The Exact Class of Graph Functions Generated by Graph Neural Networks

Mohammad Fereydounian* Hamed Hassani* Javid Dadashkarimi† Amin Karbasi†

Abstract

Given a graph function, defined on an arbitrary set of edge weights and node features, does there exist a Graph Neural Network (GNN) whose output is identical to the graph function? In this paper, we fully answer this question and characterize the class of graph problems that can be represented by GNNs. We identify an algebraic condition, in terms of the permutation of edge weights and node features, which proves to be necessary and sufficient for a graph problem to lie within the reach of GNNs. Moreover, we show that this condition can be efficiently verified by checking quadratically many constraints. Note that our refined characterization on the expressive power of GNNs are orthogonal to those theoretical results showing equivalence between GNNs and Weisfeiler-Lehman graph isomorphism heuristic. For instance, our characterization implies that many natural graph problems, such as min-cut value, max-flow value, and max-clique size, can be represented by a GNN. In contrast, and rather surprisingly, there exist very simple graphs for which no GNN can correctly find the length of the shortest paths between all nodes. Note that finding shortest paths is one of the most classical problems in Dynamic Programming (DP). Thus, the aforementioned negative example highlights the misalignment between DP and GNN, even though (conceptually) they follow very similar iterative procedures. Finally, we support our theoretical results by experimental simulations.

1 Introduction

Processing data with graph structures has become an essential tool in application domains such as as computer vision [34], natural language processing [38], recommendation systems [29], and drug discovery [16], to name a few. Graph Neural Networks (GNN) are a class of iterative-based models that can process information represented in the form of graphs. Through a message passing mechanism, GNNs aggregate information from neighboring nodes in the graph in order to update node features [10]. Such node features can be ultimately used for down-stream tasks such as classification, link prediction, clustering, etc.

Even though many variations and architectures of GNNs have been proposed in recent years to increase the representation capacity of GNNs [2, 6, 7, 11–13, 15, 22, 24, 27, 28, 31, 32, 35, 40], it is still not clear what class of functions GNNs can generate exactly. There has been a large body of work that aims to understand the expressive power of GNNs through their ability to distinguish non-isomorphic graphs and the Weisfeiler–Lehman graph isomorphism test [20, 25, 30, 31]. However, the aforementioned results do not provide much indication to practitioners whether a specific graph function (e.g., shortest paths, min-cut, etc) can be computed by a GNN. In this paper, we aim to characterize the exact class

*Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA. Emails: {mferey, hassani}@seas.upenn.edu.

†Department of Electrical Engineering, Computer Science, Statsitics & Data Science, Yale University, New Haven, CT, USA. Emails: {javid.dadashkarimi, amin.karbasi}@yale.edu.

of graph problems solvable by GNNs. Our results are analogous to those of approximation capabilities of the feedforward neural networks on the space of continuous functions [1].

More specifically, we consider graphs that consist of nodes equipped with feature vectors, along with weights assigned to *all* pairs of nodes (i.e., edges). We should note that almost all graph problems can be stated over fully connected but weighted graphs. For example, for computing the shortest path on a given graph (which may not be fully connected), we can assign a very large value to non-existing edges. A graph function takes as input a graph in the form of weight and feature matrices and assigns a vector to each node. Similarly, a GNN is an evolving graph function that updates node features iteratively through an aggregation operation. Naturally, for GNNs to be able to solve graph problems defined over weighted graphs, their message-passing iterates need to incorporate edge weights. Finally, in our setting, we do not generally consider pooling/readout operations, since such operations can considerably reduce the class of functions generated by a GNN. However, as we will discuss shortly in the related work section, our results have important implications on GNNs with readouts.

Our contributions. In this paper, 1) we provide a necessary and sufficient algebraic condition, so called *permutation-compatibility*, which states that a graph function can be generated by a GNN if and only if it is permutation-compatible. 2) We show that for any graph problem, permutation-compatibility can be verified over polynomial number of constraints. 3) We characterize the basis functions for permutation-compatible graph functions. These basis functions effectively relate the properties of aggregation operators to the expressive power of the resulting GNNs. For instance, it follows that with continuous aggregation operators, all continuous permutation-compatible functions lie within the reach of GNNs. 4) Finally, we support our results empirically on two canonical graph problems, namely, the shortest paths (as an example of a permutation-incompatible graph function) and min-cut value (as an example of a permutation-compatible graph function). We see that GNNs on very simple graphs completely fail to compute the shortest paths. In contrast, the same GNN architectures can successfully compute the min-cut.

1.1 Related Work

It is well-established that GNNs cannot assign different values to isomorphic graphs [25]. On the other hand, from [31] and [20] we know that GNNs with appropriate *aggregation* and *pooling* operators, over *unweighted* graphs, are only as powerful as the Weisfeiler–Lehman graph isomorphism test, denoted by 1-WL [30]. Due to this negative result, many follow-up works proposed more involved variants such as as adding stochastic features [8, 21, 23, 26, 37], adding deterministic distance features [14, 36], building higher order GNNs [5, 17–20], so that the expressive power of the resulting GNNs go beyond the 1-WL test. Note that in our GNN setting, we consider fully connected weighted graphs without the pooling/readout operation. As a result, the equivalence between GNNs (on unweighted graphs with readout mechanisms) and 1-WL test do not apply to our setting. Indeed, our precise characterization of graph functions generated by GNNs, namely permutation-compatibility, also allows us to shed light on some of the elusive features of GNNs.

Implications of our results. First, when a permutation invariant pooling/readout operation is applied, permutation-compatibility implies that any isomorphism-invariant representation of graphs can be generated by GNNs performing message passing on fully connected weighted graphs. Thus, the resulting GNNs are considerably more powerful than the 1-WL test. Also note that, as we elaborated earlier, GNNs in general cannot assign different values to isomorphic graphs. Thus GNNs on fully connected weighted graphs achieve the highest expressive power. Second, our results show that there is a striking contrast between Dynamic Programming on graphs and functions computed by GNNs,

despite seemingly similar procedures and recent claims [33]. In fact, we show that no matter how the node features are chosen, there is a shortest path instance (a canonical DP problem) for which no GNN can assign correct values to all nodes. We also observe this phenomenon in simple empirical experiments.

2 Preliminaries

Throughout the paper, we consider multi-dimensional arrays (sequences) of objects. By $(a_1, \ldots, a_n)$, we denote a one-dimensional array of objects $a_1, \ldots, a_n$. If $A = (a_1, \ldots, a_n)$, we refer to the i-th element of A by $[A]_i$, i.e., $[A]_i = a_i$. Similarly, if A is a two-dimensional array (e.g., a matrix), $[A]_{i,j}$ refers to its (i, j)-th element. Similarly, $[A]_{i,j,k}$ refers to the (i, j, k)-th element in a three-dimensional array A. Sets are denoted by $\{\cdot\}$. We also let $[n] = \{1, \ldots, n\}$ and $[n]_{-i} = \{1, \ldots, n\} \setminus \{i\}$, where $A \setminus B$ denotes the set difference. The set of complex numbers is denoted by $\mathbb{C}$. If $z \in \mathbb{C}$, we use the standard notation $z = \mathrm{Re}(z) + \mathrm{Im}(z)\sqrt{-1}$.

In the following, we formally define *graphs*, *graph functions*, and GNNs.

Definition 2.1 (Class $\mathcal{G}_{n,d}$ graphs)**.** An undirected graph G is a tuple $G = ([n], W, X)$, where $[n] = \{1, \ldots, n\}$ is the set of nodes, and every pair of nodes $\{i, j\}$ with $i \neq j$ forms an edge to which a weight $w_{i,j} = w_{j,i}$ is assigned. The symmetric matrix $W \in \mathbb{R}^{n \times n}$ is called the weight matrix with zeros on its diagonal. Further, each node i is associated with a row feature vector $x_i \in \mathbb{R}^d$. We call $X = (x_1^\top, \ldots, x_n^\top)$ the feature matrix. Finally, we denote by $\mathcal{G}_{n,d}$ the set of graphs of size n with feature vectors of dimension d.

Remark 2.2. For the ease of presentation, we mainly consider scalar-valued weights. However, all of results can be extended to vector-valued weights.

Definition 2.3 (Graph function)**.** A graph function over $\mathcal{G}_{n,d}$ is a function F that takes as input any graph $G = ([n], W, X) \in \mathcal{G}_{n,d}$ and is identified by its action on (W, X) via the following form: $F(W, X) = (f_1(W, X), \ldots, f_n(W, X))$, where $f_i(W, X)$, so called the node-functions, are vector-valued function in some common Euclidean vector space.

Example 2.4. *To better understand the notion of graph functions, let us consider a few examples.*

1. **Feature-Oblivious.** *Let $n = 3$, and consider a function F with $f_1(W, X) = 0$, $f_2(W, X) = w_{1,2} + w_{2,3}$, and $f_3(W, X) = \sin(w_{1,3} + w_{2,3})$.*
2. **Feature-Identity.** *Let $f_i(W, X) = x_i$, for $i \in [n]$.*
3. **Feature-Sum.** *Let $f_i(W, X) = \sum_{\ell \in [n]} x_\ell$ for every $i \in [n]$. Here, F assigns the same value, i.e., the sum of all features, to each node.*
4. **Min-Sum.** *Let $f_i(W, X) = \min(x_i, \sum_{j \in [n]_{-i}} x_j)$. Here, the features are scalar-valued, i.e., $d = 1$.*
5. **Degree.** *Let F be a function that assigns to each node its degree, i.e., $f_i(W, X) = \sum_{j \in [n]_{-i}} w_{i,j}$.*
6. **Max-Neighbor-Degree.** *Let F be a function that assigns to each node i the maximum degree of its neighbors; i.e., $f_i(W, X) = \max_{j \in [n]_{-i}} (\sum_{r \in [n]_{-j}} w_{r,j})$.*

7. **Distance-to-Node-1.** *Let F be a function that assigns to each node i the length of its shortest path to node 1. More formally, $f_1(W, X) = 0$, and for $i \in [n]_{-1}$*

$$f_i(W, X) = \min_{(j_0,\ldots,j_\ell)\in P(i,1)} \left(\sum_{r=0}^{\ell-1} w_{j_r,j_{r+1}} \right), \tag{1}$$

where $P(i, 1)$ denotes the set of all paths starting from node i and ending in node 1.

8. **Min-Cut Value.** *Let F be a function that assigns to every node the minimum-cut of the whole graph; i.e., for all $i \in [n]$*

$$f_i(W, X) = \min_{\emptyset\subsetneq A\subsetneq[n]} \left(\sum_{r\in A} \sum_{s\in[n]\setminus A} w_{r,s} \right). \tag{2}$$

Definition 2.5 (GNN)**.** A Graph Neural Network (GNN) is an iterative mechanism that generates a sequence of functions $H^{(k)}$, for $k \geq 0$, over $\mathcal{G}_{n,d}$ in the following manner. For $G = ([n], W, X) \in \mathcal{G}_{n,d}$, the function $H^{(k)}(W, X) = (h_1^{(k)}(W, X), \ldots, h_n^{(k)}(W, X))$ is given as

$$\text{if } k = 0 : \; h_i^{(0)} = x_i, \tag{3}$$

$$\text{if } k \geq 1 : \; h_i^{(k)} = \sum_{j\in[n]_{-i}} \phi_k \left(h_i^{(k-1)}, h_j^{(k-1)}, w_{i,j} \right). \tag{4}$$

We assume that the outputs of functions ϕ_k, for $k \geq 1$, lie in some Euclidean vector space. Moreover, we say that a graph function F is generated by a GNN, if there exists a GNN and a finite $k \geq 0$ such that $H^{(k)}(W, X) = F(W, X)$ for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$.

Definition 2.5 does not restrict the function-class that ϕ_k belongs to. However, it is common in practice to choose ϕ_k from the class of multi-layer perceptrons (MLPs).

The update (4) is called the aggregation operator. It is common in the literature to consider more general aggregation operators. However, as we show in the following proposition, other choices of aggregation operators do not enlarge the class of of graph functions generated by GNNs. For a more formal statement and proof, we refer to Appendix A.

Proposition 2.6 (Informal)**.** *Suppose we replace* (4) *with an aggregation operation of the form*

$$h_i^{(k)} = \text{AGG} \left(h_i^{(k-1)}, \left\{ (h_j^{(k-1)}, w_{i,j}) \mid j \in [n]_{-i} \right\} \right).$$

Then the class of functions generated by GNNs under such an aggregation is not larger than the class of functions generated by a GNN with the aggregation defined in (4)*.*

Finally, our characterization of the class of functions generated by GNNs requires formalizing the concepts and notation related to permutations.

Definition 2.7 (Permutations)**.** (i) A permutation π over $[n]$ is a mapping $\pi : [n] \to [n]$ that is bijective. The set of all permutations over $[n]$ is denoted by S_n.

(ii) For $i \in [n]$, we use ∇_i to denote the set of all permutations π over $[n]$ such that $\pi(i) = i$. More formally, $\nabla_i = \{\pi \in S_n \mid \pi(i) = i\}$. Here, for simplicity, we have dropped the dependency of ∇_i on n.

(iii) For $i, j \in [n]$, we use $\pi_{i,j}$ to denote the specific permutation over $[n]$ that swaps i and j but fixes all the other elements. More formally, $\pi_{i,j} \in S_n$ with $\pi_{i,j}(i) = j$, $\pi_{i,j}(j) = i$, and $\pi_{i,j}(\ell) = \ell$ for $\ell \in [n] \setminus \{i, j\}$.

We next define permutations on weights and features of a graph that are induced by a permutation on the nodes.

