Graph Collaborative Reasoning

Hanxiong Chen
Rutgers University
New Brunswick, NJ, US
hanxiong.chen@rutgers.edu

Yunqi Li
Rutgers University
New Brunswick, NJ, US
yunqi.li@rutgers.edu

Shaoyun Shi
Tsinghua University
Beijing, China
shisy17@mails.tsinghua.edu.cn

Shuchang Liu
Rutgers University
New Brunswick, NJ, US
shuchang.liu@rutgers.edu

He Zhu
Rutgers University
New Brunswick, NJ, US
hz375@cs.rutgers.edu

Yongfeng Zhang
Rutgers University
New Brunswick, NJ, US
yongfeng.zhang@rutgers.edu

ABSTRACT

Graphs can represent relational information among entities and graph structures are widely used in many intelligent tasks such as search, recommendation, and question answering. However, most of the graph-structured data in practice suffer from incompleteness, and thus link prediction becomes an important research problem. Though many models are proposed for link prediction, the following two problems are still less explored: (1) Most methods model each link independently without making use of the rich information from relevant links, and (2) existing models are mostly designed based on associative learning and do not take reasoning into consideration. With these concerns, in this paper, we propose Graph Collaborative Reasoning (GCR), which can use the neighbor link information for relational reasoning on graphs from logical reasoning perspectives. We provide a simple approach to translate a graph structure into logical expressions so that the link prediction task can be converted into a neural logic reasoning problem. We apply logical constrained neural modules to build the network architecture according to the logical expression and use backpropagation to efficiently learn the model parameters, which bridges differentiable learning and symbolic reasoning in a unified architecture. To show the effectiveness of our work, we conduct experiments on graph-related tasks such as link prediction and recommendation based on commonly used benchmark datasets, and our graph collaborative reasoning approach achieves state-of-the-art performance.

CCS CONCEPTS

• **Computing methodologies → Logical and relational learning**; **Machine learning**; **Neural networks**; • **Information systems → Recommender systems**.

KEYWORDS

Collaborative Reasoning; Relational Reasoning; Neural-Symbolic Learning and Reasoning; GNNs; Recommendation; Link Prediction

ACM Reference Format:
Hanxiong Chen, Yunqi Li, Shaoyun Shi, Shuchang Liu, He Zhu, and Yongfeng Zhang. 2022. Graph Collaborative Reasoning. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3488560.3498410

1 INTRODUCTION

Graph is able to describe the entities and their relations in many real-world systems and research problems, such as e-commerce user-item interactions, social networks, citation networks and knowledge graphs. Though graphs can encode rich relationships among plenty of entities, they still suffer from incompleteness [28, 39]. This issue gives rise to the link prediction task, which is to learn representations from the known data and then predict the potential valid connections. Link prediction is essential to many tasks such as knowledge graph reasoning, entity search, recommender systems and question answering.

Recent years have witness the success of knowledge graph embedding methods for link prediction [2, 5, 34, 41, 42]. The basic idea is to encode the entities and their relations into a low dimensional vector space while the inherent structure information of the graph is preserved. However, one drawback of these embedding-based models is that they usually process each *(entity, relation, entity)* triplet independently without explicitly considering the information from neighborhood links, though information from neighbourhood nodes is considered. As a result, these methods are not able to capture the rich information from the neighbor connections and hence result in less informative embeddings [1, 20].

Another line of research is graph neural networks (GNNs), which have shown the power in many graph-related problems [11, 15, 36]. These approaches are able to learn effective entity representations by aggregating its own representation and the representations of surrounding neighbors. The nodes in the graph can exchange information through message passing [7], which alleviates the problem of aforementioned embedding-based methods. Despite that GNNs could capture more information than those shallow embedding-based models, their key idea for handling link prediction tasks are actually similar—they aim to learn embeddings to capture the similarity patterns among entities, so that link prediction can be conducted by calculating the similarity for a pair of nodes over a specific relation. However, most GNN approaches are designed from a perceptual perspective and they seldom consider the logical relationship among entities and links for relational reasoning.

