G}}\|_\infty \leq \bar{b}$ *holds.*

By adopting an argument similar to that proposed in Scarselli (2009a), it is proved that Theorem 4.6 is equivalent to the following Theorem A.7, where the domain contains a finite number of graphs and the features are integers.

Theorem A.7 *For any finite set of patterns* $\{(\mathbf{G}_i, v_i)|\ \mathbf{G}_i \in \mathcal{G}, v_i \in N, 1 \leq i \leq n\}$*, with* $r = \max_{\mathbf{G}_i} |(\mathbf{G}_i)|$ *and with graphs having integer features, for any function* $\tau : \mathcal{D} \to \mathbb{R}$*, which preserves the unfolding equivalence, and for any* *real $\varepsilon > 0$, there exist continuously differentiable functions AGGREGATE$^{(k)}$, COMBINE$^{(k)}$, $\forall k \leq r+1$, s.t.*

$$\mathbf{h}_v^k = \text{COMBINE}^{(k)}\Big(\mathbf{h}_v^{k-1}, \text{AGGREGATE}^{(k)}\left(\{\{\mathbf{h}_u^{k-1},\ u \in ne[v]\}\}\right)\Big)$$

and a function READOUT, with feature dimension $m = 1$, i.e., $\mathbf{h}_v^k \in \mathbb{R}$, so that the function φ (realized by the GNN), computed after $r+1$ steps, satisfies the condition

$$|\tau(\mathbf{G}_i, v_i) - \varphi(\mathbf{G}_i, v_i)| \leq \varepsilon \tag{18}$$

for any i, $1 \leq i \leq n$.

The equivalence is formally proved by the following lemma.

Lemma A.8 *Theorem 4.6 holds if and only if Theorem A.7 holds.*

Proof Although the proof is quite identical to that contained in Scarselli (2009a), we report it here with the new notation.

Theorem 4.6 is more general than Theorem A.7, which makes this implication straightforward. Suppose instead that Theorem A.7 holds and show that this implies Theorem 4.6. Let us apply Lemma A.6 with values for P and λ equal to the corresponding values of Theorem 4.6, being δ any positive real number. It follows that there is a real $\bar{b}$ and a subset $\bar{\mathcal{D}}$ of $\mathcal{D}$ s.t. $P(\bar{\mathcal{D}}) > 1 - \lambda$. Let $\mathcal{M}$ be the subset of $\mathcal{D}$ that contains only the graphs $\mathbf{G}$ satisfying $\|\ell_{\mathbf{G}}\|_\infty \leq \bar{b}$. Note that, since $\bar{b}$ is independent of δ, then $\bar{\mathcal{D}} \subset \mathcal{M}$ for any δ. Since τ is integrable, there exists a continuous function which approximates τ, in probability, up to any degree of precision. Thus, without loss of generality, we can assume that τ is equi-continuous on $\mathcal{M}$. By definition of equi-continuity, a real $\bar{\delta} > 0$ exists, such that

$$|\tau(\mathbf{G}_1, v) - \tau(\mathbf{G}_2, v)| \leq \frac{\varepsilon}{2} \tag{19}$$

holds for any node v and for any pair of graphs $\mathbf{G}_1$, $\mathbf{G}_2$ having the same structure and satisfying $\|\ell_{\mathbf{G}_1} - \ell_{\mathbf{G}_2}\|_\infty \leq \bar{\delta}$.

