Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Logical Expressivity and Explanations for Monotonic GNNs with Scoring Functions

Matthew Morris[1] , **David J. Tena Cucala**[2] , **Bernardo Cuenca Grau**[1]
[1]Department of Compute Science, University of Oxford
[2]Department of Computer Science, Royal Holloway, University of London
{matthew.morris, bernardo.cuenca.grau}@cs.ox.ac.uk, david.tenacucala@rhul.ac.uk

Abstract

Graph neural networks (GNNs) are often used for the task of link prediction: predicting missing binary facts in knowledge graphs (KGs). To address the lack of explainability of GNNs on KGs, recent works extract Datalog rules from GNNs with provable correspondence guarantees. The extracted rules can be used to explain the GNN's predictions; furthermore, they can help characterise the expressive power of various GNN models. However, these works address only a form of link prediction based on a restricted, low-expressivity graph encoding/decoding method. In this paper, we consider a more general and popular approach for link prediction where a scoring function is used to decode the GNN output into fact predictions. We show how GNNs and scoring functions can be adapted to be monotonic, use the monotonicity to extract sound rules for explaining predictions, and leverage existing results about the kind of rules that scoring functions can capture. We also define procedures for obtaining equivalent Datalog programs for certain classes of monotonic GNNs with scoring functions. Our experiments show that, on link prediction benchmarks, monotonic GNNs and scoring functions perform well in practice and yield many sound rules.

1 Introduction

Knowledge graphs (KGs) (Hogan et al. 2022) find use in a variety of applications (Yang and Mitchell 2017; Hamilton, Ying, and Leskovec 2017; Wang et al. 2018) and are often represented as sets of unary and binary facts, where vertices correspond to constants, edges to binary facts, and vertex labels to unary facts. However, KGs are often incomplete – the field of KG completion aims to solve this by predicting missing facts that hold in its (unknown a-priori) complete version. A number of solutions have been proposed for KG completion, including embedding-based approaches with distance-based scoring functions (Abboud et al. 2020), tensor product scoring functions (Yang et al. 2015), recurrent neural networks (Sadeghian et al. 2019), differentiable reasoning (Rocktäschel and Riedel 2017; Evans and Grefenstette 2018), and language models (Xie et al. 2022).

KG completion methods based on graph neural networks (GNNs) (Ioannidis, Marques, and Giannakis 2019; Pflueger, Tena Cucala, and Kostylev 2022; Morris et al. 2024), including R-GCN (Schlichtkrull et al. 2018) and its extensions (Tian et al. 2020; Cai et al. 2019; Vashishth et al. 2019; Yu et al. 2021; Shang et al. 2019; Liu et al. 2021), are widely popular as GNNs can take advantage of the structural information in the graph. As a result, GNN models have highly desirable properties such as invariance under graph isomorphisms and the ability to be applied to graphs of any size.

A limitation of GNN models is that applying them to a KG produces real vector embeddings for the graph's vertices but leaves the edges untouched. Therefore, to predict new edges (the task known as link prediction), many approaches use a decoder based on a *scoring function* (Wang et al. 2017)—a learnable mapping of binary predicates and pairs of embeddings to a real-valued score—which can be easily applied to the embeddings of each pair of vertices in the graph. The score can then be compared against a threshold to decide whether a link should be predicted. This is a very popular approach for link prediction with GNNs (Zhang 2022), including the pioneering R-GCN (Schlichtkrull et al. 2018), which uses DistMult (Yang et al. 2015) as a decoder.

Despite their effectiveness, the predictions of models using GNNs and scoring functions cannot easily be explained, verified, and interpreted (Garnelo and Shanahan 2019). In contrast, logic-based and neuro-symbolic approaches to KG completion often yield rules which can be used to explain the model's predictions. Such methods include RuleN (Meilicke et al. 2018), AnyBURL (Meilicke et al. 2018), RNNLogic (Qu et al. 2020), Neural-LP (Yang, Yang, and Cohen 2017), and DRUM (Sadeghian et al. 2019). To ensure that the rules truly capture the reasons why the model makes a particular prediction, it is important to ensure that they are *faithful* to the model, in the sense that applying the rules to an arbitrary dataset produces the same result as the model. Thus, there is growing interest in models whose predictions can be captured by faithful rules (Tena Cucala, Cuenca Grau, and Motik 2022; Wang et al. 2023).

For GNNs specifically, Tena Cucala et al. (2021) consider GNNs with max aggregation and non-negative weights (along with other restrictions) and show that they can be faithfully characterized by programs of tree-like Datalog rules. Similarly, monotonic max-sum GNNs (Tena Cucala et al. 2023), which encompass both max and sum aggregation, can be faithfully characterised by means of tree-like Datalog programs with inequalities. However, these models do not use scoring functions; instead, they rely on an encoding-decoding strategy that can predict new edges only between constants already linked in the input graph.

On the other hand, (Huang et al. 2023) characterize the expressivity of various GNN and scoring function models using a relational Weisfeiler-Leman algorithm. They also extend the results of (Barceló et al. 2020) to GNNs that perform link prediction relative to a fixed query relation and source vertex, showing that they can only express logical classifiers from a specific fragment of first-order logic. However, their work does not provide a means for obtaining faithful rules from the GNNs. Thus, given the advantages and popularity of GNNs with scoring function decoders, there is an important open question as to how faithful rules may be obtained from them.

Our Contribution In this paper, we bound the logical expressivity of monotonic max-sum GNNs with monotonic scoring functions from above using Datalog rules with inequality, and we provide a method to extract faithful rules from these models to explain their predictions. In particular, we show how many popular scoring functions can be made *monotonically increasing*, provide a means for checking the soundness of Datalog rules that can be used to explain predictions of such models, and prove how existing results about which rule patterns are captured by different scoring functions affect the logical expressivity of the models. We also define procedures for obtaining finite equivalent programs of *tree-like* Datalog rules for (1) any monotonic GNN with max aggregation and monotonically increasing scoring function, and (2) any monotonic GNN with max-sum aggregation and a *non-negative bilinear* scoring function. We conduct experiments on the benchmark link prediction datasets of (Teru, Denis, and Hamilton 2020) and also use the rule-based LogInfer evaluation framework described in (Liu et al. 2023) and (Morris et al. 2024). We find that performance either drops slightly or increases when restricting the model to be monotonic, and many sound rules can be extracted. Therefore, our models maintain performance comparable to standard models while offering the added benefit of explainability.

2 Background

Datalog We fix a signature of $\delta \in \mathbb{N}$ unary predicates and a finite but arbitrary set of binary predicates (also referred to as relations). We also consider countably infinite sets of variables and constants, disjoint with one another and the set of predicates. A term is a variable or a constant. An atom is an expression of the form $R(t_1, ..., t_n)$, where each t_i is a term and R is a predicate with nonnegative integer arity n. A literal is an atom or any inequality $t_1 \not\approx t_2$. A literal is ground if it contains no variables. A fact is a ground atom, and a dataset D is a finite set of facts. We denote the set of constants of a dataset D with $\mathtt{con}(D)$. A (Datalog) rule is a constant-free expression of the form

$$B_1 \wedge ... \wedge B_n \rightarrow H, \tag{1}$$

where $B_1, ..., B_n$ are its body literals and H is its head atom. We do not require rules to be safe: there may be variables in the head that do not occur in a body atom. Furthermore, to avoid redundant rules, we require that each inequality in the body of a rule mentions two different terms. A (Datalog) program is a finite set of rules. A substitution ν maps finitely many variables to constants. For literal α and a substitution ν defined on each variable in α, $\alpha\nu$ is obtained by replacing each occurrence of a variable x in α with $\nu(x)$. For a dataset D and a fact B, we write $D \models B$ if $B \in D$; furthermore, given constants a_1 and a_2, we write $D \models a_1 \not\approx a_2$ if $a_1 \neq a_2$, for uniformity. We write $D \models A$ for a set or conjunction of facts A if D contains each fact in A. The immediate consequence operator T_r for a rule r of form (1) maps a dataset D to dataset $T_r(D)$ containing $H\nu$ for each substitution ν such that $D \models B_i\nu$ for each $i \in \{1, \ldots, n\}$. For a program $\mathcal{P}$, $T_\mathcal{P}(D) = \bigcup_{r\in\mathcal{P}} T_r(D)$. Notice that T_r and $T_\mathcal{P}$ are transformations from datasets to datasets.