Logical reasoning is an essential and many times a natural way to conduct reasoning on graphs for two reasons. First, many triplets in the graph may be logically related and can be modeled together through logical connections. Take knowledge graph for example, the triplet (*x, capitalOf, y*) logically implies the relation (*x, locatedIn, y*). Thus, we can use implication operations in predicate logic to describe this connection between the two triplets as (*x, capitalOf, y*) $\rightarrow$ (*x, locatedIn, y*). The logical relationship among triplets, if accurately captured, would be helpful for predicting unknown links. Second, each triplet can be naturally represented as a predicate in logical reasoning, which makes it easy to model the link prediction task as a reasoning process. For example, we can treat the target triplet (*x, locatedIn, y*) as a predicate expression *locatedIn(x,y)*. Then, the link prediction task can be formulated as answering whether the logical expression *capitalOf(x,y)* $\rightarrow$ *locatedIn(x,y)* is true, given that the predicate *capitalOf(x,y)* is true. If the logical expression is true, then we can infer that the target predicate should be true. In other words, the target triplet is a valid link.

In this paper, we explore an approach that transforms the link prediction task into a logical reasoning process on graphs. Our goal is to model the structure of a graph as simple Horn clauses so that link prediction can be conducted via logical reasoning. Inspired by [3, 31], we apply modularized logical neural networks to learn the logical operations. Instead of using explicit hand-crafted logic rules as many previous approaches did, we introduce a method to convert graph structures into Horn clauses as potential rules to be learnt. The logical relations can be captured by the neural networks so that relational reasoning can be conducted on graphs.

Technically, we propose a Graph Collaborative Reasoning (GCR) framework for relational reasoning over graphs. Specifically, we consider that links (or triplets) are potentially related to each other if they are connected by shared nodes. Based on this, we can infer a link through its neighbor links for relational reasoning. To compute the Horn clauses via deep neural networks, we encode each triplet as a predicate embedding, i.e., each entity in a given triplet is represented as a vector embedding and each relation is modeled as a neural module to encode the triplet. With the encoded predicate embeddings, we can construct the network structure using the neural modules in accordance with the modeled Horn clauses. The key benefits of our design compared to previous works are four aspects. First, we can take advantage of GNN strategies to aggregate rich information from neighbor links through message passing to make link predictions. Second, we consider logical reasoning for link prediction, which can make use of the logical relationships between links. Third, we incorporate logical reasoning without manually predefined rules, which makes our method easily adaptable to different scenarios. Finally, our model can handle uncertainty in logical reasoning. Our contributions can be summarized as follows:

- We introduce a new view of the link prediction task from logical reasoning perspectives. In this way, the link prediction task is translated into a true/false evaluation problem of predicate logical expressions.
- We propose the Graph Collaborative Reasoning (GCR) model, which conducts relational reasoning by taking advantage of the neighbor link information for message passing.
- We show the effectiveness of our approach on various graph relational reasoning tasks on several real-world graph datasets.

In the following, we will present related works in Section 2. After that, in Section 3, we formalize the link prediction task in logical language. Section 4 presents the details of our model and Section 5 gives our experimental setup and results. We will conclude this work with outlooks for future work in Section 6.

2 RELATED WORKS

Existing techniques for link prediction can be roughly classified into three categories: translation-based, tensor factorization-based, and neural network-based. The translation-based models [2, 13, 17, 41, 43] translate a head embedding into a tail embedding via a relation. The scoring function is defined as the distance between the translated head embedding and the tail embedding. Tensor factorization-based methods, such as RESCAL [23], ComplEx [34], RotatE [32], DistMult [42] and HolE [22], consider the graph as a 3D adjacency matrix, which represents the head, tail and relation embeddings along each dimension. They apply operations such as linear mapping (RotatE), bilinear mapping (DistMult and ComplEx) or circular correlation operation (HolE) to obtain low-dimensional representations for each entity and relation. The deficiency of these methods lie in treating each triplet independently and thus the rich structural information in the graph cannot be adequately used.

