Logical Distillation of Graph Neural Networks

Alexander Pluska [1] **Pascal Welke** [1] **Thomas Gärtner** [1] **Sagar Malhotra** [1]

Abstract

We present a logic based interpretable model for learning on graphs and an algorithm to distill this model from a Graph Neural Network (GNN). Recent results have shown connections between the expressivity of GNNs and the two-variable fragment of first-order logic with counting quantifiers (C^2). We introduce a decision-tree based model which leverages an extension of C^2 to distill interpretable logical classifiers from GNNs. We test our approach on multiple GNN architectures. The distilled models are interpretable, succinct, and attain similar accuracy to the underlying GNN. Furthermore, when the ground truth is expressible in C^2, our approach outperforms the GNN.

1. Introduction

We present and evaluate an algorithm for distilling *Graph Neural Networks* (GNNs) into a symbolic model. Our distillation algorithm relies on a novel model called *Iterated Decision Tree* (IDT), which is tailored to represent logical formulas represented by GNNs. GNNs play a crucial role in safety-critical applications like drug discovery and in cost-critical applications like large-scale transport routing. However, most GNN models are black-box in nature and their internal representations are opaque to any human or computer-aided formal scrutiny. Hence, interpreting and explaining GNN predictions is a fundamental problem of significant research interest. Although many results have analyzed the expressivity of GNNs in terms of formal languages like first-order logic, extracting the logical classifiers expressed by GNNs remains largely unexplored. We aim to fill this gap by developing a distillation model aimed at extracting logical classifiers expressed by GNNs.

The key motivation for our model is the close relationship between GNNs and first-order logic with only two variables and counting quantifiers C^2 (Barceló et al., 2020; Grohe, 2021; 2023). Hence our model, the IDT, is designed to express any C^2 formula. An IDT consists of a sequence of decision trees. Each decision tree expresses a number of unary C^2 formulas of quantifier depth one. Combining multiple such decision trees enables us to express formulas of larger quantifier depth. Additionally, we propose an extension of C^2 that can capture operations like mean aggregation, which are common in GNNs, and incorporate it into IDTs. Our distillation algorithm is able to exploit intermediate node representations from each message-passing layer of a GNN to iteratively learn decision trees of an IDT. Although the learning process for IDTs is guided by the GNN, our empirical results show that the logic-based inductive bias incentivizes succinct and interpretable models.

We test IDTs on multiple synthetic and real-world datasets, performing distillation on two prominent GNN architectures, Graph Isomorphism Networks (GIN) (Xu et al., 2019) and Graph Convolution Networks (GCN) (Kipf & Welling, 2017). Our algorithm consistently distills IDTs that are succinct and have comparable predictive performance to the underlying GNN. Furthermore, when the ground truth is a C^2 formula, the distilled IDT exhibits better generalization, outperforming the GNN on the test data. Qualitatively, we find that our method can provide new insights. For instance, on the AIDS dataset (Riesen & Bunke, 2008), IDTs infer a very simple high-performing rule that achieves over 99% classification accuracy. This rule classifies graphs based on their number of nodes being smaller or larger than 12. To the best of our knowledge, none of the existing GNNs or explanation methods have been able to infer this rule.

In the next section we discuss the relevant related work. In Section 3 we discuss the necessary background on graphs, logic and GNNs. We present IDTs in Section 4 and show how IDTs can be learned from GNNs in Section 5. In Section 6, we introduce an extension of C^2, which allows us to learn more expressive IDTs. Finally, we empirically evaluate IDTs in Section 7. We analyze some of the obtained logical explanations in Section B.4. We summarize our work and discuss future research directions in Section 8.

2. Related Work

Our work is related to explanation methods of GNNs (Longa et al., 2022) and to logical approaches (Barceló et al., 2020; Grohe, 2021; 2023) for analyzing their expressivity. Explanation methods aim to derive insights about the process underlying the *model predictions*. Although IDTs may aid

[1]TU Wien, Austria. Correspondence to: Alexander Pluska <alexander.pluska@tuwien.ac.at>, Sagar Malhotra <sagar.malhotra@tuwien.ac.at>.

such understanding, our goal is different. We aim to distill an interpretable classification model for the *data*, while using trained GNN model as guidance for the learning process. Hence, we want our model to not only be interpretable, but also to generalize well. In the literature, methods that provide a global explainer, i.e., an interpretable surrogate model (Azzolin et al., 2023), can be adapted to yield classification models for the underlying data. However, such models come at a significant cost to the accuracy.

Our work is also loosely connected to the general problem of learning desision trees from neural networks. This problem has already been extensively investigated for tabular data (Craven & Shavlik, 1995; Krishnan et al., 1999; Boz, 2002; Dancey et al., 2004; Setzu et al., 2021). Furthermore, recent works have also investigated the tweaking of learning process or the neural architecture itself for learning decision trees (Schaaf et al., 2019; Wu et al., 2018; Yang et al., 2018; Kontschieder et al., 2016). Although these results are related to our approach in spirit, our work is fundamentally different in its theoretical motivation and the learning procedure. GNNs expressivity is deeply connected to that of first-order logic (Barceló et al., 2020; Grohe, 2021; 2023). Hence, using logic-based decision trees is a natural choice for learning decision trees from GNNs.

Explanation methods that distill surrogate models from GNNs come closest to our approach. Azzolin et al. (2023) first derive instance-level local subgraphs as explanations and then cluster them to extrapolate a model-level Boolean formula using the subgraphs as concepts. Yuan et al. (2020) base their approach on input-optimization, i.e. globally reducing graphs to a number of instances for which the explanation is then given. Their approach requires prior domain knowledge. Most recently, Müller et al. (2024) first compute (almost) categorical layer wise node representations using GNN layers with Gumbel-Softmax update functions. Subsequently, they replace the neural networks by decision trees trained on the categorical node representations. Their approach results in an interpretable message passing scheme based on intermediate categorical node states, but requires to train a specific GNN architecture. All three approaches use graphs, sub-graphs or their combinations as the explanation model. This restricts these methods, as many simple and important constraints, e.g. a graph has more than 12 nodes, can not easily be expressed in terms of subgraphs.

3. Background

We use $[n]$ to denote the set $\{0, \dots, n-1\}$. For a matrix A we write A_{ij} for the entry in the i-th row and j-th column. A^T is the transpose of A and A^{-1} its inverse. We write I for the identity matrix, i.e., a matrix with all diagonal entries equal to one and all non-diagonal entries equal to zero. We write 1 to denote the matrix with all entries equal to one.

A *simple undirected graph* G consists of a set of nodes V and a set of edges E. Without loss of generality, a finite set of nodes V is given as $V = \{v_i\}_{i\in[|V|]}$. The adjacency matrix A of a graph is a symmetric matrix, where $A_{ij} = 1$ if there is an edge connecting v_i and v_j, otherwise $A_{ij} = 0$. We use $N(v)$ to denote the neighbors of a node v and d_v to denote its degree. In this paper, graphs are always simple and undirected. We discuss possible generalizations in Section 8.

A *tree* is a connected acyclic graph. A *rooted tree* is a tree with a designated root node. The *depth* of a node v in a rooted tree is the length of the unique path from the root to v. The *depth* of a rooted tree is defined as the maximum depth of its nodes. Each node in a rooted tree, except the root node, has a unique *parent node*, which is the only adjacent node with a smaller depth. The *children* of a node are the nodes adjacent to it with a larger depth. A *leaf node* is a node without children. A *binary tree* is a rooted tree in which each node has at most two children. A binary tree of a given depth h is perfect if it has 2^h leaves of depth h.

