Generalized Reasoning with Graph Neural Networks by Relational Bayesian Network Encodings

Raffaele Pojer
Aalborg University
rafpoj@cs.aau.dk

Andrea Passerini
University of Trento
andrea.passerini@unitn.it

Manfred Jaeger
Aalborg University
jaeger@cs.aau.dk

Abstract

Graph neural networks (GNNs) and statistical relational learning are two different approaches to learning with graph data. The former can provide highly accurate models for specific tasks when sufficient training data is available, whereas the latter supports a wider range of reasoning types, and can incorporate manual specifications of interpretable domain knowledge. In this paper we present a method to embed GNNs in a statistical relational learning framework, such that the predictive model represented by the GNN becomes part of a full generative model. This model then supports a wide range of queries, including general conditional probability queries, and computing most probable configurations of unobserved node attributes or edges. In particular, we demonstrate how this latter type of queries can be used to obtain model-level explanations of a GNN in a flexible and interactive manner.

1 Introduction

Graph-neural networks (GNNs) define the state-of-the-art in many graph learning tasks, such as node classification, link prediction, and graph generation. While achieving high performance scores for the specific task it was trained for, any given GNN is limited to that single task. This is in contrast with probabilistic generative models, where a single model can be used for a wide variety of probabilistic inferences that take a partial observation of an instance as input, and return conditional probabilities or probable configurations for the instance's unobserved components. Bayesian networks are such a class of probabilistic generative models. Originally defined as models for tabular data, Bayesian networks and other probabilistic graphical models have been adapted to deal with relational data in the field of *statistical relational learning (SRL)* [5, 15]. Even though the original motivation in SRL is mostly an integration of logic-based representations and probabilistic reasoning, the resulting frameworks can equally be understood as probabilistic models for multi-relational, attributed graphs.

Relational Bayesian networks (RBNs)[6] are a type of SRL model with a particularly close connection with current graph neural network models: as has been shown in [7], standard GNN message-passing models can be encoded as components of an RBN model, such that, e.g., a GNN model for node classification can be used to define the conditional probability of a node attribute as a component of a full generative model. This encoding is "native" in the sense that it fully translates the GNN into the original RBN representation language, and it is "computationally faithful" in the sense that the computational operations performed in forward and backpropagation operations of the GNN are in one-to-one correspondence with the computations performed by the native RBN algorithms when performing inference or learning with the encoding.

In this paper we will briefly review the RBN encoding of GNNs (Section 3). We then proceed to demonstrate different ways in which the integrated RBN/GNN model can be used to widen the scope of reasoning tasks that can be performed with a trained GNN model. Specifically, we show how for a trained node classification GNN, inference can now also be conducted from observed node labels to unobserved node attributes or link structure (Section 4), and how *most probable explanation* inference

Pojer et al., Generalized Reasoning with Graph Neural Networks by Relational Bayesian Network Encodings. *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, Virtual Event, November 27–30, 2023.

in the sense of probabilistic graphical models[2, 10] can be used to construct global explanations of a GNN (Section 5).

2 Related Work

Our work relates to the general areas of *neuro-symbolic integration* and *Bayesian deep learning*. Kautz proposed a taxonomy of six types of neural-symbolic integration [8]. Most closely related to our work are the types:

NEURO;SYMBOLIC: a neural network converts non-symbolic inputs into a symbolic representation that is then used by a symbolic solver. This system keeps the neural network and the symbolic solver separated for their complementary tasks. This is the type of integration realized by the DeepProbLog system [12, 13] within the SRL area.

Symbolic[Neuro]: a neural component is used as a sub-routine of a symbolic solver.

NEURO[SYMBOLIC]: this is the most integrated type of neuro-symbolic integration. The neural network is able to reason logically at a certain point in the execution. The symbolic component is a *subsystem* of the main neural system. Other methods that can be close to the NEURO[SYMBOLIC] type are GNNs with Knowledge Graph (KG) integration. For example, [14] impose logical constraints in the embedding space using structural relational information to transform the latent space according to the node neighborhood in the original space. Other approaches include [16], where logical constraints are applied to the loss function to encourage inter-relation structure learning. For a broader perspective, we refer to the reviews [3, 11].

Our work shares with the NEURO[SYMBOLIC] type the tight integration into a single system. However, as for the Symbolic[Neuro] type, the top-level system is the symbolic one. That said, "symbolic" in our case only means the support for high-level, logic-based model specifications. Computations performed for these models already are purely numeric: probabilistic marginalization and gradient descent optimization. This numeric processing of symbolic representations in the RBN framework is the key to the seamless integration with purely neural components.

The probabilistic nature of the integration we propose establishes a link to the field of Bayesian deep learning (BDL). In [17] BDL is presented as a form of neuro-symbolic integration of type NEURO[SYMBOLIC], where the symbolic component is a probabilistic graphical model. However, an important element of BDL also is the interpretation of neural network weights as random variables in the sense of Bayesian statistics, leading e.g. to *natural-parameter networks*[18]. In [19] multiple natural parameter networks are linked together to define a joint distribution over random variables that can be queried for arbitrary conditional probabilities. The resulting *bidirectional inference networks* are, thus, closely related to our work in that they also aim at a seamless integration of neural networks into a general reasoning framework. Distinct from our approach, this integration is for modeling tabular rather than relational data, and it is based on the specific class of natural-parameter networks, rather than aiming at the integration of a wide range of common (graph) neural networks.