Graphs We consider real-valued vectors and matrices. For $\mathbf{v}$ a vector and $i > 0$, $\mathbf{v}[i]$ denotes the i-th element of $\mathbf{v}$. For $\mathbf{A}$ a matrix and $i, j > 0$, $\mathbf{A}[i, j]$ denotes the element in row i and column j of $\mathbf{A}$ (this notation generalises to higher-order tensors). A function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is *monotonically increasing* if $x \leq y$ implies $\sigma(x) \leq \sigma(y)$. We apply univariate functions to vectors element-wise.

We fix a finite set Col of colours, with |Col| equal to the number of binary predicates in the signature. A graph G is a tuple $\langle V, \{E^c\}_{c\in\text{Col}}, \lambda\rangle$ where V is a finite vertex set, each $E^c \subseteq V \times V$ is a set of directed edges, and λ assigns to each $v \in V$ a vector of dimension δ (the number of unary predicates in the signature). We assume that there is a one-to-one correspondence between potential graph vertices and constants in the signature; we typically represent the vertex corresponding to constant a by v^a. When λ is clear from the context, we abbreviate $\lambda(v)$ as $\mathbf{v}$ for each $v \in V$. Graph G is undirected if E^c is symmetric for each $c \in$ Col and is Boolean if $\mathbf{v}[i] \in \{0, 1\}$ for each $v \in V$ and $i \in \{1, ..., \delta\}$.

Graph Neural Networks A graph neural network (GNN) $\mathcal{N}$ with $L \geq 1$ layers is a tuple

$$\langle\ \{\mathbf{A}_\ell\}_\ell,\ \{\mathbf{B}^c_\ell\}_{c,\ell},\ \{\mathbf{b}_\ell\}_\ell,\ \{\sigma_\ell\}_\ell,\ \{\text{agg}_\ell\}_\ell\ \rangle, \tag{2}$$

for each $\ell \in \{1, \ldots, L\}$ and $c \in$ Col, matrices $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ are of dimension $\delta_\ell \times \delta_{\ell-1}$ with $\delta_0 = \delta$, $\mathbf{b}_\ell$ is a vector of dimension δ_ℓ, $\sigma_\ell : \mathbb{R} \rightarrow \mathbb{R}$ is an activation function, and agg_ℓ is a function mapping real-valued multisets to real values.

Applying $\mathcal{N}$ to a graph induces a sequence of labels $\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_L$ (sometimes denoted as $\mathbf{v}_{\lambda_0}, \mathbf{v}_{\lambda_1}, ..., \mathbf{v}_{\lambda_L}$) for each vertex v in the graph as follows. First, $\mathbf{v}_0$ is the initial labelling of node v in the input graph; then, for each $1 \leq \ell \leq L$, $\mathbf{v}_\ell$ is defined by the following expression:

$$\sigma_\ell(\mathbf{b}_\ell + \mathbf{A}_\ell\mathbf{v}_{\ell-1} + \sum_{c\in\text{Col}} \mathbf{B}^c_\ell\ \text{agg}_\ell(\{\!\{\mathbf{u}_{\ell-1} \mid (v, u) \in E^c\}\!\}))$$

The output of $\mathcal{N}$ is a graph with the same vertices and edges as the input graph, but where each vertex is labelled by $\mathbf{v}_L$. Note that this definition is the same as that of (Tena Cucala et al. 2023), except without the classification function.

In this paper, we consider GNNs with max-sum aggregation functions, which generalise both max and sum aggregation. For $k \in \mathbb{N}_0 \cup \infty$, a finite real multiset S, and

$\ell := \min(k, |S|)$, we define max-k-sum$(S) = \sum_{i=1}^{\ell} s_i$, where $s_1, ..., s_\ell$ are the ℓ largest numbers of S, including repeated values. If $\ell = 0$, the sum is defined as 0. Note that max-1-sum is equivalent to max and max-∞-sum is equivalent to sum. If a GNN has agg_ℓ = max-k_ℓ-sum for all $\ell \in \{1, ..., L\}$, where $k_\ell \in \mathbb{N}_0 \cup \infty$, we refer to it as a max-sum GNN (likewise for max GNNs, when every $k_\ell = 1$).

Finally, we say that a max-sum GNN is *monotonic* if for all $\ell \in \{1, ..., L\}$ and $c \in \text{Col}$, all elements of $\mathbf{A}_\ell, \mathbf{B}_\ell^c$ are non-negative and σ_ℓ is monotonically increasing with non-negative range.

Scoring Functions A *scoring function* f with dimension $d_f \in \mathbb{N}$ and threshold $t_f \in \mathbb{R}$ defines $f(R, \mathbf{h}, \mathbf{t}) \mapsto \mathbb{R}$ for all vectors $\mathbf{h}, \mathbf{t} \in \mathbb{R}^{d_f}$ and relations R in our fixed signature. For a set of constants C and function $e : C \to \mathbb{R}^{d_f}$ that assigns an embedding to each constant, we define $\mathcal{F}_f(C, e) = \{R(a, b) \mid a, b \in C, R \text{ is a relation}, f(R, e(a), e(b)) \geq t_f\}$, i.e. the set of facts over constants of C scored as true by f. GNNs, as well as other methods, can be used to learn embedding functions e.

Throughout this paper, we will consider *families* of scoring functions—sets of functions sharing a common form and parameterized by one or more values—where each family comprises all functions generated by varying the parameter values. For example, RESCAL is the family of scoring functions of the form $f(R, \mathbf{h}, \mathbf{t}) = \mathbf{h}^\top \mathbf{M}_R \mathbf{t}$, where $\mathbf{M}_R$ is a real matrix of dimension $d_f \times d_f$.

Dataset Transformations Through GNNs and Scoring Functions A GNN $\mathcal{N}$ with L layers and a scoring function f with dimension $d_f = \delta_L$ can be used to realise a transformation $T_{\mathcal{N},f}$ from datasets to datasets — going forward, when we mention a GNN and scoring function together, we assume $d_f = \delta_L$. To perform the transformation, the input dataset must be first encoded into a graph that can be directly processed by the GNN. To this end, we adopt the so-called *canonical scheme* (Tena Cucala et al. 2023), which introduces a vertex for each constant in the dataset, and no other vertices; then, colours in graphs correspond to binary predicates and indices of feature vectors labelling each vertex to unary predicates in the signature. More precisely, U_p denotes the predicate associated to index p, for each $p \in \{1, \ldots, \delta\}$, and R^c denotes the predicate associated to colour c. Then, the canonical encoding $\texttt{enc}(D)$ of a dataset D is the Boolean graph with vertex v^a for each constant a in D, a c-coloured edge (v^a, v^b) for each fact $R^c(a, b) \in D$, and a vector $\mathbf{v}^a$ labelling each vertex v^a such that vector component $\mathbf{v}^a[p]$ is set to 1 if and only if $U_p(a) \in D$, for each $p \in \{1, \ldots, \delta\}$.

Given a scoring function f, we define the application of the decoder $\texttt{dec}_f$ to a labelled graph G as producing the dataset containing $R^c(a, b)$ for each relation R^c and pair of constants a, b corresponding to vertices in G such that $f(R^c, \mathbf{v}^a, \mathbf{v}^b) \geq t_f$. Thus, we have $\texttt{dec}_f(\mathcal{N}(\texttt{enc}(D)))$

$$= \{R(a, b) \mid a, b \in \texttt{con}(D), f(R, \mathbf{v}^{\mathbf{a}}_{\mathbf{L}}, \mathbf{v}^{\mathbf{b}}_{\mathbf{L}}) \geq t_f\}$$

The *canonical* dataset transformation induced by a GNN $\mathcal{N}$ and a scoring function f is then defined as: $T_{\mathcal{N},f}(D) = \texttt{dec}_f(\mathcal{N}(\texttt{enc}(D)))$. We often abbreviate $\mathcal{N}(\texttt{enc}(D))$ by $\mathcal{N}(D)$. The GNN and scoring function combination can be trained end-to-end.

Soundness and Completeness A Datalog program or rule α is sound for a combination $(\mathcal{N}, f)$ of a GNN $\mathcal{N}$ and a scoring function f if $T_\alpha(D) \subseteq T_{\mathcal{N},f}(D)$ for each dataset D. Conversely, α is complete for $(\mathcal{N}, f)$ if $T_{\mathcal{N},f}(D) \subseteq T_\alpha(D)$ for each dataset D. Finally, we say that α is equivalent to $(\mathcal{N}, f)$ if it is both sound and complete for $(\mathcal{N}, f)$.

