A Algebra definitions

A.1 Formal defintions for Universal Algebra

Universal algebra is the field of mathematics that studies algebraic structures, which are defined as a set A along with its own collection of operations. In this section, we recall some basic definitions and theorems from Burris and Sankappanavar (6); Day (8); Jonnson (17), about elements of universal algebra and lattice theory.

Definition A.1. N-ary function For a non-empty set A and n non-negative integer we define $A^0 = \{\emptyset\}$ and, for $n > 0$, A^n is the set of n-tuples of elements from A. An n-ary *operation* on A is any function f from A^n to A; n is the *arity* of f. An operation f on A is called an n-ary operation if its arity is n.

Definition A.2. Algebraic Structure An *algebra* $\mathbf{A}$ is a pair (A, F) where A is a non-empty set called *universe* and F is a set of finitary operations on A.

Apart from the operations on A, an algebra is further defined by axioms, that in the particular case of universal algebras are in the form of identities.

Definition A.3. A *lattice* $\mathbf{L}$ is an algebraic structure composed by a non-empty set L and two binary operations $\vee$ and $\wedge$ satisfying the following axioms and their duals obtained exchanging $\vee$ and $\wedge$:

$$x \vee y \approx y \vee x \qquad \text{(commutativity)}$$
$$x \vee (y \vee z) \approx (x \vee y) \qquad \text{(associativity)}$$
$$x \vee x \approx x \qquad \text{(idempotency)}$$
$$x \approx x \vee (x \wedge y) \qquad \text{(absorption)}$$

Theorem A.4 ((6)). *A partially ordered set L is a lattice if and only if for every $a, b \in L$ both* supremum *and* infimum *of $\{a, b\}$ exist (in L) with $a \vee b$ being the supremum and $a \wedge b$ the infimum.*

Definition A.5. Let $\mathbf{L}$ be a lattice. Then $\mathbf{L}$ is *modular* (*distributive*, $\vee$*-semi-distributive*, $\wedge$*-semi-distributive*) if it satisfies the following:

$$x \le y \to x \vee (y \wedge z) \approx y \wedge (x \vee z) \qquad \text{(modularity)}$$
$$x \vee (y \wedge z) \approx (x \vee y) \wedge (x \vee z) \qquad \text{(distributivity)}$$
$$x \vee y \approx x \vee z \to x \vee (y \wedge z) \approx x \vee y \qquad \text{($\vee$-semi-distributivity)}$$
$$x \wedge y \approx x \wedge z \to x \wedge (y \vee z) \approx x \wedge y \qquad \text{($\wedge$-semi-distributivity).}$$

Furthermore a lattice $\mathbf{L}$ is semi-distributive if is both $\vee$-semi-distributive and $\wedge$-semi-distributive

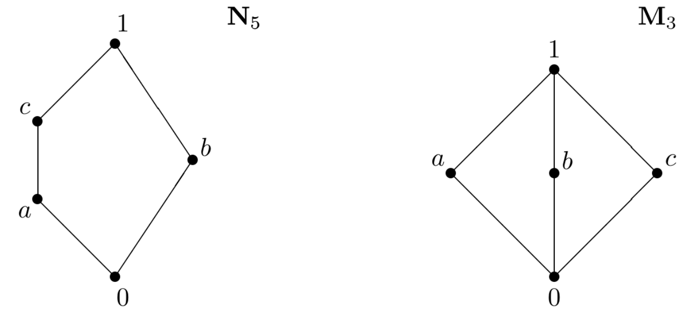

Figure 8: $\mathbf{N}_5$, a non-modular non-distributive and $\mathbf{M}_3$, a modular non-distributive lattice.

Theorem A.6. *If a lattice $\mathbf{L}$ is distributive, then $\mathbf{L}$ is also modular.*

Proof. By assuming $x \le y$, we have $x \vee y = y$. Hence, from the distributive property we get:

$$x \vee (y \wedge z) \approx (x \vee y) \wedge (x \vee z) \approx y \wedge (x \vee z)$$

□

Definition A.7. Congruence Lattice
An *equivalence relation* on a set A is a binary relation $\sim$ that satisfies three properties: reflexivity, symmetry, and transitivity.