Definition 2.8 (Induced Weight-Feature Permutation (IWFP))**.** Consider a graph $G = ([n], W, X)$ and a permutation $\pi \in S_n$ over its nodes. Then π induces a permutation σ_π over the elements of W, and a permutation λ_π over the elements of X as follows: $\sigma_\pi(W)$ is an $n \times n$ matrix whose (i, j)-th element is $w_{\pi(i),\pi(j)}$. More formally, $[\sigma_\pi(W)]_{i,j} = w_{\pi(i),\pi(j)}$. Also, given the feature matrix $X = (x_1^\top, \ldots, x_n^\top)$, we have $\lambda_\pi(X) = (x_{\pi(1)}^\top, \ldots, x_{\pi(n)}^\top)$, or equivalently $[\lambda_\pi(X)]_i = x_{\pi(i)}^\top$. For every $\pi \in S_n$, we call $(\sigma_\pi, \lambda_\pi)$, a weight-feature permutation induced by π.

3 Main Results

Warm-up. Before presenting the main results of the paper, let us consider two simple examples and see why a GNN may or may not be able to generate the correct graph functions.

Example 3.1. *Consider the graph shown in Figure 1-(a) over $n = 3$ nodes, with $x_1 = x_2 = x_3 = c \in \mathbb{R}^d$ and $w_{1,2} = w_{2,3} = w_{3,1} = 1$. Moreover, consider the function F that assigns to each node its distance to node 1 (i.e., Item 7 of Example 2.4 for $n = 3$). By letting W_0 and X_0 be the weight and feature matrices for this graph, we have $f_1(W_0, X_0) = 0$ and $f_2(W_0, X_0) = f_3(W_0, X_0) = 1$. Note that the graph structure in Example 3.1 is totally symmetric with respect to all the three nodes. Hence, any GNN obeying iterations (4) will generate identical values for all nodes; i.e., for any k we have: $h_1^{(k)} = h_2^{(k)} = h_3^{(k)}$. However, we know that $f_1(W_0, X_0) \neq f_2(W_0, X_0)$. Therefore, there is no GNN that can generate F.*

Example 3.2. *The function F defined in Item 5 of Example 2.4 can be generated by a GNN. This can be done in one iteration by letting $\phi_1(h_i^{(0)}, h_j^{(0)}, w_{i,j}) = w_{i,j}$. Hence, we have $h_i^{(1)} = \sum_{j \in [n]_{-i}} w_{i,j} = f_i$ which leads to $H^{(1)} = (h_1^{(1)}, \ldots, h_n^{(1)}) = F$.*

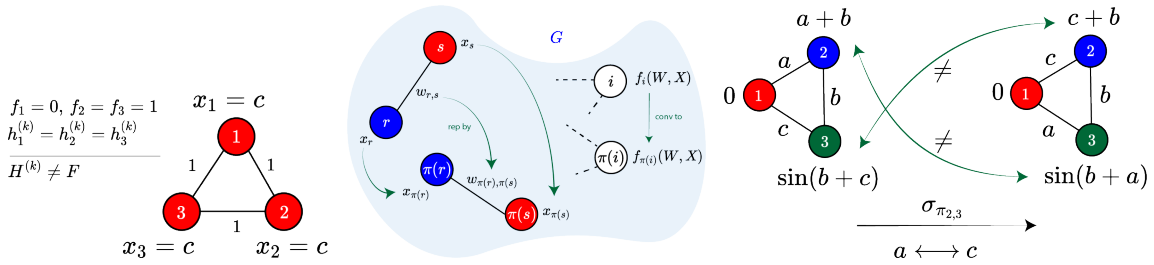
For the above toy examples, we could easily reason whether the graph functions could have been generated by a GNN. In this section, we aim to answer the same question for general graph functions, no matter how complicated they might be. To this end, let's first introduce the main algebraic structure of the paper.

Definition 3.3 (Class of permutation-compatible functions $\mathcal{F}_{n,d}$)**.** Consider a function $F = (f_1, \ldots, f_n)$ over $\mathcal{G}_{n,d}$. We say that F belongs to the class of permutation-compatible functions $\mathcal{F}_{n,d}$ if and only if for every $\pi \in S_n$, we have

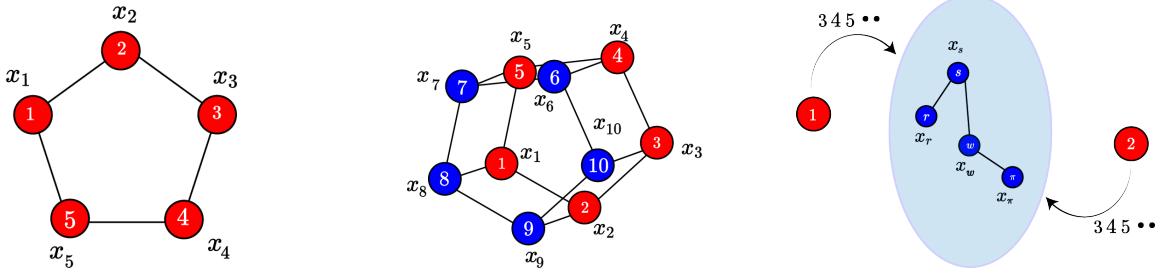
$$f_{\pi(i)}(W, X) = f_i(\sigma_\pi(W), \lambda_\pi(X)) \quad \forall i \in [n], \tag{5}$$

for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$.

We will provide in Section 3.1 several illustrative examples of functions that are permutation-compatible and incompatible. Condition (5) states that if we permute weights/features according to a permutation $\pi \in S_n$, that is replacing weights w_{rs} by $w_{\pi(r),\pi(s)}$ and features x_r by $x_{\pi(r)}$, then the function $f_i(W, X)$ will be converted to $f_{\pi(i)}(W, X)$. We refer to Figure 1 - (b) for an illustration of this condition. Having the function class $\mathcal{F}_{n,d}$, the first main result of this paper is stated as follows.



(a) Illustration of Example 3.1 (b) Illustration of Condition (5). (c) Illustration of Item 1 of Example 2.4



(d) Cycle Graph (e) Ladder Cycle Graph (f) Symmetric Nodes

Figure 1: (a) Illustration of Example 3.1 as an instance that GNNs cannot generate; (b) Illustration of (5) which states that replacing all weights $w_{r,s}$ by $w_{\pi(r),\pi(s)}$ and all features x_r by $x_{\pi(r)}$, converts $f_i(W, X)$ into $f_{\pi(i)}(W, X)$; (c) Showing that Item 1 of Example 2.4 fails to satisfy (5) for $\pi = \pi_{2,3}$. For simplicity we used $w_{1,2} = a$, $w_{2,3} = b$, and $w_{1,3} = c$; (d) A Cycle graph with 5 nodes; (e) A Ladder Cycle graph with 10 nodes and cycles of 5 nodes; (f) A visualization of two symmetric nodes (i.e. nodes 1, 2) added to an Erdös-Renyi graph. The examples (d) , (e), and (f) will be used in our numerical experiments.

.

Theorem 3.4. *A GNN over $\mathcal{G}_{n,d}$ generates a function F if and only if $F \in \mathcal{F}_{n,d}$.*

The formal proof of this theorem is provided in Appendix B. Theorem 3.4 has two directions. The necessity states that any function $H^{(k)} = (h_1^{(k)}, \ldots, h_n^{(k)})$, generated by a GNN for some $k \geq 0$, is a permutation-compatible function. This side can be formally proven by induction on k. The sufficiency result in Theorem 3.4 states that if a function F is permutation-compatible then there exists a GNN that generates it. Proving the sufficiency is far more challenging than the necessity and we refer to Section 3.2 for a sketch.

3.1 Verification of Permutation-Compatible Functions

In order to show that F lies in $\mathcal{F}_{n,d}$, one can verify Condition (5) by naively enumerating over all the elements of S_n and over all the nodes. Since S_n has $n!$ elements, this makes a total of $n \cdot n!$ constraints. It turns out that many of these constraints are redundant and can be reduced to an equivalent set of $n(n-1)/2$ constraints.

Proposition 3.5. *Consider a function $F = (f_1, \ldots, f_n)$ over $\mathcal{G}_{n,d}$. Then $F \in \mathcal{F}_{n,d}$ if and only if there exists $i_0 \in [n]$ such that both of the following conditions hold for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$:*

(i) For all $r, s \in [n]_{-i_0}$

$$f_{i_0}(\sigma_{\pi_{r,s}}(W), \lambda_{\pi_{r,s}}(X)) = f_{i_0}(W, X). \tag{6}$$

(ii) For all $j \in [n]_{-i_0}$

$$f_j(W, X) = f_{i_0}(\sigma_{\pi_{i_0,j}}(W), \lambda_{\pi_{i_0,j}}(X)). \tag{7}$$

In the following, namely Section 3.1.1 and Section 3.1.2, we go through specific examples to determine whether or not they satisfy the conditions stated in Proposition 3.5.

3.1.1 Permutation-Compatible Examples

Given Proposition 3.5, it is easy to verify that the functions given in Item 5 and Item 6 of Example 2.4 are indeed permutation-compatible functions.

Let us now consider two important special cases for which verifying permutation-compatibility is even simpler than Proposition 3.5.

Corollary 3.6. *Suppose* $F(W, X)$ *is a function over* $\mathcal{G}_{n,d}$ *that ignores* W*, i.e.,* $F(X) = (f_1(X), \ldots, f_n(X))$*. Then* $F \in \mathcal{F}_{n,d}$ *if and only if for some* $i_0 \in [n]$*:* $f_{i_0}(X) = f_{i_0}(\lambda_{\pi_{r,s}}(X))$ *for all* $r, s \in [n]_{-i_0}$ *and* $f_j(X) = f_{i_0}(\lambda_{\pi_{i_0,j}}(X))$ *for all* $j \in [n]_{-i_0}$*.*

A re-statement of Corollary 3.6 in the following form better presents the result.

Corollary 3.7. *Suppose* $F(W, X)$ *is a function over* $\mathcal{G}_{n,d}$ *that ignores* W*, i.e.,* $F(X) = (f_1(X), \ldots, f_n(X))$*. Moreover, define a function* f*, that accepts* n *inputs from* $\mathbb{R}^d$*, a quasi-permutation-invariant function if for all* $x, y_1, \ldots, y_{n-1} \in \mathbb{R}^d$*, we have* $f(x, y_{\pi(1)}, \ldots, y_{\pi(n-1)}) = f(x, y_1, \ldots, y_{n-1})$ *for all* $\pi \in S_{n-1}$*. Further, let* X_{-i} *be the sequence* $(x_1, \ldots, x_n)$ *from which* x_i *is removed. Then* $F \in \mathcal{F}_{n,d}$ *if and only if* $f_i(X) = f(x_i, X_{-i})$ *for all* $i \in [n]$*, where* f *is some quasi-permutation-invariant function.*

Corollary 3.7 is just a re-statement and an immediate result of Corollary 3.6. Using Corollary 3.7, we can immediately conclude that Items 2, 3, and 4 of Example 2.4 are permutation-compatible functions. As a side result, having a fully connected unweighted graph with features $x_1, \ldots, x_n$, any function $f(x_1, \ldots, x_n)$ that remains invariant under any permutation of $x_1, \ldots, x_n$ can be generated by a GNN. This particular case was also proven in [33] based on the analytical results from [39].

Another special case is when the graph function assigns the same value to all nodes.

Corollary 3.8. *Consider a function* $F = (f_1, \ldots, f_n)$ *over* $\mathcal{G}_{n,d}$ *and suppose* F *assigns the same value to all nodes, i.e.,* $f_i = f$ *for all* $i \in [n]$*. Moreover, assume* f *is an automorphism-invariant function, i.e., for all* $\pi \in S_n$*,* $f(W, X) = f(\sigma_\pi(W), \lambda_\pi(X))$ *for all* $G = ([n], W, X)$ *in* $\mathcal{G}_{n,d}$*. Then* $F \in \mathcal{F}_{n,d}$*.*

Remark 3.9. From Corollary 3.8 we can conclude that the min-cut value problem defined in Item 8 of Example 2.4 is permutation-compatible, and hence can be generated by a GNN. Indeed, many classical graph problems that take a graph as input, and output a value which is invariant under graph isomorphims, can be generated by GNNs. Examples include the clique number and the max-flow value.

In the following, we show graph functions that do not satisfy permutation-compatibility.

3.1.2 Permutation-Incompatible Examples

Consider the function F defined in Item 1 of Example 2.4. We claim that $F \notin \mathcal{F}_{3,d}$, i.e., F fails to satisfy (5). Let $\pi = \pi_{2,3}$, i.e., we have $\pi(1) = 1, \pi(2) = 3, \pi(3) = 2$. Let $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$. Note that W' consists of three elements $w'_{1,2}$, $w'_{1,3}$, and $w'_{2,3}$ and $X' = (x'_1, x'_2, x'_3)$. As

shown in Figure 1-(c), under the permutation σ_π on the weights, we get $w'_{1,2} = w_{1,3}$, $w'_{1,3} = w_{1,2}$, and $w'_{2,3} = w_{2,3}$, and under λ_π on the features, we get $x'_1 = x_1$, $x'_2 = x_3$, and $x'_3 = x_2$. We now apply $(\sigma_\pi, \lambda_\pi)$ on each node function as follows (note that the specific choice of F considered here totally ignores X and only depends on W):

$$f_1(\sigma_\pi(W), \lambda_\pi(X)) = f_1(W', X') = 0, \tag{8}$$

$$f_2(\sigma_\pi(W), \lambda_\pi(X)) = f_2(W', X') = w'_{1,2} + w'_{2,3} = w_{1,3} + w_{2,3}, \tag{9}$$

$$f_3(\sigma_\pi(W), \lambda_\pi(X)) = f_3(W', X') = \sin\left(w'_{1,3} + w'_{2,3}\right) = \sin\left(w_{1,2} + w_{2,3}\right). \tag{10}$$

Note that $\pi(2) = 3$ and therefore guaranteeing (5) requires $f_3(W, X) = f_2(\sigma_\pi(W), \lambda_\pi(X))$ for all W and X but this does not hold as $f_3(W, X) = \sin(w_{1,3} + w_{2,3})$ while $f_2(\sigma_\pi(W), \lambda_\pi(X)) = w_{1,3} + w_{2,3}$. Hence, $F \notin \mathcal{F}_{3,d}$.