3 Sound Rules

In this section, we propose adaptions to several existing families of scoring functions to make them amenable to rule extraction when used with a monotonic max-sum GNN—specifically, we make them *monotonically increasing*. Furthermore, we provide a means for checking whether a Datalog rule is sound for a monotonic max-sum GNN and monotonically increasing scoring function. Finally, we show how existing results about which rule patterns are captured by various scoring function families affect the soundness of rules when they are used with a GNN.

Monotonically Increasing Scoring Functions (Tena Cucala et al. 2023) show that the ability to extract Datalog rules from plain monotonic max-sum GNNs crucially depends on the fact that these models are *monotonic under injective homomorphisms*, meaning that they are agnostic to the particular constants used, and if the model is applied to an arbitrary dataset, then the output values will never decrease as new facts are added to the input. We identify a property of scoring functions which ensures that combining such a function with a monotonic max-sum GNN produces a transformation that is still monotonic in the same way.

Definition 1. *A scoring function f is* monotonically increasing *if for all relations R and vectors $\mathbf{h}, \mathbf{h}', \mathbf{t}, \mathbf{t}' \in \mathbb{R}^{d_f}$ such that $0 \leq \mathbf{h}[i] \leq \mathbf{h}'[i]$ and $0 \leq \mathbf{t}[i] \leq \mathbf{t}'[i]$ $\forall i \in \{1, ..., d_f\}$, we have that $f(R, \mathbf{h}, \mathbf{t}) \leq f(R, \mathbf{h}', \mathbf{t}')$.*

This property guarantees that a scoring function will never decrease its output for a pair of embeddings $\mathbf{h}$ and $\mathbf{t}$ if the values of those embeddings increase. We now consider various scoring function families from the literature and show that some are never monotonically increasing, whereas others can be restricted to be monotonically increasing.

There are a variety of "distance-based models" used for scoring binary facts. For example, TransE (Bordes et al. 2013) has the scoring function $f(R, \mathbf{h}, \mathbf{t}) = -||\mathbf{h} + \mathbf{r}_R - \mathbf{t}||_2$, where each $\mathbf{r}_R$ is a vector of size d_f representing a fixed embedding for relation R. Since this involves a sum in which $\mathbf{h}$ and $\mathbf{t}$ have different signs, there is no collection of vectors $\{\mathbf{r}_R, \ldots\}$ such that the corresponding scoring function is monotonically increasing. This problem occurs in all the distance-based models, including UM (Bordes et al. 2012), TransH (Wang et al. 2014), TransR (Lin et al. 2015), TransD (Ji et al. 2015), TransSparse (Ji et al. 2016), TransM (Fan et

al. 2014), ManifoldE (Xiao, Huang, and Zhu 2016), TransF (Feng et al. 2016), TransA (Xiao et al. 2015), RotatE (Sun et al. 2018), and HAKE (Zhang et al. 2020). A similar issue occurs with some non-distance-based models including ComplEx (Trouillon et al. 2016), QuatE (Zhang et al. 2019), and DualE (Cao et al. 2021), when the computation is considered in terms of real vectors, and is explained in Appendix A.1.

In contrast, the following scoring function families rely on combinations of multiplication, addition, and non-linear functions; they can be modified by restricting their main parameters—usually expressed as vectors, matrices, tensors, or neural network weights—to contain only non-negative values, and by making all non-linear functions in them monotonically increasing and non-negative (e.g., ReLU).

For example, when restricting RESCAL (Nickel et al. 2011) such that each $\mathbf{M}_R$ contains only non-negative numbers, f is monotonically increasing. This approach yields monotonically increasing scoring functions for the following other families: DistMult (Yang et al. 2015), ANALOGY (Liu, Wu, and Yang 2017), HolE (Nickel, Rosasco, and Poggio 2016), TATEC (García-Durán, Bordes, and Usunier 2014), TuckER (Balazevic, Allen, and Hospedales 2019), SimplE (Kazemi and Poole 2018), SME (Bordes et al. 2014), NTN (Socher et al. 2013), SLM (Socher et al. 2013), MLP (Dong et al. 2014), NAM (Liu et al. 2016), and ConvE (Dettmers et al. 2018). These families are fully described in Appendix A.2 and Appendix A.3. This list is non-exhaustive – there may be other scoring function families that can also be modified in this way or others to yield monotonically increasing scoring functions.

Checking Soundness of Rules We provide a means for checking whether a given Datalog rule is sound for a monotonic max-sum GNN and monotonically increasing scoring function. First, we establish that using a monotonic max-sum GNN and monotonically increasing scoring function yields a monotonic dataset transformation.

Lemma 2. *Let $\mathcal{N}$ be a monotonic max-sum GNN and f a monotonically increasing scoring function. Then for all datasets D, D' such that $D \subseteq D'$, $T_{\mathcal{N},f}(D) \subseteq T_{\mathcal{N},f}(D')$.*

The lemma is proved in Appendix B.2 and relies on vertex labels yielded by the GNN not decreasing when facts are added to the input dataset, and outputs of f not decreasing as its input vectors increase. In addition to this, max-sum GNNs and the canonical encoding are agnostic to the particular constants used in the input dataset: they depend only on the structure. These two results show that monotonic max-sum GNNs with monotonically increasing scoring functions are monotonic under injective homomorphisms. We now provide the proposition that allows one to check for sound rules, the proof of which is given in Appendix B.3 and relies on the above results, as well the fact that any dataset satisfying the body of r contains one of the following datasets D_μ (up to constant renaming).

Proposition 3. *Let $\mathcal{N}$ be a monotonic max-sum GNN and f a monotonically increasing scoring function. Let r be a Datalog rule with a binary head atom H, a (possibly empty) set A of body atoms, and a (possibly empty) set I of body inequalities. For each variable x in r, let a_x, b_x be distinct constants uniquely associated with x. Then, r is sound for $(\mathcal{N}, f)$ if and only if $H\mu \in T_{\mathcal{N},f}(D_\mu)$ for each substitution μ mapping the variables of r to constants in the set $\{a_x \mid x \text{ is a variable of } r\}$ such that $\mu(x) \neq \mu(y)$ for each inequality $x \not\approx y \in I$, and $D_\mu = A\mu \cup \{R(b_x, \mu(x)) \mid x \text{ is a variable occurring in } H \text{ but not in } A\}$ where R is an arbitrary but fixed binary predicate.*

The inclusion of each $R(b_x, \mu(x))$ in D_μ accounts for unsafe rules. A rule can be checked for soundness by enumerating each above-defined substitution μ and computing $T_{\mathcal{N},f}(D_\mu)$; then the rule is sound if and only if $H\mu \in T_{\mathcal{N},f}(D_\mu)$ for each such substitution. For example, to check the soundness of rule $r : P_1(x, y) \wedge U_1(x) \rightarrow P_2(x, y)$, we have four possible substitutions. The first two are:

1. $\mu_1 = \{x \mapsto a_x, y \mapsto a_x\}$
   $D_{\mu_1} = \{U_1(a_x), P_1(a_x, a_x)\}$
2. $\mu_2 = \{x \mapsto a_x, y \mapsto a_y\}$
   $D_{\mu_2} = \{U_1(a_x), P_1(a_x, a_y)\}$

and the others are symmetric to them. If $H\mu_i \in T_{\mathcal{N},f}(D_{\mu_i})$ for each substitution, then the soundness of r follows, since any dataset satisfying the body of r contains one of the above datasets, up to constant renaming.

Leveraging Scoring Function Results In this section, we consider GNNs (not necessarily monotonic, or even max-sum) and exploit known results (Sun et al. 2018; Abboud et al. 2020) about the ability of a scoring function family to model different types of rules (referred to as "patterns"), to bound the logical expressivity of the GNN and scoring function. (Pavlović and Sallinger 2022) formally define the capturing of rule patterns. Intuitively, a scoring function family captures a pattern *exactly* if there exists a scoring function in the family such that, for all constant embeddings, if the body of the pattern is satisfied then the head of the pattern is satisfied. To formalise this using our definition of scoring functions, we first define a rule pattern.

Definition 4. *A* rule pattern *is a safe Datalog rule containing only binary predicates from a set of meta predicates $\mathcal{M} = \{M_1, M_2, ...\}$, disjoint from the signature. A rule r* conforms *to a rule pattern ρ if it can be obtained from ρ by replacing each distinct meta binary predicate with a different binary predicate from the signature.*

We can now formalise the capturing of rules. In addition to defining "capturing exactly", we define "capturing universally", which generalises cases such as DistMult being symmetric.

