Real-World Benchmarks for Explainable GNN-Based Models

David Tena Cucala
University of Oxford
david.tena.cucala@cs.ox.ac.uk

Artem Orlovskyi
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
orlovskyi.artem@lll.kpi.ua

Sviatoslav Lushnei
Ukrainian Cathlolic University
sviatoslav.lushnei@ucu.edu.ua

Liliana Hotsko
Ukrainian Cathlolic University
liliana.hotsko@ucu.edu.ua

Abstract

Graph Neural Networks (GNNs) have been instrumental in learning transformations of graph data, yet their numeric output lacks symbolic explanation. In this paper, we use a novel approach using Monotonic Graph Neural Networks (MGNNs) to craft transformations that remain explainable via Datalog logical inferences. Our process involves encoding a knowledge graph into a numeric-feature-laden graph, processing it using an MGNN, decoding the result back into a knowledge graph, modifying and analyzing algorithms on different datasets. The critical aspect is ensuring the transformation's equivalence to Datalog rule application. MGNNs maintain monotonicity under homomorphisms, a property essential for symbolic representation. Importantly, this method guarantees that all predictions derived from an MGNN can be explained using Datalog rules, bridging machine learning and symbolic AI. We demonstrate the efficacy of this approach in knowledge graph completion tasks, showcasing competitive performance compared to state-of-the-art systems.

1 Introduction

Knowledge graphs (KGs) serve as valuable representations in various domains, often requiring transformations for enhanced utility. Existing Graph Neural Networks (GNNs) provide effective solutions but lack symbolic interpretability. Our work focuses on leveraging Monotonic Graph Neural Networks (MGNNs) [2] to create transformations that maintain symbolic explainability through Datalog rules. We delineate a process involving graph encoding, MGNN processing, and decoding that ensures predictions are logically derivable, offering a bridge between machine learning and symbolic reasoning.

This report delves into the application of Knowledge Graphs (KGs) within diverse domains like recommendation systems, node classification, and link prediction. KGs depict entities and their interrelations, often undergoing transformations to enhance various applications. For instance, in recommendation systems, augmenting user-item interactions within a graph with external KGs significantly improves recommendation accuracy and diversity. Similarly, KG completion systems enrich input graphs with missing relationships, a process often learned from examples rather than explicitly provided.

Graph Neural Networks (GNNs) stand out as pivotal machine learning models for graph data, commonly employed in three stages when dealing with KGs. Initially, the input KG gets encoded into an embedding space, followed by GNN processing through multiple layers to update feature vectors. Finally, these vectors are decoded to form the output KG.

Despite the effectiveness of these approaches, comprehending the transformations facilitated by GNNs remains challenging due to their numeric calculation-based predictions. There's a rising interest in elucidating GNN predictions, with numerous methods focusing on identifying relevant subgraphs but falling short in explaining predictions symbolically through logical inferences, as offered by formalisms like Datalog.

This paper's primary contribution uses a novel class of GNN-based KG transformations that not only can be trained from examples but also allow for symbolic explanations of predictions using Datalog rules - monotonic GNNs (MGNNs). These MGNNs possess a crucial property ensuring that when input feature vector values increase, no values in the output feature vector decrease. This transformation maintains monotonicity under homomorphisms, a fundamental property aligning with key aspects of Datalog rule application.

In contrast to existing methods, all predictions generated by this transformation can be explained using Datalog rules. Each MGNN corresponds to an equivalent set of rules, enabling a fusion of machine learning and symbolic AI. This integration allows for leveraging MGNNs to make predictions while ensuring that these predictions can always be explained through rules, offering new avenues in decision-making and transparency.

The paper validates this approach's effectiveness by applying it to KG completion classification tasks, showcasing its competitive performance against other systems.

2 Theory

2.1 Definition of MGNN

Monotonic Graph Neural Networks (MGNNs) are a specialized class of Graph Neural Networks (GNNs) designed to ensure both effective learning from graph data and the ability to provide symbolic explanations for their predictions. Unlike conventional GNNs whose predictions are often complex and challenging to interpret, MGNNs offer a unique property: they enable the extraction of Datalog rules that precisely capture their reasoning process. This capability bridges the gap between the numeric computations of GNNs and the symbolic explanations provided by logical rules in Datalog.

At the core of MGNNs lies a structured approach to transforming input knowledge graphs into understandable symbolic representations. Let's delve into the mechanisms of MGNNs and how they encode information from graphs into a format suitable for computation.