Next, let us consider the function Distance-to-Node-1 defined in Item 7 of Example 2.4. Earlier, by using the graph given in Example 3.1 we argued that F cannot be generated by a GNN. Here, we verify this using Theorem 3.4 – i.e., we show that $F \notin \mathcal{F}_{3,d}$. Consider the matrices W and X of the graph of Example 3.1. Let $\pi = \pi_{1,2}$. As all the weights and all the features are equal, any permutation keeps W and X the same: i.e., $\sigma_\pi(W) = W$ and $\lambda_\pi(X) = X$. Suppose $F \in \mathcal{F}_{3,d}$. Condition (5) for $\pi = \pi_{1,2}$ implies that $f_{\pi(1)}(W, X) = f_1(\sigma_\pi(W), \lambda_\pi(X))$ leading to $f_2(W, X) = f_1(W, X)$ which contradicts the fact that $f_1 = 0$ and $f_2 = 1$.

The above argument can be extended to provide a general negative result on the learnability of shortest path by GNNs.

Proposition 3.10. *Let F be the "distance to node 1" function defined in Item 7 of Example 2.4 and assume some $X_0 \in \mathbb{R}^{d\times n}$ is given. Then there exists no GNN with output $H^{(k)}$ (for some finite k) for which $H^{(k)}(W, X_0) = F(W, X_0)$ for all $G = ([n], W, X_0) \in \mathcal{G}_{n,d}$. In particular, $F \notin \mathcal{F}_{n,d}$.*

The following simple corollary shows how Condition (5) constrains a single node function f_i as well as pairs of node functions f_i and f_j. This leads to a necessity condition whose failure leads to F not being permutation-compatible.

Corollary 3.11. *Consider a function $F = (f_1, \ldots, f_n)$ over $\mathcal{G}_{n,d}$ and assume $F \in \mathcal{F}_{n,d}$. Then, for any $G = ([n], W, X) \in \mathcal{G}_{n,d}$, we have:*

(i) For every $i \in [n]$

$$f_i(\sigma_\pi(W), \lambda_\pi(X)) = f_i(W, X) \quad \forall \pi \in \nabla_i. \tag{11}$$

(ii) For every $i, j \in [n]$ with $i \neq j$

$$f_j(W, X) = f_i(\sigma_{\pi_{i,j}}(W), \lambda_{\pi_{i,j}}(X)). \tag{12}$$

Note that condition (11) follows from (5) since $\forall \pi \in \nabla_i : \pi(i) = i$. Also, condition (12) follows from (5) by choosing $\pi = \pi_{i,j}$.

3.2 Characterization of $\mathcal{F}_{n,d}$ and Proof Sketch

In this section, we study a characterization of $\mathcal{F}_{n,d}$ that provides a deeper insight about what GNNs can represent, and paves the path for proving Theorem 3.4.

We start by introducing the notion of *multiset-equivalent functions* (MEF) and provide useful candidates for such functions. Based on MEFs, we define what we call a *basis* function for which we show that any permutation-compatible function, i.e., any element of $\mathcal{F}_{n,d}$, can be written in terms of this basis function. This observation eventually leads to proving the sufficient condition of Theorem 3.4.

Definition 3.12 (Multiset-Equivalent Function (MEF))**.** For positive integers n and m, we call the function $\psi : \mathbb{R}^m \to \mathbb{R}^p$ a multiset-equivalent function (MEF), if for all $v_1, \ldots, v_n, v'_1, \ldots, v'_n \in \mathbb{R}^m$, the equation

$$\sum_{i=1}^{n} \psi\left(v_i\right) = \sum_{i=1}^{n} \psi\left(v'_i\right), \tag{13}$$

holds if and only if there exists a permutation $\pi \in S_n$ such that $(v'_1, \ldots, v'_n) = (v_{\pi(1)}, \ldots, v_{\pi(n)})$. For fixed m and n, the class of all such functions is denoted by $\Psi_{m,n}$. Moreover, we refer to p, the dimension of the co-domain of the function ψ as $p(\psi)$.

The function ψ aims to generate an algebraic form for a multiset of vectors. The term "multiset" refers to a generalisation of a set in which repetition of elements is permitted. The name multiset-equivalent function for ψ in Definition 3.12, relates to the fact that the summation of ψ over a sequence of vectors preserves all the data up to a permutation and thus, this sum is equivalent to the "multiset" of them. It is not trivial to find a MEF. For instance, note that the identity function (which leads to $\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} v'_i$) is not a MEF for $n > 1$. This is because when $m = 1$ and $n = 2$, we have $2 + 5 = 3 + 4$ but $(2, 5)$ is not a permutation of $(3, 4)$. The function ψ is basically a tool to "translate" multiset of vectors to algebraic summations of them. A natural question here is whether such function exists at all. The following proposition introduces candidate elements of $\Psi_{m,n}$ for all positive integers n and m.

Proposition 3.13. *The followings are specific constructions of MEFs for (i) $m = 1$, and (ii) $m > 1$:*

(i) Consider the function $\psi : \mathbb{R} \to \mathbb{R}^n$ such that for $v \in \mathbb{R}$, $\psi(v) = (v, v^2, \ldots, v^n)$. Then $\psi \in \Psi_{1,n}$. Moreover, note that $p(\psi) = n$.

(ii) Let $m > 1$ and consider $v = (v_1, \ldots, v_m) \in \mathbb{R}^m$. Define $\psi(v) = \psi(v_1, \cdots, v_m)$ to be an $n \times m \times m$ array (tensor) with real elements such that for every $\ell \in [n]$ and $r, s \in [m]$ with $r < s$

$$[\psi(v_1, \cdots, v_m)]_{\ell,r,s} = \mathrm{Re}\left(\left(v_r + v_s\sqrt{-1}\right)^{\ell}\right), \tag{14}$$

$$[\psi(v_1, \cdots, v_m)]_{\ell,s,r} = \mathrm{Im}\left(\left(v_r + v_s\sqrt{-1}\right)^{\ell}\right). \tag{15}$$

Note that $\psi(v) \in \mathbb{R}^{n \times m \times m}$. Let us re-shape $\psi(v)$ into a long vector in $\mathbb{R}^{m^2 n}$, and consider $\psi : \mathbb{R}^m \to \mathbb{R}^{m^2 n}$. Then $\psi \in \Psi_{m,n}$. Moreover, note that $p(\psi) = m^2 n$.

To construct valid MEFs, the idea behind specific formula of ψ in Proposition 3.13 for $m = 1$ is that when ψ is applied to the sum of n scalars, it encodes them into the roots of a unique polynomial, and thus it preserves the data up to a permutation. For $m > 1$, this idea is extended by encoding vectors of arbitrary size into the roots of a system of complex polynomials.

In the following, we construct a basis function over $\mathcal{G}_{n,d}$ through MEFs.

Definition 3.14 (Basis Function)**.** Define the graph function $\mathcal{B} = (\beta_1, \ldots, \beta_n)$ over $\mathcal{G}_{n,d}$ such that for $i \in [n]$

$$\beta_i\left(W, X\right) = \left(x_i, \sum_{j \in [n]_{-i}} \psi_2\left(x_j, w_{i,j}, \sum_{\ell \in [n]_{-j}} \psi_1\left(w_{j,\ell}\right)\right)\right), \tag{16}$$

where $\psi_1 \in \Psi_{1,n-1}$ and $\psi_2 \in \Psi_{p(\psi_1)+d+1,n-1}$ is the same for all $i \in [n]$. We call $\mathcal{B}$ a basis function over $\mathcal{G}_{n,d}$.

This specific structure of a basis function leads to an important property which is stated and formally proven in the following proposition.

Proposition 3.15. *Let $\mathcal{B} = (\beta_1, \dots, \beta_n)$ be a basis function over $\mathcal{G}_{n,d}$. Then $\mathcal{B} \in \mathcal{F}_{n,d}$ with the following additional property: For every $i \in [n]$, if $\beta_i(W, X) = \beta_i(W', X')$ then there exists $\pi \in \nabla_i$ such that $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$.*

The fact that ψ_1 and ψ_2 are MEFs is crucial in showing that the specific structure of the basis function in (16) leads to Proposition 3.15. We omit the details here and refer to the proof of Proposition 3.15 in Appendix F.

A consequence of Proposition 3.15 is the following result which states that any node function f_i of a permutation-compatible function F can be written in terms of β_i.

Theorem 3.16. *Suppose $F \in \mathcal{F}_{n,d}$ with $F = (f_1, \dots, f_n)$ and let $\mathcal{B} = (\beta_1, \dots, \beta_n)$ be a basis function over $\mathcal{G}_{n,d}$. Then, there exists a function ρ such that for every $i \in [n]$, the following holds for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$:*

$$f_i(W, X) = \rho\left(\beta_i(W, X)\right). \tag{17}$$

The sufficiency of Theorem 3.4 follows from Theorem 3.16. To this end, suppose $F = (f_1, \dots, f_n) \in \mathcal{F}_{n,d}$ is given. Then consider some basis function over $\mathcal{G}_{n,d}$ of the form given in (16) with some MEFs ψ_1 and ψ_2. For instance, one can use candidates introduced in Proposition 3.13. Having these, Theorem 3.16 ensures the existence of a function ρ satisfying (17). Note that due to (17), f_i's can be written in terms of ψ_1, ψ_2, and ρ. To show that $F = (f_1, \dots, f_n)$ is GNN-generatable, we directly construct a GNN with 3 iterations. For such a construction, it suffices to provide specific candidates for functions ϕ_1, ϕ_2, ϕ_3, defined in Definition 2.5. In the first iteration, we consider

$$\phi_1\left(h_j^{(0)}, h_\ell^{(0)}, w_{j,\ell}\right) = \left(\frac{1}{n-1} h_j^{(0)},\ \psi_1\left(w_{j,\ell}\right)\right). \tag{18}$$

In the second iteration, we set ϕ_2 as follows.

$$\phi_2\left(h_i^{(1)}, h_j^{(1)}, w_{i,j}\right) = \left(\frac{1}{n-1}\left[h_i^{(1)}\right]_{1:d}, \psi_2\left(\left[h_j^{(1)}\right]_{1:d}, w_{i,j}, \left[h_j^{(1)}\right]_{d+1:d+p(\psi_1)}\right)\right). \tag{19}$$

In Equation (19), the notation $[v]_{a:b}$ for $v = (v_1, \dots, v_m)$ means $[v]_{a:b} = (v_a, v_{a+1}, \dots, v_b)$. Finally, in the third iteration, we let ϕ_3 be

$$\phi_3(h_i^{(2)}, h_j^{(2)}, w_{i,j}) = \frac{1}{n-1}\rho\left(h_i^{(2)}\right). \tag{20}$$

It is straightforward to show that the choices (18), (19), and (20) lead to $h_i^{(3)}(W, X) = \rho(\beta_i(W, X)) = f_i(W, X)$. Hence, $H^{(3)} = F$ which concludes the sufficient condition of Theorem 3.4.

Theorem 3.16 provides an insight that can potentially be used to prove other results. For instance, using Theorem 3.16 helps to translate the constraints on ϕ_k to constraints on F. For instance, knowing the fact that multi-layer perceptrons (MLPs) are good approximates to continuous functions, one can restrict the functions ϕ_k to be continuous and ask what function class will be generated accordingly. This is answered in the following result.

Corollary 3.17. *If $F(W, X)$ in Theorem 3.4 is continuous with respect to (W, X), then F can be generated by a GNN with continuous functions ϕ_k.*

4 Experiments

In this section, we provide experimental results on two canonical graph problems, namely, the shortest path length (as an example of a permutation-incompatible graph function) and min-cut value (as an example of a permutation-compatible graph function). Our code is available online[1].

Settings: Our main dataset consists four classes of graphs: (i) Erdös-Renyi graphs (ERG); (ii) ERG ensemble with two additional symmetric nodes (ERGS), where these two nodes are randomly connected to the exact same nodes in the original Erdös-Renyi graph, i.e., one is the copy of the other one as shown in Figure 1-(f). We refer to this case as "symmetric nodes" in the plots; (iii) Cycle graphs (CG) with size n, shown in Figure 1-(d); and (iv) Ladder Cycle graphs (LCG), shown in Figure 1-(e). We study two main functions: (i) shortest path length analogous to Item 7 of Example 2.4 and (ii) minimum cut value analogous to Item 8 of Example 2.4. In both cases, we investigate random and identical initialization.

To build an ERG, we first fix the number of nodes to $n = 100$ and connect any two nodes i and j independently with probability 0.1. We build up between 200 to $10k$ ERG samples for training. Similarly, for ERGS we first create ERG with size $n - 2$ and then connect a random subset of nodes to the nodes $n - 1$ and n. Hence, these two nodes become totally symmetric with respect to the remaining of the graph. Finally, a CG is created by linking all n nodes in a row and accordingly LCGs are constructed by linking mirrored nodes from a CG with depth k to CG with depth $k + 1$.

We consider both random and identical initializations. In the case of identical initialization, we assign to all nodes, the same vector which is arbitrarily chosen, e.g., the vector of all-zeros or all-ones. In the case of random initialization, we assign to each node a vector of independent random elements, which are also independent from other initialization vectors in the graph. Further, each element of the vector is chosen uniformly at random between 0 and 1. It is important to note that under random initialization for the ERGS case, we generate a random vector and assign it to both of the aforementioned symmetric nodes.