Let us apply Lemma A.6 again, where, now, the δ of the hypothesis is set to $\bar{\delta}$, i.e., $\delta = \bar{\delta}$. From then on, $\bar{\mathcal{D}} = \mathcal{G}_i \times \{v_i\}$, $1 \leq i \leq n$, represents the set obtained by the new application of Lemma A.6 and $I_i^{\bar{b},\bar{\eta}}$, $1 \leq i \leq 2d$, denote the corresponding intervals defined in the proof of the same lemma. Let $\theta : \mathbb{R} \to \mathbb{Z}$ be a function that encodes reals into integers as follows: for any i and any $z \in I_i^{\bar{b},\bar{\eta}}$, $\theta(z) = i$. Thus, θ assigns to all the values of an interval $I_i^{\bar{b},\bar{\eta}}$ the index i of the interval itself. Since the intervals do not overlap and are not contiguous, θ can be continuously extended to the entire $\mathbb{R}$. Moreover, θ can be extended also to vectors, being $\theta(\mathbf{Z})$ the vector of integers obtained by encoding all the components of $\mathbf{Z}$. Finally, let $\Theta : \mathcal{G} \to \mathcal{G}$ represent the function that transforms each graph by replacing all the feature labels with their coding, i.e., $\mathbf{L}_{\Theta(\mathbf{G})} = \theta(\mathbf{L}_{\mathbf{G}})$. Let $\bar{\mathbf{G}}_1, \dots, \bar{\mathbf{G}}_v$ be graphs, each one extracted from a different set $\mathcal{G}_i$. Note that, according to points 3, 4, 5 of Lemma A.6, Θ produces an encoding of the sets $\mathcal{G}_i$. More precisely, for any two graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ of $\bar{\mathcal{D}}$, we have $\Theta(\mathbf{G}_1) = \Theta(\mathbf{G}_2)$ if the graphs belong to the same set, i.e., $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{G}_i$, while $\Theta(\mathbf{G}_1) \neq \Theta(\mathbf{G}_2)$ otherwise. Thus, we can define a decoding function Γ s.t. $\Gamma(\Theta(\bar{\mathbf{G}}_i), v_i) = (\bar{\mathbf{G}}_\mathbf{i}, v_i)$, $1 \leq i \leq n$.

Consider, now, the problem of approximating $\tau \circ \Gamma$ on the set $(\Theta(\bar{\mathbf{G}}_1), v_1), \dots, (\Theta(\bar{\mathbf{G}}_n), v_n)$. Theorem A.7 can be applied to such a set, because it contains a finite number of graphs with integer labels. Therefore, there exists a GNN that implements a function $\bar{\varphi}$ s.t. for each i

$$|\tau(\Gamma(\Theta(\bar{\mathbf{G}}_i), v_i)) - \bar{\varphi}(\Theta(\bar{\mathbf{G}}_i), v_i)| \leq \frac{\varepsilon}{2}. \tag{20}$$

However, this means that there is also another GNN that produces the same result operating on the original graphs $\mathbf{G}_i$, namely a GNN for which

$$\varphi(\mathbf{G}_i, v_i) = \bar{\varphi}(\Theta(\bar{\mathbf{G}}_i), v_i) \tag{21}$$

holds. Actually, the graphs $\mathbf{G}_i$ and $\bar{\mathbf{G}}_i$ are equal except that the former has the coding of the feature labels attached to the nodes, while the latter contains the whole feature labels. Thus, the GNN that operates on $\bar{\mathbf{G}}_i$ is that suggested by Theorem A.7, except that $\overline{\text{AGGREGATE}}^{(0)}$ preliminary creates a coding of $\theta(\ell_v)$.

Putting together the above equality with Eqs. (19) and (20), it immediately follows that for any $(\mathbf{G}, v) \in \bar{\mathcal{D}}_i$:

$$\begin{aligned}
&|\tau(\mathbf{G}, v) - \varphi(\mathbf{G}, v)| \\
&\quad = |\tau(\mathbf{G}, v) - \tau(\bar{\mathbf{G}}_i, v) + \tau(\bar{\mathbf{G}}_i, v) - \varphi(\mathbf{G}, v)| \\
&\quad \leq |\tau(\bar{\mathbf{G}}_i, v) - \varphi(\mathbf{G}, v)| + \frac{\varepsilon}{2} \\
&\quad = |\tau(\Gamma(\Theta(\bar{\mathbf{G}}_i), v)) - \bar{\varphi}(\Theta(\bar{\mathbf{G}}_i), v)| + \frac{\varepsilon}{2} \leq \varepsilon.
\end{aligned}$$

Thus, the GNN described by Eq. (21) satisfies $|\tau(\mathbf{G}, v) - \varphi(\mathbf{G}, v)| \leq \varepsilon$ in the restricted domain $\bar{\mathcal{D}}$. Since $P(\bar{\mathcal{D}}) \geq 1 - \lambda$, we have

$$P(\|\tau(\mathbf{G}, v) - \varphi(\mathbf{G}, v)\| \leq \varepsilon) \geq 1 - \lambda,$$

which proves the lemma.