> Reflexivity: For every element a in A, a is related to itself, denoted as $a \sim a$;
>
> Symmetry: For any elements a and b in A, if $a \sim b$, then $b \sim a$;
>
> Transitivity: For any elements a, b, and c in A, if $a \sim b$ and $b \sim c$, then $a \sim c$.

In other words, an equivalence relation partitions the set A into subsets, called *equivalence classes*, such that elements within the same class are equivalent to each other under the relation $\sim$.

Let $\mathbf{A}$ be an algebra. A *congruence θ* of $\mathbf{A}$ is a equivalent relation on A, that is compatible with the operations of $\mathbf{A}$. Formally, for every n-ary operation f of $\mathbf{A}$: if $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n) \in \theta$, then $(f(a_1, a_2, \ldots, a_n), f(b_1, b_2, \ldots, b_n)) \in \theta$. For every algebra $\mathbf{A}$ on the set A, the identity relation on A, and $A \times A$ are trivial congruences. An algebra with no other congruences is called *simple*. Let $\mathrm{Con}(\mathbf{A})$ be the set of congruences on the algebra $\mathbf{A}$. Since congruences are closed under intersection, we can define a meet operation: $\wedge : \mathrm{Con}(\mathbf{A}) \times \mathrm{Con}(\mathbf{A}) \to \mathrm{Con}(\mathbf{A})$ by simply taking the intersection of the congruences $E_1 \wedge E_2 = E_1 \cap E_2$. Congruences are not closed under union, however we can define the following closure operator of a binary relation E, with respect to a fixed algebra $\mathbf{A}$, such that its image is congruence: $\langle E \rangle_{\mathbf{A}} = \bigcap\{F \in \mathrm{Con}(\mathbf{A}) \mid E \subseteq F\}$. Note that the closure of a binary relation is a congruence and thus depends on the operations in $\mathbf{A}$, not just on the base set. Now define $\vee : \mathrm{Con}(\mathbf{A}) \times \mathrm{Con}(\mathbf{A}) \to \mathrm{Con}(\mathbf{A})$ as $E_1 \vee E_2 = \langle E_1 \cup E_2 \rangle_{\mathbf{A}}$. For every algebra $\mathbf{A}$, $(\mathrm{Con}(\mathbf{A}), \wedge, \vee)$ with the two operations defined above forms a lattice, called the *congruence lattice* of $\mathbf{A}$.

A *type* $\mathcal{F}$ is defined as a set of operation symbols along with their respective arities. Each operation symbol represents a specific operation that can be performed on the elements of the algebraic system. To refer to the specific operation performed by a given symbol f on an algebra $\mathbf{A}$ of type $\mathcal{F}$, we denote it as $f^{\mathbf{A}}$. This notation allows us to differentiate and access the particular operation carried out by f within the context of $\mathbf{A}$.

Definition A.8. Subalgebra Let $\mathbf{A}$ and $\mathbf{B}$ be two algebras of the same type. Then $\mathbf{B}$ is a *subalgebra* of $\mathbf{A}$ if $B \subseteq A$ and every fundamental operation of $\mathbf{B}$ is the restriction of the corresponding operation of $\mathbf{A}$, i.e., for each function symbol f, $f^{\mathbf{B}}$ is $f^{\mathbf{A}}$ restricted to $\mathbf{B}$.

Definition A.9. Homomorphic image Suppose $\mathbf{A}$ and $\mathbf{B}$ are two algebras of the same type $\mathcal{F}$, i.e. for each operation of $\mathbf{A}$, there exists a corresponding operation $\mathbf{B}$ with the same arity, and vice versa. A mapping $\alpha : A \to B$ is called a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ if

$$\alpha f^{\mathbf{A}}(a_1, \ldots, a_n) = f^{\mathbf{B}}(\alpha a_1, \ldots, \alpha a_n)$$

for each n-ary f in $\mathcal{F}$ and each sequence $a_1, \ldots, a_n$ from $\mathbf{A}$. If, in addition, the mapping α is onto then $\mathbf{B}$ is said to be a *homomorphic image* of $\mathbf{A}$.