MGNN architecture

The MGNN is a graph neural network, a denoising autoencoder by the architecture design. It operates in multiple layers, with each layer performing aggregations over the neighbours of a node with the transformation of the input vector of a given node. The linear and graph convolutional layers are updated then with a monotonically increasing activation function. The formula that defines how an MGNN updates the node labellings in the encoded graph is defined below:

$$\mathbf{v}^{(l)} = \sigma(\mathbf{A}^{(l)}\mathbf{v}^{(l-1)} + \sum_{c \in Col} \mathbf{B}_c^{(l)} \cdot max\{\mathbf{w}^{(l-1)} | w \in N_c(v)\} + \mathbf{b}^{(l)})$$

where the variables mean the following:

- $\mathbf{v}^{(l)}$: Feature vectors at layer l
- $\mathbf{A}^{(l)}$: Matrix for layer l
- $\mathbf{B}_c^{(l)}$: Matrix for color c in layer l
- $\mathbf{b}^{(l)}$: Bias vector for layer l
- σ: Monotonically increasing activation function
- $N_c(v)$: Neighbors of vertex v connected by edges of color c
- $\mathbf{w}^{(l-1)}$: vectors of neighbour vertices
- w: neighbour vertices
- Col: the set of the colours present in the encoding

Significance of MGNNs

The significance of MGNNs lies in their ability to bridge the gap between complex numeric computations of GNNs and the symbolic, logical explanations provided by Datalog rules. By encoding, processing, and decoding knowledge graphs, MGNNs enable the extraction of rules that precisely capture the reasoning of the model in a human-interpretable manner. This not only aids in understanding predictions but also ensures transparency and verifiability in decision-making processes.

The integration of MGNNs with Datalog rules presents a promising pathway to unify machine learning and symbolic artificial intelligence, enabling applications to leverage both worlds effectively.

2.2 Datalog

Datalog is a declarative logic programming language that is often used as a query language for deductive databases. It is based on the principles of first-order logic and allows you to express what you want to know about your data, rather than how to compute it.

Datalog programs are made up of rules that express relationships between facts. These rules can be used to infer new facts from existing ones. Datalog programs are guaranteed to terminate and never go into infinite loops, this makes them very reliable for querying large datasets.

Here is an example of a Datalog program that computes the transitive closure of a relation called friends:

$friends(alice, bob).$

$friends(bob, charlie).$

$transitive\_friends(X, Y) : -friends(X, Y).$

$transitive\_friends(X, Y) : -friends(X, Z), transitive\_friends(Z, Y).$

Datalog is a powerful and versatile language that can be used for a wide range of applications. It is particularly well-suited for tasks that involve reasoning about relationships between data.

Datalog can be used to combine data from multiple sources into a single, unified view, to represent complex knowledge graphs, such as those used in artificial intelligence systems, to analyze the behavior of programs, such as finding bugs or verifying properties.

Limitations and drawbacks to consider when applying it to large-scale data analysis:

1. Datalog relies on repeatedly applying rules to existing facts to derive new ones. This can be inefficient for large datasets, especially for complex queries that involve many rules and joins.
2. Depending on the implementation and query, Datalog might require storing all intermediate results in memory, leading to scalability issues with massive datasets.
3. Compared to specialized big data processing tools, Datalog might lack built-in optimization techniques for handling large data volumes efficiently.
4. Scaling to massive datasets across multiple nodes can be challenging compared to distributed frameworks specifically designed for big data. For complex analysis of big datasets, Datalog might require integration with external libraries and tools which can add complexity.

Datalog remains a valuable tool for specific tasks, especially for logical reasoning and relational queries. However, when dealing with big datasets, its limitations in performance, scalability, and suitability for some analysis tasks should be considered.

2.3 Encoding

In a given GNN-based transformation method for knowledge graphs (KGs), considering KGs as finite sets of facts denoted as A(a) or R(a1, a2), where A represents unary predicates, R stands for binary predicates, and a, a1, a2 are constants representing entities within an application domain.

The transformation, denoted as TM and parameterized by a GNN M, operates by encoding a dataset D into a graph GD labeled with numeric feature vectors. This graph is processed by GNN M and then decoded back to an output dataset TM(D).

The encoding of the dataset involves creating a graph where vertices represent constants and relationships between constants are expressed through edges with different colors. Each predicate is assigned a position in feature vectors, representing unary and binary predicates based on the presence or absence of facts in the dataset. This encoding process allows for the distinction of different types of connections between entities.