□

Now, we can proceed to prove Theorem A.7.

Proof of Theorem A.7 For the sake of simplicity, the theorem will be proved assuming $n = 1$, i.e., $\tau(\mathbf{G}, v) \in \mathbb{R}$. However, the result can be easily extended to the general case when $\tau(\mathbf{G}, v) \in \mathbb{R}^o$. Indeed, in this case, the GNN that satisfies the theorem can be defined by stacking n GNNs, each one approximating a component of $\tau(\mathbf{G}, v)$.

According to Theorem 4.5, there exists a function κ s.t. $\tau(\mathbf{G}, v) = \kappa(\mathbf{T}_v)$. Therefore, an unfolding tree of depth $2r-1$, where r is the maximum number of nodes in the graph domain, is enough to store the graph information, so that κ can be designed to satisfy $\tau(\mathbf{G}, v) = \kappa(\mathbf{T}_v) = \kappa(\mathbf{T}_v^{2r-1})$; moreover, according to Theorem 4.3, the depth of the truncated unfolding tree is enough to respect the unfolding equivalence over all the nodes of every graph in the domain. Consequently, the main idea of the proof consists in designing a GNN that is able to encode the unfolding tree into the node features, i.e., for each node v, we want to have $\mathbf{h}_v = \triangledown(\mathbf{T}_v^{2r-1})$, where $\triangledown$ is an encoding function that maps trees into real numbers. More precisely, the encodings are constructed recursively by AGGREGATE$^{(k)}$ and COMBINE$^{(k)}$ functions using the neighborhood information. After t steps, the node features contain the encoding of the unfolding tree $\triangledown(\mathbf{T}_v^t)$ of depth t. Then, after a number of steps $\bar{t}$ larger than the number of nodes of the graph, the GNN, by the READOUT function, can produce the desired output $\kappa(\mathbf{T}_v^n)$.

Accordingly, the theorem can be proved provided that we can implement the above-mentioned procedure, which means that there exist appropriate functions $\triangledown$, AGGREGATE$^{(k)}$, COMBINE$^{(k)}$ and READOUT. The existence of the READOUT function is obvious, since, given that unfolding trees can be encoded in node features, READOUT has just to decode the representation and compute the target output. Then, let use focus on the other functions. They will be defined in two steps. Initially, AGGREGATE$^{(k)}$, COMBINE$^{(k)}$, and READOUT will be defined without taking into account that they have to be continuously differentiable. Later, this farther constraint will be considered.

The coding function $\triangledown$

Let $\triangledown$ be a composition of any two injective functions α and β, $\alpha \circ \beta$, with the properties described in the following.

- α is an injective function from the domain of the unfolding trees $\mathcal{T}^N$, calculated on the nodes of the graph $\mathbf{G}_i$, to the Cartesian product $\mathbb{N} \times \mathbb{N}^P \times \mathbb{Z}^{\ell \mathbf{v}} = \mathbb{N}^{P+1} \times \mathbb{Z}^{\ell \mathbf{v}}$, where N is the number of nodes of the graph and P is the maximum number of nodes a tree could have.
  Intuitively, in the Cartesian product, $\mathbb{N}$ represents the tree structure, $\mathbb{N}^P$ denotes the node numbering, while, for each node, an integer vector $\in \mathbb{Z}^{\ell \mathbf{v}}$ is used to encode the node features. Note that α exists and is injective, since the maximum information contained in an unfolding tree is given by the union of all its node features and all its structural information, which is exactly equal to the codomain size of α.
- β is an injective function from $\mathbb{N}^{P+1} \times \mathbb{Z}^{\ell \mathbf{v}}$ to $\mathbb{R}$, whose existence is guaranteed by the cardinality theory, since the two sets have the same cardinality.

Since α and β are injective, also the existence and the injectiveness of $\triangledown$ are ensured.