3 GNN-RBN Integration

3.1 Relational Bayesian Networks

We briefly review the main elements of the RBN framework by an example. We here only cover simple "non-recursive" RBNs [6] that only allow probabilistic dependency models at the attribute/relation level, as opposed to auto-regressive dependencies for the values of a single attribute at different nodes. Suppose, then, that we model graphs with a single *link* relation, and two Boolean node attributes *verified* and *influential* (RBNs directly support only Boolean attributes; multi-valued attributes can be handled in the form of one-hot-encodings). To define a generative model for graphs with these two attributes, the attributes and relations are first arranged in a directed, acyclic graph representing the conditional dependencies. Here we may choose to consider *link* and *verified* as roots of the graph, and *influential* as their common child. An RBN then defines for each attribute/relation its conditional distribution given its parent attributes/relations. This conditional distribution is defined by a *probability formula*, which essentially is an expression in a special-purpose functional programming language.

1. $F_{\textit{verified}}(x) = 0.1$
2. $F_{\textit{link}}(x, y) = 1/\#\textit{nodes}$
3. $F_{\textit{triangle}}(x, y, z) =$
   $\textit{link}(x, y)\&\textit{link}(x, z)\&\textit{link}(y, z)$
4. $F_{\textit{triangle\_count}}(x) =$
   `combine` 1.0
   `with sum`
   `forall` y, z
   `where` $F_{\textit{triangle}}(x, y, z)$
5. $F_{\textit{influential}}(x) =$
   `combine` $0.6 \cdot F_{\textit{triangle\_count}}(x)$
   $0.3 \cdot \textit{verified}(x)$
   -3.0
   `with logistic regression`

Table 1: RBN example

For the root attribute *verified* we can specify a formula $F_{\textit{verified}}(x)$ that evaluates to the constant probability 0.1 for all nodes x (see Table 1, line 1.). For the root relation *link* we may want to use a sparse graph model, according to which the probability of a link between any pair of nodes x, y is inversely proportional to the number of nodes in the graph (line 2.; we here display formulas using some syntactic abbreviations instead of their actual representation in the RBN language primitives). For the *influential* attribute we define a conditional probability model according to which *verified* nodes that are part of many triangles are more likely to be influential. This conditional model is given by an incrementally constructed probability formula as follows: first, we define a formula that represents the Boolean condition for three nodes x, y, z to form a triangle (line 4., illustrating the support for symbolic knowledge representation). The next formula then defines for a single node x the number of triangles it is part of. This formula illustrates the key *combination function* construct that aggregates values of already defined formulas for sets of related nodes. Finally, the *triangle_count* and *verified* features are used as input features of a logistic regression model for the *influential* attribute (line 5.).

A very useful capability of the RBN framework is to impose logical constraints on the generated graphs. As an example, consider graphs where nodes are equipped with three Boolean attributes *red*,*green*,*blue*. If the intention is that these three Booleans actually are a one-hot-encoding of a single *color* attribute, then we want to impose the constraint that for every node x exactly one of the three Booleans is true. For this we can introduce a new Boolean *constraint* node attribute, defined by the probability formula

$$F_{\textit{constraint}}(x) = (\textit{red}(x) \vee \textit{green}(x) \vee \textit{blue}(x)) \wedge \neg(\textit{red}(x) \wedge \textit{green}(x)) \wedge \ldots \wedge \neg(\textit{green}(x) \wedge \textit{blue}(x)).$$

For any node j, conditioning the model on $\textit{constraint}(j) = \textit{true}$ then ensures that with probability 1 the node j has exactly one color. We use this capability in Section 5.2 to interactively generate explanations with different desired properties.

3.2 GNN encodings

Generally, probability formulas define scalar features for nodes, pairs of nodes, or tuples of more than two nodes. To see the relationship with node feature vectors constructed in a GNN, consider a basic message-passing update:

$$\boldsymbol{h}^{k+1}(x) = \sigma\Big(\sum_{y \in N_x} U\boldsymbol{h}^k(y)\Big), \tag{1}$$

where the $\boldsymbol{h}^k$ are l-dimensional feature vectors at the k'th network layer, the $\boldsymbol{h}^{k+1}$ are m-dimensional feature vectors, U is an $m \times l$-dimensional parameter matrix, and σ is a general activation function. Breaking this vector-matrix specification down to the scalar level, we obtain for the i'th component of $\boldsymbol{h}^{k+1}(x)$:

$$\boldsymbol{h}_i^{k+1}(x) = \sigma\Big(\sum_{y \in N_x} \sum_{j=1}^{l} U_{i,j}\boldsymbol{h}_j^k(y)\Big). \tag{2}$$