Definition 5. *Let f be a scoring function, C a set of constants, and $e : C \rightarrow \mathbb{R}^{d_f}$ an embedding function. Then the tuple (f, C, e)* captures *a rule $B \rightarrow H$ if for every substitution μ such that $\mathcal{F}_f(C, e) \models B\mu$, we have $\mathcal{F}_f(C, e) \models H\mu$.*

A family of scoring functions F universally captures *a rule pattern ρ if for all $f \in F$, constants C, $e : C \rightarrow \mathbb{R}^{d_f}$, and any rule r conforming to ρ, we have (f, C, e) captures r.*

A family of scoring functions F exactly captures *a rule pattern ρ if for any rule r conforming to ρ, there exists $f \in F$*

such that for all constants C and $e : C \to \mathbb{R}^{d_f}$, (f, C, e) captures r.

We give a sample of some of the pattern capturing results from the literature. Consider the following rule patterns:

1. Hierarchy: $M_1(x, y) \to M_2(x, y)$
2. Symmetry: $M_1(x, y) \to M_1(y, x)$
3. Intersection: $M_1(x, y) \wedge M_2(x, y) \to M_3(x, y)$
4. Inversion: $M_1(x, y) \leftrightarrow M_2(y, x)$
5. Composition: $M_1(x, y) \wedge M_2(y, z) \to M_3(x, z)$

For example, DistMult universally captures symmetry and exactly captures hierarchy (Abboud et al. 2020). TransE exactly captures inversion, composition, and intersection.

We now analyse the effect that a scoring function tuple capturing a rule has when used with a GNN. We find that when a rule is captured across all constants and embeddings, certain rules being sound for the GNN and scoring function implies that there are other rules (not necessarily logically entailed by them) which must also be sound. This limits the class of equivalent programs for the model, and thus its logical expressivity. This is formalised in the following proposition, which is proved in Appendix B.4.

Proposition 6. *Let f be a scoring function and $B \to H$ a safe rule such that for all constants C and $e : C \to \mathbb{R}^{d_f}$, (f, C, e) captures $B \to H$. Let $\mathcal{N}$ be a GNN. Assume that for each atom B_i of B, there is an inequality-free rule $A_i \to B_i$ that is sound for $(\mathcal{N}, f)$. Then the rule $\bigwedge_i A_i \to H$ is sound for $(\mathcal{N}, f)$.*

As a direct consequence of this, notice that when using a scoring function family that universally captures a pattern, the result applies to any rule conforming to the pattern and scoring function in the family.

Corollary 7. *Let F be a scoring function family that universally captures a rule pattern ρ, $B \to H$ a rule conforming to ρ, $\mathcal{N}$ a GNN, and $f \in F$. Assume that for each atom B_i of B, there is an inequality-free rule $A_i \to B_i$ sound for $(\mathcal{N}, f)$. Then the rule $\bigwedge_i A_i \to H$ is sound for $(\mathcal{N}, f)$.*

For example, let F be a scoring function family that universally captures symmetry. Then for any GNN $\mathcal{N}$ and scoring function $f \in F$, if a rule $A \to P_1(x, y)$ is sound for $(\mathcal{N}, f)$, so is $A \to P_1(y, x)$.

A similar result applies to patterns captured exactly, except that it shows that there *exists* a scoring function from the family such that when certain rules are sound, others must also be sound, regardless of the parameters of the GNN.

Corollary 8. *Let F be a scoring function family that exactly captures a rule pattern ρ, and $B \to H$ a rule conforming to ρ. Then there exists $f \in F$ such that: for each GNN $\mathcal{N}$ and inequality-free rules $A_i \to B_i$ sound for $(\mathcal{N}, f)$ for each B_i in B, the rule $\bigwedge_i A_i \to H$ is sound for $(\mathcal{N}, f)$.*

Note that the above $f \in F$ is the same that witnesses the capturing of the rule in the definition of capturing exactly. For example, let F be a scoring function family that exactly captures intersection. Then, there exists $f \in F$ such that for any GNN $\mathcal{N}$, if rules $A_1 \to P_1(x, y)$ and $A_2 \to P_2(x, y)$ are sound for $(\mathcal{N}, f)$, then so is $A_1 \wedge A_2 \to P_3(x, y)$.

As an application of the results in this section, Corollaries 7 and 8 allow one to use rules already known to be sound to derive new sound rules. This avoids both searching for the new rules and using Proposition 3 to verify their soundness.

4 Equivalent Programs

In this section, we provide a means to obtain finite equivalent Datalog programs from GNNs and scoring functions: we do this for arbitrary monotonic max GNNs with monotonically increasing scoring functions, and for monotonic max-sum GNNs with a particular subclass of monotonically increasing scoring functions. This provides an upper bound on the expressive power of such models. To define the rule space, we first provide the definition of a (p, o)-tree-like formula φ_x for variable x.

Definition 9. *For each root variable x:*

1. *$\top$ is tree-like for x;*
2. *for each unary predicate U, $U(x)$ is tree-like for x;*
3. *for all formulas φ_1, φ_2 that share no variables and are tree-like for x, $\varphi_1 \wedge \varphi_2$ is tree-like for x;*
4. *for each variable x, binary predicate R, and tree-like formulas $\varphi_1, ..., \varphi_n$ for distinct variables $y_1, ..., y_n$ where no φ_i contains x and no φ_i and φ_j with $i \neq j$ share a variable, the following is tree-like for x:*

$$\bigwedge_{i=1}^{n} \Big(R(x, y_i) \wedge \varphi_i \Big) \wedge \bigwedge_{1 \leq i \leq j \leq n}^{n} y_i \not\approx y_j \qquad (3)$$

Let φ be a tree-like formula and let x be a variable in φ. The fan-out *of x in φ is the number of distinct variables y_i for which $R(x, y_i)$ is a conjunct of φ. The* depth *of x is the maximal n for which there exist variables $x_0, ..., x_n$ and predicates $R_1, ..., R_n$ such that $x_n = x$ and $R(x_{i-1}, x_i)$ is a conjunct of φ for each $1 \leq i \leq n$. The* depth *of φ is the maximum depth of a variable in φ.*

For p and o natural numbers, a tree-like formula φ is (p, o)-tree-like if, for each variable x in φ, the depth i of x is at most p and the fan-out of x is at most $o(p - i)$.

Using this, we define a *tree-like* rule.

Definition 10. *A Datalog rule is* (p, o)-tree-like *if it is of the form $\varphi_x \wedge \varphi_y \to R(x, y)$, where φ_x, φ_y share no variables, φ_x is (p, o)-tree-like for x, and φ_y is (p, o)-tree-like for y.*

Monotonic Max GNNs We now provide a theorem describing a Datalog program equivalent to a given monotonic max GNNs and monotonically increasing scoring function.

Theorem 11. *Let $\mathcal{N}$ be a monotonic max GNN and f a monotonically increasing scoring function. Let $\mathcal{P}_\mathcal{N}$ be the Datalog program containing, up to variable renaming, each $(L, |Col| \cdot \delta_\mathcal{N})$-tree-like rule without inequalities that is sound for $(\mathcal{N}, f)$, where $\delta_\mathcal{N} = max(\delta_0, ..., \delta_L)$. Then $T_{\mathcal{N},f}$ and $\mathcal{P}_\mathcal{N}$ are equivalent.*

The full proof is given in Appendix B.7. Although the form of the rules extracted is similar to that for plain monotonic max-sum GNNs, the proof of the theorem differs substantially from that of (Tena Cucala et al. 2023, Theorem

13) due to (1) the inclusion of two variables x, y (with corresponding vertices and constants) in the base case and (2) the use of the monotonicity of the scoring function.

Proof sketch. We show that $T_{\mathcal{N},f}(D) = T_{P_\mathcal{N}}(D)$ holds for every dataset D. $T_{P_\mathcal{N}}(D) \subseteq T_{\mathcal{N},f}(D)$ is trivial. So let α be an arbitrary binary fact in $T_{\mathcal{N},f}(D)$. Observe that α can be of the form $R(s,t)$, for distinct constants s, t, or of the form $R(s,s)$, for a constant s. In this sketch, we will consider only the $R(s,t)$ case – the other is similar. We construct a $(L, |\text{Col}| \cdot \delta_\mathcal{N})$-tree-like rule without inequalities r, such that $\alpha \in T_r(D)$ and r is sound for $(\mathcal{N}, f)$. Together, these will imply that $\alpha \in T_{P_\mathcal{N}}(D)$, as required for the proof.