Formally, the encoding generates a (Col, δ)-graph GD, where Col represents a set of colors and δ signifies the dimension of the feature vectors. The transformation involves processing this graph with a Monotonic Graph Neural Network (MGNN), which is a specific type of GNN that ensures monotonicity, and its application to the graph induces a sequence of vertex labellings. The output of the MGNN applied to the graph is a labeled graph with the same structure but with vertices labeled by a classification function.

This MGNN operates through layers, utilizing matrices and bias vectors as learnable parameters. It transforms the initial graph by computing feature vectors for each vertex, ensuring specific restrictions such as non-negative weights and monotonic activation and classification functions.

Encoding Graphs for MGNNs

Consider a dataset represented as a knowledge graph consisting of entities and relationships between them. The goal is to transform this graph using an MGNN in such a way that the transformations and predictions made by the model can be explained logically using Datalog rules.

Encoding Process

1. **Graph Representation**: The dataset, comprising facts like relationships between entities (e.g., Likes(Alex, NovelX), Author(BookX, AuthorY)), is encoded into a graph representation.
2. **Vertex Labeling**: Each entity in the knowledge graph becomes a vertex in the encoded graph. Each vertex is labeled with a numeric feature vector that represents various aspects of the entity (e.g., its properties or relationships).
3. **Edge Assignment**: Relationships between entities are represented as edges between vertices in the graph. Different types of relationships are distinguished using colors assigned to the edges.
4. **Predicate Representation**: Unary and binary predicates (representing entity types and relationships) are encoded into feature vectors that label the vertices. For instance, if "Likes" and "Author" are predicates, specific positions in the feature vectors will represent these predicates.

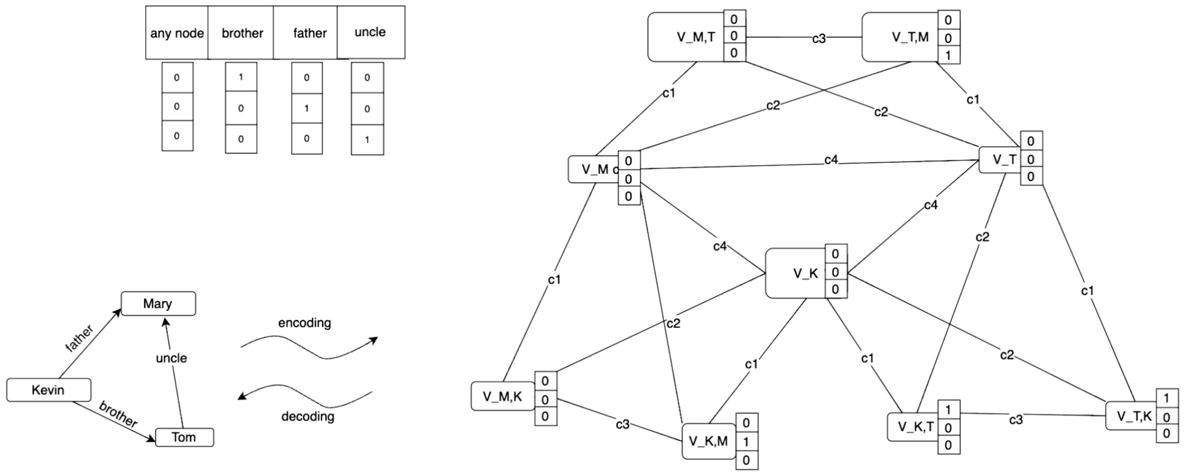

Figure 1: Encoding

3 Properties of the fact explanation algorithm

The objective of this project is to provide real-world knowledge graph datasets, where the facts are given in the form of triples $< subject, predicate, object >$ for the model to train from for graph completion and node classification tasks. The corresponding model will be trained for each dataset, and the equivalent rules will be extracted from the dataset. This implies that the application of a set of Datalog program rules to the dataset will return the same output as the application of a monotonic graph neural network.

The program extracted from the network, as a set of rules, will provide the same results as the network itself, which is guaranteed from the theoretical perspective in the paper about the equivalence of Datalog and MGNN operators [2]. The program extracted is constant free, which leads to the idea that the learned operator is monotonic under homomorphisms. Monotonicity is explained as the invariance of the program operator for the addition of new facts to the datasets, which means a rule will be still extracted if another fact is added. As for the another property, applying the variable-free rule on another arbitrary dataset with the same predicates will not tell a user that the constants were replaced, thus, a new fact from the same rule will be returned.