Assuming that the scalar features $\boldsymbol{h}_j^k$ $(j = 1, \ldots, l)$ are already defined by probability formulas $F_{k,j}$, and that σ is the sigmoid activation function, then the right-hand side of (2) can be expressed by the

probability formula:

$$
\begin{aligned}
F_{k+1,i}(x) = \texttt{combine}\ & U_{i,1} \cdot F_{k,1}(y), \\
& U_{i,2} \cdot F_{k,2}(y), \\
& \cdots \\
& U_{i,l} \cdot F_{k,l}(y) \\
\texttt{with logistic regression}\ & \\
\texttt{forall}\ & y \\
\texttt{where}\ & \textit{link}(x, y)
\end{aligned}
\tag{3}
$$

Formula (3) shows the core of RBN encodings of GNNs by an iterative construction of probability formulas encoding the computations at each GNN layer. For ease of exposition we here have assumed the very simplistic form of message-passing update (1). Additional elements like a special treatment of the dependence of $\boldsymbol{h}^{k+1}(x)$ on $\boldsymbol{h}^{k}(x)$, or the addition of bias terms, can be included in this encoding.

A GNN for node classification defines the conditional distribution $P(A_0|A_1, \ldots, A_r, E_0)$ for a node attribute A_0 (usually considered as a distinguished class label), given other node attributes $A_1, \ldots, A_r$, and the edge relation E_0. The RBN probability formula encoding of the GNN then defines the same conditional distribution as part of a full generative model in which $A_1, \ldots, A_r, E_0$ are the parents of A_0 in the underlying dependency graph. Note that the generative model may also include other attributes $A_{r+1}, \ldots$ and additional edge relations $E_1, \ldots$, which are not part of the conditional model for A_0. The RBN encoding of a GNN contains the same parameters as the original GNN. As the following theorem states, also learning these parameters by training the GNN is equivalent to learning them inside the RBN. To state the result more precisely, we also have to consider parameters in the RBN that are not part of the GNN encoding.

Theorem 3.1 Let $A_0, \ldots, A_r, A_{r+1}, \ldots, A_p$ and $E_0, \ldots, E_q$ be a set of node attributes and edge relations. Let $G_1, \ldots, G_N$ be a set of graphs for these attributes and relations, and $\bar{G}_1, \ldots, \bar{G}_N$ their reductions to the attributes $A_0, \ldots, A_r$, and the single relation E_0. Let g be a node classification GNN with target A_0 and inputs $A_1, \ldots, A_r, E_0$. Let $\boldsymbol{\phi}$ denote the parameters of g. Let r be a RBN defining a generative model for graphs over all attributes and relations $A_0, \ldots, \ldots, A_p, E_0, \ldots, E_q$, such that in the underlying dependency graph $A_1, \ldots, A_r, E_0$ are the parents of A_0, and the conditional distribution of A_0 is given by the encoding of g. Let $(\boldsymbol{\phi}, \boldsymbol{\psi})$ denote the parameters of r. Then for any instantiation $\boldsymbol{\phi}^* \in \mathbb{R}^{|\boldsymbol{\phi}|}$ of the parameters $\boldsymbol{\phi}$ the following are equivalent:

A. $\boldsymbol{\phi}^*$ minimizes cross-entropy loss of g for training data $\bar{G}_1, \ldots, \bar{G}_N$.

B. There exists an instantiation $\boldsymbol{\psi}^*$ of $\boldsymbol{\psi}$, such that $(\boldsymbol{\phi}^*, \boldsymbol{\psi}^*)$ maximizes the log-likelihood score of r for training data $G_1, \ldots, G_N$.

Proof: The log-likelihood for r given $G_1, \ldots, G_N$ decomposes according to the chain rule into a sum of conditional log-likelihoods for each A_i and E_j. Since the conditional distribution for A_0 is defined by the encoding of g, it only depends on the parameters $\boldsymbol{\phi}$, and the reductions $\bar{G}_h$ of the training graphs. Since the $\boldsymbol{\phi}$ do not occur in any other log-likelihood term than the one for A_0, the local optimization of the conditional log-likelihood for A_0 is part of the optimal solution for the full log-likelihood function. Finally, maximizing the log-likelihood for A_0 is equivalent to minimizing the cross-entropy loss. □

Note that the theorem considers the case of training under a "pure" loss/likelihood objective without regularization or procedural elements like dropout or batch-normalization. The theorem shows that the following two learning approaches for a full generative RBN model are equivalent: encode a GNN model for the conditional distribution of a specific attribute A_0 as an RBN probability formula, and use the native likelihood-optimization methods for RBN learning to learn all RBN parameters jointly, or to use an external GNN library to learn the GNN parameters, and import them into the RBN model. Even though both approaches will also lead to exactly the same computational steps in the optimization (assuming the same optimization technique, e.g. Adam[9]), the latter turns out to be much more efficient in practice, and is therefore the approach we have adopted in our experiments.

3.3 Implementation

For representing and reasoning with RBNs we use the Primula tool.[1] Our current GNN-to-RBN compiler supports the class of ACR-GNNs [1] with sigmoid activation functions. Extensions to cover other GNN modeling elements like ReLu activation and attention mechanisms are quite straightforward in principle, but may also require some minor extensions to the Primula tool. GNNs are implemented and trained in PyTorch Geometric [4].[2] The code for reproducing the experiments is available at `https://github.com/raffaelepojer/GNN-RBN-reasoning`.