3.1. Graph Neural Networks

Graph neural networks (GNNs) are a class of deep learning models that operate on graphs. We will consider message passing GNNs consisting of a sequence of layers that iteratively combine the feature vector of every node with the multiset of feature vectors of its neighbors. Formally, let agg_k and comb_k for $k \in [l]$ be *aggregation* and *combination* functions. We assume that each node has an associated initial Boolean feature vector $x_v = x_v^{(0)}$. A GNN computes a vector $x_v^{(k)}$ for every node v via the following recursive formula

$$x_v^{(k+1)} = \mathsf{comb}_{k+1}(x_v^{(k)}, \mathsf{agg}_{k+1}(\{\!\{x_w^{(k)} : w \in N(v)\}\!\})), \tag{1}$$

where $k \in [l]$. The vectors $x_v^{(l)}$ are then *pooled*

$$\hat{y} = \mathsf{pool}(\{\!\{x_v^{(l)} : v \in V\}\!\}) \tag{2}$$

to give a single graph vector $\hat{y}$, the output of the GNN. Note that in our theoretical framework, pool is not limited to common pooling operations such as mean but can also include more complicated operations, e.g., a neural network.

Example 3.1. A special case of the above architecture pattern is the GCN (Kipf & Welling, 2017), in which

$$\mathsf{agg}(\{\!\{x_w^{(k)} : w \in N(v)\}\!\}) = \sum_{w\in N(v)} \frac{x_w^{(k)}}{\sqrt{d_w}}$$

$$\mathsf{comb}(x_v^{(k)}, a) = \sigma\left(W^{(k)}\left(\frac{x_v^{(k)}}{d_v} + \frac{a}{\sqrt{d_v}}\right)\right)$$

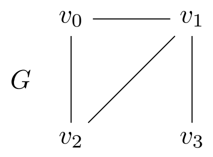
$$\mathsf{pool}(\{\!\{x_v^{(l)}\}\!\}) = \mathsf{MLP}\left(\sigma'\left(\frac{1}{|V|}\sum_{v\in V} x_v^{(l)}\right)\right)$$

where: $W^{(k)}$ are learned weight matrices; σ and σ' are non-linear activation functions and MLP is a function computed by a Multilayer Perceptron.

3.2. Graphs and Logic

C^2 is the fragment of first-order logic with only two variables v, w. Besides the usual existential ($\exists$) and universal ($\forall$) quantifiers, C^2 also admits counting quantifiers of the form $\exists^{\geq n}, \exists^{\leq n}$ and $\exists^{=n}$, which stand for *exist at least* n, *exist at most* n and *exist exactly* n (Cai et al., 1989). All C^2 sentences can be expressed in first-order logic without counting quantifiers albeit using more than two variables (Grohe, 2023). In this paper, we assume a first-order language on graphs consisting of exactly one binary predicate E and m unary predicates $\{U_j\}_{j\in[m]}$. $E(v,w)$ denotes that there is an edge between nodes v and w. $U_j(v)$ denotes that the j-th node-attribute is true for the node v in the graph. We use U to represent a binary matrix with $|V|$ rows and m columns. An entry U_{ij} in U is 1 if $U_j(v_i)$ holds in a given graph. Hence, the matrix U completely represents the interpretation of the atoms $\{U_j\}$ on a given graph. With a slight abuse of notation, we use U_j also to denote the j-th column of U. Note that for a given graph G, its adjacency matrix A completely encodes the interpretation of the predicate E.

Example 3.2. Consider the graph G (shown below) with unary predicates U_0, U_1 where $U_0(v)$ is true if $v \in \{v_1, v_3\}$ and $U_1(v)$ holds if $v \in \{v_0, v_3\}$. Then we have



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix}$$

Here, $U_0 = (0101)^T$ and $U_1 = (1001)^T$. The matrix product AU_j gives us the number of neighbors of each node satisfying U_j. Similarly, $(1-A)U_j$ gives us all the non-neighbors of each node satisfying U_j, where 1 is a matrix with all entries equal to 1.

We will consider node and graph classifiers. For instance, the C^2 formula $\exists^{=2}y\ E(x,y)$ is a logical node classifier which characterizes nodes of a given graph with degree exactly 2. The formula has exactly one free variable x and can, therefore, be evaluated on the nodes of a given graph. On the other hand, $\forall x \exists^{=2} y\ E(x,y)$ does not have any free variables. Therefore, it constitutes a graph classifier. It characterizes 2-regular graphs. Barceló et al. (2020) and Grohe (2021) have shown multiple connections between C^2 and expressivity of GNNs. A key result is the following:

Theorem 3.3 (Barceló et al. (2020), Theorem 5.2)**.** *Any classifier expressible in* C^2 *can be computed by a GNN.*

Note that GNNs can also compute classifiers that are inexpressible in first-order logic. Whether every first-order classifier computed by a GNN is expressible in C^2 is an open question. However, for GNNs without pool operation, first-order logic expressivity is completely characterized by the guarded fragment of C^2 (Barceló et al., 2020, Theorem 4.2). The proof of Theorem 3.3 builds on the equivalence between C^2 and the modal logic $\mathcal{EMLC}$ (Lutz et al., 2001). $\mathcal{EMLC}$ allows us to define a convenient grammar and serves as the motivation for our proposed model. We follow Barceló et al. (2020, Appendix D) and introduce a language similar to their Lemma D.4.

Definition 3.4. A *modal parameter* S is one of the following

$$0, 1, I, A, 1-I, 1-A, I+A, 1-I-A.$$

Given a graph G and vertex v, the interpretation $\varepsilon_S(v)$ of S on v is defined as

$$\begin{aligned} \varepsilon_0(v) &:= \emptyset & \varepsilon_{1-I}(v) &:= V \setminus \{v\} \\ \varepsilon_1(v) &:= V & \varepsilon_{1-A}(v) &:= V \setminus N(v) \\ \varepsilon_I(v) &:= \{v\} & \varepsilon_{I+A}(v) &:= \{v\} \cup N(v) \\ \varepsilon_A(v) &:= N(v) & \varepsilon_{1-I-A}(v) &:= V \setminus (\{v\} \cup N(v)) \end{aligned}$$

An $\mathcal{EMLC}$ formula is then built by the following grammar:

$$\varphi ::= U_j \mid \top \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid S\varphi > n$$

where U_j ranges over all the unary predicates, $\top$ represents the predicate which is always true, S ranges over modal parameters, and n ranges over $\mathbb{N}$. The semantics of the logical connectives ($\top$, $\wedge$, $\vee$, and $\neg$) are defined as usual. We say that $(G, v) \models S\varphi > n$ if there are more than n vertices $w \in \varepsilon_S(v)$ with $(G, w) \models \varphi$. We say $G \models \varphi$ if $(G, v) \models \varphi$ for all nodes $v \in G$. The depth of a formula is the maximal number of nested modal parameters.