As the model operator is equivalent to Datalog operator, the rules extracted share the same constraints as Datalog. For example, as Datalog does not support negation operators or disjunctions, the same constraints are on the facts extracted from the MGNN model as well. Still, this is a fundamental part of why the rule extraction algorithm will work only in these conditions.

Mentioning another property, the rules are learned to be generalized. This is a part of the optimized fact explanation algorithm. For instance, let's say that there were two extracted rules from the naïve algorithm: $B(x) \leftarrow A(x)$ and $B(x) \leftarrow A(x) \wedge C$ where C is an arbitrary conjunction. In this case, only the first rule will be left as the another one is redundant in the presence of the rule of a larger set.

Among the primary properties to be discussed is the constraint which rules can be learned by the network. As it is described in the paper, the learned rules are tree-like rules in the form of $A(x) \leftarrow C_x$ or $R(x, y) \leftarrow C_x \wedge B_{x,y} \wedge C_y$ where C_x, C_y are tree-like conjunctions with no common variables and $B_{x,y}$ is a non-empty constant free conjunction of binary atoms. This statement implies a consequential outcome that the model will not learn chain-like rules as were mentioned in the description of Datalog. For example, the rule $grandparent(X, Z) \leftarrow parent(X, Y) \wedge parent(Y, Z)$ cannot be learned by the design of the network since the model cannot capture second-hop neighbourhood variables. Mostly this property of the network will be checked in the extracted rules. Research on the shape of rules and their properties as the structure of a tree will be conducted.

Some practical properties are special for the node classification task, which are described below:

1. **Task description**
   Node Classification is a machine learning task in graph-based data analysis, where the goal is to assign labels to nodes in a graph based on the properties of nodes and the relationships between them. The MGNN model is also applicable for explaining the facts with node-class correspondence. It can be done in a format to check if a concrete node is a member of a specific class, applied for the unary predicates. Meaning that it is done in the following way: For example, for node X and a specific class, it is translated into:

$$is\_class(X) = node.is\_type(class) = R(node,\, is\_type\_relation,\, class)$$

2. **Model specifics**
   A unary predicate correspondence is translated into the binary relation "node special_rdf_istype_relation type class". For datasets with only binary predicates, we can also run explanations for the node correspondence after some reprocessing: creating duplicates of the predicates into unary format. For instance, if there is a relation $X\, daughter\, Y$, a unary predicate $is\_daughter$ can be introduced by extending the existing dataset with relations of that type. Additionally, the tests are added into positive and negative examples.

   Example of the fact to perform node classification explanation:

$$Node\ -\ is\_type\ -\ class$$

$$('2331', 'rdf - syntax - ns\#type', 'is\_daughter')$$

$$'http : //dl - learner.org/carcinogenesis\#d78', 'rdf - syntax - ns\#type', 0.0$$

3. **Encoding properties**
   An additional aspect in node classification is that the model, when trained on a classical encoding scheme, can also undergo rule explanation through our fact explanation algorithm. However, it exhibits sub-optimal results; for instance, it may generate empty rule bodies for certain rules, whereas the encoding scheme used in ICLR22, while potentially offering suggestions that are incorrect or lengthy, still manages to produce some results.

4 Datasets

The rules for the dataset selection were described in the following reasoning. Datasets must be suitable for the task of link prediction and node classification without any time axis or component, so all temporal graph datasets were excluded from the search. A lot of graph datasets are structured in a way of only containing a single predicate, for example whether a user is a friend with someone in a social network graph dataset, a graph of cited papers, movie recommendations, as a person likes a media content or not.

The other approach was to find a dataset with a defined ontology, that would come as a pair of RDF (Resource Description Framework) and OWL (Web Ontology Format) files. The issue behind that idea was in a broader expressiveness of ontology than Datalog. Datalog rules are limited in operations, they cannot be applied for description logic rule extraction. To mention a simpler way to describe an issue with ontologies, Datalog cannot express predicates like owl:SameAs, as they do not recognize the equality predicate as a separate predicate. The expressive power of OWL allows OWL many deductions that cannot be represented in Datalog. There is an another reason why we cannot take the ontology-based datasets for the research. Apart from Datalog, the MGNN model has its constraints as well, so as the same example with owl:Same predicate, can the model learn the equality predicate? That will mean that the graph neural network should treat some nodes as the same, but is not possible with the current architecture and the present encoding. Due to the reasons above, the search for possible datasets with ontologies was limited.