Let $G = (V, E, \lambda)$ be the canonical encoding of D and let $\lambda_0, ..., \lambda_L$ the vertex labelling functions arising from the application of $\mathcal{N}$ to G. We inductively construct Γ, a conjunction of two formulas, one of which is tree-like for x and the other for y. During the inductive construction, we also define a substitution ν from the variables of Γ to the set of constants in D, and a graph U (without vertex labels) where each vertex in U is of the form u^x for x a variable and each edge has a colour in Col. We assign to each vertex in U a *level* between 0 and L.

For the base case, we introduce fresh variables x, y, define $\nu(x) = s, \nu(y) = t$, introduce vertices u^x and u_y, and extend Γ with $U(x)$ for each $U(s) \in D$ (likewise for y for t). For the induction step, consider $1 \leq \ell \leq L$ and assume that all vertices of level greater than or equal to ℓ have been already defined. We then consider each vertex of the form u^x of level ℓ. Let $t = \nu(x)$. For each colour $c \in \text{Col}$, each layer $1 \leq \ell' < \ell$, and each dimension $j \in \{1, ..., \delta_{\ell'-1}\}$ for which there exists some $(v^t, w) \in E^c$, let w be a maximum c-coloured neighbour of v^t in feature vector index j at layer ℓ' (when applying $\mathcal{N}$ to G). Note that w must be of the form v^s for some constant s in D.

We then introduce a fresh variable y and define $\nu(y) = s$. We introduce a vertex u^y of level $\ell - 1$ and an edge $E^c(u^x, u^y)$ to U. Finally, we append to Γ the conjunction $E^c(x, y) \wedge \bigwedge_{U(s) \in D} U(y)$, where U stands in for arbitrary unary predicates. This completes the inductive construction. We then let $H = R(x, y)$, and define our tree-like rule to be $\Gamma \rightarrow H$. The construction of ν means $D \models \Gamma\nu$. So we have $H\nu \in T_r(D)$, with $H\nu = \alpha$, so $\alpha \in T_r(D)$, as required.

We now show that r is sound for $(\mathcal{N}, f)$. Let D' be an arbitrary dataset and α' an arbitrary ground atom such that $\alpha' \in T_r(D')$. Then there exists a substitution ν' such that $D' \models \Gamma\nu'$. Let the graph $G = (V', E', \lambda')$ be the canonical encoding of D' and $\lambda'_0, ..., \lambda'_L$ the functions labelling the vertices of G' when $\mathcal{N}$ is applied to it.

The following statement is proved by induction, which relies on the monotonicity of $\mathcal{N}$: for each $0 \leq \ell \leq L$ and each vertex u^x of U whose level is at least ℓ, we have $\mathbf{v}_{\lambda_\ell}[i] \leq \mathbf{p}_{\lambda'_\ell}[i]$ for each $i \in \{1, ..., \delta_\ell\}$, where $v = v^{\nu(x)}$ and $p = v^{\nu'(x)}$.

Note α' has the form $R(s', t')$ with $s' = \nu'(x)$ and $t' = \nu'(y)$. Now let $v = v^s, w = v^t, p = v^{s'}, q = v^{t'}$. Then $R(s,t) \in T_{\mathcal{N},f}(D)$ implies $f(R, \mathbf{v}_{\lambda_L}, \mathbf{w}_{\lambda_L}) \geq t_f$. The above property ensures that $\mathbf{v}_{\lambda_L}[i] \leq \mathbf{p}_{\lambda'_L}[i]$ and $\mathbf{w}_{\lambda_L}[i] \leq \mathbf{q}_{\lambda'_L}[i]$ for all $i \in \{1, ..., \delta_L\}$. Thus, since f is monotonically increasing, we have $f(R, \mathbf{v}_{\lambda_L}, \mathbf{w}_{\lambda_L}) \leq f(R, \mathbf{p}_{\lambda'_L}, \mathbf{q}_{\lambda'_L})$. So $f(R, \mathbf{p}_{\lambda'_L}, \mathbf{q}_{\lambda'_L}) \geq t_f$, which implies that $R(s', t') \in T_{\mathcal{N},f}(D')$. □

The program $\mathcal{P}_\mathcal{N}$ is computable in principle by enumerating all rules in the finite class of tree-like rules mentioned in the theorem, and then filtering out rules that do not pass the soundness check in Proposition 3. This brute-force procedure can be optimised in practice: for example, by excluding all rules that subsume a known sound rule, since they are redundant; or by exploiting results about the expressiveness of scoring functions to avoid unnecessary soundness checks, as discussed at the end of Section 3.

Note that as a consequence of this theorem, rules conforming to common rule patterns (e.g. symmetry, inversion, hierarchy, composition, intersection, triangle, diamond (Liu et al. 2023), fork, and cup (Morris et al. 2024)) can only be sound if they are subsumed by a sound tree-like rule. For example, if a composition rule of the form $R_1(x, z) \wedge R_2(z, y) \rightarrow R_3(x, y)$ is sound for a monotonic max GNN and a monotonic scoring function, there must exist another sound tree-like rule that subsumes it, such as $R_1(x, z_1) \wedge R_2(z_2, y) \rightarrow R_3(x, y)$.

Monotonic Max-Sum GNNs When seeking to obtain equivalent programs for the larger class of monotonic max-sum GNNs instead of max GNNs, we can no longer consider all monotonically increasing scoring functions. This is because there are a potentially unbounded number of neighbours that contribute to the computation during the execution of the max-sum GNN, so if we tried to follow the same approach as for monotonic max GNNs, we would have to consider an infinite number of tree-like rules. To address this, we prove that an argument of the same type as that of (Tena Cucala et al. 2023) for max-sum GNNs can also be made when a scoring function is considered. We show that the number of neighbours considered in each max-sum aggregation step can be restricted to a finite number without changing the output of the GNN and scoring function on any dataset. This requires, however, an additional restriction on the scoring function: namely, it must be *non-negative bilinear*. This property ensures that there is a scalar value α such that once a vertex feature output by a GNN is greater than it, increasing any input to the scoring function will not result in a change of the output of $T_{\mathcal{N},f}$.

Definition 12. *A* bilinear scoring function f *is one such that for all relations R and vectors $\mathbf{h}, \mathbf{t} \in \mathbb{R}^{d_f}$, we have $f(R, \mathbf{h}, \mathbf{t}) = \mathbf{h}^\top \mathbf{M}_R \mathbf{t}$, where $\mathbf{M}_R \in \mathbb{R}^{d_f \times d_f}$ is a matrix conditioned on R. We say that f is* non-negative *if each $\mathbf{M}_R$ contains only non-negative entries.*

Notice that RESCAL, DistMult, and ANALOGY are bilinear scoring functions. Also, TuckER can be rewritten as one, as shown in Appendix A.2. In addition to restricting the scoring function, we also add the restriction that the activations functions $\{\sigma_\ell\}_{1 \leq \ell \leq L}$ of $\mathcal{N}$ must be unbounded.

The essence of our approach is to compute, for each layer ℓ of the GNN, a non-negative integer $\mathcal{C}_\ell$, called a *capacity*,

such that replacing the number k_ℓ in the aggregation function by $\mathcal{C}_\ell$ does not change the output of the model on any dataset. We compute the capacities $\mathcal{C}_\ell$ using a more complex variant of the algorithm of (Tena Cucala et al. 2023, Algorithm 1), where the role of their GNN classification threshold is replaced by α, defined as follows. In the following, the set $\mathcal{X}_{\ell,i}$ consists of all real numbers that can occur in label vector index i at layer ℓ, when a max-sum GNN $\mathcal{N}$ is applied to a dataset; these sets can be computed as shown in (Tena Cucala et al. 2023, Definition 7).

Definition 13. *Consider a monotonic max-sum GNN $\mathcal{N}$ and a non-negative bilinear scoring function f. For each binary relation R in the signature, where $\mathbf{M}_R$ is its relation matrix in f, we define: $\alpha_R := 1$ if all elements of $\mathbf{M}_R$ are 0 or $\bigcup_i \mathcal{X}_{L,i} = \{0\}$. Otherwise, we define $\alpha_R :=$ the least natural number such that $\alpha_R \cdot w \cdot \epsilon \geq t_f$, where w is the least non-zero element of $\mathbf{M}_R$ and ϵ is the least non-zero element of $\bigcup_i \mathcal{X}_{L,i}$. Then, we define $\alpha :=$ max$\{\alpha_R \mid$ for each binary relation R in the signature$\}$.*

The following theorem then establishes that replacing the aggregation number k_ℓ by the capacity $\mathcal{C}_\ell$ leaves the output of the GNN and scoring function unchanged on any dataset.