Functions AGGREGATE$^{(k)}$ *and* COMBINE$^{(k)}$

Functions AGGREGATE$^{(k)}$ and COMBINE$^{(k)}$ must satisfy

$$\begin{aligned}\triangledown(\mathbf{T}_v^t) = \mathbf{h}_v^t &= \text{COMBINE}^{(k)}\big(\mathbf{h}_v^{k-1},\\ &\quad \text{AGGREGATE}^{(k)}(\{\{\mathbf{h}_u^{k-1},\ u \in ne[v]\}\})\big)\\ &= \text{COMBINE}^{(k)}\big(\triangledown(\mathbf{T}_v^{k-1}),\\ &\quad \text{AGGREGATE}^{(k)}(\{\{\triangledown(\mathbf{T}_u^{k-1}),\ u \in ne[v]\}\})\big)\end{aligned}$$

$\forall k \leq N$, where N is the number of nodes. In a simple solution, AGGREGATE$^{(k)}$ decodes the trees of the neighbor $\mathbf{T}_u^{t-1}$ of v and stores them into a data structure to be accessed by COMBINE$^{(k)}$. For example, the trees can be collected into the coding of a new tree, i.e., AGGREGATE$^{(k)}(\triangledown(\mathbf{T}_u^{t-1}), u \in ne[v])$ $= \triangledown(\bigcup_{u \in ne[v]} \triangledown^{-1}(\triangledown(\mathbf{T}_u^{t-1})))$, where $\bigcup_{u \in ne[v]}$ denotes an operator that constructs a tree, with a root having void features, from a set of sub-trees (see Fig. 6). Then, COMBINE$^{(k)}$ assigns the correct features to the root by extracting them from $\mathbf{T}_v^{t-1}$, that is

$$\begin{aligned}&\text{COMBINE}^{(k)}(\triangledown(\mathbf{T}_v^{t-1}), b)\\ &\quad = \triangledown(\text{ATTACH}(\triangledown^{-1}(\triangledown(\mathbf{T}_v^{t-1})), \triangledown^{-1}(b))),\end{aligned}$$

where ATTACH is an operator that construct a tree following the procedure depicted in Fig. 6 and b is the result of the AGGREGATE$^{(k)}$ function.

Unfortunately, with this definition, AGGREGATE$^{(k)}$, COMBINE$^{(k)}$, and READOUT may not be differentiable. Nevertheless, Eq. (18) has to be satisfied only for a finite number of graphs, namely $\mathbf{G}_i$. Thus, we can specify other functions $\overline{\text{AGGREGATE}}^{(k)}$, $\overline{\text{COMBINE}}^{(k)}$, and $\overline{\text{READOUT}}$, which produce exactly the same computations when they are applied on the graphs $\mathbf{G}_i$, but that can be extended to the rest of their domain, so that they are continuously differentiable. Obviously, such an extension exists, since those functions are only constrained to interpolate a finite number of points.[6] □

[6] It is worth noting that a similar extension can also be applied to the coding function $\triangledown$ and to the decoding function $\triangledown^{-1}$. In this case, the coding function is not injective on the whole domain, but only on the graphs mentioned in the theorem.

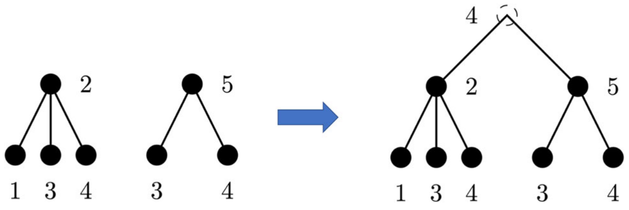


Fig. 6 The ATTACH operator on trees

Proof sketch of Theorem 4.8

Proof As in the proof of Theorem 4.6, without loss of generality, we will assume that the feature dimension is $m = 1$. First of all, note that Theorem 4.6 ensures that we can find $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\forall k \le N$, and $\overline{\text{READOUT}}$, so that for the corresponding function $\bar\varphi$ implemented by the GNN

$$P(\|\tau(\mathbf{G}, v) - \bar\varphi(\mathbf{G}, v)\| \le \varepsilon/2) \ge 1 - \lambda \tag{22}$$

holds. Let us consider the corresponding transition function $\bar f$, defined by

$$\begin{aligned}&\bar f^k(\mathbf{h}_v^{k-1}, \{\{\mathbf{h}_u^{k-1}, u \in ne[v]\}\})\\&\quad = \overline{\text{COMBINE}}^{(k)}\big(\mathbf{h}_v^{k-1},\\&\qquad \overline{\text{AGGREGATE}}^{(k)}(\{\{\mathbf{h}_u^{k-1}, u \in ne[v]\}\})\big).\end{aligned}$$