Definition A.10. Direct product Let $\mathbf{A}_1$ and $\mathbf{A}_2$ be two algebras of the same type $\mathcal{F}$. We define the direct product $\mathbf{A}_1 \times \mathbf{A}_2$ to be the algebra whose universe is the set $A_1 \times A_2$, and such that for $f \in \mathcal{F}$ and $a_i \in A_1$, $a'_i \in A_2$, $1 \leq i \leq n$,

$$f^{\mathbf{A}_1 \times \mathbf{A}_2}(\langle a_1, a'_1 \rangle, \ldots, \langle a_n, a'_n \rangle) = \langle f^{\mathbf{A}_1}(a_1, \ldots, a_n), f^{\mathbf{A}_2}(a'_1, \ldots, a'_n) \rangle$$

The collection of algebraic structures defined by equational laws are called varieties. (15)

Definition A.11. Variety A nonempty class K of algebras of type $\mathcal{F}$ is called a *variety* if it is closed under subalgebras, homomorphic images, and direct products.

B Algorithm 1 details

In the following, we report some technical details on how the dataset generator sketched in Algorithm 1 is actually implemented. First, notice that a brute-force approach is infeasible for large lattices, as given a set of n nodes, the number of binary relations on this set is 2^{n^2}. To cope with this issue, first for each candidate lattice $\mathbf{L}$ we consider a squared $n \times n$ matrix $\leq_L$ representing its order that has 1 value in a position (i, j) if and only if the element i is less or equal to the element j in $\mathbf{L}$ (i.e. $i \leq_L j$) and 0 otherwise. Then we constraint each matrix to have 1 in the diagonal (reflexivity), 0 in each (i, j) with $j < i$, where "$<$" denotes denotes the order on $\mathbb{N}$ (this choice both prunes the majority of isomorphic lattices and yields anti-symmetricity). All the other pairs of elements (i, j) can either be such that $i \leq_L j$ or incomparable (i.e. $i \nleq_L j$ and $j \nleq_L i$), and we consider all these possible cases. Finally, we apply matrix multiplications to get a transitive closure of the order relation (convergence guaranteed in at most $n - 2$ steps) and hence $\leq_L$ represents a partial order. To assure that the order represents a lattice, we have to check that each pair of elements (i, j) admits a (unique!) infimum and supremum. This step and checking lattice equational properties are implemented tensorially to leverage GPU quicker computations, hence being particularly advantageous when the dimensions of the lattices is such that the computational cost of Algorithm 1 surpasses the overhead of GPU communication. Finally, we notice that even avoiding the isomorphic lattices, for $n = 18$ there are around 165Bn non-isomorphic different lattices (13). This is why we sampled a fixed number of lattices as n increases, e.g. 20 samples for cardinality after a certain threshold, instead of keeping generating all the possible lattices for each value of n, which is not particularly relevant for our task. Whereas it allows us to study the strong generalization capability of GNNs trained on e.g. up to $n = 8$ nodes and then evaluated on lattices of higher dimensions, e.g. $n = 50$ nodes (see Figure 5). Notice that checking e.g. the distributivity for $n = 50$ is deterministic but requires checking 50^3 identities.

Running time. The running time of the algorithm increases polynomially in the size n of the given lattice (it is $O(n^3)$ for checking each of the equational properties, e.g. distributivity/modularity, and $O(n^4)$ to check if a candidate binary relation is actually a lattice). In a machine with a single quad-core CPU, it requires 20 minutes to generate all the lattices up to $N = 8$ and sampling $n_s = 20$ lattices for $n \in [9, 50]$ (the dataset we used in the paper).

C Baselines' details

In practice, we train all models using eight message passing layers and different embedding sizes ranging from 16 to 64. We train all models for 200 epochs with a learning rate of 0.001. For interpretable models, we set the Gumbel-Softmax temperature to the default value of 1 and the activation behavior to "hard," which generates one-hot encoded embeddings in the forward pass, but computes the gradients using the soft scores. For the hierarchical model, we set the internal loss weight to 0.1 (to score it roughly 10% less w.r.t. the main loss). Overall, our selection of baselines aims at embracing a wide set of training setups and architectures to assess the effectiveness and versatility of GNNs for analyzing lattice properties in universal algebra. To demonstrate the robustness of our approach, we implemented different types of message passing layers, including graph convolution and GIN.