For the computation of conditional probabilities (Sections 4.1) we use an importance sampling method provided by Primula. Computation of most probable configurations (Sections 4.2 and 5.2) are performed using Primula's *most probable explanation (MPE)* module. This module uses a semi-greedy strategy with limited lookahead to find a maximum likelihood joint configuration of a list of target attributes and/or edges. The search starts with a random configuration, and random restarts can be used to optimize the likelihood of the solution.

4 Generalized inference

In this section we demonstrate the enhanced inferential capabilities afforded by embedding a trained GNN model into a fully generative RBN. Our examples make use of synthetic node labels with an underlying deterministic definition expressed in first-order logic, as introduced by Barceló et al.[1]. Originally conceived to explore logic-based characterizations of GNN expressivity, we adopt Barceló et al.'s example because the underlying well-defined logical ground-truth facillitates the demonstration and evaluation of probabilistic inferences. Concretely, we consider graphs whose nodes x have a *color* attribute with five different values, and a label $\alpha(x)$ defined as

$$\alpha(x) := \exists^{[2,3]} y(\text{Blue}(y) \wedge \neg\text{edge}(x, y)). \tag{4}$$

Thus, a node x belongs to the α class if and only if there exist 2 or 3 blue nodes not connected to x (if x itself is blue, and does not have a self-loop, then x itself will also count towards these blue non-neighbors). We generate small random graphs with 5-8 nodes, random node coloring, and the α label as defined above.

We trained a one-layer ACR-GNN model with 20 hidden units, which achieves a test-set accuracy of 0.9944. The trained GNN was compiled into a probability formula, and embedded in a RBN where all the Boolean color attributes were given a prior probability of 0.5, and edges given a prior probability p_{edge} that is varied in some inference tasks.

4.1 Inverse inference: from labels to attributes

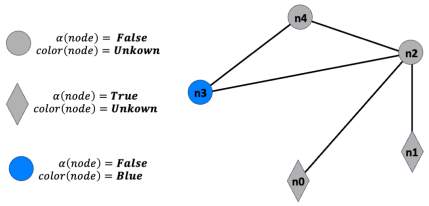
In our first demonstration we show how the RBN embedding of the GNN enables us to invert the usual direction of (predictive) inference: given our model trained on the α prediction task, we now consider an inference scenario where the α labels are observed, but information about the node attributes is incomplete. Figure 1 shows a small graph where node labels and the graph structure are observed, but there is only very partial information on the color attributes. Since *blue* is the color of most interest, we are interested in inferring the probabilities of *blue* for all nodes.

The table on the right of Figure 1 shows the computed probabilities. The logical specification (4) in conjunction with the partial observation of the graph imply that n4 must be blue (otherwise n0 and n1 could not be α nodes, and that n0 and n1 cannot be blue (otherwise n3 and n4 would have to be α nodes). This is well reflected in the computed probabilities. The probabilities do not attain the exact values 0 and 1, because the trained GNN model only is a somewhat noisy representation of (4). The given observations impose no constraints on the *blue* value of n2, which therefore retains its prior probability of 0.5.

4.2 Inferring the graph structure

Next we consider a scenario where node properties (attributes, labels) are fully observed, but the graph structure is unknown. Figure 2 at the top left shows such a scenario: five nodes with full observation

[1] `https://github.com/manfred-jaeger-aalborg/primula3`
[2] `https://pyg.org/`



Query Atoms	P(n)
blue(n0)	0.1291
blue(n1)	0.1228
blue(n2)	0.5086
blue(n3)	1.0
blue(n4)	0.8283

Figure 1: Left: graph with known labels and edges, and partial assignment to the *blue* attribute. **Right:** posterior probabilities for the *blue* attribute for all nodes

on their α class membership and *blue* attribute are given as input. Similar as in Section 4.1 one could query the probability of each possible edge. However, of greater interest is the joint configuration of all edges. For this we employ MPE inference to obtain a most probable edge configuration (there may be many solutions).

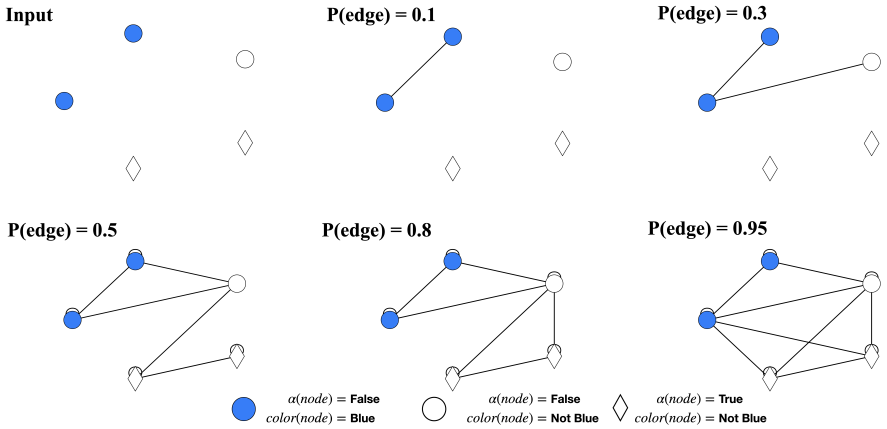


Figure 2: Results of the generated graph under the different edge probability given. Graphs with edge probability of 0.1 and 0.95 are exampls that are not consistent with (4). With 0.5 there is no bias towards sparser or denser graph.