Since $\overline{\text{COMBINE}}^{(k)}$ and $\overline{\text{AGGREGATE}}^{(k)}$ are continuously differentiable, $\bar f^k$ is continuously differentiable. Considering that the theorem has to hold only in probability, we can also assume that the domain is bounded, so that $\bar f^k$ is bounded and has a bounded Jacobian. Let B be a bound on the Jacobian/derivative of $\bar f^k$ for any k and any input. The same argument can also be applied to the function $\overline{\text{READOUT}}$, which is continuously differentiable w.r.t. its input and can be assumed to have a bounded Jacobian/derivative. Let us assume that B is also a bound for the Jacobian/derivative of $\overline{\text{READOUT}}$. Moreover, let $\text{COMBINE}_w^{(k)}$ and $\text{AGGREGATE}_w^{(k)}$ be functions implemented by universal neural network that approximate $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\forall k \le r$, respectively, such that

$$\begin{aligned}&f_w^k(\mathbf{h}_v^{k-1}, \{\{\mathbf{h}_u^{k-1},\ u \in ne[v]\}\})\\&\quad = \text{COMBINE}_w^{(k)}\big(\mathbf{h}_v^{k-1}, \text{AGGREGATE}_w^{(k)}\big(\{\{\mathbf{h}_u^{k-1}, u \in ne[v]\}\}\big)\big),\end{aligned}$$

and let us assume that

$$\|\bar f^k - f_w^k\|_\infty \le \eta \tag{23}$$

holds for every k and a $\eta > 0$. Let READOUT_w be the function implemented by a universal neural network that approximates $\overline{\text{READOUT}}$, so that

$$\|\overline{\text{READOUT}} - \text{READOUT}\|_\infty \le \eta.$$

In the following, it will be shown that, when η is sufficiently small, the GNN implemented by the approximating neural networks is sufficiently close to the GNN of Theorem 4.6, so that the thesis is proved.

Let $\bar F^k$, F_w^k be the global transition functions of the GNNs that are obtained by stacking all the $\bar f^k$ and f_w^k for all the nodes of the input graph. The node features are computed at each step by $\bar H^k = \bar F^k(\bar H^{k-1})$, $H^k = F_w^k(H^{k-1})$, where $\bar H^k$,H^k denote the stacking of all the node features of the graph obtained by the two transition functions, respectively. Then

$$\|\bar H^1 - H^1\|_\infty = \|\bar F^1(H^0) - F_w^1(H^0)\|_\infty \le \eta N, \tag{24}$$

where $N = |\mathbf{G}|$ is the number of nodes in the input graph. Moreover

$$\begin{aligned}&\|\bar H^2 - H^2\|_\infty\\&\quad = \|\bar F^2(\bar H^1) - F_w^2(H)\|_\infty\\&\quad = \|\bar F^2(\bar H^1) - \bar F^2(H^1) + \bar F^2(H^1) - F_w^2(H^1)\|_\infty\\&\quad \le \|\bar F^2(\bar H^1) - \bar F^2(H^1)\|_\infty + \|\bar F^2(H^1) - F_w^2(H^1)\|_\infty\\&\quad \le \eta N B + \eta N = \eta N(B+1).\end{aligned}$$

Here, $\|\bar F^2(\bar H^1) - \bar F^2(H^1)\|_\infty \le \eta N B$ holds because of Eq. (24), which bounds the difference between $\bar H^1$ and H^1, and due to the fact that the Jacobian/derivative of $\bar F^2$ is bounded by B. Moreover, $\|\bar F^2(H^1) - F_w^2(H^1)\|_\infty \le \eta N$ holds by Eq. (23).