D Generalization results details

Here we report the raw numbers for the weak and strong generalization results reported as a figure in the main paper. The results are obtained by setting maximum lattice size to 8 in training and using lattices of size 9 or larger during evaluation. All models provide near perfect performance for binary classification and perform slightly worse but still very competitively for multi-label classification. This demonstrates the strong generalization capability of GNNs for universal algebra tasks, and may be an ideal starting point to finding new relevant patterns in UA properties.

In Table 1 we also include a quantitative comparison with GSAT (30), and observe that GSAT and iGNNs obtain comparable results in terms of task generalization and concept completeness (cf. Table 4) when trained on the proposed UA's tasks.

E Concept completeness and purity

Our experimental results (Tables 2 & 4) demonstrate that interpretable GNNs produce concepts with high completeness and low purity, which are standard quantitative metrics used to evaluate the quality of concept-based approaches. Completeness score is the accuracy of a classifier, such as decision

Table 1: Generalization performance of different graph neural models in solving universal algebra's tasks. Values represents the mean and the standard error of the mean of the area under the receiver operating curve (AUCROC, %).

	weak generalization				**strong generalization**			
	GCExplainer	**GSAT**	**iGNN**	**HiGNN**	**GCExplainer**	**GSAT**	**iGNN**	**HiGNN**
Distributive	99.80 ± 0.04	96.73 ± 0.44	99.56 ± 0.12	99.45 ± 0.06	99.51 ± 0.20	97.26 ± 0.30	99.44 ± 0.05	99.42 ± 0.04
Join Semi Distributive	99.49 ± 0.02	98.33 ± 0.06	98.31 ± 0.15	98.28 ± 0.04	98.77 ± 0.15	97.76 ± 0.07	97.50 ± 0.14	97.48 ± 0.14
Meet Semi Distributive	99.52 ± 0.04	98.36 ± 0.04	98.19 ± 0.06	98.25 ± 0.08	98.90 ± 0.03	97.85 ± 0.10	97.18 ± 0.14	96.89 ± 0.37
Modular	99.77 ± 0.02	96.62 ± 0.31	99.18 ± 0.11	99.35 ± 0.09	99.32 ± 0.22	96.35 ± 0.31	99.21 ± 0.14	99.11 ± 0.22
Semi Distributive	99.66 ± 0.03	98.76 ± 0.04	98.57 ± 0.02	98.50 ± 0.06	99.19 ± 0.04	98.14 ± 0.12	97.28 ± 0.48	96.88 ± 0.47
Multi Label	99.60 ± 0.02	95.00 ± 0.52	96.32 ± 0.34	95.98 ± 0.50	98.62 ± 0.43	94.45 ± 0.38	95.29 ± 0.55	95.27 ± 0.32

tree, which takes concepts as inputs and predicts a label. Purity score is the number of graph edits, such as node/edge addition/eliminations, necessary to match two graphs in a cluster. A concept space is said to be pure if the purity score is zero.

We employ decision tree as the classifier, but compute recall instead of accuracy to calculate completeness score since the datasets are heavily unbalanced towards the negative labels. We compute purity scores for each cluster and report the average of those scores as the final purity score. Our approach achieves at least 73% and up to 87% recall, which shows that our interpretable models consistently avoid false negatives in the abundance of negative labels. We obtain around 3-4 purity scores, which suggests that our interpretable models extract relatively pure concept spaces in the presence of large lattices.

Furthermore, the hierarchical structure of interpretable GNNs enables us to evaluate the quality of intermediate concepts layer by layer. This hierarchy provides insights into why we may need more layers, and it can be used as a valuable tool to find the optimal setup and tune the size of the architecture. Additionally, it can also be used to compare the quality of concepts at different layers of the network. To that end, we compare the purity scores of the concept spaces obtained by the second layer and the final layer of HiGNN. As shown in Table 3, deeper layers may produce higher quality concepts for distributivity and join semi-distributivity whereas earlier layers may result in more reliable concepts for the remaining properties. Overall, these results quantitatively assess and validate the high quality of the concepts learned by the interpretable GNNs, highlighting the effectiveness of this approach for learning and analyzing complex algebraic structures.

Table 2: Concept purity scores of graph neural models in solving universal algebra's tasks. Lower is better.