To train GNN – for both Shortest-Path Length and Min-Cut Value functions – we use a basic architecture of graph convolutional networks with two hidden layers and define the output channel based on the number of classes linked with the problem. We choose the dimension d of each node feature from the set $\{20, 40, 60, 80, 100\}$ based on cross-validation on a left out dataset. In addition, we use the ADAM optimizer – with learning rate 0.01, weight decay $5e-4$, and negative log likelihood loss –, and repeat training for many epochs including $\{100, 200, 500, 1000, 5000\}$. Batch sizes, indicating the number of graphs in one single message passing, is also determined based on cross-validation ranging among $\{16, 32, 64, 128\}$.

Minimum-cut value problem: We run Ford–Fulkerson method [9] on all examples to obtain the min-cut value of the whole graph and then train a GNN to compute the min-cut value function. As shown in Figure 2, it can be observed that the training error has a decreasing manner which demonstrates the progress in training. Accordingly, GNN replicates exact values derived from Ford-Fulkerson (i.e., accuracy $\geq 85\%$) in all ERG, ERGS, CG, and LCG cases under both random and identical initializations (see Figure 3(a) and Figure 3(b), respectively).

Shortest-path length problem: To compute the shortest-path length to one node of the graph, we run the Bellman-Ford dynamic program [3]. Similarly, we train a GNN, aiming to learn the shortest-path length function. Here, we observe that the training error exhibits a constant loss as shown in Figure 2 which demonstrates the inability of the GNN to make progress in training. Furthermore, the training accuracy using random initialization is as low as 71% for ERG, 0% for ERGS, 32% for

[1] `https://github.com/dadashkarimi/gnn`

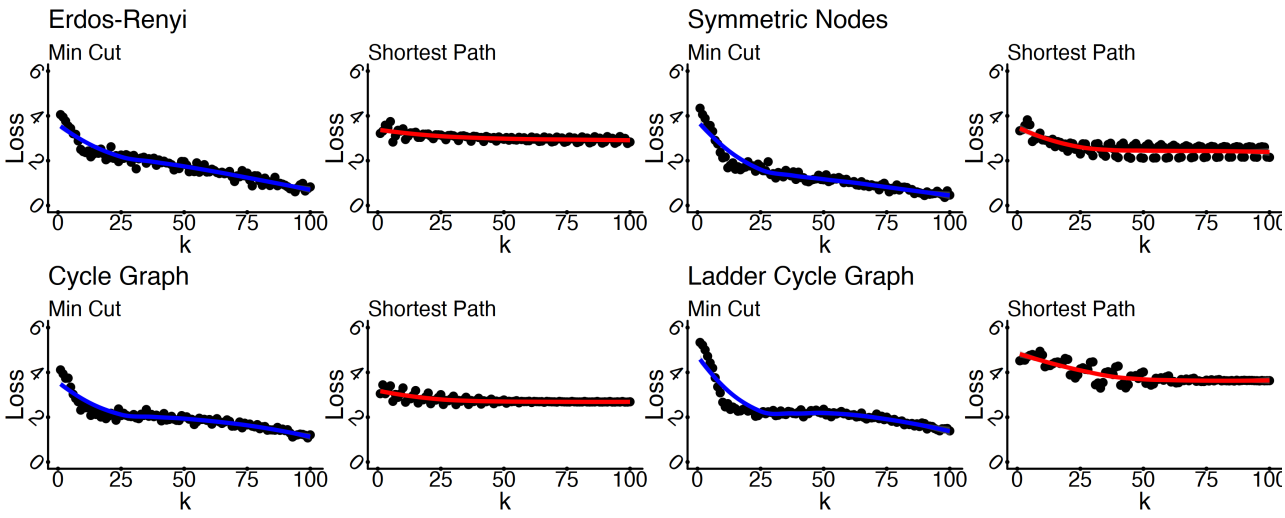


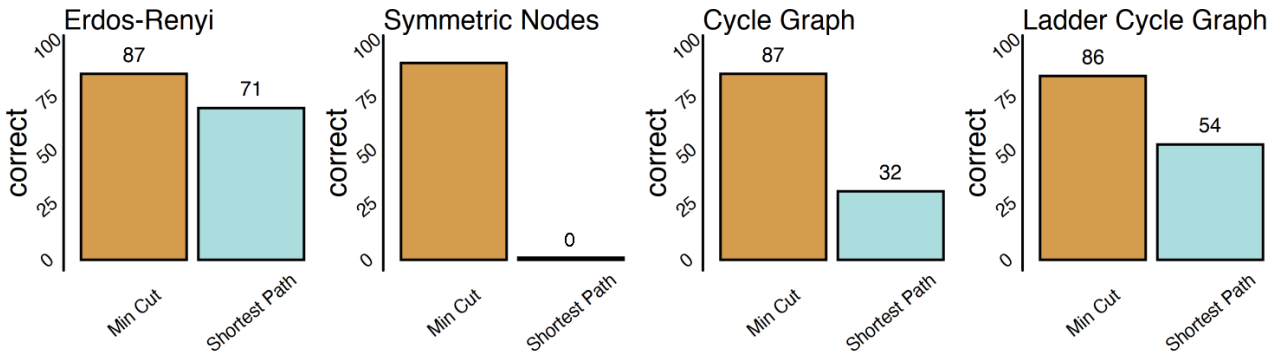
Figure 2: Training error over 100 iterations of GNN using a training set with size 200. We observe decreasing curves for the Min-Cut Value and steady curves for Shortest-Path Length problems for different examples.

CG, and 54% for LCG (see Figure 3(a)). When we set all the features identical, the training accuracy further drops to 69% for ERG, again 0% for ERGS, 1% for CG, and 1% for LCG (see Figure 3(b)). In the ERGS case, i.e., the Erdös-Renyi graph over $n-2$ nodes plus two additional symmetric nodes, the length of the shortest path to one of the symmetric nodes is computed and the output of the GNN is evaluated depending on whether GNN predicts true labels for both symmetric nodes, simultaneously. In other cases, the shortest-path length to node 1 is computed.

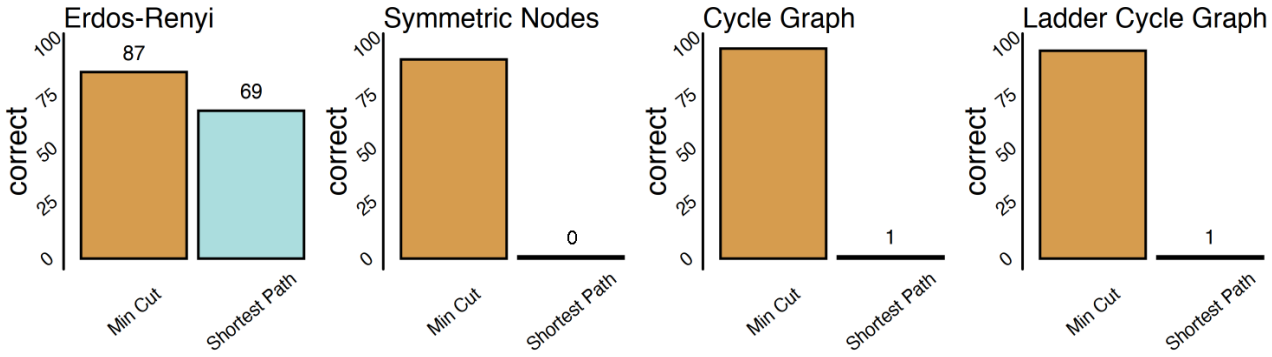
Parameter sensitivity: Here we study the effect of free parameters including (i) number of iterations and (ii) training size on GNN outputs (see Figure 4). Each plot exhibits accuracy of a trained GNN associated with Min-Cut Value (the first and third plots) and Shortest-Path Length problem (the second and fourth plots) using appropriate parameters. The results range between 0 (center) to 1 (outermost). As illustrated, a GNN with $5k$ iterations yields Min-Cut Value equally well across all examples (i.e., accuracy $= 1$). Similar results hold for the training size. Yet, the same GNN architecture is unable to build Shortest-Path Length function for any of the examples over all ranges of parameters.

Alignment of experimental results with theory: We know from the theory that Min-Cut Value is a permutation-compatible (see Remark 3.9) and the Shortest-Path Length is a permutation-incompatible function (see Proposition 3.10). Being permutation-compatible, due to Theorem 3.4, means that a graph function can be represented by a GNN defined in Definition 2.5. This implies that under ideal choice of functions ϕ_k, the resulting GNN equals to the graph function over all graphs of the corresponding size n and feature-dimension d, i.e., 100% training accuracy over a training set including all such graphs. Note that this result is not about the generalization ability of GNNs and thus the performance of GNNs in terms of test error is not of interest here. We seek to evaluate the training accuracy over sufficiently large training sets. Hence, in the Min-Cut Value case, high training accuracy (Figure 3) and decreasing loss-curves (Figure 2) align with permutation-compatibility of this function.

The case of Shortest-Path Length needs further explanation. Note that the motivation to use GNN towards generating the output of a dynamic program (DP) (e.g., generating the shortest-path length as the output of Bellman-Ford dynamic program) is only justifiable if GNN can generate the



(a) Random initialization.



(b) Identical initialization.

Figure 3: Training accuracy of GNN for Min-Cut Value problem and the Shortest-Path Length problem under; (a) random initialization for node features and (b) identical initialization for node features. In Shortest-Path Length over ERGS (Erdös-Renyi over $n-2$ nodes plus two additional symmetric nodes), accuracy is determined by whether both symmetric nodes are correctly classified or not, whereas for the other graph examples, it is the average over all nodes.

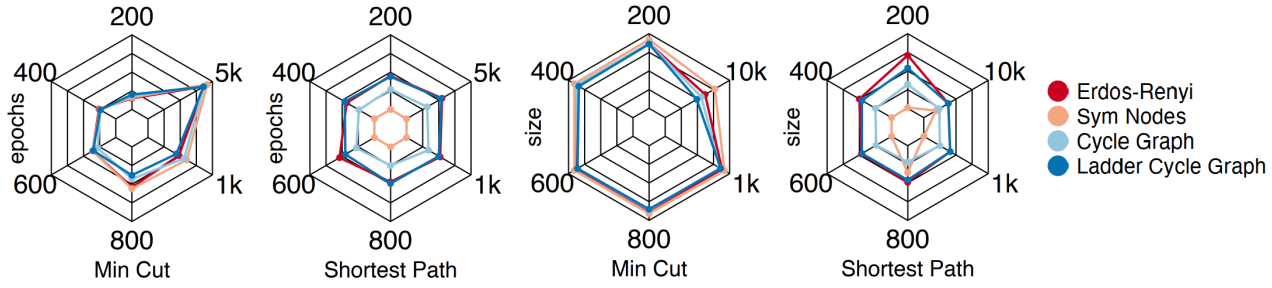


Figure 4: Parameter sensitivity to the number of iterations (the first and second plots) and parameter sensitivity to the training size (the third and fourth plots) addressing the accuracy of GNN ranging between 0 (centers) and 1 (outermost).

output regardless of the initialization setup. Otherwise, one would use the original DP with the appropriate initialization and no need for a GNN to get involved. Following this line of reasoning, Proposition 3.10 shows that for any initialization, GNN fails to represent the shortest path on some graphs. This, in particular, implies that the training accuracy cannot reach 100% under initialization methods that are not aware of the original DP in advance (e.g., identical or random initializations).

These theoretical understandings are also in alignment with the steady loss-curves (Figure 2) and low training accuracy results (Figure 3) for Shortest-Path Length function. As the next step, let us explain extremely low accuracy results in EGRS, CG, and LCG cases. In any graph, we call two nodes symmetric if exchanging their names results in a graph that is isomorphic to the the original graph. This, in particular, requires the identity of the feature vectors of these two nodes. In any graph with two symmetric nodes 1 and 2, considering the permutation $\pi_{1,2}$ and the corresponding IWFP $(\sigma_{\pi_{1,2}}, \lambda_{\pi_{1,2}})$, we can conclude that a permutation-compatible function (in particular, the output of a GNN) produces identical values on nodes 1 and 2. In the ERGS case, this implies that the GNN cannot produce true (different) labels for both symmetric nodes, simultaneously, leading to 0% accuracy (see Figure 3(a) and Figure 3(b)). Similarly, in the CG case, GNN can produce at most two true labels, and in LCG, at most four true labels under identical initialization. However, under random initialization, these accuracy rates may go higher but always away from 100% as discussed above. All these results are verified in the experiments (see Figure 3). The whole discussion above also highlights the misalignment between DP and GNN.

Conclusion

In this paper, we provided a necessary and sufficient algebraic condition, dubbed *permutation-compatibility*, that implies which graph problems can be generated by GNNs. We also showed that for any graph problem, permutation-compatibility can by verified be checking quadratically many constraints and elaborated this concept through many examples and special cases. Further, we characterized the basis functions for the class of permutation-compatible graph functions which effectively relate the properties of aggregation operators to the expressive power of the resulting GNNs. Finally, we supported our findings through empirical evaluations on two canonical graph problems, namely, the length of shortest paths and min-cut value. Compatible with our theory, we observed that GNNs on very simple graphs completely failed to compute the length shortest paths. In contrast, the same GNN architectures could successfully compute the min-cut value.

References

[1] Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, USA, 1st edition, 2009.

[2] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction Networks for Learning About Objects, Relations and Physics. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[3] Richard Bellman. On a Routing Problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[4] P. B. Bhattacharya, S. K. Jain, and S. R. Nagpaul. *Basic Abstract Algebra*. Cambridge University Press, 2 edition, 1994.

[5] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the Equivalence Between Graph Isomorphism Testing and Function Approximation with GNNs. *CoRR*, abs/1905.12560, 2019.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

[7] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[8] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking Graph Neural Networks. *arXiv preprint arXiv:2003.00982*, 2020.