Theorem 14. *Let $\mathcal{N}$ be a monotonic max-sum GNN with unbounded activation functions and f a non-negative bilinear scoring function. Let $\mathcal{N}'$ be the GNN obtained from $\mathcal{N}$ by replacing k_ℓ with the capacity $\mathcal{C}_\ell$ for each $\ell \in \{1, ..., L\}$. Then for each dataset D, it holds that $T_{\mathcal{N},f}(D) = T_{\mathcal{N}',f}(D)$.*

The proof is given in Appendix B.9 and depends on the structure of non-negative bilinear scoring functions. Using this, we prove the main result of this section, which is analogous to Theorem 11. The theorem shows that an equivalent program can be constructed for any monotonic max-sum GNN $\mathcal{N}$ and non-negative bilinear scoring function f using rules taken from the finite space of all $(L, |\text{Col}| \cdot \delta_\mathcal{N} \cdot \mathcal{C}_{\mathcal{N},f})$-tree-like rules, where $\mathcal{C}_{\mathcal{N},f} = \max\{\mathcal{C}_1, ..., \mathcal{C}_L\}$. The proof is given in Appendix B.10.

Theorem 15. *Let $\mathcal{N}$ be a monotonic max-sum GNN with unbounded activation functions and f a non-negative bilinear scoring function. Let $\mathcal{P}_\mathcal{N}$ be the Datalog program containing, up to variable renaming, each $(L, |Col| \cdot \delta_\mathcal{N} \cdot \mathcal{C}_{\mathcal{N},f})$-tree-like rule that is sound for $(\mathcal{N}, f)$, where $\delta_\mathcal{N} =$ max$(\delta_0, ..., \delta_L)$. Then $T_{\mathcal{N},f}$ and $\mathcal{P}_\mathcal{N}$ are equivalent.*

5 Experiments

We train monotonic max-sum GNNs with monotonically increasing scoring functions across several link prediction datasets, showing that sound rules can be extracted in practice and that the restriction to monotonicity does not significantly decrease performance. For the model architecture, we fix a hidden dimension of 50, 2 layers, and ReLU activation functions. The GNN definition given in Section 2, chosen to correspond to that of (Tena Cucala et al. 2023) and for ease of presentation, describes aggregation in the reverse direction of the edges. In our experiments, we use the standard approach and aggregate in the direction of the edges.

We use GNNs with max aggregation, and GNNs with sum aggregation, as well as experimenting with 4 different varieties of scoring function families: RESCAL, DistMult, TuckER, and NAM. We train each model for 8000 epochs, except for ones that use TuckER, which we train for 4000, given our computational constraints and how much slower the TuckER models are. For all trained models, we compute standard classification metrics, such as precision, recall, accuracy, F1 score, and area under the precision-recall curve (AUPRC). To select the scoring function threshold, we evaluate the model on the validation set across candidate thresholds and select the one which maximises accuracy. The candidate thresholds are the set of all scores produced on the validation input set.

Each training epoch, for each positive target fact, 10 negative facts are generated by randomly corrupting the binary predicate. These facts are filtered to ensure they do not contain any false negatives (i.e. facts that appear in the training set). We originally corrupted the constants in the positive fact, but found that predicate corruption leads to significantly better performance by the baseline models on standard benchmarks (Teru, Denis, and Hamilton 2020). We train all our models using binary cross entropy with logits (BCE) loss and the Adam optimizer with a standard learning rate of 0.001 and weight decay of $5e^{-4}$.

We train models without restrictions as baselines (denoted by "Standard"), as well as restricting the models to having non-negative weights (denoted by "Monotonic"), by clamping negative weights to 0 after each optimizer step, as in the approach of (Tena Cucala et al. 2021; Morris et al. 2024). When clamping weights, we multiply by 50 the term in the BCE loss function corresponding to the positive examples: without this, we found there to be insufficient positive signal for the training, leading to consistently positive gradients and thus weights that only tended to 0 as training progressed; the value of 50 was arrived at by hyperparameter tuning on standard benchmark datasets. We run each experiment across 5 different random seeds and present the aggregated metrics. Experiments are run using PyTorch Geometric, with a CPU on a Linux server.

Datasets We use 3 standard benchmarks: WN18RRv1, fb237v1, and nellv1 (Teru, Denis, and Hamilton 2020), each of which provides datasets for training, validation, and testing, as well as negative examples and positive targets. Importantly, these benchmarks are also inductive, meaning that the validation and testing sets contain constants not seen during training, so approaches where embeddings are learned for each constant do not work for them.

We also utilize LogInfer (Liu et al. 2023), a framework which augments a dataset by considering Datalog rules conforming to a particular pattern and adding the consequences of the rules to the dataset (we call these *injected* rules). We use the datasets LogInfer-WN-hier (WN-hier) and LogInfer-WN-sym (WN-sym) (Liu et al. 2023), which are enriched with the hierarchy and symmetry patterns, respectively. We also use LogInfer-WN-cup_nmhier (Morris et al. 2024), which was created using a mixture of monotonic and non-monotonic rules: rules from the "cup" ($R(x, y) \wedge S(y, z) \wedge T(w, x) \rightarrow P(x, y)$) and "non-monotonic hierar-

Dataset	Decoder	Model	%Acc	%Prec	%Rec	%F1	AUPRC	Loss	%SO	#1B	#2B
WN-hier	DistMult	Standard	89.87	89.1	90.86	89.97	0.9423	0.11	-	-	-
		Monotonic	85.05	79.18	95.52	86.43	0.809	1.41	52	171	26532
	RESCAL	Standard	92.69	**90.79**	95.02	92.86	**0.9597**	0.09	-	-	-
		Monotonic	88.59	85.3	93.26	89.1	0.863	1.23	**60**	104	17212
	NAM	Standard	92.1	90.22	94.46	92.28	0.9481	0.09	-	-	-
		Monotonic	67.87	68.33	65	64.77	0.6577	1.64	48	141	19244
	TuckER	Standard	**93**	90.51	**96.08**	**93.21**	0.9542	0.09	-	-	-
		Monotonic	87.5	82.91	94.88	88.39	0.8206	1.36	56	89	14498
WN-sym	DistMult	Standard	**96.6**	**95.79**	97.5	**96.63**	**0.9929**	0.09	-	-	-
		Monotonic	92.34	86.92	**99.68**	92.86	0.9561	1.32	88	81	12845
	RESCAL	Standard	96.1	94.71	97.66	96.16	0.9889	0.07	-	-	-
		Monotonic	94.27	89.72	100	94.58	0.9355	1.12	**92**	42	7537
	NAM	Standard	95.72	94.58	97.02	95.78	0.9897	0.06	-	-	-
		Monotonic	72.16	76.71	67.3	68.82	0.6775	1.79	40	88	12457
	TuckER	Standard	96.24	95.03	97.58	96.29	0.9888	0.07	-	-	-
		Monotonic	91.31	85.97	98.84	91.95	0.8892	1.27	80	81	12825
WN-cup_nmhier	DistMult	Standard	74.98	78.44	68.88	73.35	0.8139	0.17	-	-	-
		Monotonic	69.69	71.21	66.45	68.61	0.6953	1.77	40	143	21366
	RESCAL	Standard	**77.98**	77.38	79.1	**78.22**	0.8152	0.15	-	-	-
		Monotonic	73.21	75.91	67.99	71.72	0.7058	1.61	40	31	5006
	NAM	Standard	76.72	74.56	**81.34**	77.72	**0.833**	0.14	-	-	-
		Monotonic	59.74	59.04	63.1	57.47	0.5697	2.41	**45**	354	50014
	TuckER	Standard	77.46	77.61	77.17	77.38	0.8144	0.15	-	-	-
		Monotonic	71.51	**80.15**	57.27	66.78	0.664	1.68	40	22	3772
fb237v1	DistMult	Standard	55.55	53.08	97.1	68.62	0.9611	0.02	-	-	-
		Monotonic	81.85	79.48	86	82.59	0.6884	0.80	-	35366	-
	RESCAL	Standard	58.8	55.38	96.5	70.23	**0.9625**	0.01	-	-	-
		Monotonic	**91**	**96.72**	84.9	**90.41**	0.6114	0.70	-	4797	-
	NAM	Standard	60.5	56.44	96.8	71.17	0.9552	0.01	-	-	-
		Monotonic	64	58.66	97	73.03	0.9557	0.07	-	155348	-
	TuckER	Standard	54.35	52.31	**98.9**	68.42	0.9597	0.01	-	-	-
		Monotonic	90.5	92.07	88.7	90.34	0.6305	-	-	8511	-
WN18RRv1	DistMult	Standard	89.76	83.13	**100**	90.75	0.9337	0.02	-	-	-
		Monotonic	91.52	85.49	**100**	92.18	0.7922	0.79	-	97	11995
	RESCAL	Standard	91.52	85.61	99.88	92.19	0.9306	0.01	-	-	-
		Monotonic	**95.82**	**92.38**	99.88	**95.98**	0.7878	0.77	-	52	6488
	NAM	Standard	78.61	70.96	**100**	82.75	0.9287	0.01	-	-	-
		Monotonic	74.73	66.51	99.88	79.83	0.9251	0.10	-	202	20874
	TuckER	Standard	91.76	85.94	**100**	92.41	**0.9402**	0.01	-	-	-
		Monotonic	92.73	88.3	98.91	93.23	0.7787	0.81	-	65	8134
nellv1	DistMult	Standard	57.65	54.51	92.71	68.56	0.9271	0.09	-	-	-
		Monotonic	65.53	59.24	**100**	74.39	0.9081	1.35	-	631	125094
	RESCAL	Standard	58.94	55.01	99.06	70.72	0.9076	0.04	-	-	-
		Monotonic	70.59	**72.23**	66.59	69.05	0.5372	1.05	-	144	30154
	NAM	Standard	65.53	61.05	86.82	71.15	0.9127	0.03	-	-	-
		Monotonic	53.24	51.69	**100**	68.15	**0.929**	0.44	-	970	176836
	TuckER	Standard	55.53	53.04	99.06	69.06	0.9103	0.04	-	-	-
		Monotonic	**76.82**	68.57	99.29	**81.1**	0.6893	1.21	-	265	53418