Another strategy to find the real benchmarks was the use of large language models as knowledge graph generators from the natural language text. The example of such a knowledge graph you can find in Figure 2 is the knowledge graph of the Wikipedia page about the fictional character James Bond. The model that was used for this generation for the ChatGPT-3.5 Turbo. Regarding that reasons, there were selected two datasets for the analysis: **Family**, **Inferwiki16k**[1].

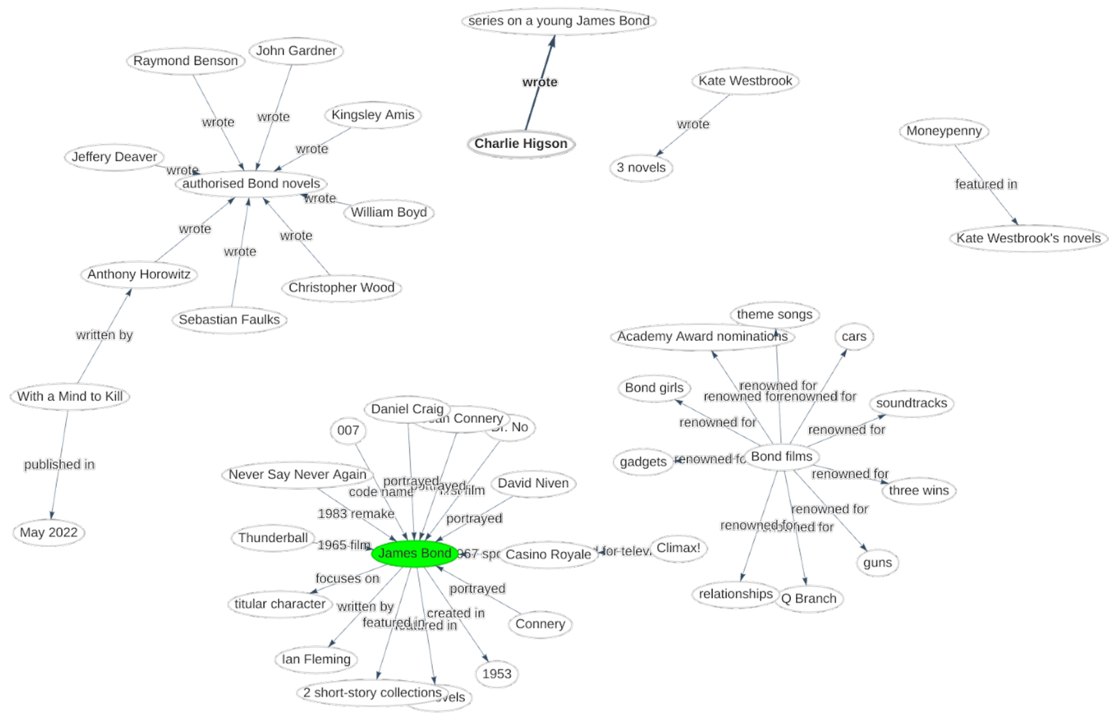

Figure 2: Example of LLM-generated knowledge graph

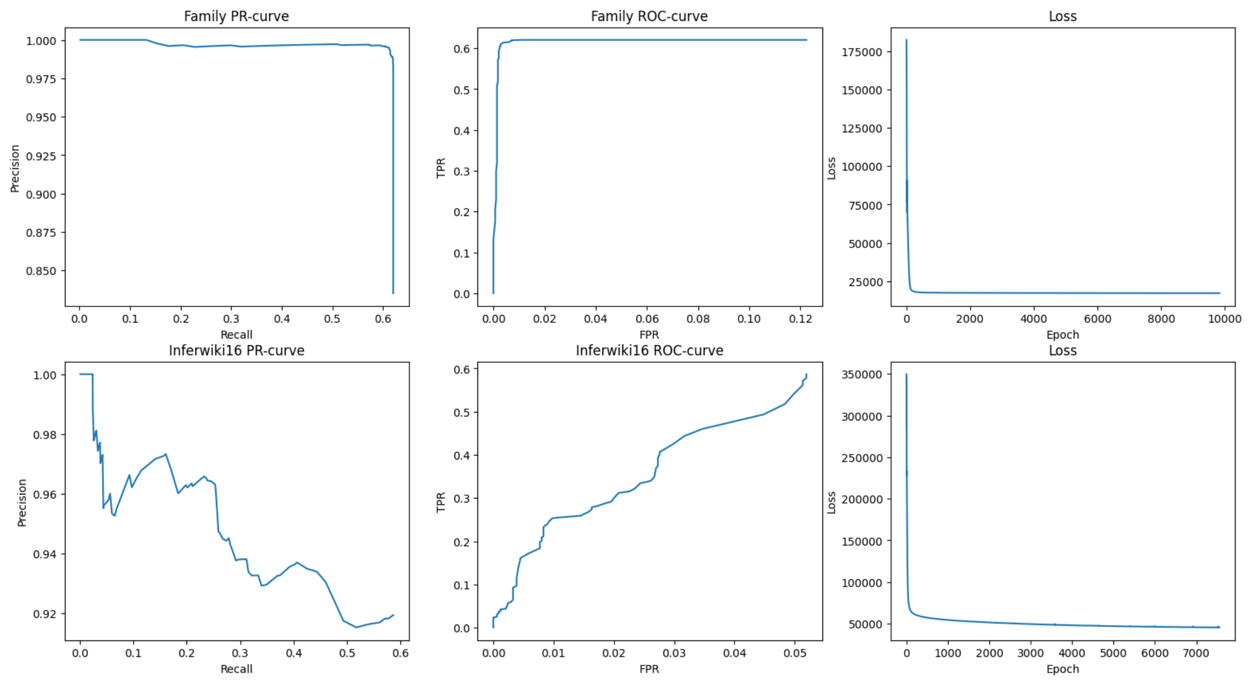

Figure 3: ROC,PR curves with loss results on Family and Inferwiki16k datasets

5 Extracted rules

5.1 Node classification

Let's examine the outcomes of node classification on the family dataset previously mentioned.

1. **Simple Explanations**:
   Individuals correspond to certain classes if they have a relationship with that property. For example, a person X is a mother if there is a relation $X\ mother\ Y$, and the same principle applies to all predicates. Example:

   $$('1189','rdf-syntax-ns\#type','is\_sister') \quad <is\_sister>[?X1]:-<sister>[?X1,?Y1].$$
   $$('306','rdf-syntax-ns\#type','is\_aunt') \quad <is\_aunt>[?X1]:-<aunt>[?X1,?Y1].$$

2. **Gender Revealing**:
   Another interesting aspect is that the model can detect the gender of a person through specific relations. For instance, for the daughter class, it can be revealed if there is a $wife$ relation, $aunt$ relation, $sister$ relation, or even more complex ones like another person being its $wife$. The same principle stands for the is_son predicate with symmetric rules. Examples:

   $$('1086','rdf-syntax-ns\#type','is\_son') \quad <is\_son>[?X1]:-<father>[?X1,?Y1].$$
   $$('693','rdf-syntax-ns\#type','is\_daughter') \quad <is\_daughter>[?X1]:-<wife>[?X1,?Y1].$$

3. **Inverse Rules**:
   If X and Y belong to different ends of the same relation, for example, if X is the son/daughter of Y, and Y is the $father/mother$, or in $wife-husband$ relations. Additionally, more complex rules involving gender identification, such as X being the sister of Y and X being the $husband$, can determine someone's brother (to be more specific, the brother of Y). Examples:

   $$('692','rdf-syntax-ns\#type','is\_father') \quad <is\_father>[?X1]:-<daughter>[?Y1,?X1].$$
   $$('2247','rdf-syntax-ns\#type','is\_husband') \quad <is\_husband>[?X1]:-<wife>[?Y1,?X1].$$

4. **More complex rules**:
   Going beyond simple relations, the model also handles more intricate familial connections, such as explanations for is_aunt and is_uncle, as well as is_nephew is_niece in more complex variants requiring more connections for example is_niece - $is\_daughter$ and it mother has $sister$ relation or for example - X being the aunt of if X has a sister Y and Y is a $mother$. The model can discern these relations and provide meaningful explanations for them. To provide the examples:

$$('214','rdf-syntax-ns\#type','is\_aunt') \quad <is\_aunt>[?X1]:-<sister>[?Y5,?X1],$$
$$<sister>[?Y4,?X1],<sister>[?X1,?Y3],<father>[?Y2,?X1],<brother>[?Y1,?X1].$$

$$('2856','rdf-syntax-ns\#type','is\_aunt')$$
$$<is\_aunt>[?X1]:-<sister>[?X1,?Y2],<father>[?Y1,?X1].$$