[9] Lester Randolph Ford and Delbert R Fulkerson. Maximal Flow Through a Network. *Canadian journal of Mathematics*, 8:399–404, 1956.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural Message Passing for Quantum Chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[12] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular Graph Convolutions: Moving Beyond Fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[13] Thomas N. Kipf and Max Welling. Semi-Supervised Classification With Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR*, 2017.

[14] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. In *Advances in Neural Information Processing Systems: NeurIPS33*, volume 33, pages 6–12, 2020.

[15] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations, ICLR*, 2016.

[16] Tengfei Ma, Jie Chen, and Cao Xiao. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. In *Advances in Neural Information Processing Systems*, 2018.

[17] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably Powerful Graph Networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[18] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and Equivariant Graph Networks. In *International Conference on Learning Representations*, 2019.

[19] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the Universality of Invariant Networks. In *International conference on machine learning*, pages 4363–4371. PMLR, 2019.

[20] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[21] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs. In *International Conference on Learning Representations*, 2019.

[22] Adam Santoro, Felix Hill, David Barrett, Ari Morcos, and Timothy Lillicrap. Measuring Abstract Reasoning in Neural Networks. In *International Conference on Machine Learning*, pages 4477–4486, 2018.

[23] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random Features Strengthen Graph Neural Networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

[24] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[25] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational Capabilities of Graph Neural Networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2009.

[26] Balasubramaniam Srinivasan and Bruno Ribeiro. On the Equivalence Between Positional Node Embeddings and Structural Graph Representations. In *International Conference on Learning Representations*, 2020.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

[28] Saurabh Verma and Zhi-Li Zhang. Graph Capsule Convolutional Neural Networks. *arXiv preprint arXiv:1805.08090*, 2018.

[29] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge Graph Attention Network for Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 950–958, 2019.

[30] Boris Weisfeiler and Andrei Leman. The Reduction of a Graph to Canonical Form and the Algebra Which Appears Therein. *NTI, Series*, 2(9):12–16, 1968.

[31] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? In *International Conference on Learning Representations*, 2019.

[32] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[33] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What Can Neural Networks Reason About? In *International Conference on Learning Representations*, 2020.

[34] Xu Yang, Kaihua Tang, Hanwang Zhang, and Jianfei Cai. Auto-Encoding Scene Graphs for Image Captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10685–10694, 2019.

[35] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

[36] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-Aware Graph Neural Networks. *arXiv preprint arXiv:2101.10320*, 2021.

[37] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-Aware Graph Neural Networks. In *International Conference on Machine Learning*, pages 7134–7143. PMLR, 2019.

[38] Lingfei Wu Yu Chen and Mohammed J. Zaki. Reinforcement Learning Based Graph-to-Sequence Model for Natural Question Generation. In *8th International Conference on Learning Representations, ICLR*, 2020.

[39] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[40] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An End-to-End Deep Learning Architecture for Graph Classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Notation for Appendix

Definition .1 (Multiset)**.** Multisets generalize the concept of a set in which the repetition of elements is allowed. A multiset is a pair $Y = (S, m)$, where S is the underlying set of the distinct elements of Y and $m : S \to \mathbb{Z}_{\geq 1}$ is the function that indicates the multiplicity of each element.

Definition .2 (Notation $\#\{\cdot\}$)**.** Suppose $y_1, \ldots, y_n$ are some objects with possibly repeated elements, then the multiset containing $y_1, \ldots, y_n$ is denoted by $\#\{y_1, \ldots, y_n\}$. More formally, if the set of distinct elements among $y_1, \ldots, y_n$ is $\{y_{i_1}, \ldots, y_{i_r}\}$ with $m(y_{i_1}), \ldots, m(y_{i_n})$ denoting the multiplicity of the elements, then

$$\#\{y_1, \ldots, y_n\} = (\{y_{i_1}, \ldots, y_{i_r}\}, m). \tag{21}$$

The notation $\#\{\cdot\}$ considers the repetition but ignores the order of the elements. Therefore, for the sequences $(y_1, \ldots, y_n)$ and $(z_1, \ldots, z_m)$ of possibly repeated elements, the equation

$$\#\{y_1, \ldots, y_n\} = \#\{z_1, \ldots, z_m\} \tag{22}$$

holds if and only if $m = n$ and there exists a permutation $\pi \in S_n$ such that $(y_1, \ldots, y_n) = \left(z_{\pi(1)}, \ldots, z_{\pi(n)}\right)$.

Notation $f \circ g$. We denote by $f \circ g$, the combination of functions f and g, meaning that $f \circ g(x) = f(g(x))$.

A Formal Statement and Proof of Proposition 2.6

To compare the effect of different aggregations, we define Extended-GNN analogous to Definition 2.5 as follows.

Definition A.1 (Extended-GNN)**.** An Extended Graph Neural Network (Extended-GNN) is an iterative mechanism that generates a sequence of functions $H^{(k)}$, $k \geq 0$, over $\mathcal{G}_{n,d}$ in the following manner. For $G = ([n], W, X) \in \mathcal{G}_{n,d}$, the function $H^{(k)}(W, X) = (h_1^{(k)}(W, X), \ldots, h_n^{(k)}(W, X))$ is given as

$$\text{if } k = 0 : \; h_i^{(0)} = x_i, \tag{23}$$

$$\text{if } k \geq 1 : \; h_i^{(k)} = \Phi_k \left(h_i^{(k-1)}, \# \left\{ (h_j^{(k-1)}, w_{i,j}) \mid j \in [n]_{-i} \right\} \right), \tag{24}$$

for some functions Φ_k, $k \geq 1$, where for each k, the outputs of Φ_k lie in some Euclidean vector space. Moreover, we say that a graph function F is generated by a Extended-GNN, if there exists a Extended-GNN and a finite $k \geq 0$ such that $H^{(k)}(W, X) = F(W, X)$ for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$.

Proposition A.2 (Formal Version of Proposition 2.6)**.** *Function F over $\mathcal{G}_{n,d}$ can be generated by a GNN if and only if it can be generated by an Extended-GNN.*

Proof. Suppose $h_i^{(k)} \in \mathbb{R}^{\alpha_k}$ for $k \geq 0$ and fix some MEF $\psi_k \in \Psi_{\alpha_{k-1}+1, n-1}$, defined in Definition 3.12. Also note that candidates for MEFs are provided in Proposition 3.13. As the first step, we claim that there exists a function Θ_k such that

$$\Phi_k \left(h_i^{(k-1)}, \# \left\{ (h_j^{(k-1)}, w_{i,j}) \mid j \in [n]_{-i} \right\} \right) = \Theta_k \left(h_i^{(k-1)}, \sum_{j \in [n]_{-i}} \psi_k \left(h_j^{(k-1)}, w_{i,j} \right) \right), \tag{25}$$

over all valid inputs. To prove (25), it suffices to show that having

$$\left(h_i^{(k-1)}, \sum_{j\in[n]_{-i}} \psi_k\left(h_j^{(k-1)}, w_{i,j}\right)\right) = \left(\hat{h}_i^{(k-1)}, \sum_{j\in[n]_{-i}} \psi_k\left(\hat{h}_j^{(k-1)}, \hat{w}_{i,j}\right)\right) \tag{26}$$

leads to

$$\Phi_k\left(h_i^{(k-1)}, \#\left\{(h_j^{(k-1)}, w_{i,j}) \mid j\in[n]_{-i}\right\}\right) = \Phi_k\left(\hat{h}_i^{(k-1)}, \#\left\{(\hat{h}_j^{(k-1)}, \hat{w}_{i,j}) \mid j\in[n]_{-i}\right\}\right). \tag{27}$$

To show that (26) leads to (27), note that from (26), we have $h_i^{(k-1)} = \hat{h}_i^{(k-1)}$, and

$$\sum_{j\in[n]_{-i}} \psi_k\left(h_j^{(k-1)}, w_{i,j}\right) = \sum_{j\in[n]_{-i}} \psi_k\left(\hat{h}_j^{(k-1)}, \hat{w}_{i,j}\right). \tag{28}$$

Having (28), the definition of an MEF (see Definition 3.12) implies that

$$\#\left\{(h_j^{(k-1)}, w_{i,j}) \mid j\in[n]_{-i}\right\} = \#\left\{(\hat{h}_j^{(k-1)}, \hat{w}_{i,j}) \mid j\in[n]_{-i}\right\}. \tag{29}$$

Equation (29) together with $h_i^{(k-1)} = \hat{h}_i^{(k-1)}$ proves (27). Hence, we showed that there exists a function Θ_k satisfying (25). As the second step, in what follows, we show that having (25) proves the statement. Note that (4) is a specific case of (24) and thus any function that is generated by a GNN can be generated by an Extended-GNN as well. Now, suppose $\hat{H}^{(L)} = (\hat{h}_1^{(L)} \dots, \hat{h}_n^{(L)})$ is generated by an Extended-GNN using $\Phi_1, \dots, \Phi_L$. Let $\psi_1, \dots, \psi_L$ and $\Theta_1, \dots, \Theta_L$ be the corresponding functions according to (25). Having this, we construct a GNN with $2L$ iterations that generates $H^{(2L)}$ and show that $H^{(2L)} = \hat{H}^{(L)}$. To this aim, for $r \in [L]$, define

$$\phi_{2r-1}\left(h_i^{(2r-2)}, h_j^{(2r-2)}, w_{i,j}\right) = \left(\frac{1}{n-1} h_i^{(2r-2)}, \psi_r\left(h_j^{(2r-2)}, w_{i,j}\right)\right), \tag{30}$$

$$\phi_{2r}\left(h_i^{(2r-1)}, h_j^{(2r-1)}, w_{i,j}\right) = \frac{1}{n-1}\Theta_k\left(h_i^{(2r-1)}\right). \tag{31}$$

Due to (30) and (31), for all $i \in [n]$, we have

$$h_i^{(2r-1)} = \sum_{j\in[n]_{-i}} \phi_{2r-1}\left(h_i^{(2r-2)}, h_j^{(2r-2)}, w_{i,j}\right) = \left(h_i^{(2r-2)}, \sum_{j\in[n]_{-i}} \psi_r\left(h_j^{(2r-2)}, w_{i,j}\right)\right), \tag{32}$$

$$h_i^{(2r)} = \sum_{j\in[n]_{-i}} \frac{1}{n-1}\Theta_k\left(h_i^{(2r-1)}\right) = \Theta_k\left(h_i^{(2r-2)}, \sum_{j\in[n]_{-i}} \psi_r\left(h_j^{(2r-2)}, w_{i,j}\right)\right). \tag{33}$$

Therefore,

$$h_i^{(2r)} = \Phi_k\left(h_i^{(2r-2)}, \#\left\{(h_j^{(2r-2)}, w_{i,j}) \mid j\in[n]_{-i}\right\}\right). \tag{34}$$

Note that (34) implies that even iterations of the introduced GNN coincide the iterations of the Extendedn-GNN. Next, we claim that $h_i^{(2r)} = \hat{h}_i^{(r)}$ for all $r \in [L]$. We prove this by induction on r. For $r = 0$, it is given by definition of GNNs and Extended-GNNs. Assuming it is true for $r-1$, we have $h_i^{(2r-2)} = \hat{h}_i^{(r-1)}$. Replacing this in (34) leads to $h_i^{(2r)} = \hat{h}_i^{(r)}$. Hence, we showed that $h_i^{(2r)} = \hat{h}_i^{(r)}$ for all $i \in [n]$ and all $r \in [L]$. In particular, $h_i^{(2L)} = \hat{h}_i^{(L)}$ for all $i \in [n]$ which gives $H^{(2L)} = \hat{H}^{(L)}$. □

B Proof of Theorem 3.4

Proof. **Necessity**. Suppose a GNN over $\mathcal{G}_{n,d}$ generates the function $H^{(k)}(W,X) = (h_1^{(k)}(W,X), \ldots, h_n^{(k)}(W,X))$ for some $k \geq 0$. We want to show that $H^{(k)} \in \mathcal{F}_{n,d}$. To this aim, we must show that for every given $\pi \in S_n$ the following holds for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$:

$$h_{\pi(i)}^{(k)}(W,X) = h_i^{(k)}\left(\sigma_\pi(W), \lambda_\pi(X)\right). \tag{35}$$

We prove the statement by induction on k. For $k = 0$, we have $h_i^{(0)}(W,X) = x_i$ for all i. Hence,

$$h_i^{(0)}\left(\sigma_\pi(W), \lambda_\pi(X)\right) = x_{\pi(i)} = h_{\pi(i)}^{(0)}\left(W,X\right). \tag{36}$$

Suppose the induction hypothesis holds for $k-1$, we must show that it holds for k. From (4), we have

$$\begin{aligned}
h_{\pi(i)}^{(k)}(W,X) &= \sum_{\ell \in [n]_{-\pi(i)}} \phi_k\left(h_{\pi(i)}^{(k-1)}\left(W,X\right), h_\ell^{(k-1)}\left(W,X\right), w_{\pi(i),\ell}\right) && (37)\\
&= \sum_{j \in [n]_{-i}} \phi_k\left(h_{\pi(i)}^{(k-1)}\left(W,X\right), h_{\pi(j)}^{(k-1)}\left(W,X\right), w_{\pi(i),\pi(j)}\right) && (38)\\
&= \sum_{j \in [n]_{-i}} \phi_k\left(h_i^{(k-1)}\left(\sigma_\pi(W), \lambda_\pi(X)\right), h_j^{(k-1)}\left(\sigma_\pi(W), \lambda_\pi(X)\right), w_{\pi(i),\pi(j)}\right) && (39)\\
&= h_i^{(k)}\left(\sigma_\pi(W), \lambda_\pi(X)\right), && (40)
\end{aligned}$$

where equation (38) is obtained by using the fact that π is injective and surjective and putting $j = \pi^{-1}(\ell)$, equation (39) is due to the induction hypothesis for $k-1$, and equation (40) follows from definition of $h_i^{(k)}$ in (4).