Figure 2 shows the MPE solutions obtained for various settings of the prior edge probability. At the default value of 0.5 there is no bias towards sparser or denser graphs. The computed solution is consistent with (4) and the observed node attributes and labels. Increasing or decreasing the edge probability leads to denser and sparser solutions, respectively, which still stay consistent with (4). Only at the more extreme values $p_{edge} = 0.1$ and $p_{edge} = 0.95$ do the inferred graph topologies become inconsistent with (4). The bias induced by the edge priors here overrules the imperfect GNN representation of (4).

5 MPE inference for model-level explanation

Explanation techniques for GNNs have received a significant amount of attention. Much work is dedicated to instance-level explanations (i.e., explaining a specific prediction), but a few authors also have addressed the problem of model-level explanation (explaining the underlying predictive rules)[20, 21]. Model-level explanation in the form of constructing an input graph that maximizes the output probability for a specific target graph label is a special case of the general *most probable explanation (MPE)* inference task for RBNs. Thus, the GNN-RBN embedding together with the Primula RBN toolbox provides an out-of-the-box GNN explainer.

When considering GNN explanation techniques, one has to carefully consider different levels and objectives of explanations. At one level, the objective is strictly limited to explaining the functioning of a given GNN, e.g. with a view towards troubleshooting or robustness analysis. In the context of popular molecule classification scenarios, the question would be: "what makes the GNN predict

a graph to be toxic?". At another level, the objective is to better understand the actual domain. The question then is: "what makes a molecule toxic"? Two conditions have to be fulfilled for the second type of objective: the explanation graphs have to resemble actual molecules, and the GNN model has to be highly accurate. The first condition implies that the explainer cannot be based on the GNN as the only input; it also requires information on the real-world data distribution. In the GNNInterpreter[20], for instance, this information is provided by access of the interpreter to the training data of the GNN. In our approach, we focus on the first level explanations that only take a given model as input. However, human domain knowledge can be used to bias or condition the MPE inference towards graphs with desired properties.

5.1 Synthetic dataset

To test the feasibility of our approach, we first conduct an experiment with synthetic data that allows us to identify correct explanations. Inspired by molecule data, we generate undirected random graphs with 5 to 20 nodes which are labeled with a *type* attribute with possible values $\{A, B, C, D, E, F, G\}$. A graph belongs to the *positive* class, if it contains the two motifs $B - A - B$ and $D - C - D$. We trained a three-layer ACR-GNN with 10, 8, and 6 hidden units on a dataset containing 7000 graphs for each class. The trained network achieved a validation accuracy of 0.9904 ($\pm$ 0.0012).

We use the approach described in Section 3.1 to enforce the constraint that for all nodes exactly one of the *type* values is set to true in the one-hot-encoding. We create an RBN containing the encoding of the trained GNN, uniform prior for the *type* attribute, and a prior edge probability of 0.5. We run MPE inferences where the input consists of the number of nodes in the desired explanation graph, and the target class *positive* or *negative*. Table 2 in the top row shows explanations obtained for $n = 2, 4, 6, 8$ nodes, as well as the target class probability returned by the GNN for these inputs. Since the MPE inference depends on a random initial state, we allow 10 random restarts and select the solution with the highest target class probability. Once n reaches the minimum size $n = 6$ for the positive class, the explanations for the positive class consist of graphs containing the two motifs. Explanations for the negative class do not contain any of the motifs. The computed explanations are quite dense. In order to obtain sparser explanations, we reduce the prior edge probability. The results shown in the second row of Table 2 now identify the two motifs in a more parsimonious manner. However, the lower probabilities show that the GNN's confidence in the positive class prediction is increased by a denser graph structure.

For comparison, we have also applied XGNN[21] to provide explanations for the trained GNN. XGNN only allows to specify a maximum number of nodes in the explanation, and it also contains a random element producing different solutions in different runs. We have rerun XGNN as many times as allowed in the time budget defined by the time required for the 10 restarts in RBN-MPE inference. The results are shown in the bottom row of Table 2. XGNN was not able in this case to provide any high probability explanations for the positive class.