Definition 3.4 can be used to express other comparison symbols, for instance one can define $S\varphi < n$ as $\neg(S\varphi > n-1)$ and $S\varphi = n$ as $\neg(S\varphi < n) \wedge \neg(S\varphi > n)$. Computationally, $\mathcal{EMLC}$ formulas can be interpreted as matrix operations. The modal parameters are naturally interpreted as matrices, i.e., 0 and 1 as the square matrices with numerical entries zero and one everywhere, I as the identity matrix, and A as the adjacency matrix. Suppose φ is given as a binary vector, i.e., the i-th entry of φ is 1 if and only of $(G, v_i) \models \varphi$. Then the i-th entry of the matrix-vector product $S\varphi$ is the number of $w \in \varepsilon_S(v_i)$ such that $(G, w) \models \varphi$. Vectorizing $\wedge, \vee, \neg$ and $>$ gives a convenient method for determining which nodes satisfy a given $\mathcal{EMLC}$ formula.

Example 3.5. We have the following C^2 formula φ with a free variable v

$$\varphi := \exists^{>1} w (E(v,w) \wedge \neg\exists^{=1} v (E(w,v) \wedge U_1(v))). \quad (3)$$

Hence, φ is true for a node if it has more than one neighbor w satisfying

$$\neg\exists^{=1} v(E(w,v) \wedge U_1(v)),$$

that is, w must not have exactly one neighbor satisfying U_1. Here, we have re-used the variable v. The inner-quantification is bound to the quantifier $\exists^{=1}$. Whereas the first occurance of v is free. Note that φ can be equivalently written as the following $\mathcal{EMLC}$ formula

$$A(\neg(AU_1 = 1)) > 1. \tag{4}$$

When evaluating the formula (4) on the graph given in Example 3.2, we have

$$\begin{aligned} A(\neg(AU_1 = 1)) > 1 &= A(\neg(A(1001)^T = 1)) > 1 \\ &= A(\neg((0210)^T = 1))) > 1 \\ &= A(\neg((0010)^T)) > 1 \\ &= A(1101)^T > 1 \\ &= (1221)^T > 1 \\ &= (0110)^T \end{aligned}$$

The second entry of $AU_1 = A(1001)^T = (0210)^T$ represents that v_1 has two neighbors w for which $U_1(w)$ is true. Similarly, the third entry represents that v_2 has exactly one such neighbor and the zero entries reflect that both v_0 and v_3 have no neighbors satisfying U_1. The subsequent computation shows that v_1 and v_2 satisfy φ while v_0 and v_3 don't.

The following result makes the connection between $\mathcal{EMLC}$ and C^2 explicit. A *unary* C^2 formula is one in which, at most, one variable occurs freely.

Theorem 3.6 (Barceló et al. (2020), Theorem D.3, Lemma D.4)**.** *For every $\mathcal{EMLC}$ formula there is an equivalent unary C^2 formula. Conversely, for every unary C^2 formula there is an equivalent $\mathcal{EMLC}$ formula.*

3.3. Decision Trees

A *decision tree* is a hierarchical, binary-tree structured, decision model. It assigns labels to samples through a sequence of binary decisions. Each leaf of a decision tree can be interpreted as a logical formula consisting of the conjunction of decisions from the root of the tree to the leaf. All samples satisfying this formula are assigned the label of the leaf. For example, the leaf labeled v_2 in Figure 1 can be interpreted as $\neg U_1 \wedge \neg U_0$, the leaf labeled v_1 as $\neg U_1 \wedge U_0$. Note that since paths to a leaf of the decision tree correspond to a conjunction of decisions, sets of leaf nodes can express disjunctions of conjunctions, e.g. the set of red leaves in Figure 1 express the following disjunction of conjunctions

$$(\neg U_1 \wedge \neg U_0) \vee (\neg U_1 \wedge U_0) \vee (U_1 \wedge U_0).$$

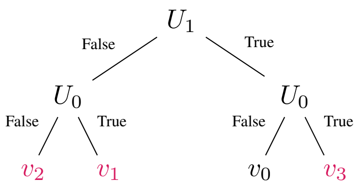


Figure 1. A simple decision tree. Each set of leaves can be interpreted as a formula, e.g. set of red leaves can be interpreted as $(\neg U_1 \wedge \neg U_0) \vee (\neg U_1 \wedge U_0) \vee (U_1 \wedge U_0)$.

Since two leaves in the set have the same parent node, we can simplify the formula to

$$\neg U_1 \vee (U_1 \wedge U_0)$$

in the above example. Formally, we have the following:

Definition 3.7. A *decision tree* is a binary rooted tree T in which each inner node (i.e. not a leaf) u is labeled with a splitting decision φ_u. For each leaf v we then define

$$\chi_v := \bigwedge\{\varphi_u : u \in \text{path}(v)\}$$

where path(v) is the path from the root to v and for each leaf set M we define

$$\chi_M := \bigvee\{\chi_v : v \in M\}.$$

When learning from data, we can obtain a decision tree by iteratively partitioning the sample space. Given a feature matrix U we choose a feature j such that each of the partitions $\{v_i : U_{ij} = 1\}$ and $\{v_i : U_{ij} = 0\}$ have maximal homogeneity according to some measure, e.g. partitions that minimize the variance of the numerical labels. If U does not only contain binary but also numerical features, in addition to the feature j we determine a threshold t such that the homogeneity of $\{v_i : U_{ij} \leq t\}$ and $\{v_i : U_{ij} > t\}$ is maximal. This process is then recursively repeated for each partition until a stopping criterion is met.

Example 3.8. Let us forget for a moment the graph structure from Example 3.2 and consider just the feature matrix U, consisting of four samples v_0, v_1, v_2, v_3, as well as the following target values Y:

$$U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \qquad Y = \begin{pmatrix} 0.2 \\ 0.8 \\ 0.9 \\ 0.1 \end{pmatrix}$$

Given our features U_0 and U_1, there are two possible splits, splitting on U_0 gives us the partition containing the sets $\{v_0, v_2\}$ and $\{v_1, v_3\}$, corresponding to U_0 being true and false respectively. While splitting on U_1 gives us a partition containing the sets $\{v_1, v_2\}$ and $\{v_0, v_3\}$, corresponding to U_1 being true and false respectively. Note that when splitting to minimize the variance of the Y values within each partition, U_1 constitutes the preferable splitting criterion.

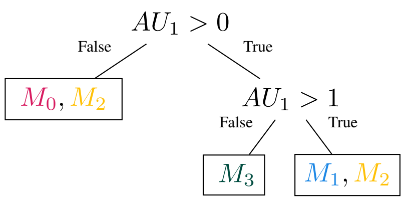
4. Iterated Decision Trees

In this section, we introduce the formal structure of our distillation model – the *Iterated Decision Tree* (IDT). IDTs consist of a sequence of decision trees. Each leaf set of a decision tree *layer* in an IDT represents an $\mathcal{EMLC}$ formula with a free variable. Each subsequent decision tree layer adds a modal parameter or a new Boolean combination of leaf sets of the previous layer. Hence, a k layer IDT can represent an $\mathcal{EMLC}$ formula with up to k nested model parameters. In the following we formally define a single IDT layer.

Definition 4.1. An iterated decision tree layer L consists of

- a decision tree T with splitting decisions of the form $S\varphi > n$ where S is a modal parameter, and $n \in \mathbb{N}$,
- and a set of leaf sets $\{M_j\}_{j\in[l]}$ of T.