Sufficiency. Suppose $F \in \mathcal{F}_{n,d}$ and let $F(W,X) = (f_1(W,X), \ldots, f_n(W,X))$. Consider a basis function $\mathcal{B}(W,X) = (\beta_1(W,X), \ldots, \beta_n(W,X))$ over $\mathcal{G}_{n,d}$. Due to Theorem 3.16, we can conclude that there exists a function ρ such that for every $i \in [n]$, the following holds for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$

$$f_i(W,X) = \rho\left(\beta_i(W,X)\right). \tag{41}$$

Now, we introduce a GNN with three iterations, i.e., we introduce functions ϕ_1, ϕ_2, ϕ_3 such that $h_i^{(3)} = f_i(W,X)$ for all $i \in [n]$ and as a result, $H^{(3)} = F$. First consider the definition of $\beta_i(W,X)$ in Definition 3.14 and re-write (41) as

$$f_i(W,X) = \rho\left(\beta_i(W,X)\right) = \rho\left(x_i, \sum_{j \in [n]_{-i}} \psi_2\left(x_j, w_{i,j}, \sum_{\ell \in [n]_{-j}} \psi_1\left(w_{j,\ell}\right)\right)\right). \tag{42}$$

For $k = 1$, define the function ϕ_1 of the GNN as

$$\phi_1\left(h_j^{(0)}, h_\ell^{(0)}, w_{j,\ell}\right) = \left(\frac{1}{n-1} h_j^{(0)},\ \psi_1\left(w_{j,\ell}\right)\right). \tag{43}$$

This leads to the following formula for all $j \in [n]$:

$$h_j^{(1)} = \sum_{\ell \in [n]_{-j}} \phi_1\left(h_j^{(0)}, h_\ell^{(0)}, w_{j,\ell}\right) = \sum_{\ell \in [n]_{-j}} \left(\frac{1}{n-1} h_j^{(0)},\ \psi_1\left(w_{j,\ell}\right)\right) = \left(x_j, \sum_{\ell \in [n]_{-j}} \psi_1\left(w_{j,\ell}\right)\right). \tag{44}$$

To introduce ϕ_2 and ϕ_3, we need to set a notation. Having a vector $v = (v_1, \ldots, v_m)$, by $[v]_{a:b}$ for $a \leq b$, we denote the sub-vector consisting of the elements with indices starting from a to b, that is, $[v]_{a:b} = (v_a, v_{a+1}, \ldots, v_b)$. Moreover, $[v]_{a:\text{end}} = [v]_{a:m}$. Using this notation, for $k = 2$, define the function ϕ_2 of the GNN as

$$\phi_2\left(h_i^{(1)}, h_j^{(1)}, w_{i,j}\right) = \left(\frac{1}{n-1}\left[h_i^{(1)}\right]_{1:d},\ \psi_2\left(\left[h_j^{(1)}\right]_{1:d}, w_{i,j}, \left[h_j^{(1)}\right]_{d+1:\text{end}}\right)\right). \tag{45}$$

Hence,

$$h_i^{(2)} = \sum_{j\in[n]_{-i}} \phi_2\left(h_i^{(1)}, h_j^{(1)}, w_{i,j}\right) = \sum_{j\in[n]_{-i}} \left(\frac{1}{n-1}\left[h_i^{(1)}\right]_{1:d},\ \psi_2\left(\left[h_j^{(1)}\right]_{1:d}, w_{i,j}, \left[h_j^{(1)}\right]_{d+1:\text{end}}\right)\right) \tag{46}$$

$$= \left(x_i, \sum_{j\in[n]_{-i}} \psi_2\left(x_j, w_{i,j}, \sum_{\ell\in[n]_{-j}} \psi_1\left(w_{j,\ell}\right)\right)\right). \tag{47}$$

Finally, For $k = 3$, define the function ϕ_3 of the GNN as

$$\phi_3\left(h_i^{(2)}, h_j^{(2)}, w_{i,j}\right) = \frac{1}{n-1}\rho\left(h_i^{(2)}\right), \tag{48}$$

which results in

$$h_i^{(3)} = \sum_{j\in[n]_{-i}} \phi_3\left(h_i^{(2)}, h_j^{(2)}, w_{i,j}\right) \tag{49}$$

$$= \sum_{j\in[n]_{-i}} \frac{1}{n-1}\rho\left(h_i^{(2)}\right) \tag{50}$$

$$= \rho\left(x_i, \sum_{j\in[n]_{-i}} \psi_2\left(x_j, w_{i,j}, \sum_{\ell\in[n]_{-j}} \psi_1\left(w_{j,\ell}\right)\right)\right) \tag{51}$$

$$= \rho(\beta_i(W, X)) \tag{52}$$

$$= f_i(W, X). \tag{53}$$

Hence, we showed that for every $i \in [n]$, $h_i^{(3)} = f_i(W, X)$ and thus $H^{(3)} = F$. This means the GNN with functions ϕ_1, ϕ_2, and ϕ_3 can generate F. □

C Proof of Proposition 3.10

Proof. Let $F = (f_1, \ldots, f_n)$ be the "distance to node 1" function described in Item 7 of Example 2.4. We use proof by contradiction. Suppose there exists a GNN with output $H^{(k)} = (h_1^{(k)}, \ldots, h_n^{(k)})$ for some k that $H^{(k)}(W, X_0) = F(W, X_0)$, for all $G = ([n], W, X_0) \in \mathcal{G}_{n,d}$. Note that $f_1(W, X_0) = 0$ and

$$f_2(W, X_0) = \min_{(j_0,\ldots,j_\ell)\in P(2,1)} \left(\sum_{r=0}^{\ell-1} w_{j_r, j_{r+1}}\right). \tag{54}$$

Due to the necessity part of Theorem 3.4, we know that $H^{(k)}$ satisfies (5). Letting $\pi = \pi_{1,2}$ and considering (5) for $H^{(k)}$, we have $h_2(W, X) = h_1(\sigma_{\pi_{1,2}}(W), \lambda_{\pi_{1,2}}(X))$ for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$. In

particular, when $X = X_0$, we have $h_2(W, X_0) = h_1(\sigma_{\pi_{1,2}}(W), \lambda_{\pi_{1,2}}(X_0))$ for all $G = ([n], W, X_0) \in \mathcal{G}_{n,d}$. We already know that $H^{(k)}(W, X_0) = F(W, X_0)$, for all $G = ([n], W, X_0) \in \mathcal{G}_{n,d}$. As a result, $f_2(W, X_0) = f_1(\sigma_{\pi_{1,2}}(W), \lambda_{\pi_{1,2}}(X_0))$ for all $G = ([n], W, X_0) \in \mathcal{G}_{n,d}$. Therefore,

$$0 = \min_{(j_0,\dots,j_\ell)\in P(2,1)} \left(\sum_{r=0}^{\ell-1} w_{j_r,j_{r+1}} \right), \tag{55}$$

for all $G = ([n], W, X_0) \in \mathcal{G}_{n,d}$ which is impossible, e.g., consider the case where all the weights are equal to 1. In such a case, the right-hand side equals to 1. Due to this contradiction, the original statement is proved. □

D Proof of Proposition 3.5

Lemma D.1. *Consider graph $G = ([n], W, X)$ and suppose $\pi, \pi_1, \pi_2 \in S_n$. Then $\lambda_{\pi_2\circ\pi_1}(X) = \lambda_{\pi_1} \circ \lambda_{\pi_2}(X)$ and $\sigma_{\pi_2\circ\pi_1}(W) = \sigma_{\pi_1} \circ \sigma_{\pi_2}(W)$. In particular, $\lambda_{\pi^{-1}}(X) = \lambda_\pi^{-1}(X)$ and $\sigma_{\pi^{-1}}(W) = \sigma_\pi^{-1}(W)$.*

Proof. Note that $\lambda_\pi(X)$ is in general, a sequence of objects. To show that two sequences are equal, it suffices to show that their corresponding elements are equal. To this aim, fix $i \in [n]$ and note that $[\lambda_{\pi_2\circ\pi_1}(X)]_i = x_{\pi_2\circ\pi_1(i)}$. Moreover, let $y_i = [\lambda_{\pi_2}(X)]_i = x_{\pi_2(i)}$ and $Y = (y_1, \dots, y_n) = \lambda_{\pi_2}(X)$. Therefore, $[\lambda_{\pi_1} \circ \lambda_{\pi_2}(X)]_i = [\lambda_{\pi_1}(Y)]_i = y_{\pi_1(i)} = x_{\pi_2\circ\pi_1(i)}$. Hence, we showed that $[\lambda_{\pi_2\circ\pi_1}(X)]_i = x_{\pi_2\circ\pi_1(i)} = [\lambda_{\pi_1} \circ \lambda_{\pi_2}(X)]_i$ for all $i \in [n]$, which leads to $\lambda_{\pi_2\circ\pi_1}(X) = \lambda_{\pi_1} \circ \lambda_{\pi_2}(X)$.

The argument for $\sigma_\pi(W)$ is similar. For fixed distinct $i, j \in [n]$, note that $[\sigma_{\pi_2\circ\pi_1}(W)]_{i,j} = w_{\pi_2\circ\pi_1(i),\pi_2\circ\pi_1(j)}$. Define $Z = \sigma_{\pi_2}(W)$ whose (i,j)-th element is $z_{i,j}$. This means $z_{i,j} = w_{\pi_2(i),\pi_2(j)}$. Now $[\sigma_{\pi_1} \circ \sigma_{\pi_2}(W)]_{i,j} = [\sigma_{\pi_1}(Z)]_{i,j} = z_{\pi_1(i),\pi_1(j)} = w_{\pi_2\circ\pi_1(i),\pi_2\circ\pi_1(j)}$. Hence, we showed that $[\sigma_{\pi_2\circ\pi_1}(W)]_{i,j} = w_{\pi_2\circ\pi_1(i),\pi_2\circ\pi_1(j)} = [\sigma_{\pi_1} \circ \sigma_{\pi_2}(W)]_{i,j}$ for all distinct $i, j \in [n]$, which leads to $\sigma_{\pi_2\circ\pi_1}(W) = \sigma_{\pi_1} \circ \sigma_{\pi_2}(W)$.

Note that $\lambda_\pi(\cdot)$ and $\sigma_\pi(\cdot)$ are permutations and thus their inverse exist and right and left inverses coincide. Based on the first part of the statement, we can write $X = \lambda_{\text{id}}(X) = \lambda_{\pi\circ\pi^{-1}}(X) = \lambda_{\pi^{-1}} \circ \lambda_\pi(X)$, where id is the identity permutation. Hence, $\lambda_{\pi^{-1}}(X) = \lambda_\pi^{-1}(X)$. Similarly for $\sigma_\pi(W)$, we have $X = \sigma_{\text{id}}(X) = \sigma_{\pi\circ\pi^{-1}}(X) = \sigma_{\pi^{-1}} \circ \sigma_\pi(X)$. Therefore, $\sigma_{\pi^{-1}}(W) = \sigma_\pi^{-1}(W)$. □

Lemma D.2. *For a graph $G = ([n], W, X)$ and function $f(W, X)$, the following statements are equivalent:*

$$\textit{(i)} \quad f(\sigma_{\pi_{r,s}}(W), \lambda_{\pi_{r,s}}(X)) = f(W, X) \quad \forall r, s \in [n]_{-i}. \tag{56}$$

$$\textit{(ii)} \quad f(\sigma_\pi(W), \lambda_\pi(X)) = f(W, X) \quad \forall \pi \in \nabla_i. \tag{57}$$

Proof. Since for $r, s \in [n]_{-i}$, we have $\pi_{r,s} \in \nabla_i$, we conclude that (57) follows from (56). Therefore, it suffices to show that (57) follows from (56). To this aim, suppose $\pi \in \nabla_i$. Then π fixes i, i.e., $\pi(i) = i$ and induces a permutation over $[n]_{-i}$. Call this induced permutation $\tilde{\pi}$. It is known that any permutation can be written as a combination of swapping permutations (see [4]). This means there exists a sequence of swapping permutations over $[n]_{-i}$ whose combination is π. More formally, there exist $(r_1, s_1), \dots, (r_m, s_m)$ with $r_\ell \neq s_\ell$ (note that $r_1, \dots, r_m$ may or may not be distinct and the same is true for $s_1, \dots, s_m$), where $s_\ell, r_\ell \in [n]_{-i}$ for all ℓ, such that $\pi = \pi_{r_1,s_1} \circ \pi_{r_2,s_2} \circ \dots \circ \pi_{r_m,s_m}$. Having this, for $\pi \in \nabla_i$, we can write

$$\begin{aligned} f(\sigma_\pi(W), \lambda_\pi(X)) &= f\left(\sigma_{\pi_{r_1,s_1}\circ\dots\circ\pi_{r_m,s_m}}(W), \lambda_{\pi_{r_1,s_1}\circ\dots\circ\pi_{r_m,s_m}}(X)\right) && (58)\\ &= f\left(\sigma_{\pi_{r_m,s_m}} \circ \dots \circ \sigma_{\pi_{r_1,s_1}}(W), \lambda_{\pi_{r_m,s_m}} \circ \dots \circ \lambda_{\pi_{r_1,s_1}}(X)\right) && (59)\\ &= f(W, X), && (60) \end{aligned}$$

where (59) holds due to applying Lemma D.1, m times and (60) follows from applying (56), m times. □

Proof of Proposition 3.5. The conditions (6) and (7) are particular cases of (5). Therefore, they hold trivially if $F \in \mathcal{F}_{n,d}$. Now suppose both of the conditions (6) and (7) hold. We want to show (5) for all $\pi \in S_n$. For a given $\pi \in S_n$ and $r \in [n]$, we want to show that

$$f_{\pi(r)}(W, X) = f_r(\sigma_\pi(W), \lambda_\pi(X)). \tag{61}$$

First note due to Lemma D.2, condition (6) is equivalent to (57), that is

$$f(\sigma_\pi(W), \lambda_\pi(X)) = f(W, X) \quad \forall \pi \in \nabla_i. \tag{62}$$

Having (62) as an equivalent of condition (6), we proceed as follows. Given $r \in [n]$, let $s = \pi(r)$ and consider the following cases. In each case, we show that (61) holds.