5.2 Real-world dataset

We now consider the real-world *Mutagenicity*[3] dataset. The data consists of 4337 molecules with 4 to 417 atoms (30.3 on average). Atoms are labeled with one out of 14 elements Ca,Na,H,F,S,Li,N,Cl,P,K,C,O,I,Br, and molecules with a *mutagenic* class label. We trained a small 3-layer ACR-GNN with 16, 8, and 8 hidden units. The trained network achieved a validation accuracy of 0.7929 and F1 score of 0.7688. Table 3 in the top row shows the explanations we obtained from the GNN-RBN explainer with prior edge probability 0.5. As for the synthetic data, explanations for both classes are rather dense. Already a slight reduction in the edge probability leads to much sparser explanations (second row). This leads to a substantial decrease in the class probabilities for the mutagenic class, hinting at the relevance of connections in characterizing the positive class. On the other hand, completely disconnected graphs have the same class probability as dense ones in the non-mutagenic class, suggesting that the element distribution is what characterizes negative examples in the learned model. Both the dense and sparse explanations are composed almost entirely of uncommon elements, leading to a possible conjecture that the GNN has mostly learned to base its prediction on characteristic occurrences of uncommon elements. In order to test this conjecture, we modify the prior probabilities for the elements, such that the common elements C,O,H,N are much more probable than the uncommon ones. We then obtain solutions that are composed entirely of

[3] https://chrsmrrs.github.io/datasets/docs/datasets/

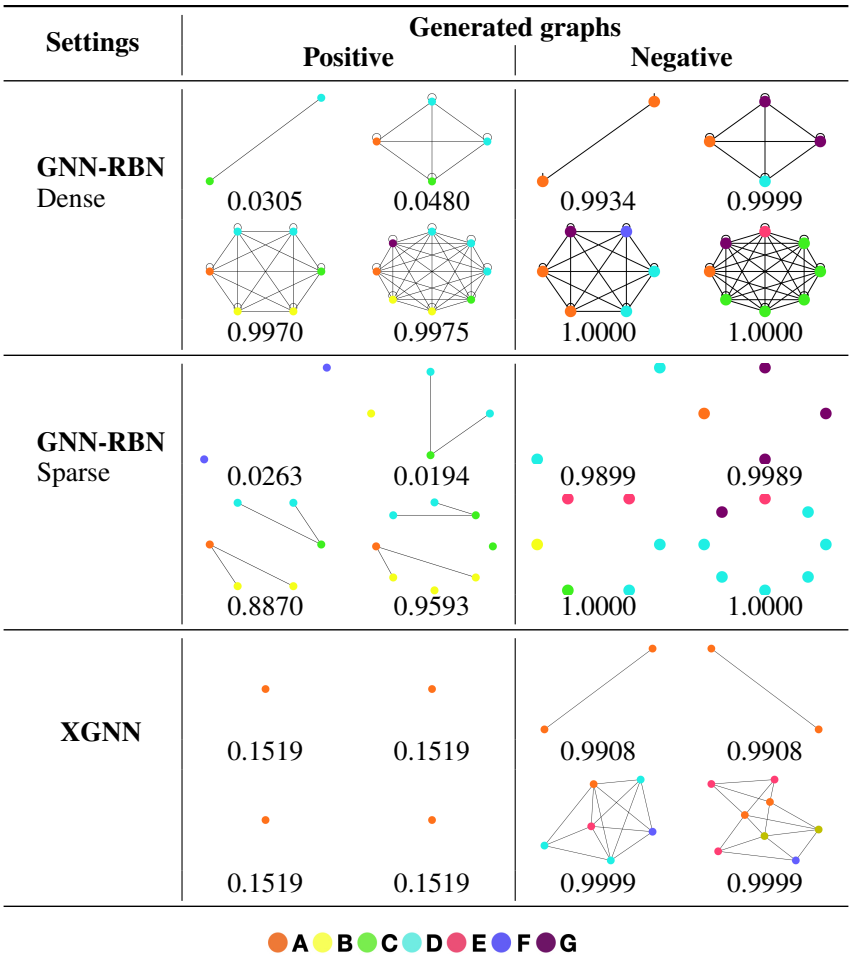


Table 2: Synthetic data: computed explanation graphs for positive and negative class with target class probabilities assigned by the GNN. **First row**: dense GNN-RBN explanations, achieved setting P(edge)=0.5. **Second row**: sparse GNN-RBN explanations, achieved setting P(edge)=0.3. **Third row**: best results obtained through multiple runs of XGNN (same time budget of GNN-RBN).

these four elements, and which retain essentially the same probabilities as the original solutions (third row in Table 3), thus showing that against the preliminary conjecture, the GNNs predictions are not based on uncommon elements alone. The results highlight the utility of the GNN-RBN in supporting an interactive analysis of learned GNN models that can confirm or disconfirm conjectures about the features being learned, something beyond the reach of typical GNN explainers.

We again also applied XGNN to this task allowing an equivalent time budget as used for GNN-RBN. For the mutagenic class here XGNN did not succeed in producing any viable solutions, whereas for the non-mutagenic class it produced explanations whose probabilities almost precisely match the probabilities of the GNN-RBN solutions, and which also are mostly composed of uncommon elements.

6 Conclusion

We have presented an approach to integrate GNNs into a SRL modeling and inference framework. In the resulting fully generative model one can exploit the high predictive accuracy of the GNN in the discriminative task it was trained on to also perform a wider range of probabilistic inferences. In particular, we have shown how MPE inference can be used to reconstruct unobserved graph structures from observed node data, and as a way to find model-level explanations. Our current implementation

is limited to dealing with GNNs of small size. However, there do not seem to be any fundamental complexity issues arising in the GNN-RBN integration, as the computations performed on the GNN encoding are identical to standard forward and backpropagations. Scaling our implementation to larger GNNs therefore is an objective of future work.