Neural network-based methods, such as CNN-based [5, 21] and GNN-based [30, 35] methods, use neural network structures to capture the rich information among the links. CNN-based methods, such as ConvE [5], use 2D convolution layers to extract the relationships between head entity embeddings and relation embeddings. The relations are represented as multiple feature maps, which are obtained through various filters. Then all these feature maps are concatenated and fed into a fully connected layer to get the projected embeddings for similarity calculation with the tail entity embeddings. These models still consider each triplet independently which also suffer from the aforementioned problem. GNN-based models, such as GCN [15], GAT [36] and GraphSAGE [11], can help to resolve this issue by using message passing strategy to aggregate information from neighbor nodes so as to enrich the vector representation of each entity. Since the original design of these models are based on homogeneous graphs, they are unable to handle multi-relational link prediction tasks. Later, an extension of GCN named R-GCN [30] is proposed to deal with multi-relational data. However, none of the above methods consider the logical relationships between nodes/links in the graph for relational reasoning.

Recently, there have been some research works on integrating logic into link prediction. The related approaches can be broadly classified into hard-logic-based and soft-logic-based methods. The hard-logic-based methods focus on applying hard logic rules to the learning process [4, 9, 27, 37, 38]. The problem of using hard logic rules is that the model does not tolerate to any violation. As a result, the logic rules need to be carefully designed and the application scenarios can be limited. For example, the hard-rule-based methods are able to handle rules like "x is the capital of y implies x is located in y," however, they can hardly deal with rules like "user purchased a cellphone x implies that user probably will purchase a phone case y," since the rule can be violated in some cases.

To solve the problem, soft-logic-based methods try to handle this uncertainty by using soft logic constraints, which assign probabilities to the logic rules to make the model more tolerate to exceptions

[8, 10, 12, 24, 25, 44, 45]. One powerful model is pLogicNet [24], which is based on Markov Logic Network. It can learn the weight for each predefined logic rule to handle uncertainty and noise. However, these models usually need to ground the logic rules by traversing all potential valid links in a graph, which makes these methods difficult to scale to large graphs. Though recent works try to get rid of the grounding process by directly adding rule-based constraints on the relation vector representations [6, 8, 19], they can only deal with simple rules such as $(x, hypernym, y) \rightarrow (y, hyponym, x)$.

All of the aforementioned logical rule-based methods need explicitly predefined logic rules either as part of a pipelined framework or as a constraint of the learning process. This makes the model highly dependent on the effectiveness of the predefined logic rules. An open challenge, as mentioned in [8], is to design models that can handle not only simple (manually) designed rules but also complex learned rules while considering the scalability and uncertainty. Although soft-logic-based methods can be more flexible than hard-logic-based approaches, these works all need the background knowledge of the data so that logical rules can be created reasonably, which needs considerable manual efforts.

3 PROBLEM FORMULATION

The link prediction task predicts the potential connections among nodes/entities from the known information in a graph. Different from previous works which treat each triplet independently, we consider that triplets may have potential relations to each other if they have shared nodes. This information is helpful in many cases. For example, in a social network, the reason that Alice and Bob follow each other is probably because of their common habits. That means the triplet (*Alice, follows, Bob*) is valid due to (*Alice, likes, Pop*) and (*Bob, likes, Pop*), which can be represented as the logical expression $likes(Alice, Pop) \wedge likes(Bob, Pop) \rightarrow follows(Alice, Bob)$. Based on this, we can take advantage of the neighbor information to help link prediction. To realize this idea, we model the link prediction task in three steps: 1) convert the graph structure into a logic expression; 2) use neural modules to encode triplets as predicate embeddings; 3) apply logical constrained modules to generate ranking scores. The details for step 2) and 3) will be given in Section 4. In this section, we focus on how to convert an graph structure into a logic expression and how to formulate the link prediction task as a true/false evaluation problem of logical expressions.