Table 1: Results for max GNNs. Loss is from the final epoch on the training set. AUPRC is from the validation set. Other metrics are computed on the test set. %SO is the percentage of LogInfer rules that are sound for the model. #1B and #2B are the number of sound rules with one and two body atoms respectively.

chy" ($R(x, y) \wedge \neg S(y, z) \rightarrow T(x, y)$) patterns, in this case. We use the dataset to test whether the monotonic models can recover the monotonic rules in the dataset, despite the presence of non-monotonic rules. For the LogInfer datasets, each training epoch, 10% of the input facts are randomly set aside and used as ground truth positive targets, whilst the rest of the facts are used as input to the model.

Given that the datasets have no unary predicates, we introduce a dummy unary predicate that holds for every constant in the dataset, yielding the same initial embedding for every node in the encoding. More details on this and other approaches to initial node features in the absence of unary predicates can be found in Appendix A.4.

Rule Extraction On all datasets, we iterate over each Datalog rule in the signature with up to two body atoms and a binary head predicate, and count the number of sound rules, using Proposition 3 to check soundness. On the fb237v1 dataset, we only check rules with one body atom, since the large number of predicates means that searching the space of rules with two body atoms intractable. For datasets created with LogInfer (Liu et al. 2023), which are obtained by appending the consequences of a known set of Datalog rules to a pre-existing dataset, we also check if these rules are sound.

Results The experimental results for max GNNs are provided in Table 1. For each dataset and metric, we highlight in bold the best value achieved by a model variant for that metric. Our findings definitively show that restricting GNNs and scoring functions to be monotonic does not adversely affect model performance in most cases (with the exception of models that use NAM). In fact, in some scenarios, the restrictions cause a significant increase in performance. Similar conclusions can be drawn from our results for sum GNNs as for max GNNs; for completeness, full results for monotonic sum GNNs are given in Table 2, of Appendix C.

Some general patterns emerge. Firstly, loss is always lower for the standard model than its monotonic counterpart. This is due to the restriction on model weights preventing gradient descent from optimising the parameters to the same extent. On the other hand, the monotonicity may help to prevent overfitting on the training set. Likewise, AUPRC is consistently higher on the standard model than the monotonic version. Used as an indication for model performance on the validation set, this suggests that the standard models may be overfitting on the validation facts and struggling to generalise to the test set, in comparison to the monotonic models. Finally, we see that models using NAM as a scoring function consistently struggled when restricted to being monotonic: other scoring functions are thus more suitable when monotonicity is desired.

We obtained a number of sound rules for every monotonic model (columns #1B and #2B in Table 1), showing the efficacy of our rule-checking methodology. On the LogInfer-WN datasets, there are 605 possible rules with one body atom and 90508 with two. On fb237v1, there are 162000 possible rules with one body atom. On WN18RRv1, there are 405 possible rules with one body atom and 49572 with two. On nellv1, there are 980 possible rules with one body atom and 186592 with two. Notice that, for example, nearly all possible rules are sound when using monotonic NAM on nellv1, which corresponds to the perfect model recall and poor precision. We show some randomly sampled sound rules in Table 3 of Appendix C.

On the LogInfer datasets, we find consistently small drops in performance ($\approx 5\%$ less accuracy) when monotonic models are used. However, the restriction enables us to check sound rules for the model: we find many sound rules with one or two body atoms, and also that around half (or sometimes more) of the injected LogInfer rules are sound for the model. On WN-cup_nmhier, the monotonic models showed their ability to still learn and recover some of the injected monotonic rules, even in the presence of some injected rules that are explicitly non-monotonic.

On fb237v1, we see significantly better performance by monotonic models over their standard counterparts. The same is seen on nellv1. On WN18RRv1, the monotonic models again outperform the standard ones, but by smaller margins. These results are somewhat surprising, but encouraging, since one would expect monotonic models to have a greater advantage on the LogInfer datasets, where the underlying patterns in the data are explicitly monotonic. Finally, we note that the monotonic model performance on fb237v1 and WN18RRv1 is significantly better than those of MGNNs (Tena Cucala et al. 2021), highlighting the benefits of using a scoring function as a decoder instead of their dataset encoding-decoding scheme, since the recall of their models is upper-bounded by the number of positive test pairs of constants that also appear in the input dataset.

6 Conclusion

In this paper, we showed how scoring functions can be made monotonically increasing, how sound Datalog rules can be found for monotonic GNNs with monotonic scoring functions, and showed how existing results about scoring functions capturing rule patterns impact the expressivity of GNN models that use those scoring functions. We also provided ways to obtain an equivalent program for any monotonic GNN with max aggregation and monotonically increasing scoring function, and any monotonic GNN with max-sum aggregation and a scoring function that is non-negative bilinear. We showed through our experiments that, in practice, the performance of GNNs with scoring functions does not drop substantially when applying the restrictions that make them monotonic, and in some cases increase the model performance. We also showed that, in practice, many sound rules can be recovered from these monotonic models, which can be used to explain their predictions.

A limitation of this work is that we only consider non-negative bilinear scoring functions when obtaining equivalent programs for max-sum GNNs: the approach we describe may also work for other classes of monotonically increasing scoring functions, such as SimplE. For future work, we aim to consider more advanced training paradigms for the monotonic models.

Acknowledgments

Matthew Morris is funded by an EPSRC scholarship (CS2122_EPSRC_1339049). This work was also supported by Samsung Research UK, the EPSRC projects UKFIRES (EP/S019111/1) and ConCur (EP/V050869/1). The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work http://dx.doi.org/10.5281/zenodo.22558.

For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

References

Abboud, R.; Ceylan, I.; Lukasiewicz, T.; and Salvatori, T. 2020. Boxe: A box embedding model for knowledge base completion. *Advances in Neural Information Processing Systems* 33:9649–9661.

Balazevic, I.; Allen, C.; and Hospedales, T. 2019. Tucker: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J.-P. 2020. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*.

Berg, R. v. d.; Kipf, T. N.; and Welling, M. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*.

Bordes, A.; Glorot, X.; Weston, J.; and Bengio, Y. 2012. Joint learning of words and meaning representations for open-text semantic parsing. In *Artificial intelligence and statistics*, 127–135. PMLR.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26.

Bordes, A.; Glorot, X.; Weston, J.; and Bengio, Y. 2014. A semantic matching energy function for learning with multi-relational data: Application to word-sense disambiguation. *Machine Learning* 94:233–259.

Cai, L.; Yan, B.; Mai, G.; Janowicz, K.; and Zhu, R. 2019. Transgcn: Coupling transformation assumptions with graph convolutional networks for link prediction. In *Proceedings of the 10th international conference on knowledge capture*, 131–138.