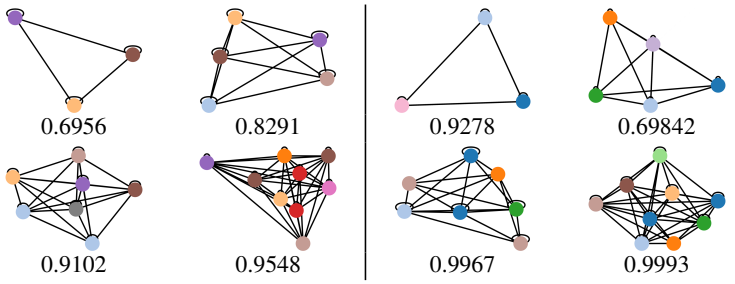
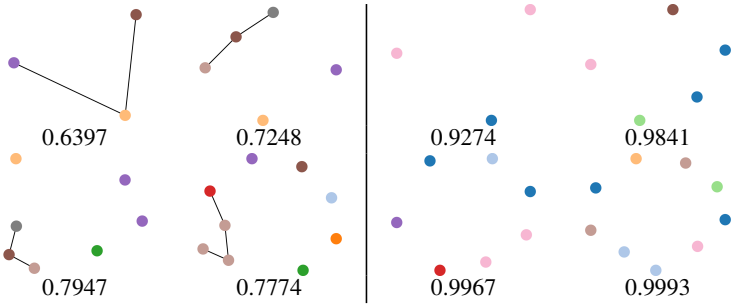
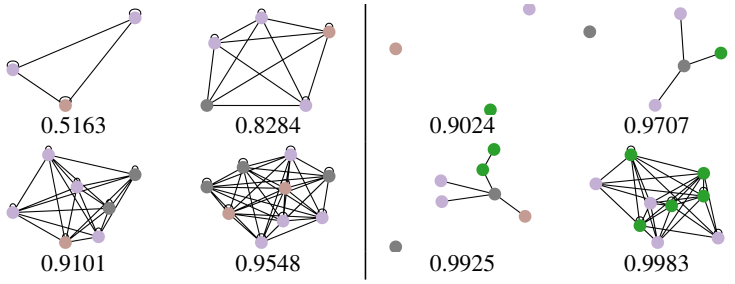
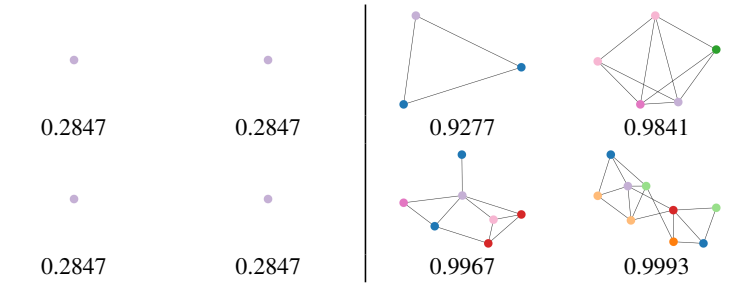
Settings	Generated graphs	
	Mutagenic	Non-mutagenic
GNN-RBN Dense		
GNN-RBN Sparse		
GNN-RBN C,O,H,N only		
XGNN		

Table 3: Mutagenicity data: computed explanation graphs for positive and negative class with target class probabilities assigned by the GNN. **First row**: dense GNN-RBN explanations, achieved setting P(edge)=0.5. **Second row**: sparse GNN-RBN explanations, achieved setting P(edge)=0.3. **Third row**: GNN-RBN explanations using frequent atoms only, achieved setting probability of rare atoms to 0.01 (with the probability for common atoms and edges being 0.5). **Last row**: best results obtained through multiple runs of XGNN, all executed within the same time budget of GNN-RBN.

Acknowledgments

This research was supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation program under GA No 952215.

References

[1] Pablo Barceló, Egor Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020. 5, 11

[2] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009. 2

[3] Lauren Nicole DeLong, Ramon Fernández Mir, Matthew Whyte, Zonglin Ji, and Jacques D Fleuriot. Neurosymbolic ai for reasoning on graph structures: A survey. *arXiv preprint arXiv:2302.07200*, 2023. 2

[4] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 5

[5] L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, 2007. 1

[6] Manfred Jaeger. Relational Bayesian networks. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13)*, pages 266–273, Providence, USA, 1997. Morgan Kaufmann. ISBN 1-55860-485-5. 1, 2

[7] Manfred Jaeger. Learning and Reasoning with Graph Data: Neural and Statistical-Relational Approaches. In Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza, editors, *International Research School in Artificial Intelligence in Bergen (AIB 2022)*, volume 99 of *Open Access Series in Informatics (OASIcs)*, pages 5:1–5:42. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. ISBN 978-3-95977-228-0. 1

[8] Henry Kautz. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125, 2022. 2

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[10] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 2

[11] Luís C Lamb, Artur Garcez, Marco Gori, Marcelo Prates, Pedro Avelar, and Moshe Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. *arXiv preprint arXiv:2003.00330*, 2020. 2