The above reasoning can then be applied recursively to prove that

$$\|\bar H^k - H_w^k\|_\infty \le \eta N \sum_{i=0}^{k-1} B^i.$$

Since the output of the GNN is computed using the encoding at step N, we have

$$\begin{aligned}&\|\bar\varphi(\mathbf{G}, v) - \varphi_w(\mathbf{G}, v)\|_\infty\\&\quad = \|\overline{\text{READOUT}}(\bar H^N) - \text{READOUT}_w(H^N)\|_\infty\\&\quad \le \eta N + B\left(\eta N \sum_{i=0}^{N} B^i\right).\end{aligned}$$

Finally, since we can consider the maximum number of nodes N as bounded[7], then we can find a GNN based on neural networks, so that η is small enough to achieve

[7] For the sake of simplicity, we skip over a very formal proof of this claim. Intuitively, note that the theorem has to be proved and Lemma A.6 clarifies that any graph domain can be covered in high probability by a finite number of structures, which obviously have a bounded number of nodes.

$$\|\overline{\varphi}(\mathbf{G}, v) - \varphi_w(\mathbf{G}, v)\|_\infty \leq \epsilon/2,$$

which, together with Eq. (22), produces the bound of Theorem 4.6. □

References

Abboud R, Ceylan İİ, Grohe M, Lukasiewicz T(2020) The surprising power of graph neural networks with random node initialization. arXiv preprint arXiv:2010.01179

Alon U, Yahav E (2020) On the bottleneck of graph neural networks and its practical implications. arXiv preprint arXiv:2006.05205

Angluin D (1980) Local and global properties in networks of processors (extended abstract). In: Proceedings of the 12th annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 82–93

Azizian W, Lelarge M (2020) Expressive power of invariant and equivariant graph neural networks. arXiv preprint arXiv:2006.15646

Bandinelli N, Bianchini M, Scarselli F (2010) Learning long-term dependencies using layered graph neural networks. Proc IJCNN 2010:1–8

Barceló P et al (2020) The logical expressiveness of graph neural networks. In: Proceedings of the 8th international conference on learning representations (ICLR 2020)

Battaglia P et al (2018) Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261

Bianchini M, Gori M (2001) Theoretical properties of recursive neural networks with linear neurons. IEEE Trans Neural Netw 12:953–967

Bodnar C et al (2021a) Weisfeiler and Lehman go topological: message passing simplicial networks (PMLR), pp 1026–1037

Bodnar C et al (2021b) Weisfeiler and Lehman go cellular: CW networks. Adv Neural Inf Process Syst 34:2625–2640

Bouritsas G, Frasca F, Zafeiriou S, Bronstein MM (2020) Improving graph neural network expressivity via subgraph isomorphism counting. arXiv preprint arXiv:2006.09252

Brugiapaglia S, Liu M, Tupper P (2020)Generalizing outside the training set: when can neural networks learn identity effects? arXiv preprint arXiv:2005.04330

Brugiapaglia S, Liu M, Tupper P (2022) Invariance, encodings, and generalization: learning identity effects with neural networks. Neural Comput 34:1756–1789

Bruna J, Zaremba W, Szlam A, LeCun Y (2014) Spectral networks and locally connected networks on graphs. In: Proceedings of ICLR 2014

Dell H, Grohe M, Rattan G (2018) Lovász meets Weisfeiler and Leman. arXiv preprint arXiv:1802.08876

D'Inverno GA, Brugiapaglia S, Ravanelli M (2023)Generalization limits of graph neural networks in identity effects learning. arXiv preprint arXiv:2307.00134

Garg V, Jegelka S, Jaakkola T (2020) Generalization and representational limits of graph neural networks. In: Proceedings of ICML 2020 (PMLR), pp 3419–3430

Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. In: Proceedings of ICML 2017 (PMLR), pp 1263–1272

Gori M, Monfardini G, Scarselli F (2005) A new model for learning in graph domains. In: Proceedings of IJCNN 2005, vol 2, pp 729–734

Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. Adv Neural Inf Process Syst 30:13481

Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4:251–257

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2:359–366

Jegelka S (2022) Theory of graph neural networks: representation and learning. arXiv preprint arXiv:2204.07697