   The rule body can also have class correspondence relation as unary predicates:

$$('77','rdf-syntax-ns\#type','is\_sister') \quad <is\_sister>[?X1]:-<mother>[?Y4,?X1],$$
$$<uncle>[?Y3,?X1],<mother>[?Y2,?X1],\ <is\_aunt>[?['Y2']],<is\_sister>[?['X1']],$$
$$<is\_aunt>[?['Y1']],<niece>[?X1,?Y1].$$

5. **Bias Detected**:
   There are rule explanations that can identify biases in the dataset. For instance, X is a $mother$ if it has a $husband$, and X has an is_wife relation if there is a son relation. The same observation was made in more complex relations, such as X being an $aunt$ if it has a sister Y, and Y has a $husband$. Moreover, there was no detection in the other way for $husband$ and $father$ relations and classes. Examples:

$$('771','rdf-syntax-ns\#type','is\_wife') \quad <is\_wife>[?X1]:-<son>[?Y1,?X1].$$
$$('1282','rdf-syntax-ns\#type','is\_mother') \quad <is\_mother>[?X1]:-<husband>[?Y1,?X1].$$

5.2 Link prediction

The link prediction is the task of predicting missing links from the original dataset, which can be known as graph completion. The different types in a transductive setting and the missing links on any graph with the set of the same predicates in an inductive learning setting. The link prediction is possible only with the encoding that was presented in the MGNN paper, which is noted in this report as ICLR22 encoding. The results with the same parts as in the node classification section are written below. These are the results on a InferWiki16k dataset:

1. **Symmetric rules**
   In these rules, the learned properties of symmetric predicates were learned. For example, as individual X is a spouse to Y, then by symmetrical property Y is a spouse to X. The same logic applies to sibling predicate.

$$('Q727580','spouse','Q240756')<spouse>[?X1,?X2]:-<spouse>[?X2,?X1].$$

$$('Q3838','shares\_border\_with','Q3844')$$
$$<shares\_border\_with>[?X1,?X2]:-<shares\_border\_with>[?X2,?X1].$$

2. **Inverse rules**
   By a similar principle as in the node classification section, the predicate meaning can be learned by the reverse edge connections.

$$('Q83286','part\_of','Q36704')<part\_of>[?X1,?X2]:-<has\_part>[?X2,?X1].$$
$$('Q15180','follows','Q2895')<follows>[?X1,?X2]:-<followed\_by>[?X2,?X1].$$

3. **Complex rules**
Multiple-atom rules from InferWiki16 provide insights about the data structure of real-world knowledge. For example:

$$('Q3889','country','Q1029') < country > [?X1, ?X2] :- < capital > [?X2, ?X1],$$

$$< headquarters\_location > [?X2, ?X1], < country > [?Y1, ?X2], < capital\_of > [?X1, ?Y2].$$

This rule introduces a new predicate of $< headquarters\_location >$ into the explanation of $< country >$ predicate, which can mean the connection of a country to a certain business and its location in it.
Another worth-mentioning moment in this rule is the presence of Y variables, which appear during decoding and unfolding the ICLR22 encoding to the original form. The properties of unfolding are still in the process of exploring.

4. **Threshold rule manipulation** As the threshold for contributions is lower, the rules exhibited a reduced length. For example, these are the results from the fact explanation algorithm with the threshold of 0.001 in a family dataset:

$$('451','daughter','597') \quad < daughter > [?X1, ?X2] :- < father > [?X2, ?X1].$$

$$('2131','uncle','2416') \quad < uncle > [?X1, ?X2] :- < niece > [?X2, ?X1].$$

However, as the threshold is elevated, the rules sharing the same grounding exhibit a tendency to be not only longer, but to be not true as well:

$$('2131','uncle','2416') \quad < uncle > [?X1, ?X2] :- < husband > [?X1, ?X2], < niece > [?X2, ?X1].$$

It is worth mentioning that as an atom $< niece > [?X2, ?X1]$ is on the first position in the rule of the lower threshold, it is shifted to a second place in a rule of same grounding fact with a higher threshold. In theory, the conjunction operator is symmetrical and the order of atoms does not matter as the rule is the same if the atoms are swapped. Nevertheless, it means that the fact explanation algorithm handles atom retrieval in a not predictable way, as we do not know which position this atom will take in a high-threshold version of a rule. We are only certain that this atom will be present within that context due to the network's monotonicity.