Suppose we have a graph $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{T})$, where $\mathcal{V}$ is the vertex set, $\mathcal{R}$ is the relation set, and the known triplets (edges) in the graph are represented as $\mathcal{T}$. For any $v_i, v_j \in \mathcal{V}$ and a relation $r_k \in \mathcal{R}$, we need to predict if the target triplet $T_x = (v_i, r_k, v_j)$ is valid, where $T_x \notin \mathcal{T}$. To solve this problem, we first get the neighbors of both v_i and v_j and get all the triplets $\mathcal{T}_{ij}$ that contain either v_i or v_j.

$$\begin{aligned} \mathcal{T}_{ij} &= \{(v_i, r_{in}, v_n) | v_n \in \mathcal{N}_i\} \cup \{(v_j, r_{jm}, v_m) | v_m \in \mathcal{N}_j\} \\ &= \{r_{in}(v_i, v_n) | v_n \in \mathcal{N}_i\} \cup \{r_{jm}(v_j, v_m) | v_m \in \mathcal{N}_j\} \end{aligned} \tag{1}$$

where $\mathcal{N}_i$ and $\mathcal{N}_j$ are the neighbor vertex sets of node v_i and v_j, respectively, and the link is considered as a predicate. Since it is possible that not all the triplets in $\mathcal{T}_{ij}$ are the reasons of the target triplet T_x, we apply the OR operator to model the prediction task. The intuition here is that: the reason that T_x holds could be any of its neighbour links or any combination of its neighbour links. We translate this idea into the following expression:

$$\begin{aligned} &(T_1 \rightarrow T_x) \vee (T_2 \rightarrow T_x) \vee \cdots \vee (T_n \rightarrow T_x) \\ \vee\ &(T_1 \wedge T_2 \rightarrow T_x) \vee (T_1 \wedge T_3 \rightarrow T_x) \vee \cdots \vee (T_{n-1} \wedge T_n \rightarrow T_x) \\ \vee\ &(T_1 \wedge T_2 \wedge T_3 \rightarrow T_x) \vee \cdots \vee (T_{n-2} \wedge T_{n-1} \wedge T_n \rightarrow T_x) \\ &\cdots \\ \vee\ &(T_1 \wedge T_2 \wedge \cdots \wedge T_n \rightarrow T_x) \end{aligned} \tag{2}$$

where $T_1, T_2 \cdots T_n$ are triplets in $\mathcal{T}_{ij}$, and "$\rightarrow$" is called the implication operation[1]. This expression contains not only simple Horn clauses, such as $(T_1 \rightarrow T_x)$, but also higher-order Horn clauses, such as $(T_1 \wedge T_2 \rightarrow T_x)$ and $(T_1 \wedge T_2 \wedge \cdots \wedge T_n \rightarrow T_x)$. Based on this definition, we have the following theorem:

THEOREM 1. *Equation* (2) *is true if and only if* T_x *is true.*

To show why, we first have the following lemma:

LEMMA 2. *Let the premise* p *be true, then the clause* $p \rightarrow q$ *is true if and only if the conclusion* q *is true.*

The lemma naturally follows from the definition of the implication operation: $p \rightarrow q \Leftrightarrow \neg p \vee q$. Now back to Theorem 1, since all of the known triplets in the training data are valid, we know that each $T_* \in \mathcal{T}_{ij}$ is true, and thus any conjunction among T_* is also true. As a result, if T_x is true, then Eq.(2) must be true, and if Eq.(2) is true, we know that at least one of the Horn clauses in Eq.(2) must be true, and thus T_x must be true, meaning that T_x is a valid triplet. Now the problem of judging if a target triplet T_x is valid or not becomes answering the question that whether the logic expression in Eq.(2) is true given the known triplets. The intuition here is that T_x is true as long as at least one of its known neighbour connections or their conjunctions can imply T_x.