Cao, Z.; Xu, Q.; Yang, Z.; Cao, X.; and Huang, Q. 2021. Dual quaternion knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 6894–6902.

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Dong, X.; Gabrilovich, E.; Heitz, G.; Horn, W.; Lao, N.; Murphy, K.; Strohmann, T.; Sun, S.; and Zhang, W. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 601–610.

Evans, R., and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research* 61:1–64.

Fan, M.; Zhou, Q.; Chang, E.; and Zheng, F. 2014. Transition-based knowledge graph embedding with relational mapping properties. In *Proceedings of the 28th Pacific Asia conference on language, information and computing*, 328–337.

Feng, J.; Huang, M.; Wang, M.; Zhou, M.; Hao, Y.; and Zhu, X. 2016. Knowledge graph embedding by flexible translation. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

García-Durán, A.; Bordes, A.; and Usunier, N. 2014. Effective blending of two and three-way interactions for modeling multi-relational data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I 14*, 434–449. Springer.

Garnelo, M., and Shanahan, M. 2019. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences* 29:17–23.

Hamaguchi, T.; Oiwa, H.; Shimbo, M.; and Matsumoto, Y. 2017. Knowledge transfer for out-of-knowledge-base entities: a graph neural network approach. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 1802–1808.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30.

Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; de Melo, G.; Gutierrez, C.; Kirrane, S.; Gayo, J. E. L.; Navigli, R.; Neumaier, S.; Ngomo, A. N.; Polleres, A.; Rashid, S. M.; Rula, A.; Schmelzeisen, L.; Sequeda, J. F.; Staab, S.; and Zimmermann, A. 2022. Knowledge graphs. *ACM Comput. Surv.* 54(4):71:1–71:37.

Huang, X.; Romero, M.; Ceylan, I.; and Barceló, P. 2023. A theory of link prediction via relational weisfeiler-leman on knowledge graphs. *Advances in Neural Information Processing Systems* 36:19714–19748.

Ioannidis, V. N.; Marques, A. G.; and Giannakis, G. B. 2019. A recurrent graph neural network for multi-relational data. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8157–8161. IEEE.

Ji, G.; He, S.; Xu, L.; Liu, K.; and Zhao, J. 2015. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint*

conference on natural language processing (volume 1: Long papers), 687–696.

Ji, G.; Liu, K.; He, S.; and Zhao, J. 2016. Knowledge graph completion with adaptive sparse transfer matrix. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Kazemi, S. M., and Poole, D. 2018. Simple embedding for link prediction in knowledge graphs. *Advances in neural information processing systems* 31.

Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

Liu, Q.; Jiang, H.; Evdokimov, A.; Ling, Z.-H.; Zhu, X.; Wei, S.; and Hu, Y. 2016. Probabilistic reasoning via deep learning: Neural association models. *arXiv preprint arXiv:1603.07704*.

Liu, S.; Grau, B.; Horrocks, I.; and Kostylev, E. 2021. Indigo: Gnn-based inductive knowledge graph completion using pair-wise encoding. *Advances in Neural Information Processing Systems* 34:2034–2045.

Liu, S.; Cuenca Grau, B.; Horrocks, I.; and Kostylev, E. V. 2023. Revisiting inferential benchmarks for knowledge graph completion. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 461–471.

Liu, H.; Wu, Y.; and Yang, Y. 2017. Analogical inference for multi-relational embeddings. In *International conference on machine learning*, 2168–2178. PMLR.

Meilicke, C.; Fink, M.; Wang, Y.; Ruffinelli, D.; Gemulla, R.; and Stuckenschmidt, H. 2018. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17*, 3–20. Springer.

Morris, M.; Tena Cucala, D.; Cuenca Grau, B.; and Horrocks, I. 2024. Relational graph convolutional networks do not learn sound rules. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 21, 897–908.

Nickel, M.; Tresp, V.; Kriegel, H.-P.; et al. 2011. A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, 3104482–3104584.

Nickel, M.; Rosasco, L.; and Poggio, T. 2016. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Pavlović, A., and Sallinger, E. 2022. Expressive: A spatio-functional embedding for knowledge graph completion. In *The Eleventh International Conference on Learning Representations*.

Pflueger, M.; Tena Cucala, D. J.; and Kostylev, E. V. 2022. Gnnq: A neuro-symbolic approach to query answering over incomplete knowledge graphs. In *International Semantic Web Conference*, 481–497. Springer.

Qu, M.; Chen, J.; Xhonneux, L.-P.; Bengio, Y.; and Tang, J. 2020. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *International Conference on Learning Representations*.

Rocktäschel, T., and Riedel, S. 2017. End-to-end differentiable proving. *Advances in neural information processing systems* 30.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. Drum: End-to-end differentiable rule mining on knowledge graphs. *Advances in Neural Information Processing Systems* 32.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*, 593–607. Springer.

Shang, C.; Tang, Y.; Huang, J.; Bi, J.; He, X.; and Zhou, B. 2019. End-to-end structure-aware convolutional networks for knowledge base completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 3060–3067.

Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems* 26.

Sun, Z.; Deng, Z.-H.; Nie, J.-Y.; and Tang, J. 2018. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*.

Tena Cucala, D.; Cuenca Grau, B.; Kostylev, E. V.; and Motik, B. 2021. Explainable gnn-based models over knowledge graphs. In *International Conference on Learning Representations*.

Tena Cucala, D.; Cuenca Grau, B.; Motik, B.; and Kostylev, E. V. 2023. On the correspondence between monotonic max-sum gnns and datalog. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 658–667.

Tena Cucala, D. J.; Cuenca Grau, B.; Motik, B.; and Kostylev, E. V. 2025. From monotonic graph neural networks to datalog and back: Expressive power and practical applications. Under review: not yet published.

Tena Cucala, D.; Cuenca Grau, B.; and Motik, B. 2022. Faithful approaches to rule learning. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 484–493.

Teru, K.; Denis, E.; and Hamilton, W. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, 9448–9457. PMLR.

Tian, A.; Zhang, C.; Rang, M.; Yang, X.; and Zhan, Z. 2020. Ra-gcn: Relational aggregation graph convolutional network for knowledge graph completion. In *Proceedings of the 2020 12th international conference on machine learning and computing*, 580–586.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *International conference on machine learning*, 2071–2080. PMLR.

Vashishth, S.; Sanyal, S.; Nitin, V.; and Talukdar, P. 2019. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*.

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28.

Wang, Q.; Mao, Z.; Wang, B.; and Guo, L. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29(12):2724–2743.

Wang, H.; Zhang, F.; Wang, J.; Zhao, M.; Li, W.; Xie, X.; and Guo, M. 2018. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM international conference on information and knowledge management*, 417–426.

Wang, X.; Tena Cucala, D.; Cuenca Grau, B.; and Horrocks, I. 2023. Faithful rule extraction for differentiable rule learning models. In *The Twelfth International Conference on Learning Representations*.

Xiao, H.; Huang, M.; Hao, Y.; and Zhu, X. 2015. Transa: An adaptive approach for knowledge graph embedding. *arXiv preprint arXiv:1509.05490*.

Xiao, H.; Huang, M.; and Zhu, X. 2016. From one point to a manifold: knowledge graph embedding for precise link prediction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 1315–1321.

Xie, X.; Zhang, N.; Li, Z.; Deng, S.; Chen, H.; Xiong, F.; Chen, M.; and Chen, H. 2022. From discrimination to generation: Knowledge graph completion with generative transformer. In *Companion Proceedings of the Web Conference 2022*, 162–165.

Yang, B., and Mitchell, T. 2017. Leveraging knowledge bases in lstms for improving machine reading. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1436–1446.

Yang, B.; Yih, S. W.-t.; He, X.; Gao, J.; and Deng, L. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable learning of logical rules for knowledge base reasoning. *Advances in neural information processing systems* 30.

Yu, D.; Yang, Y.; Zhang, R.; and Wu, Y. 2021. Knowledge embedding based graph convolutional network. In *Proceedings of the Web Conference 2021*, 1619–1628.

Zhang, M., and Chen, Y. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31.

Zhang, S.; Tay, Y.; Yao, L.; and Liu, Q. 2019. Quaternion knowledge graph embeddings. *Advances in neural information processing systems* 32.

Zhang, Z.; Cai, J.; Zhang, Y.; and Wang, J. 2020. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 3065–3072.

Zhang, M. 2022. Graph neural networks: link prediction. *Graph Neural Networks: Foundations, Frontiers, and Applications* 195–223.