Example 4.2. Consider the following iterated decision tree layer



where at each leaf the respective sets that contain it are indicated. That is, the left leaf appears in the leaf set M_0 and M_2, the middle leaf in only the leaf set M_3, and the right leaf in the leaf sets M_1 and M_2. The resulting formulas according to Definition 3.7 are then as follows:

$$\begin{aligned}
\chi_{M_0} &\iff \neg(AU_1 > 0)\\
\chi_{M_1} &\iff (AU_1 > 0) \wedge (AU_1 > 1)\\
\chi_{M_2} &\iff \neg(AU_1 > 0) \vee ((AU_1 > 0) \wedge (AU_1 > 1))\\
\chi_{M_3} &\iff \neg(AU_1 > 0) \wedge \neg(AU_1 > 1)
\end{aligned}$$

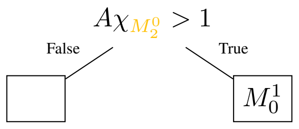
They simplify as follows:

$$\begin{aligned}
\chi_{M_0} &\iff (AU_1 = 0)\\
\chi_{M_1} &\iff (AU_1 > 1)\\
\chi_{M_2} &\iff \neg(AU_1 = 1)\\
\chi_{M_3} &\iff (AU_1 = 1)
\end{aligned}$$

Definition 4.3. A sequence of k iterated decision tree layers $\{L_i\}_{i\in[k]}$ is an *iterated decision tree* if for every $i \in [k]$ and splitting decision $S\varphi > n$ occurring in L_i, φ is of depth 0 or $\varphi \Leftrightarrow \chi_M$ for some leaf set M of L_l for $l < i$. We write M^i_j for the j-th leaf set of L_i and χ^i_j for $\chi_{M^i_j}$.

Hence, the l^{th} IDT-layer represents $\mathcal{EMLC}$ formulas with one free variable and depth of up to l. Every subsequent layer can add one additional modal parameter to the formulas represented by previous IDT layers and can also create Boolean combinations of these new formulas.

Example 4.4. Consider an iterated decision tree consisting of two layers, the first layer as in Example 4.2. Now we can add a second layer that checks if there is more than one neighbor for which $\chi_{M_2} = \chi_{M^0_2}$ is true.



It has a single leaf set M^1_0 containing only the right leaf. Then

$$\chi^1_0 \iff A(\neg(AU_1 = 1)) > 1$$

Hence the combined formula labels any node one that has more than one neighbor w such that w does not have exactly one neighbor satisfying the node attribute U_1.

We now show that any $\mathcal{EMLC}$ formula can be expressed as an iterated decision trees.

Lemma 4.5. *Given a finite set of $\mathcal{EMLC}$ formulas $\{\psi_i\}_{i\in[l]}$ of depth at most $k > 0$ there is an iterated decision tree with k layers such that χ^{k-1}_i is equivalent to ψ_i.*

A proof can be found in appendix A. The converse of Lemma 4.5 holds by definition. Hence, we have the following theorem.

Theorem 4.6. *For every $\mathcal{EMLC}$ formula ψ of depth $k > 0$ there is an iterated decision tree with k layers such that χ^k_0 is equivalent to ψ. Conversely, for every iterated decision tree with $k > 0$ layers and associated formula χ^{k-1}_j, there is an equivalent $\mathcal{EMLC}$ formula ψ of depth k.*

5. Learning Iterated Decision Trees

We now show how an iterated decision tree (IDT) can be learned from a given GNN. We are given a set of attributed graphs $\mathcal{G}$ and a l layer GNN learned on $\mathcal{G}$, using the true labels of $\mathcal{G}$. We use $\mathcal{X}^{(k)}$ to denote the set of all node representations, for all the graphs in $\mathcal{G}$, computed after k iterations of the GNN (see Equation (1)). Note that $\mathcal{X}^{(0)}$ are simply the original node attribute matrices of all the graphs in $\mathcal{G}$. We define $\mathcal{X}^{(l+1)}$ as the graph labels returned by the GNN. Finally, let $\mathcal{X}_{\text{GNN}} = \{\mathcal{X}^{(k)} \mid k \in [l+1]\}$. Algorithm 1 assumes that a function $\mathsf{LearnIDTLayer}(\mathcal{G}, \mathcal{U}, \mathcal{X}^{(k+1)})$ is able to learn an iterated decision tree layer given a set of graphs $\mathcal{G}$, a set $\mathcal{U}$ containing node attribute matrices for each graph in $\mathcal{G}$, and the labels $\mathcal{X}^{(l+1)}$. The function $\mathsf{LeafSets}$ accepts an IDT layer and returns its set of leaf sets $\{M_j\}_{j\in[l]}$ (see Definition 4.1). We discuss the subroutine $\mathsf{LearnIDTLayer}$ in Section 5.1.

Algorithm 1 Learning Procedure for Iterated Decision Trees

```
1:  procedure LEARNIDT($\mathcal{G}, \mathcal{X}_{\text{GNN}}$)
2:      IDT$\leftarrow \emptyset$
3:      $\mathcal{U} \leftarrow \mathcal{X}^{(0)}$
4:      for $k \in [l+1]$ do
5:          $L \leftarrow$ LearnIDTLayer($\mathcal{G}, \mathcal{U}, \mathcal{X}^{(k+1)}$)
6:          for $M \in$ LeafSets($L$) do
7:              $\mathcal{U} \leftarrow$ Append($\mathcal{U}$, FeatureVector($\mathcal{G}, \chi_M$))
8:          end for
9:          IDT$\leftarrow$ Append(IDT, $L$)
10:     end for
11:     return IDT
12: end procedure
```

In Algorithm 1, we initialize an empty IDT and the set $\mathcal{U}$ with the original node attribute matrices. Each iteration of the main loop (Line 4–10) computes a new IDT layer L (Line 5), using the graphs $\mathcal{G}$ with updated node-attribute matrices $\mathcal{U}$, and target labels $\mathcal{X}^{(k+1)}$ obtained from the GNN. Each leaf set M identified by $\mathsf{LeafSets}$ (Line 6) corresponds to a disjunction of conjunctions χ_M (cf. Definition 3.7). We evaluate χ_M for every node in $\mathcal{G}$ and add an explicit feature vector to the node attribute matrices in $\mathcal{U}$ (Line 7). The new set of node-attribute matrices $\mathcal{U}$ is then used as the node attributes matrix in the subsequent loop iteration.

Example 5.1. Recall the graph from Example 3.2. Assume that $\mathcal{G}$ consists of only this graph and graph label. Suppose that $\mathsf{LearnIDTLayer}(\mathcal{G},\mathcal{U},\mathcal{X}_0)$ yields the IDT layer shown in Example 4.2. Then we extend the node attributes with binary vectors representing χ_M for each leaf set $M \in \{M_0, M_1, M_2, M_3\}$. For M_0 we add

$$\begin{aligned}(AU_1 = 0) &= (A(1\ 0\ 0\ 1)^T = 0)\\ &= ((0\ 2\ 1\ 0)^T = 0)\\ &= (1\ 0\ 0\ 1)^T\end{aligned}$$

For M_1, M_2, M_3 we add

$$\begin{aligned}(AU_1 > 1) &= (0\ 1\ 0\ 0)^T\\ \neg(AU_1 = 1) &= (1\ 1\ 0\ 1)^T\\ (AU_1 = 1) &= (0\ 0\ 1\ 0)^T\end{aligned}$$

respectively. The result is an extended node attribute matrix

$$\mathcal{U} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0\\ 1 & 0 & 0 & 1 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 1 & 1 & 1 & 0 & 1 & 0\end{pmatrix}$$

which is then used for learning the next IDT layer.