6 Proposals and future work

Still, there is a vast array of opportunities for enhancements and improvements that can be implemented. To begin with, we will list the actions related to benchmarking and explain the results without altering the model. Subsequently, we will conclude our considerations with proposals to enhance the model.

Firstly, we can extend analyses of the MGNN model and extracted results:

1. **Different aggregations**. Exploring a variety of aggregation methods (ex. 'sum-sum', 'sum-max') to enhance the summarization of results and improve overall analysis. As there might uncover some nuances and surprising results in the data.
2. **Results comparison**.
   (a) **of different encodings**: Encoding serves as a pivotal factor in the model's performance and explainability. For example, 'canonical' encoding is only applicable for rules explanation of node classification form and 'iclr22' performs better in testing. So it is important to evaluate the impact of various data encodings.
   (b) **with other models**: Conducting comparative analyses with alternative models (as there are plenty of other rule extraction algorithms) can give some insights about model strengths and weaknesses.
3. **Metrics**. Developing specialized metrics for evaluating the quality of extracted rules, providing a quantitative assessment of rule effectiveness and informativeness. Despite the necessity for human oversight and analysis of the extracted rules, the integration of metrics will augment a more robust and systematic assessment.
4. **Procuring novel datasets for research purposes**. Incorporating new datasets into the analysis to make the results of the model's performance across diverse data sources. Even though it is the most clear and essential strategy to enhance the quality of the results, it is simultaneously a complex endeavour, requiring a solid selection and a lot of preprocessing efforts.

And now talking about future work related to potential model improvements:

1. **Bayesian search of hyper-parameters selection**. During the training steps, we have tried to change slightly some model parameters, and have observed better results in training performance and obtained metrics. Hence, there exists an opportunity for enhancing the model in this direction, potentially yielding more precise outcomes.
2. **Alternative encoding**. As encoding is a crucial step in model application, and some of the datasets can not be explained due to the inability to convert them to the graph that can be explained by our model (for example happened with KG20c [3] dataset) it may be reasonable to find new encoding techniques.
3. **Optimizing fact explanation algorithm**. Even though there was made some significant optimizations were made, there is room for a more in-depth analysis of its performance to identify further enhancements and explore ways to increase its speed. It is worth noting that it is not a prior step.
4. **Other model architectures**. The current model comprises two hidden layers with a ReLU activation function. Exploring alternative model architectures (ex. defining some more layers, activation functions, and their structure) and assessing their performance could provide valuable insights.
5. **Generalizing fact extraction algorithm**. The explanation algorithm is currently optimized for a specific model with its designated activations, layer count, and aggregation steps. It would be beneficial to generalize the algorithm to a more uniform format to make its application broader. Additionally, considering the potential development of models with different architectures and encoding techniques, such generalization may be crucial for maintaining effectiveness.

7 Conclusion

In this report, we proposed new benchmarks for the rule extraction pipeline with a monotonic graph neural network model to test. The theoretical side of the model-introductory paper explained the equivalence of the extracted rules to the correspondent Datalog ones. The application of benchmarks empirically proved the theoretical properties of the algorithm. Simultaneously, it prompted inquiries into its rule extraction process on real-world benchmarks and opened questions about the meaning of the extracted rules in current and subsequent models.

Acknowledgement

This work was organized by The Center for Responsible AI at the Tandon School of Engineering at New York University (NYU), with support from the Faculty of Applied Sciences of the Ukrainian Catholic University (UCU) in collaboration with a supervisor from University of Oxford.

The Ukrainian students conducting the research would like to express their gratitude to their mentor, David Tena Cucala, for his guidance throughout the project, as well as to the organizers of the RAI research program.

References

[1] Yixin Cao, Xiang Ji, Xin Lv, Juanzi Li, Yonggang Wen, and Hanwang Zhang. Are missing links predictable? an inferential benchmark for knowledge graph completion. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6855–6865, Online, August 2021. Association for Computational Linguistics.

[2] David Jaime Tena Cucala, Bernardo Cuenca Grau, Egor V. Kostylev, and Boris Motik. Explainable GNN-based models over knowledge graphs. In *International Conference on Learning Representations*, 2022.

[3] Hung Nghiep Tran and Atsuhiro Takasu. Exploring scholarly data by semantic query on knowledge graph embedding space, 2022.