[12] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming, 2018. 2

[13] Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, and Luc De Raedt. From statistical relational to neural symbolic artificial intelligence: a survey. *arXiv preprint arXiv:2108.11451*, 2021. 2

[14] Giuseppe Marra, Michelangelo Diligenti, and Francesco Giannini. Relational reasoning networks, 2023. 2

[15] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016. 1

[16] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 conference of the north American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, 2015. 2

[17] Hao Wang and Dit-Yan Yeung. A survey on bayesian deep learning. *ACM computing surveys (csur)*, 53(5):1–37, 2020. 2

[18] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Natural-parameter networks: A class of probabilistic neural networks. *Advances in neural information processing systems*, 29, 2016. 2

[19] Hao Wang, Chengzhi Mao, Hao He, Mingmin Zhao, Tommi S Jaakkola, and Dina Katabi. Bidirectional inference networks: A class of deep bayesian networks for health profiling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 766–773, 2019. 2

[20] Xiaoqi Wang and Han Wei Shen. Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2022. 6, 7

[21] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 430–438, 2020. 6, 7

A Results on the MAP inference

In this appendix, we provide additional details regarding the experimental procedures employed to investigate the effects of random initialization in Most Probable Explanation (MPE). In our study, we followed a standard practice for MPE, where the initial state of the optimization algorithm is initialized randomly. This randomness in the initial state allows for exploration of various regions in the solution space. We performed likelihood analysis across multiple runs, specifically, we conducted 100 restarts. With the likelihood, we was able to measure the presence of the two motifs inside the graph. Our analysis of the likelihood scores across 100 runs revealed noteworthy insights: the majority of the results obtained from the multiple runs consistently contained at least one motif. This observation indicates that the algorithm are effective in discovering motifs across a wide range of initial conditions. To achieve the optimal solution, characterized by the best likelihood score, a higher number of runs (e.g., 10 restarts) was typically necessary. The likelihood analysis demonstrated that, with 10 restarts, the probability of obtaining the optimal solution increased significantly, with a probability around 0.8. This suggests that the use of multiple restarts can enhance the likelihood of finding both of the two motifs.

In conclusion, our experiments in MPE involved random initialization, and the robustness and reliability of the results were ensured through multiple experimental runs. The likelihood analysis indicated that the algorithms are effective in discovering motifs, and achieving the perfect solution is feasible with a higher number of restarts. These findings provide valuable insights into the practical application of MPE for motif discovery.

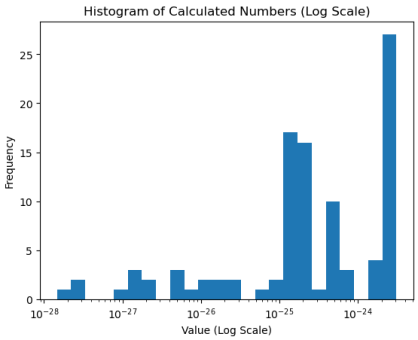
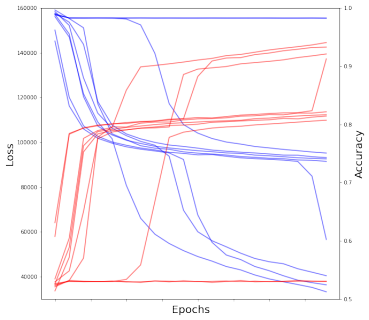


Figure 3: Histograms of the likelihood collected in 100 runs in a Log scale for the *x-axis* (best solutions have an higher likelihood).

B Training: PyTorch vs. Primula

To illustrate the computational equivalence between GNN training and likelihood optimization of its RBN encoding, we here show learning curves obtained by training on a dataset of 5000 graphs provided by Barceló et al.[1] for the synthetic α label (Section 4). We used a PyTorch Geometric (PyG) implementation of an ACR-GNN architecture, and an RBN encoding of that architecture. We used Adam stochastic gradient descent both for training the ACR-GNN in PyG, and to learn the parameters of the RBN encoding using Primula. We rand stochastic gradient descent for 20 epochs,

PyTorch Geometric



Primula

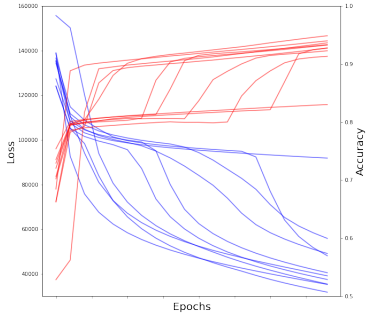


Figure 4: Training performance of PyTorch Geometric (left) and Primula (right) with multiple restarts. It is possible to see how the training curve are analogous. Red curves are accuracy, blue are the loss.

and restarted the learning 10 times with different random initialization. The following plots show the development of accuracy (right y-axes) and loss (left y-axes) during training over the epochs (x-axes).

The learning curves clearly exhibit an analogous behavior of the optimization process. However, in terms of actual time, the 20 epochs in Primula took about 50 times longer than the 20 epochs in PyG. While a certain discrepancy in computation time may be expected due to the matrix-vector based operations in PyTorch vs. object-oriented data representations and computations at the scalar level in Primula, there is at this point no full explanation for the order of magnitude of this discrepancy.