5.1. Learning Iterated Decision Tree Layers

We will now describe the procedure $\mathsf{LearnIDTLayer}$. It consists of two subprocedures, learning the underlying decision tree and choosing the leaf sets.

5.1.1. Obtaining the Decision Tree

Given a set of graphs $\mathcal{G}$, a set of node attribute matrices $\mathcal{U}$ and node representations $\mathcal{X}^{(k+1)}$, we organize them into a table as illustrated in Figure 2:

- There is a row for each node in the dataset.
- For each modal parameter S and considered feature U_j, there is one column labeled SU_j. Additionally, there is a final column for the GNN node representations $\mathcal{X}^{(k+1)}$.
- The entry at the intersection of the row associated with a vertex v and the column labelled with SU_j contains the number of nodes $w \in \varepsilon_S(v)$ such that $U_j(w)$ holds.
- The entries in the final column are the node representations $\mathcal{X}^{(k+1)}$ which are prediction targets for learning the decision tree.

We can learn a decision tree from this data using conventional decision tree learning algorithms such as Pedregosa et al. (2011). Note that splitting on the first column associated with modal parameter S and feature U_j with threshold n corresponds to splitting on the $\mathcal{EMLC}$ formula $SU_j > n$.

5.1.2. Obtaining the Leaf Sets

Finally, to obtain an IDT layer, we need to determine the leaf sets. If we consider all 2^k possible sets of leaves, this leads to an explosion of the number of node attributes. We have observed that most formulas obtained in this manner are also not used in the subsequent layers. Hence, we propose a heuristic:

Since the decision tree was learned with numeric prediction targets, each leaf of the decision tree is associated with a numeric prediction. We group leaves with similar predictions since they are represented similarly by the GNN. To this end, we perform agglomerative, hierarchical clustering (Gan et al., 2007). We start with all leaf sets containing a single leaf. Then we iteratively merge the pair of leaf sets with the closest numerical values until one set remains. This approach yields $2k-1$ instead of 2^k leaf sets.

Our approach allows us to go from the formulas associated with singleton leaf sets to formulas associated with larger leaf sets, and finally the formula associated with the set of all leaves, i.e. $\top$ which is always true. Adding $\top$ as a feature allows representing the degree of a node in the next layer, since $(G, v) \models A\top > n$ if and only if $d_v > n$.

		IU_0	AU_0	$\cdots$	$\mathcal{X}^{(k+1)}$
v_0	$\cdots$	0	4	$\cdots$	$(0.32, \dots, 0.82)^T$
v_1	$\cdots$	1	1	$\cdots$	$(0.92, \dots, 0.12)^T$
$\vdots$		$\vdots$	$\vdots$		$\vdots$
v_n	$\cdots$	0	2	$\cdots$	$(0.24, \dots, 0.55)^T$

Figure 2. Schematic representation of the data table.

5.2. Practical Considerations

We have made a number of choices to arrive at a practical implementation of our proposed algorithm.

1. For all but the final iterated decision tree layer, we consider only the modal parameters $I, A, I + A$, which correspond to modal parameters quantifying only on the local neighborhood. As the aggregation operation of the GNN does not have access to features of non-neighbors, local quantification captures operations performed by real-world GNNs. Limiting ourselves to these modal parameters did not make a notable difference in the model performance and significantly reduced the computational cost.

2. For the final iterated decision tree layer, we only consider the modal parameter 1, corresponding to sum pooling.

3. We limit the depth of all but the final iterated decision tree layers to two to enhance interpretability. However, for the final IDT layer, we don't limit the depth but instead perform minimal cost complexity pruning (Breiman et al., 1984).

4. Instead of a single decision tree, we train multiple decision trees with a randomized subset of the features at each layer and add the formulas corresponding to each of their leaf sets. Empirically, this leads to better generalization.

6. Relative Modal Parameters

In GNN architectures, such as the GCN, the aggregation is normalized by the degree of the node. However, this normalization is not expressible in $\mathcal{EMLC}$ and can, as such, not be captured by the defined iterated decision tree. We argue that a property like *"more than half of the neighbors of v satisfy U_0"* should be expressible in our model. Therefore, we propose the following extension of $\mathcal{EMLC}$:

Definition 6.1. An $\mathcal{EMLC}\%$ formula is defined by the following grammar:

$$\varphi ::= U_j \mid \top \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid S\varphi > n \mid S\varphi > p$$

where S ranges over modal parameters (cf. Definition 3.4), n over $\mathbb{N}$ and p over the open interval $(0, 1)$. The semantics of the first six cases agree with $\mathcal{EMLC}$, while $(G, v) \models S\varphi > p$ holds if and only if there are more than $p \cdot |\varepsilon_S(v)|$ vertices $w \in \varepsilon_S(v)$ such that $(G, w) \models S\varphi$. Note that we can always distinguish between $S\varphi > n$ and $S\varphi > p$, since n and p range over disjoint sets.

Example 6.2. Using the semantics defined in Definition 6.1, the formula

$$1U_0 > 0.5$$

holds if more than half the nodes in G satisfy U_0.

Theorem 6.3. *$\mathcal{EMLC}\%$ is more expressive than $\mathcal{EMLC}$.*

Proof. Using Theorem 3.6 we have that every $\mathcal{EMLC}$ formula is equivalent to a C^2 formula. Furthermore, every C^2 formula can be captured in first-order logic (Grohe, 2021). We will now show that there is an $\mathcal{EMLC}\%$ formula that can not be expressed in first-order logic. Our proof relies on zero-one laws on images as proved in Coupier et al. (2004).

Consider graphs with only one unary node attribute U_0. Assume for each node v that $U(v)$ is true with a probability 0.5. Define $\varphi := IU_0 > 0.5$. The probability of φ holding in a graph converges to exactly 0.5 as the graph cardinality grows. Hence, φ does not obey a zero-one law. Therefore, φ is not definable in first-order logic. $\square$

The following result transfers from Barceló et al. (2020, Theorem 5.2) by adapting case 4 of their proof to relative modal parameters, which can be easily done by adding comparisons of the form $S\varphi > p$:

Theorem 6.4. *$\mathcal{EMLC}\%$ formulas can be computed by GNNs.*

However, the property of a node having more red than blue neighbors is still not expressible in $\mathcal{EMLC}\%$. Whereas, this property can be computed by a GNN (Grohe, 2023). This puts the expressivity of $\mathcal{EMLC}\%$ strictly between the expressivity of C^2 and GNNs.

It is straightforward to extend IDTs to $\mathcal{EMCL}\%$ since the computational difference is a single division operation.

7. Experiments

We have implemented IDTs and briefly highlight our empirical results. A detailed analysis of the experiments can be found in Appendix B. Our code is available online.

We measure accuracy, F1-score, and fidelity of Graph Convolutional Networks (GCNs), Graph Isomorphism Networks (GINs), and Iterative Decision Tree (IDT) variants on several real world and synthetic datasets. IDTs achieve high accuracy and F1-scores, outperforming both GCN and GIN models in many cases. Further analysis of fidelity indicates that the outputs of different GNNs are closer to each other

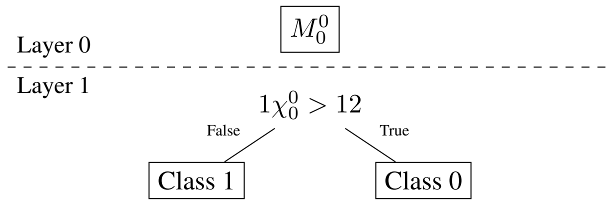


Figure 3. Distilled IDT for AIDS. At layer 0, we have just one leaf set M_0^0 containing only the tree root. Hence, the formula χ_0^0 is equivalent to $\top$. The rule derived for class 0 is thus $1\top > 12$. It expresses that the graph has more than 12 nodes.