	WEAK PURITY			**STRONG PURITY**		
	GCExplainer	**iGNN**	**HiGNN**	**GCExplainer**	**iGNN**	**HiGNN**
Distributive	3.30 ± 0.36	3.64 ± 0.30	3.09 ± 0.56	3.29 ± 0.38	4.00 ± 0.77	4.15 ± 0.67
Join Semi Distributive	2.38 ± 0.37	3.96 ± 0.51	3.74 ± 0.62	3.45 ± 0.34	3.98 ± 0.68	4.29 ± 0.61
Meet Semi Distributive	3.24 ± 0.63	3.55 ± 0.62	3.39 ± 0.29	3.36 ± 0.32	4.25 ± 0.39	4.97 ± 0.44
Modular	3.10 ± 0.35	3.50 ± 0.46	4.44 ± 0.56	3.14 ± 0.24	3.19 ± 1.01	4.25 ± 0.69
Semi Distributive	2.84 ± 0.51	3.70 ± 0.54	4.11 ± 0.46	3.70 ± 0.55	3.92 ± 0.28	4.08 ± 0.85

Table 3: Concept purity scores of different layers of HiGNN. Lower is better.

	WEAK PURITY		**STRONG PURITY**	
	2nd Layer	**Last Layer**	**2nd Layer**	**Last Layer**
Distributive	3.26 ± 0.43	3.09 ± 0.56	4.66 ± 0.98	4.15 ± 0.67
Join Semi Distributive	4.25 ± 0.69	3.74 ± 0.62	4.30 ± 0.39	4.29 ± 0.61
Meet Semi Distributive	3.64 ± 0.39	3.39 ± 0.29	4.41 ± 0.27	4.97 ± 0.44
Modular	3.89 ± 0.63	4.44 ± 0.56	4.19 ± 0.56	4.25 ± 0.69
Semi Distributive	3.55 ± 0.58	4.11 ± 0.46	3.16 ± 0.59	4.08 ± 0.85

Table 4: Concept completeness scores of graph neural models in solving universal algebra's tasks. Higher is better.

	WEAK COMPLETENESS				**STRONG COMPLETENESS**			
	GCExplainer	**iGNN**	**GSAT**	**HiGNN**	**GCExplainer**	**iGNN**	**GSAT**	**HiGNN**
Distributive	96.53 ± 0.72	99.54 ± 0.13	58.36 ± 5.22	99.42 ± 0.09	95.61 ± 0.76	99.48 ± 0.06	66.59 ± 1.19	99.46 ± 0.06
Join Semi-Distributive	96.64 ± 0.14	98.45 ± 0.25	83.72 ± 4.82	98.19 ± 0.11	93.98 ± 0.97	97.59 ± 0.13	90.57 ± 0.66	97.51 ± 0.31
Meet Semi-Distributive	96.46 ± 0.11	98.17 ± 0.11	78.78 ± 1.45	98.30 ± 0.03	95.18 ± 0.44	97.20 ± 0.14	85.46 ± 2.68	96.36 ± 0.43
Modular	96.28 ± 0.86	99.08 ± 0.03	61.62 ± 1.24	99.40 ± 0.12	94.49 ± 1.08	99.10 ± 0.20	66.83 ± 2.60	99.33 ± 0.07
SemiDistributive	97.47 ± 0.04	98.62 ± 0.08	83.58 ± 1.79	98.57 ± 0.09	96.31 ± 0.02	97.07 ± 0.81	85.68 ± 2.23	97.00 ± 0.75
Multilabel	95.79 ± 0.47	88.21 ± 0.88	74.85 ± 1.35	87.44 ± 3.85	93.33 ± 0.87	87.16 ± 0.31	82.35 ± 1.30	86.86 ± 1.29