- Case $r = s = i_0$. In this case $\pi(i_0) = i_0$ and thus $\pi \in \nabla_{i_0}$. Therefore, due to condition (i), we have

$$f_{i_0}(W, X) = f_{i_0}(\sigma_\pi(W), \lambda_\pi(X)). \tag{63}$$

- Case $r = i_0$ and $s \neq i_0$. We have

$$\begin{aligned} f_s(W, X) &= f_{i_0}(\sigma_{\pi_{s,i_0}}(W), \lambda_{\pi_{s,i_0}}(X)) && (64)\\ &= f_{i_0}(\sigma_{\pi\circ\pi^{-1}\circ\pi_{s,i_0}}(W), \lambda_{\pi\circ\pi^{-1}\circ\pi_{s,i_0}}(X)) && (65)\\ &= f_{i_0}(\sigma_{\pi^{-1}\circ\pi_{s,i_0}} \circ \sigma_\pi(W), \lambda_{\pi^{-1}\circ\pi_{s,i_0}} \circ \lambda_\pi(X)), && (66) \end{aligned}$$

  where (64) follows from (7), (65) holds because $\pi \circ \pi^{-1} \circ \pi_{s,i_0} = \pi_{s,i_0}$ and (66) is a results of Lemma D.1. Note that $\pi^{-1} \circ \pi_{s,i_0} \in \nabla_{i_0}$ because $\pi^{-1} \circ \pi_{s,i_0}(i_0) = \pi^{-1}(s) = i_0$ and thus, due to (62), we have

$$f_{i_0}(\sigma_{\pi^{-1}\circ\pi_{s,i_0}}(W'), \lambda_{\pi^{-1}\circ\pi_{s,i_0}}(X')) = f_{i_0}(W', X'). \tag{67}$$

  Replace $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$ in (67) to get

$$f_{i_0}(\sigma_{\pi^{-1}\circ\pi_{s,i_0}} \circ \sigma_\pi(X), \lambda_{\pi^{-1}\circ\pi_{s,i_0}} \circ \lambda_\pi(X)) = f_{i_0}(\sigma_\pi(W), \lambda_\pi(X)). \tag{68}$$

  Hence, (66) and (68) together lead to

$$f_s(W, X) = f_{i_0}(\sigma_\pi(W), \lambda_\pi(X)). \tag{69}$$

- Case $r \neq i_0$ and $s = i_0$. We want to show that $f_{i_0}(W, X) = f_r(\sigma_\pi(W), \lambda_\pi(X))$. This is equivalent to $f_r(W, X) = f_{i_0}(\sigma_\pi^{-1}(W), \lambda_\pi^{-1}(X)) = f_{i_0}(\sigma_{\pi^{-1}}(W), \lambda_{\pi^{-1}}(X))$, which follows from the previous case ($r = i_0$ and $s \neq i_0$) by swapping r and s and replacing π with π^{-1}.

- Case $s \neq i_0$ and $r \neq i_0$.

$$\begin{aligned} f_r(\sigma_\pi(W), \lambda_\pi(X)) &= f_{i_0}(\sigma_{\pi_{r,i_0}} \circ \sigma_\pi(W), \lambda_{\pi_{ri_0}} \circ \lambda_\pi(X)) && (70)\\ &= f_{i_0}(\sigma_{\pi\circ\pi_{r,i_0}}(W), \lambda_{\pi\circ\pi_{r,i_0}}(X)) && (71)\\ &= f_{i_0}(\sigma_{\pi_{s,i_0}\circ\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}}(W), \lambda_{\pi_{s,i_0}\circ\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}}(X)) && (72)\\ &= f_{i_0}(\sigma_{\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}} \circ \sigma_{\pi_{s,i_0}}(W), \lambda_{\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}} \circ \lambda_{\pi_{s,i_0}}(X)), && (73) \end{aligned}$$

where (70) follows from (7), (72) holds because $\pi_{s,i_0}^{-1} = \pi_{s,i_0}$ and thus $\pi_{s,i_0} \circ \pi_{s,i_0} \circ \pi \circ \pi_{r,i_0} = \pi \circ \pi_{r,i_0}$, and (71) and (73) are results of Lemma D.1. Note that $\pi_{s,i_0} \circ \pi \circ \pi_{r,i_0} \in \nabla_{i_0}$ because $\pi_{s,i_0} \circ \pi \circ \pi_{r,i_0}(i_0) = \pi_{s,i_0} \circ \pi(r) = \pi_{s,i_0}(s) = i_0$ and thus due to (62), we have

$$f_{i_0}(\sigma_{\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}}(W'), \lambda_{\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}}(X')) = f_{i_0}(W', X'). \tag{74}$$

Replace $W' = \sigma_{\pi_{s,i_0}}(W)$ and $X' = \lambda_{\pi_{s,i_0}}(X)$ in (74) to get

$$f_{i_0}(\sigma_{\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}} \circ \sigma_{\pi_{s,i_0}}(W), \lambda_{\pi_{s,i_0}\circ\pi\circ\pi_{r,i_0}} \circ \lambda_{\pi_{s,i_0}}(X)) = f_{i_0}(\sigma_{\pi_{s,i_0}}(W), \lambda_{\pi_{s,i_0}}(X)). \tag{75}$$

Moreover, for the right-hand side of (75), we use (7) to write

$$f_{i_0}(\sigma_{\pi_{s,i_0}}(W), \lambda_{\pi_{s,i_0}}(X)) = f_s(W, X). \tag{76}$$

Putting together (73), (75), and (76), we have

$$f_r(\sigma_\pi(W), \lambda_\pi(X)) = f_s(W, X). \tag{77}$$

□

E Proof of Proposition 3.13

Lemma E.1. *If for* $v_1, \ldots, v_n, v'_1, \ldots, v'_n \in \mathbb{C}$,

$$\sum_{i=1}^{n} v_i^r = \sum_{i=1}^{n} v_i'^r \quad \forall r \in \{1, \ldots, n\}, \tag{78}$$

then $\#\{v_1, \ldots, v_n\} = \#\{v'_1, \ldots, v'_n\}$.

Proof. We use the known *elementary symmetric polynomials* over n variables which are defined as follows:

$$p_r(v_1, \ldots, v_n) = \sum_{i=1}^{n} v_i^r, \qquad e_r(v_1, \ldots, v_n) = \begin{cases} 1, & r = 0, \\ \sum_{1 \le i_1 < i_2 < \ldots < i_r \le n} v_{i_1} \ldots v_{i_r}, & 1 \le r \le n. \end{cases} \tag{79}$$

Using this notation, we have $p_r(v_1, \ldots, v_n) = p_r(v'_1, \ldots, v'_n)$ for all $r \in \{1, \ldots, n\}$. First we show that $e_r(v_1, \ldots, v_n) = e_r(v'_1, \ldots, v'_n)$ also holds for all $r \in \{1, \ldots, n\}$. To this aim, we use induction on r. For $r = 1$, e_1 coincides p_1, i.e.,

$$e_1(v_1, \ldots, v_n) = \sum_{i=1}^{n} v_i = p_1(v_1, \ldots, v_n). \tag{80}$$

Hence,

$$e_1(v_1, \ldots, v_n) = p_1(v_1, \ldots, v_n) = p_1\left(v'_1, \ldots, v'_n\right) = e_1\left(v'_1, \ldots, v'_n\right). \tag{81}$$

For $r > 1$, having the result for all values less than r, we need to prove it for r. From Newton's identity for elementary symmetric polynomial, we have

$$e_r(v_1, \ldots, v_n) = \frac{1}{r} \sum_{i=1}^{r} (-1)^{i-1} e_{r-i}(v_1, \ldots, v_n)\, p_i(v_1, \ldots, v_n). \tag{82}$$

The identity in (82) together with the induction hypothesis and the assumption on p_r, results in

$$e_r(v_1,\dots,v_n) = \frac{1}{r}\sum_{i=1}^{r}(-1)^{i-1}e_{r-i}(v_1,\dots,v_n)\,p_i(v_1,\dots,v_n) \tag{83}$$

$$= \frac{1}{r}\sum_{i=1}^{r}(-1)^{i-1}e_{r-i}\left(v'_1,\dots,v'_n\right)p_i\left(v'_1,\dots,v'_n\right) \tag{84}$$

$$= e_r\left(v'_1,\dots,v'_n\right). \tag{85}$$

Hence, for all $r \in \{0,\dots,n\}$, we have

$$e_r(v_1,\dots,v_n) = e_r\left(v'_1,\dots,v'_n\right). \tag{86}$$

Now consider the polynomial $\prod_{i=1}^{n}(x-v_i)$ over the complex variable x. We can decompose this polynomial and get the identity $\prod_{i=1}^{n}(x-v_i) = \sum_{r=0}^{n}(-1)^r e_r(v_1,\dots,v_n)\,x^{n-r}$. Replacing (86) in this identity, leads to

$$\prod_{i=1}^{n}(x-v_i) = \sum_{r=0}^{n}(-1)^r e_r(v_1,\dots,v_n)\,x^{n-r} = \sum_{r=0}^{n}(-1)^r e_r\left(v'_1,\dots,v'_n\right)x^{n-r} = \prod_{i=1}^{n}\left(x-v'_i\right). \tag{87}$$

Hence, $\#\{v_1,\dots,v_n\} = \#\{v'_1,\dots,v'_n\}$. □

Proof of Proposition 3.13. (a) We need to show that

$$\sum_{i=1}^{n}\psi(v_i) = \sum_{i=1}^{n}\psi\left(v'_i\right) \quad\Longleftrightarrow\quad \#\{v_1,\dots,v_n\} = \#\{v'_1,\dots,v'_n\}. \tag{88}$$

Note that the $\sum_{i=1}^{n}\psi(v_i) = \sum_{i=1}^{n}\psi(v'_i)$ is equivalent to (78) and thus the statement follows from Lemma E.1.

(b) If $\#\{v_1,\dots,v_n\} = \#\{v'_1,\dots,v'_n\}$, then (13) holds trivially. Therefore, suppose (13) holds for ψ, i.e.,

$$\sum_{i=1}^{n}\psi(v_i) = \sum_{i=1}^{n}\psi\left(v'_i\right). \tag{89}$$

We claim that $\#\{v_1,\dots,v_n\} = \#\{v'_1,\dots,v'_n\}$. Without loss of generality, consider the function ψ in its tensor-form rather than its linearized vector-form. Then due to (14) and (15), for the (ℓ,r,s)-th and (ℓ,s,r)-th coordinate of both sides of the equation (89), for all $\ell \in [n]$, and all $r,s \in [m]$ with $r<s$, we have

$$\sum_{i=1}^{n}\mathrm{Re}\left(\left([v_i]_r + [v_i]_s\sqrt{-1}\right)^{\ell}\right) = \sum_{i=1}^{n}\mathrm{Re}\left(\left(\left[v'_i\right]_r + \left[v'_i\right]_s\sqrt{-1}\right)^{\ell}\right) \tag{90}$$

$$\sum_{i=1}^{n}\mathrm{Im}\left(\left([v_i]_r + [v_i]_s\sqrt{-1}\right)^{\ell}\right) = \sum_{i=1}^{n}\mathrm{Im}\left(\left(\left[v'_i\right]_r + \left[v'_i\right]_s\sqrt{-1}\right)^{\ell}\right). \tag{91}$$

Hence, for every $r,s \in [m]$ with $r<s$, we have

$$\sum_{i=1}^{n}\left([v_i]_r + [v_i]_s\sqrt{-1}\right)^{\ell} = \sum_{i=1}^{n}\left(\left[v'_i\right]_r + \left[v'_i\right]_s\sqrt{-1}\right)^{\ell} \quad \forall \ell \in [n]. \tag{92}$$

Using Lemma E.1, for every $r, s \in [m]$, the equation (92) leads to

$$\#\left\{[v_i]_r + [v_i]_s \sqrt{-1} \mid i \in [n]\right\} = \#\left\{\left[v'_i\right]_r + \left[v'_i\right]_s \sqrt{-1} \mid i \in [n]\right\}, \tag{93}$$

which is equivalent to

$$\#\left\{([v_i]_r, [v_i]_s) \mid i \in [n]\right\} = \#\left\{(\left[v'_i\right]_r, \left[v'_i\right]_s) \mid i \in [n]\right\}. \tag{94}$$

Since (94) holds for every two coordinates $r, s \in [m]$ with $r < s$, we conclude that $\#\{v_1, \ldots, v_n\} = \#\{v'_1, \ldots, v'_n\}$.