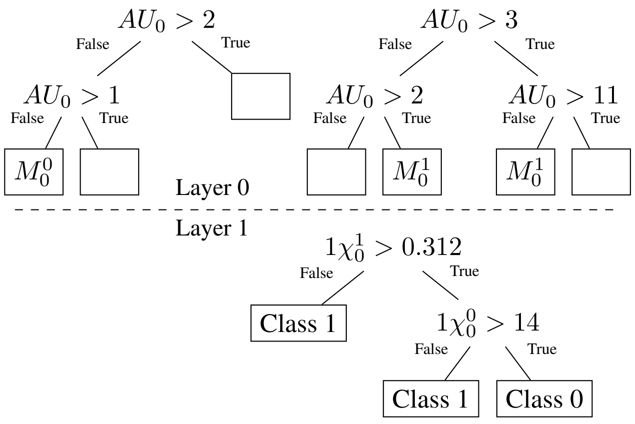


Figure 4. Distilled IDT for BAMultiShapes. $\chi_0^0 \Leftrightarrow AU_0 < 2$ and $\chi_0^1 \Leftrightarrow (AU_0 > 2) \wedge (AU_0 < 12)$. The rule derived for class 0 is $(1((AU_0 > 2) \wedge (AU_0 < 12)) > 0.312) \wedge (1(AU_0 < 2) > 14)$. It expresses that for more than 31.2% of nodes their degree d_v satisfies $2 < d_v < 12$ and for more than 14 nodes $d_v < 2$.

than the outputs of a GNN and its corresponding IDT, indicating that IDTs function differently from GNNs. This is further supported by the superior performance of IDTs on synthetic datasets where the ground truth is an $\mathcal{EMLC}\%$ formula.

A qualitative evaluation of the IDTs demonstrates their interpretability. The IDT for the AIDS dataset, shown in Figure 3, simplifies the decision-making process to a rule based on the number of nodes in a graph. If a graph has more than twelve nodes, it is assigned to Class 0. Similarly, the IDT for the BAMultiShapes dataset, shown in Figure 4 effectively uses degree-based features to classify graphs. If at least 31.2% of all nodes in a graph have degree between three and eleven and at least fifteen nodes have degree zero or one, a graph is assigned to Class 0. This showcases the model's ability to derive meaningful, simple decision rules.

8. Conclusion

We have presented Iterated Decision Trees (IDTs), a novel model tailored towards distilling interpretable logical formulas from Graph Neural Networks (GNNs). IDTs can express any logical formula expressible in first order logic with two variables and counting quantifiers (C^2) — a fragment of first-order logic that is closely connected to the logical expressivity of GNNs. We have also introduced an extension of C^2 that captures operations commonly used in GNNs. This extension was easily incorporated into IDTs without any significant computational overhead. The distilled IDTs often surpass the accuracy of the underlying GNN while providing insight into the decision process. They also outperform the considered GNNs when the ground truth is itself a logical formula. The classification decisions of the IDT are interpretable and enable us to extract insights on multiple datasets.

In this work, we have applied IDTs only to simple undirected graphs. Loops and multi-edges, however, can be incorporated into our model by allowing entries on the diagonal of the adjacency matrix A and allowing integer entries respectively. Generalizing to directed graphs is also possible. In this case the adjacency matrix is no longer symmetric, so the modal parameter A only represents out-neighbors. Thus, we need to introduce a new modal parameter A^T to represent in-neighbors and its combinations with the other modal parameters. Extending the proposed method to multi-relational graphs and edge labels would require further changes, which we will consider in future work.

While our approach has allowed us insights into the AIDS and BAMultiShapes datasets, we found it more difficult to extract meaningful explanations for other datasets. We plan to further analyze the obtained explanations and apply regularization techniques in order to obtain more human-readable explanations. As IDTs perform reasonably even without GNNs, we also look forward to further assessing the merit of IDTs as an independent architecture.

Acknowledgments

This work was funded in part by the Vienna Science and Technology Fund (WWTF), project StruDL (ICT22-059); by the Austrian Science Fund (FWF), project NanOX-ML (6728); and by the by the European Unions Horizon Europe Doctoral Network programme under the Marie-Skłodowska-Curie grant, project Training Alliance for Computational systems chemistry (101072930).

References

Azzolin, S., Longa, A., Barbiero, P., Liò, P., and Passerini, A. Global explainability of gnns via logic combination of learned concepts. In *International Conference on Learning Representations, (ICLR)*, 2023.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J. L., and Silva, J. P. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations, (ICLR)*, 2020.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, 2005.

Boz, O. Extracting decision trees from trained neural networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth, 1984. ISBN 0-534-98053-8.

Cai, J., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identification. In *Symposium on Foundations of Computer Science (FOCS)*, 1989.

Cai, T., Luo, S., Xu, K., He, D., Liu, T.-y., and Wang, L. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning (ICML)*, 2021.

Coupier, D., Desolneux, A., and Ycart, B. A zero-one law for first-order logic on random images. In *Mathematics and Computer Science III: Algorithms, Trees, Combinatorics and Probabilities*, pp. 495–505. Springer, 2004.

Craven, M. W. and Shavlik, J. W. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1995.

Dancey, D., McLean, D., and Bandar, Z. Decision tree extraction from trained neural networks. In *International Florida Artificial Intelligence Research Society Conference*, 2004.

Gan, G., Ma, C., and Wu, J. *Data clustering: theory, algorithms, and applications*. SIAM, 2007.

Grohe, M. The logic of graph neural networks. In *Symposium on Logic in Computer Science (LICS)*, 2021.

Grohe, M. The descriptive complexity of graph neural networks. In *Symposium on Logic in Computer Science (LICS)*, 2023.

Howson, C. *Logic with trees: an introduction to symbolic logic*. Routledge, 2005.

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bulò, S. R. Deep neural decision forests. In *International Joint Conference on Artificial Intelligence, (IJCAI)*, 2016.

Krishnan, R., Sivakumar, G., and Bhattacharya, P. Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12):1999–2009, 1999.

Longa, A., Azzolin, S., Santin, G., Cencetti, G., Liò, P., Lepri, B., and Passerini, A. Explaining the explainers in graph neural networks: a comparative study. *arXiv*, 2210.15304, 2022.

Lutz, C., Sattler, U., and Wolter, F. Modal logic and the two-variable fragment. In *International Workshop on Computer Science Logic (CSL)*, 2001.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. In *Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

Müller, P., Faber, L., Martinkus, K., and Wattenhofer, R. GraphChef: Decision-tree recipes to explain graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2024.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Riesen, K. and Bunke, H. IAM graph database repository for graph based pattern recognition and machine learning. In *International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 2008.

Schaaf, N., Huber, M. F., and Maucher, J. Enhancing decision tree based interpretation of deep neural networks through L1-orthogonal regularization. In *International Conference On Machine Learning And Applications (ICMLA)*, 2019.

Setzu, M., Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., and Giannotti, F. GLocalX - from local to global explanations of black box AI models. *Artificial Intelligence*, 294:103457, 2021.