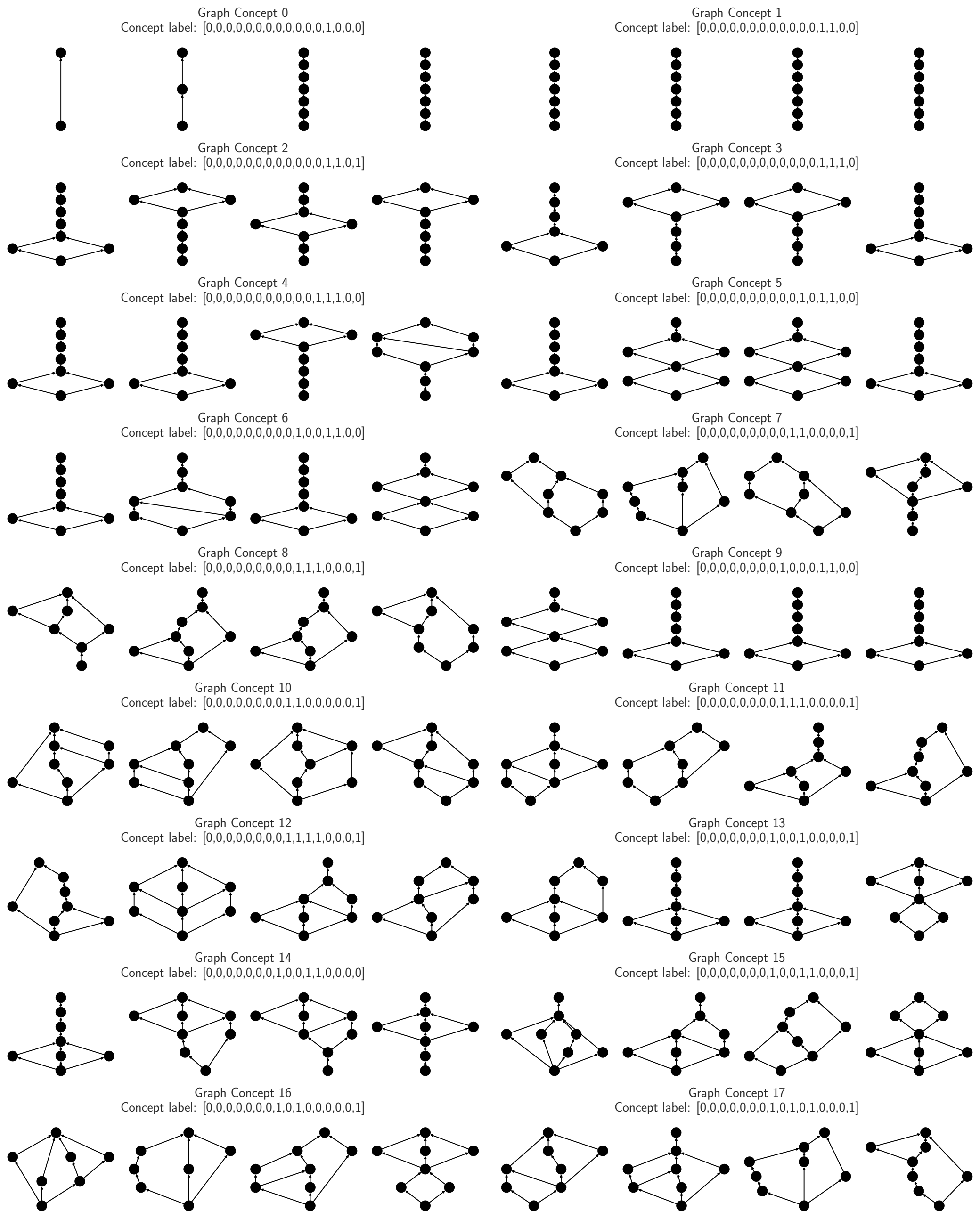
Graph Concept 0
Concept label: [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
Graph Concept 1
Concept label: [0,0,0,0,0,0,0,0,0,0,0,1,1,0,0]
Graph Concept 2
Concept label: [0,0,0,0,0,0,0,0,0,0,0,1,1,0,1]
Graph Concept 3
Concept label: [0,0,0,0,0,0,0,0,0,0,0,1,1,1,0]
Graph Concept 4
Concept label: [0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0]
Graph Concept 5
Concept label: [0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,0]
Graph Concept 6
Concept label: [0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0]
Graph Concept 7
Concept label: [0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1]
Graph Concept 8
Concept label: [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,1]
Graph Concept 9
Concept label: [0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0]
Graph Concept 10
Concept label: [0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1]
Graph Concept 11
Concept label: [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1]
Graph Concept 12
Concept label: [0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,1]
Graph Concept 13
Concept label: [0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1]
Graph Concept 14
Concept label: [0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0]
Graph Concept 15
Concept label: [0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,1]
Graph Concept 16
Concept label: [0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1]
Graph Concept 17
Concept label: [0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1]

Figure 9: Examples of graph concepts.