□

F Proof of Proposition 3.15

Definition F.1. Consider the graph $G = ([n], W, X)$ and for each $s \in [n]$. let $W_s = \#\{w_{s,r} \mid r \in [n]\}$. Then for a given $i \in [n]$, define $\Delta_i(W, X)$ as

$$\Delta_i(W, X) = (x_i, \#\left\{(x_j, w_{i,j}, W_j) \mid j \in [n]_{-i}\right\}). \tag{95}$$

Lemma F.2. *Consider the graphs $G = ([n], W, X)$ and $G = ([n], W', X')$. Then $\Delta_i(W, X) = \Delta_i(W', X')$ if and only if there exists $\pi \in \nabla_i$ such that $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$.*

Proof. Denote the objects in the multiset by $\alpha_j = (x_j, w_{i,j}, W_j)$ and $\alpha'_j = (x'_j, w'_{i,j}, W'_j)$ for every $j \in [n]_{-i}$. To prove the sufficiency part, note that if $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$ for some $\pi \in \nabla_i$, then $x'_i = x_i$ (because $\pi(i) = i$), and $x'_j = x_{\pi(j)}$, and $w'_{i,j} = w_{i,\pi(j)}$ for $j \neq i$. Since $w'_{j,r} = w_{\pi(j),\pi(r)}$, the weights of edges connected to node j will be mapped to weights of edges connected to node $\pi(j)$ and thus $W'_j = W_{\pi(j)}$. Putting these together, we showed that $\alpha'_j = (x'_j, w'_{i,j}, W'_j) = (x_{\pi(j)}, w_{i,\pi(j)}, W_{\pi(j)}) = \alpha_{\pi(j)}$. This leads to $\#\{\alpha_j \mid j \in [n]_{-i}\} = \#\{\alpha'_j \mid j \in [n]_{-i}\}$ and thus $\Delta_i(W, X) = \Delta_i(W', X')$.

To prove the necessity part, note that if $\Delta_i(W, X) = \Delta_i(W', X')$, then both of the following conditions hold

$$x_i = x'_i, \tag{96}$$

$$\#\left\{(x_j, w_{i,j}, W_j) \mid j \in [n]_{-i}\right\} = \#\left\{(x'_j, w'_{i,j}, W'_j) \mid j \in [n]_{-i}\right\}. \tag{97}$$

From (97), we conclude that there exists a permutation $\tilde{\pi}$ over $[n]_{-i}$ such that $\alpha'_j = \alpha_{\tilde{\pi}(j)}$. $\tilde{\pi}$ can be extended to a permutation π over $[n]$ by defining $\pi(i) = i$ and $\pi(j) = \tilde{\pi}(j)$ for $j \neq i$. Note that $\pi \in \nabla_i$. We now claim that $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$. Note that $X' = \lambda_\pi(X)$ trivially holds because for $j \neq i$, $\alpha'_j = \alpha_{\tilde{\pi}(j)} = \alpha_{\pi(j)}$ leads to $x'_j = x_{\pi(j)}$ and we already have $x'_i = x_i = x_{\pi(i)}$ from (96). Hence, $X' = \lambda_\pi(X)$.

To show that $W' = \sigma_\pi(W)$, it suffices to show that for all $r, s \in [n]$ with $r \neq s$, we have $w'_{r,s} = w_{\pi(r),\pi(s)}$. If $r = i$, $w'_{i,j} = w_{i,\tilde{\pi}(j)} = w_{\pi(i),\pi(j)}$ for $j \neq i$, follows from $\alpha'_j = \alpha_{\tilde{\pi}(j)}$. Thus, it remains to prove $w'_{r,s} = w_{\pi(r),\pi(s)}$ when $r \neq i$. To this aim, we proceed as follows. From $\alpha'_j = \alpha_{\tilde{\pi}(j)} = \alpha_{\pi(j)}$, we conclude that

$$W'_j = W_{\pi(j)} \quad \text{for } j \in [n]_{-i}. \tag{98}$$

The value of $w'_{i,j}$, $j \in [n]_{-i}$ is obtained above as $w'_{i,j} = w_{i,\pi(j)} = w_{\pi(i),\pi(j)}$. Now, consider all $w_{r,s}$ and $w'_{r,s}$ as algebraic variables. Therefore, (98) can be seen as a system of set-equations over a pair of $(n-1)(n-2)/2$ algebraic variables, i.e., $w_{r,s}$ and $w'_{r,s}$ for $r, s \in [n]_{-i}$, $r < s$. We want to show that (98)

leads to the relation $w'_{r,s} = w_{\pi(r),\pi(s)}$ among these variables. We already know that if $w'_{r,s} = w_{\pi(r),\pi(s)}$ holds for $r, s \in [n]_{-i}$, $r < s$, then (98) holds too. Thus, (98) has a solution. Now, we claim that having (98) enforces $w'_{r,s} = w_{\pi(r),\pi(s)}$ for $r, s \in [n]_{-i}$, $r < s$. Since we considered $w_{r,s}$ and $w'_{r,s}$ as algebraic variables, every W_j as well as every W'_j consist of $n-1$ distinct elements and there will not be any repetition in them, and thus they can be seen as "sets" (rather than multisets) of $n-1$ elements (algebraic variables). Note that for distinct $r, s \in [n]_{-i}$, the sets W_r and W_s have exactly one element in common, that is $w_{r,s}$. Now suppose $r, s \in [n]_{-i}$ with $r < s$ are given. From (98), we know that $W'_r = W_{\pi(r)}$ and $W'_s = W_{\pi(s)}$. Taking the intersection between the left-hand side of these two equations as well as the right-hand side leads to $\{w'_{r,s}\} = W'_r \cap W'_s = W_{\pi(r)} \cap W_{\pi(s)} = \{w_{\pi(r),\pi(s)}\}$. Putting all the arguments together, we showed that $w'_{r,s} = w_{\pi(r),\pi(s)}$ holds for all $r, s \in [n]$. Therefore, $W' = \sigma_\pi(W)$. □

Lemma F.3. *Suppose $G = ([n], W, X)$ and $G' = ([n], W', X')$ are two graphs. Then $\beta_i(W, X) = \beta_i(W', X')$ if and only if $\Delta_i(W, X) = \Delta_i(W', X')$.*

Proof. Suppose $G = ([n], W, X)$ and $G' = ([n], W', X')$ are two graphs. Having Δ_i from Definition F.1, as the first step, we show that $\beta_i(W, X) = \beta_i(W', X')$ if and only if $\Delta_i(W, X) = \Delta_i(W', X')$. To this aim, note that $\beta_i(W, X) = \beta_i(W', X')$ holds if and only if both of the following conditions hold

$$x_i = x'_i, \tag{99}$$

$$\sum_{j \in [n]_{-i}} \psi_2 \left(x_j, w_{i,j}, \sum_{\ell \in [n]_{-j}} \psi_1 \left(w_{j,\ell} \right) \right) = \sum_{j \in [n]_{-i}} \psi_2 \left(x'_j, w'_{i,j}, \sum_{\ell \in [n]_{-j}} \psi_1 \left(w'_{j,\ell} \right) \right). \tag{100}$$

Since ψ_2 is a MEF (defined in Definition 3.12), The equation (100) holds if and only if

$$\# \left\{ \left(x_j, w_{i,j}, \sum_{\ell \in [n]_{-j}} \psi_1 \left(w_{j,\ell} \right) \right) \mid j \in [n]_{-i} \right\} = \# \left\{ \left(x'_j, w'_{i,j}, \sum_{\ell \in [n]_{-j}} \psi_1 \left(w'_{j\ell} \right) \right) \mid j \in [n]_{-i} \right\}. \tag{101}$$

Due to the fact that ψ_1 is a MEF, we know that

$$\sum_{\ell \in [n]_{-j}} \psi_1 \left(w_{j,\ell} \right) = \sum_{\ell \in [n]_{-j}} \psi_1 \left(w'_{j,\ell} \right) \quad \Longleftrightarrow \quad W_j = W'_j. \tag{102}$$

Therefore, (101) holds if and only if

$$\# \left\{ (x_j, w_{i,j}, W_j) \mid j \in [n]_{-i} \right\} = \# \left\{ \left(x'_j, w'_{i,j}, W'_j \right) \mid j \in [n]_{-i} \right\}. \tag{103}$$

Knowing that (100) is equivalent to (103), $\beta_i(W, X) = \beta_i(W', X')$ holds if and only if

$$x_i = x'_i, \tag{104}$$

$$\# \left\{ (x_j, w_{i,j}, W_j) \mid j \in [n]_{-i} \right\} = \# \left\{ \left(x'_j, w'_{i,j}, W'_j \right) \mid j \in [n]_{-i} \right\}. \tag{105}$$

Hence, having Δ_i from Definition F.1, the equations (104) and (105) hold if and only if $\Delta_i(W, X) = \Delta_i(W', X')$. So far, we showed that $\beta_i(W, X) = \beta_i(W', X')$ if and only if $\Delta_i(W, X) = \Delta_i(W', X')$. Moreover, from Lemma F.2, $\Delta_i(W, X) = \Delta_i(W', X')$ holds if and only if there exists $\pi \in \nabla_i$ such that $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$. □

Proof of Proposition 3.15. We prove the additional property first. Note that if $\beta_i(W,X) = \beta_i(W',X')$, then due to Lemma F.3, $\Delta_i(W,X) = \Delta_i(W',X')$ and thus, from Lemma F.2, there exists $\pi \in \nabla_i$ such that $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$.

It remains to prove that $\mathcal{B} \in \mathcal{F}_{n,d}$. To this aim, we use Proposition 3.5. First we prove the condition (6) for $\mathcal{B}$. To do so, pick an arbitrary $i_0 \in [n]_{-i_0}$. Given $r,s \in [n]_{-i_0}$, let $W' = \sigma_{\pi_{r,s}}(W)$ and $X' = \lambda_{\pi_{r,s}}(X)$ and note that $\pi_{r,s} \in \nabla_{i_0}$. Therefore, due to Lemma F.2, $\Delta_{i_0}(W,X) = \Delta_{i_0}(W',X')$ and then from Lemma F.3, $\beta_{i_0}(W,X) = \beta_{i_0}(W',X')$. As a next step, we prove the condition (7). Due to (16), we have

$$\beta_{i_0}(W,X) = \left(x_{i_0}, \sum_{r\in[n]_{-i_0}} \psi_2 \left(x_r, w_{i_0,r}, \sum_{\ell\in[n]_{-r}} \psi_1(w_{r,\ell}) \right) \right). \tag{106}$$

Now replace W by $\sigma_{\pi_{j,i_0}}(W)$ and X by $\lambda_{\pi_{j,i_0}}(X)$ in (106) to get

$$\beta_{i_0}\left(\sigma_{\pi_{j,i_0}}(W), \lambda_{\pi_{j,i_0}}(X)\right) = \left(x_j, \sum_{r\in[n]_{-j}} \psi_2 \left(x_r, w_{j,r}, \sum_{\ell\in[n]_{-r}} \psi_1(w_{r,\ell}) \right) \right) = \beta_j(W,X). \tag{107}$$

Hence, $\mathcal{B}$ satisfies condition (7) as well as condition (6) leading to $\mathcal{B} \in \mathcal{F}_{n,d}$. □

G Proof of Theorem 3.16

Proof. Pick $i_0 \in [n]$ arbitrarily. Due to Corollary 3.11 part (i), we know that

$$f_{i_0}(\sigma_\pi(W), \lambda_\pi(X)) = f_{i_0}(W,X) \quad \forall \pi \in \nabla_{i_0}. \tag{108}$$

We claim that there exists a function ρ such that for all $G = ([n], W, X) \in \mathcal{G}_{n,d}$, we have $f_{i_0}(W,X) = \rho(\beta_{i_0}(W,X))$.

To see this, consider graphs $G = ([n], W, X)$ and $G' = ([n], W', X')$ in $\mathcal{G}_{n,d}$. It suffices to show that $\beta_{i_0}(W',X') = \beta_{i_0}(W,X)$ results in $f_{i_0}(W',X') = f_{i_0}(W,X)$. If $\beta_{i_0}(W',X') = \beta_{i_0}(W,X)$, from Proposition 3.15, we conclude that there exists $\pi \in \nabla_{i_0}$ such that $W' = \sigma_\pi(W)$ and $X' = \lambda_\pi(X)$. Hence, (108) leads to

$$f_{i_0}\left(W',X'\right) = f_{i_0}\left(\sigma_\pi(W), \lambda_\pi(X)\right) = f_{i_0}\left(W,X\right). \tag{109}$$

Therefore,

$$f_{i_0}(W,X) = \rho\left(\beta_{i_0}(W,X)\right). \tag{110}$$

It suffices to prove that $f_j(W,X) = \rho(\beta_j(W,X))$ holds for $j \in [n]_{-i_0}$. To this aim, note that due to Proposition 3.15, $\mathcal{B} \in \mathcal{F}_{n,d}$ and thus due to Corollary 3.11, by replacing $i = i_0$, for all $j \neq i_0$ we have

$$f_j(W,X) = f_{i_0}(\sigma_{\pi_{j,i_0}}(W), \lambda_{\pi_{j,i_0}}(X)), \tag{111}$$

$$\beta_j(W,X) = \beta_{i_0}(\sigma_{\pi_{j,i_0}}(W), \lambda_{\pi_{j,i_0}}(X)). \tag{112}$$

As a result, for all $j \neq i_0$

$$f_j(W,X) = f_{i_0}(\sigma_{\pi_{j,i_0}}(W), \lambda_{\pi_{j,i_0}}(X)) = \rho\left(\beta_{i_0}(\sigma_{\pi_{j,i_0}}(W), \lambda_{\pi_{j,i_0}}(X))\right) = \rho\left(\beta_j(W,X)\right). \tag{113}$$

□

H Proof of Corollary 3.17

Proof. Consider the sufficiency part of the proof of Theorem 3.4. Due to the choice of ϕ_1, ϕ_2, and ϕ_3 in (43), (45), and (48), the continuity of ϕ_1, ϕ_2, and ϕ_3 follows from continuity of ψ_1, ψ_2, and ρ. The MEFs introduced in Proposition 3.13 are continuous and thus ψ_1 and ψ_2 can be chosen from the continuous class of functions. Moreover, when ψ_1 and ψ_2 are continuous, β_i is continuous. This means the function ρ in (41) must be continuous over the range of β_i. Therefore, there are continuous ϕ_1, ϕ_2, and ϕ_3 that can generate F. □