Sutherland, J. J., O'brien, L. A., and Weaver, D. F. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6): 1906–1915, 2003.

Wu, M., Hughes, M. C., Parbhoo, S., Zazzi, M., Roth, V., and Doshi-Velez, F. Beyond sparsity: Tree regularization of deep models for interpretability. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Yang, Y., Morillo, I. G., and Hospedales, T. M. Deep neural decision trees. *arXiv*, 1806.06988, 2018.

Yuan, H., Tang, J., Hu, X., and Ji, S. Xgnn: Towards model-level explanations of graph neural networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2020.

A. Proof of Lemma 4.5

Let us first recall the claim:

Lemma 4.5. *Given a finite set of $\mathcal{EMLC}$ formulas $\{\psi_i\}_{i\in[l]}$ of depth at most $k > 0$ there is an iterated decision tree with k layers such that χ_i^{k-1} is equivalent to ψ_i.*

Before we prove it, we show the following auxiliary result:

Lemma A.2. *Let $\Phi = \{\varphi_j\}_{j\in[h]}$ be a finite set of formulas. There is a decision tree with splitting decisions in Φ such that for every Boolean combination ψ of formulas in Φ there is a leaf set M such that $\chi_M \Leftrightarrow \psi$.*

Proof. Let T be the complete binary tree of depth h such that each node of depth j is labeled with the splitting decision φ_j. Observe that for each conjunction c of the form $l_0 \wedge \cdots \wedge l_{h-1}$ where l_j is $\neg\varphi_j$ or φ_j there exists a leaf t in T such that $\chi_t = c$.

Suppose ψ is a Boolean combination of the φ_j. By the Disjunctive Normal Form Theorem (Howson, 2005) there is a set $\mathcal{C}$ of conjunctions of the form $l_0 \wedge \cdots \wedge l_{h-1}$ where l_j is $\neg\varphi_j$ or φ_j such that

$$\psi \Leftrightarrow \bigvee \mathcal{C}.$$

The claim then follows by Definition 3.7. □

Proof of Lemma 4.5. Note that $I\varphi > 0$ is equivalent to φ for any formula φ. Therefore, we can rewrite any formula of depth i as a formula of depth j for any $j \geq i$. Thus, we can view any formula ψ of depth $k > 0$ as a Boolean combination of formulas of the form $S\varphi > n$, where S is a modal parameter and φ is of depth $k-1$. For instance, consider the following formula of depth 2:

$$(1(AU_0 = 0) > 2) \wedge (AU_1 = 1) \wedge U_0.$$

We can rewrite it to the equivalent formula

$$(1(AU_0 = 0) > 2) \wedge (I(AU_1 = 1) > 0) \wedge (I(IU_0 > 0) > 0).$$

We proceed by induction on the maximal depth of the formulas in $\{\psi_i\}_{i\in[l]}$, i.e., k.

Suppose $k = 1$. That is, each formula ψ_i is of depth at most 1. As explained earlier, we can always add the modal parameter I to get a formula of depth 1 from a formula of depth zero. Hence, we may assume that there are formulas $\{\varphi_j\}_{j\in[h]}$ such that

- each φ_j is of the form $S\varphi > n$ where φ has depth 0
- and each ψ_i is a Boolean combination of a subset of $\{\varphi_j\}_{j\in[h]}$.

By Lemma A.2 there is an IDT Layer L_0 with splitting decisions in $\{\varphi_j\}_{j\in[l]}$ and leaf sets $M_0^0 \dots M_l^0$ such that $\psi_i \Leftrightarrow \chi_i^0$ for $i \in [l]$. Thus L_0 gives us the desired 1-layer IDT.

Now suppose the claim holds for a given k and the depth of each ψ_i is bounded by $k+1$. Again, we may assume a set of formulas $\{\varphi_j\}_{j\in[h]}$ such that each ψ_i is a Boolean combination of a subset of these formulas and each φ_j is of the form $S\varphi > n$ where φ has depth k. Applying the induction hypothesis, there is an IDT $L_0 \dots L_{k-1}$ with k layers and leaf sets $M_0^{k-1} \dots M_h^{k-1}$ such that $\varphi_j \Leftrightarrow \chi_0^{k-1}$. Lemma A.2 gives us a decision tree with splitting decisions in $\{\varphi_j\}_{j\in[h]}$ and leaf sets $M_0^k \dots M_l^k$ such that $\chi_i^k \Leftrightarrow \psi_i$. Thus $L_0 \dots L_k$ is the desired IDT. □

B. Extended Experiments

There are seven models we consider for experiments[1]:

- We evaluate two GNN architectures, GCN+GraphNorm and GIN+GraphNorm as described in Cai et al. (2021), simply called GCN and GIN from here on. The number of layers, hidden dimensions, and the learning rate are determined experimentally.
- Two IDTs as described in Section 5. One leveraging the GCN node representations, IDT(GCN), and the other one leveraging the GIN node representations, IDT(GIN).
- Two IDTs as above. However, the final layer of each IDT is learned using the true labels of the dataset instead of the GNN outputs, denoted as IDT(GCN+True) and IDT(GIN+True). Using the true labels for the final layer allows for increased accuracy while still leveraging the information of the underlying GNN.

[1]Available at github.com/lexpk/LogicalDistillationOfGNNs.

Table 1. Accuracy of our proposed IDT method and of GNN models we distill from.

Test Accuracy	GCN	GIN	IDT (GCN)	IDT (GCN + True)	IDT (GIN)	IDT (GIN+True)	IDT (True)
AIDS	0.92 ± 0.02	0.92 ± 0.03	0.99 ± 0.01	1.00 ± 0.02	0.98 ± 0.01	1.00 ± 0.00	1.00 ± 0.00
BZR	0.81 ± 0.06	0.80 ± 0.08	0.79 ± 0.08	0.83 ± 0.06	0.80 ± 0.09	0.83 ± 0.06	0.81 ± 0.04
PROTEINS	0.72 ± 0.04	0.73 ± 0.04	0.73 ± 0.04	0.74 ± 0.03	0.73 ± 0.03	0.72 ± 0.02	0.71 ± 0.03
ψ_0	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
ψ_1	0.88 ± 0.02	0.88 ± 0.02	0.93 ± 0.02	0.97 ± 0.07	0.92 ± 0.04	0.95 ± 0.07	0.96 ± 0.04
ψ_2	0.81 ± 0.02	0.82 ± 0.01	0.94 ± 0.01	0.96 ± 0.01	0.95 ± 0.01	0.94 ± 0.05	0.99 ± 0.03
BAMulti	0.99 ± 0.02	0.97 ± 0.03	1.00 ± 0.01	1.00 ± 0.00	0.99 ± 0.01	1.00 ± 0.00	1.00 ± 0.00

Table 2. F1-Score with macro aggregation.