Keriven N, Peyré G (2019) Universal invariant and equivariant graph neural networks. In: Advances in neural information processing systems (NeurIPS 2019)

Kiefer S (2020) Power and limits of the Weisfeiler–Lehman algorithm. Ph.D. thesis, Dissertation, RWTH Aachen University

Kiefer S, McKay BD (2020) The iteration number of colour refinement. In: Proceedings of the 47th international colloquium on automata, languages, and programming (ICALP 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik

Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of ICLR 2017

Krebs A, Verbitsky O (2015) Universal covers, color refinement, and two-variable counting logic: Lower bounds for the depth. In: Proceedings of the 30th annual ACM/IEEE symposium on logic in computer science (IEEE), pp 689–700

Lehman AA, Weisfeiler B (1968) A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsiya 2:12–16

Li Y et al (2015) Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493

Li Y, Tarlow D, Brockschmidt M, Zemel R (2015) Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493

Linial N (1992) Locality in distributed graph algorithms. SIAM J Comput 21:193–201

Loukas A (2019) What graph neural networks cannot learn: depth vs width. arXiv preprint arXiv:1907.03199

Maron H, Ben-Hamu H, Shamir N, Lipman Y (2018) Invariant and equivariant graph networks. arXiv preprint arXiv:1812.09902

Maron H, Ben-Hamu H, Serviansky H, Lipman Y (2019) Provably powerful graph networks. Adv Neural Inf Process Syst 32:472

Micheli A (2009) Neural network for graphs: a contextual constructive approach. IEEE Trans Neural Netw 20:498–511

Morris C et al (2019) Weisfeiler and Lehman go neural: higher-order graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 4602–4609

Naor M, Stockmeyer L (1993) What can be computed locally?. In: Proceedings of the 25th annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 184–193

Puny O, Ben-Hamu H, Lipman Y (2020) From graph low-rank global attention to 2-FWL approximation. CoRR https://arxiv.org/abs/2006.07846

Ramakrishnan R, Dral PO, Rupp M, Von Lilienfeld OA (2014) Quantum chemistry structures and properties of 134 kilo molecules. Sci Data 1:1–7

Rossi A et al (2018) Inductive–transductive learning with graph neural networks. In: Proceedings of IAPR workshop on artificial neural networks in pattern recognition. Springer, New York, pp 201–212

Ruddigkeit L, Van Deursen R, Blum LC, Reymond J-L (2012) Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. J Chem Inf Model 52:2864–2875

Sato R (2020) A survey on the expressive power of graph neural networks. arXiv preprint arXiv:2003.04078

Sato R, Yamada M, Kashima H (2021) Random features strengthen graph neural networks. In: Proceedings of SDM21

Scarselli F et al (2009a) Computational capabilities of graph neural networks. IEEE Trans Neural Netw 20:81–102

Scarselli F et al (2009b) The graph neural network model. IEEE Trans Neural Netw 20:61–80

Scarselli F, Chung Tsoi A (1998) Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results. Neural Netw 11:15–37

Scarselli F, Tsoi AC, Hagenbuchner M (2018) The Vapnik–Chervonenkis dimension of graph and recursive neural networks. Neural Netw 108:248–259

Sperduti A, Starita A (1997) Supervised neural networks for the classification of structures. IEEE Trans Neural Netw 8:714–735

Veličković P et al (2018) Graph attention networks. In: Proceedings of ICLR 2018

Wu Z et al (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32:4–24

Xu K, Hu W, Leskovec J, Jegelka S (2018) How powerful are graph neural networks?. In: Proceedings of the ICLR 2018

You J, Gomes-Selman J, Ying R, Leskovec J (2021) Identity-aware graph neural networks. In: Proceedings of the conference on artificial intelligence (AAAI 21)

Zhang M, Li P (2021) Nested graph neural networks. Adv Neural Inf Process Syst 34:15734–15747

Zhou X, Wang H (2021) The generalization error of graph convolutional networks may enlarge with more layers. Neurocomputing 424:97–106. https://www.sciencedirect.com/science/article/pii/S0925231220317367

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.