F Concept visualization

Figure 9 visualizes 18 randomly sampled graph concepts (out of the 7896 graph concepts represented by different graph encodings) following the visualization procedure introduced by (28). The figure shows for each concept an example of four (randomly sampled) graphs having the same concept label in the 7-th layer of the hierarchical iGNN trained on the multilabel dataset. Graphs belonging to the same concept show a coherency in their structure and similar patterns. These patterns represent the knowledge extracted and discovered by the hierarchical iGNN.

G Explanations of post-hoc explainers

We compared our Explainable Hierarchical GNN against a standard explainer (namely GNNExplainer (44)) to further support our results. GNNExplainer is the first general, model-agnostic approach for providing interpretable explanations for predictions of any GNN-based model on any graph-based machine learning task and it is widely used in the scientific community as one of the staple explainers in GNN's XAI. In this particular setting, GNNExplainer was configured as follows: model-wise explanation on multiclass-node level classification task, with HiGNN as the model of choice, and GNNExplainer as the desired algorithm, trained for 200 epochs. The explainer takes as input a single graph in the dataset and outputs and explanation for its classification. GNNExplainer will enforce a classification based on the presence or omission of M_3 and/or N_5 and it is possible to visualize the subgraph that lead to this classification by leveraging the `visualize_graph` function. By doing this, we retrieve the following visualizations:

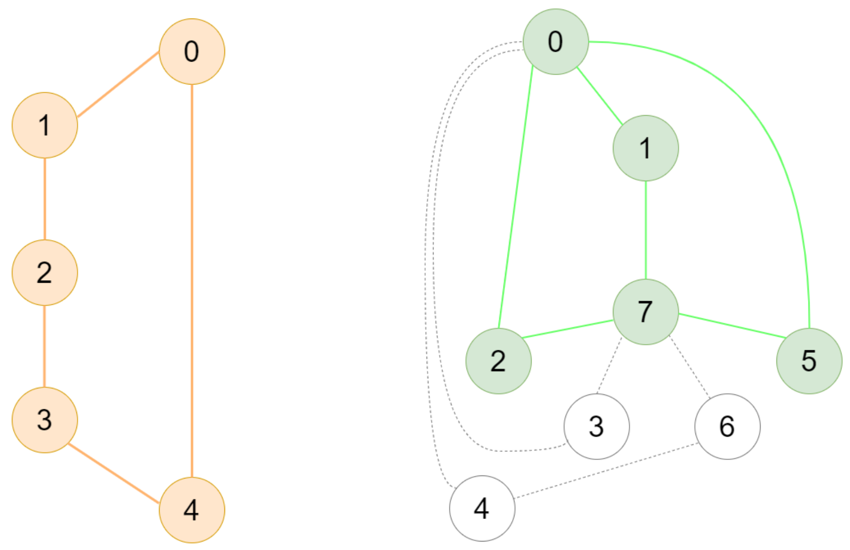

Figure 10: Visualizations obtained with GNNExplainer on weak distributive generalization (on the left) and strong multiclass generalization (on the right)

On the right, the substructure identified as $N5$ by `GNNExplainer` which lead to the classification of said graph as non modular and non distributive. On the right, in green $M3$. Our hierarchical model arrives to the same conclusions as the standard explainer but can also be augmented with a standard explainer.

H Code, Licences, Resources

Libraries For our experiments, we implemented all baselines and methods in Python 3.7 and relied upon open-source libraries such as PyTorch 1.11 (33) (BSD license) and Scikit-learn (34) (BSD license). To produce the plots seen in this paper, we made use of Matplotlib 3.5 (BSD license). We will release all of the code required to recreate our experiments in an MIT-licensed public repository.

Resources All of our experiments were run on a private machine with 8 Intel(R) Xeon(R) Gold 5218 CPUs (2.30GHz), 64GB of RAM, and 2 Quadro RTX 8000 Nvidia GPUs. We estimate that approximately 100-GPU hours were required to complete all of our experiments.