F1-Score (macro)	GCN	GIN	IDT (GCN)	IDT (GCN + True)	IDT (GIN)	IDT (GIN+True)	IDT (True)
AIDS	0.88 ± 0.04	0.87 ± 0.03	0.98 ± 0.02	1.00 ± 0.00	0.97 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
BZR	0.73 ± 0.07	0.72 ± 0.10	0.65 ± 0.12	0.63 ± 0.08	0.67 ± 0.13	0.64 ± 0.09	0.68 ± 0.05
PROTEINS	0.71 ± 0.04	0.72 ± 0.04	0.72 ± 0.04	0.73 ± 0.03	0.72 ± 0.04	0.70 ± 0.02	0.69 ± 0.04
ψ_0	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
ψ_1	0.86 ± 0.03	0.86 ± 0.02	0.92 ± 0.02	0.96 ± 0.09	0.91 ± 0.04	0.94 ± 0.09	0.95 ± 0.05
ψ_2	0.80 ± 0.02	0.81 ± 0.01	0.94 ± 0.01	0.95 ± 0.01	0.94 ± 0.01	0.94 ± 0.05	0.99 ± 0.03
BAMulti	0.99 ± 0.02	0.97 ± 0.03	1.00 ± 0.01	1.00 ± 0.00	0.99 ± 0.01	1.00 ± 0.00	1.00 ± 0.01

Table 3. Fidelity with regard to the predictions of the GCN.

Fidelity (GCN)	GIN	IDT (GCN)	IDT (True)
AIDS	0.92 ± 0.02	0.92 ± 0.02	0.92 ± 0.02
BZR	0.90 ± 0.05	0.80 ± 0.06	0.79 ± 0.05
PROTEINS	0.90 ± 0.05	0.84 ± 0.04	0.80 ± 0.06
ψ_0	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
ψ_1	0.94 ± 0.01	0.92 ± 0.01	0.85 ± 0.02
ψ_2	0.86 ± 0.01	0.83 ± 0.02	0.81 ± 0.02
BAMulti	0.97 ± 0.02	0.99 ± 0.02	0.98 ± 0.02

Table 4. Fidelity with regard to the predictions of the GIN.

Fidelity (GIN)	GCN	IDT (GIN)	IDT (True)
AIDS	0.92 ± 0.02	0.91 ± 0.03	0.91 ± 0.02
BZR	0.90 ± 0.05	0.80 ± 0.07	0.77 ± 0.05
PROTEINS	0.90 ± 0.05	0.87 ± 0.03	0.80 ± 0.04
ψ_0	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
ψ_1	0.94 ± 0.01	0.91 ± 0.03	0.85 ± 0.03
ψ_2	0.86 ± 0.01	0.82 ± 0.02	0.82 ± 0.01
BAMulti	0.97 ± 0.02	0.97 ± 0.03	0.97 ± 0.03

- As a baseline we consider an IDT in which every layer is learned with the true labels of the dataset. This model operates purely on the data and is called IDT(True).

B.1. Datasets

Real world graph classification datasets are obtained from the TU Dortmund collection (Morris et al., 2020). They are commonly used in the GNN literature.

- AIDS (Riesen & Bunke, 2008) contains 2000 graphs representing molecular compounds. The label represents activity against HIV.
- BZR (Sutherland et al., 2003) contains 405 graphs representing ligands for the benzodiazepine receptor. The label represents whether a threshold measuring binding affinity is crossed.
- PROTEINS (Borgwardt et al., 2005) contains 1113 graphs representing proteins. The label represents whether a given protein is an enzyme or not.

Synthetic datasets are based on $\mathcal{EMLC}$. We generate 1000 Erdős-Rényi graphs with $n = 13$ and $p = 0.5$ and add two node features. The feature U_0 is always 1 and the feature U_1 is 1 with probability 0.5 for each node. Then, we label the graphs according to an $\mathcal{EMLC}\%$ formula. The following formulas of increasing complexity are considered:

- $\psi_0 := 1U_1 > 0.5$.
  "More than half of the nodes satisfy U_1."
- $\psi_1 := 1((AU_0 < 4) \vee (AU_0 > 9)) > 0$.
  "There is a node v such that $d_v < 4$ or $d_v > 9$."
- $\psi_2 := 1(A(AU_0 > 6) > 0.5) > 0.5$
  "For at least half the nodes at least half of their neighbors have degree greater than 6"

BAMultiShapes (Azzolin et al., 2023) is a synthetic dataset based on subgraph motifs. The samples are generated from Barabási-Albert graphs. Each node has a feature U_0 which is always 1. To each graph, either nothing, a wheel graph, a house graph, or a grid graph is attached with a single edge, or a combination of these shapes. Class zero consists of all graphs with exactly two such shapes added. Class one of graphs with zero, one, or all three shapes added. Generally, the existence of such shapes is not expressible in $\mathcal{EMLC}\%$.

B.2. Metrics

We report the mean and standard deviation over a 10-fold cross-validation. The same splits are used for all models. The training is inductive, i.e., the test set is completely separated from the training process. We use three metrics for evaluation:

- The accuracy of the model.
- Macro F1-score: The F1-score of each class is the harmonic mean of precision and recall. These scores are then averaged, resulting is a suitable metric for more imbalanced datasets, such as AIDS.
- Fidelity with regard to each GNN. This is the proportion of predictions where the model and the GNN agree.

B.3. Quantitative Results

IDTs are able to match and beat the performance of the GNNs. Table 1 shows that IDTs trained on GCN representations and true labels consistently achieve the highest accuracy, followed by IDTs trained on GIN representations and true labels. This ranking remains the same under the F1-score as shown in Table 2. While the performance benefits are consistent, training on the true labels alone gives only slightly worse results, suggesting that there could be merit to IDTs as a stand-alone model.

Table 3 and Table 4 show the fidelity to the GCN and GIN model, respectively. Using the GNN activations generally results in IDT outputs which are closer to the GNN model, than just using the training labels. However, the two GNNs generally have larger fidelity between each other then when compared to the IDTs. This suggests that the IDTs operate in a fundamentally different way than GNNs. This point is further reinforced by the fact that on synthetic datasets the IDTs outperform both the GNNs.

B.4. Qualitative Results

We will now look at the interpretability of the distilled IDTs. First, we discard all decision tree layers and node sets which are not used for the final prediction. This procedure is automated and results in an equivalent, more compact IDT.

B.4.1. AIDS

Figure 3 shows an IDT for the AIDS dataset. The IDT is remarkably small. The decision tree in the first layer does not have a split condition. Hence $M_0^0 = \top$ and χ_0^0 is true for every node. The second layer, therefore, simply computes if there are more than twelve nodes in the graph.

B.4.2. BAMultiShapes

Figure 4 shows one extracted IDT for the BAMultiShapes dataset that achieves an accuracy of 1.0 on the test set. After some calculation, we have

$$\chi_0^0 \iff AU_0 < 2$$
$$\chi_0^1 \iff (AU_0 > 2) \land (AU_0 < 12)$$

A graph is then classified as class 0, i.e., having exactly two shapes, if

$$(1\chi_0^1 > 0.312) \land (1\chi_0^0 > 14)$$

is satisfied. Recall that U_0 is true for all vertices in BAMultiShapes. Hence, if at least 31.2% of all nodes in a graph have degree between three and eleven and at least fifteen nodes have degree zero or one, a graph is assigned to Class 0. Due to the construction of the dataset, these observations about the degree distribution correlate strongly with the label.

Other Datasets For the other real-world datasets BZR and PROTEINS, the IDTs are more complex, often containing more than 10 decision trees. Still, by carefully examining the trees it is possible to deduce explainable logical information which we leave for future work. For the synthetic datasets labeled with $\mathcal{EMLC}$ formulas ψ_0 and ψ_1, we recover the ground truth, i.e., an IDT equivalent to ψ_0 and ψ_1 respectively. For the deeper formula ψ_2 we are not able to recover the ground truth formula. The IDTs identify a more complex formula that approximates ψ_2.