However, one problem is that the size of the expression is huge, which is equal to $O(2^n)$—the size of the power set of $\mathcal{T}_{ij}$, making it impractical to implement Eq.(2). Fortunately, we can simplify the expression in Eq.(2) through implication rule and De Morgan's Law[2], which translates Eq.(2) into following simplified form:

$$\neg T_1 \vee \neg T_2 \vee \cdots \vee \neg T_n \vee T_x \tag{3}$$

Compare to the $O(2^n)$ complexity of Expression (2), the complexity of Expression (3) is only $O(n)$. We will use Expression (3) for our model implementation. In the next section, we will introduce how to encode triplets into embeddings and then build logic neural networks to generate ranking scores for relational reasoning.

4 GRAPH COLLABORATIVE REASONING

Our GCR framework views a graph from the edge perspective and aims to learn the relationship between adjacent edges that are connected by common nodes. Instead, traditional GNN views a graph from the node perspective and aims to learn the relationship between nodes that are connected by common edges. In Figure 1, we use an example to show how a link prediction task on a heterogeneous graph can be viewed from logical perspective. In this example, we hope to predict if node v_1 and v_2 could be

[1]In classical logic, $p \rightarrow q$ is equivalent to $\neg p \vee q$

[2]De Morgan's Law, in formal language, is written as $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$ and $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$

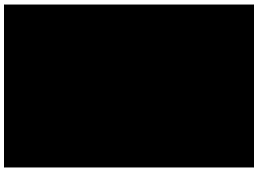


Figure 1: An example of link prediction on a heterogeneous graph. From a logical view, $r_x(v_1, v_2)$ to be true could result from any order of combinations of the neighbor links, e.g. first-order $r_1(v_1, v_4)$, second-order $r_1(v_1, v_4) \wedge r_2(v_2, v_6)$ or even higher-order $r_1(v_1, v_4) \wedge r_2(v_1, v_3) \wedge \ldots \wedge r_3(v_2, v_5)$.

connected by relation r_x. Intuitively, $r_x(v_1, v_2)$ could be true due to: 1) any first-order implication, e.g. $r_1(v_1, v_4) \rightarrow r_x(v_1, v_2)$ or $r_3(v_2, v_5) \rightarrow r_x(v_1, v_2)$ is true, or 2) any second-order implication, e.g. $r_1(v_1, v_4) \wedge r_2(v_2, v_6) \rightarrow r_x(v_1, v_2)$ is true, or even higher-order implication, e.g. $r_1(v_1, v_4) \wedge r_2(v_1, v_3) \wedge \ldots \wedge r_3(v_2, v_5) \rightarrow r_x(v_1, v_2)$ is true. With Eq.(3), this problem can be simplified as predicting if the following expression consisting of all neighbour links is true:

$$\neg r_1(v_1, v_4) \vee \neg r_2(v_1, v_3) \vee \neg r_2(v_2, v_6) \vee \neg r_3(v_2, v_5) \vee \neg r_4(v_2, v_7) \vee r_x(v_1, v_2) \tag{4}$$

In the following subsections, we will show the details of our graph collaborative reasoning framework.

4.1 Node and Link Encoding

We treat each type of relation in the graph as a predicate, e.g., each of the previously mentioned relations such as *capitalOf*, *locatedIn*, *follows*, *likes* is a predicate. We learn each node as a vector embedding, same as traditional graph neural networks. Meanwhile, we learn each predicate (relation type) as a small neural module. The predicate serves as a function that converts the two connected nodes into a latent vector in the reasoning space, e.g., to process the link $(Alice, likes, Pop)$, we write it as the predicate form $likes(Alice, Pop)$, then the node embeddings of $Alice$ and Pop are fed into the neural module of $likes$ to get the output representation for this link. More specifically, the encoding process is given as:

$$\mathbf{e}^r_{h,t} = P_r(\mathbf{e}_h, \mathbf{e}_t) = \mathbf{W}^r_2 \phi(\mathbf{W}^r_1(\mathbf{e}_h; \mathbf{e}_t) + \mathbf{b}^r_1) + \mathbf{b}^r_2 \tag{5}$$

where $P_r(\cdot, \cdot)$ is the predicate function for relation $r \in \mathcal{R}$; $\mathbf{e}_h, \mathbf{e}_t \in \mathbb{R}^d$ are embeddings for head and tail entities; $(\cdot;\cdot)$ is concatenation operation; $\phi(\cdot)$ is ReLU activation function; $\mathbf{W}^r_1, \mathbf{W}^r_2 \in \mathbb{R}^{n \times 2d}$ and $\mathbf{b}^r_1, \mathbf{b}^r_2 \in \mathbb{R}^n$ are network parameters and bias terms. Here $\mathbf{e}^r_{h,t}$ is the predicate embedding of the triplet (v_h, r, v_t). One thing we need to clarify here is that the order of the head and tail entity embeddings must be correctly sorted during the implementation, because we use concatenation operation to combine the head and tail embeddings, different ordering of head and tail concatenation will result in different outputs. However, this can be a problem for undirected graphs where the triplet (h, r, t) should have the same vector representation as (t, r, h). In our implementation, we solve this problem by assigning a unique ID to each vertex in the graph and sort their ID in ascending order. This will make sure that the triplet always comes with the smaller ID entity as the head entity while the bigger ID entity as the tail entity. For directed graphs, we will not conduct the sorting operation since the ordering is part of the graph information.

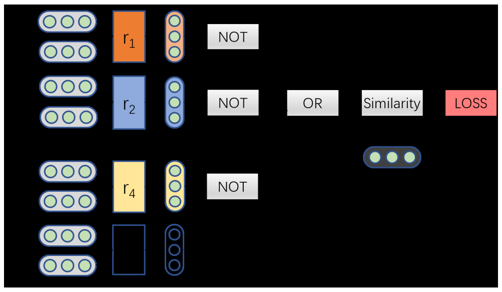


Figure 2: The logical network structure of the link prediction task given in Figure 1. The network is assembled using the logical equivalent expression which is converted via De Morgan's Law.

4.2 Logical Reasoning Modules

After obtaining all the encoded triplet vectors, we can rewrite the Expression (3) in the predicate embedding form:

$$(\neg \mathbf{e}^{r_{in_1}}_{i,n_1} \vee \neg \mathbf{e}^{r_{in_2}}_{i,n_2} \vee \ldots \vee \neg \mathbf{e}^{r_{jm_1}}_{j,m_1} \vee \neg \mathbf{e}^{r_{jm_2}}_{j,m_2}) \vee \mathbf{e}^{r_x}_{i,j} \tag{6}$$

Here $\mathbf{e}^{r_x}_{i,j}$ represents the predicate embedding for the target triplet $T_x = (v_i, r_x, v_j)$. Since the target triplet is unknown and need to be predicted, we use r_x instead of $r_{i,j}$ to make the notation concise. $\mathbf{e}^{r_{in_k}}_{i,n_k}$ and $\mathbf{e}^{r_{jm_k}}_{j,m_k}$ are the encoded predicate embeddings for the known neighbour triplets in the graph that contain either v_i or v_j. Our goal is to predict if the above logical expression is true in a continuous reasoning space. We define a constant vector $\mathbf{T}$, which is an anchor vector in the reasoning space that represents true. It is randomly initialized and kept unchanged during model training. We expect that the final vector representation of the entire expression is close to this true vector $\mathbf{T}$ if the target triplet T_x is valid. Otherwise, the vector representation of the logical expression should be far from $\mathbf{T}$.

To achieve this goal, we create neural modules $\text{OR}(\cdot, \cdot)$ and $\text{NOT}(\cdot)$ to represent the logical operations $\vee$ and $\neg$, where each module is an MLP with ReLU as activation function. To allow the neural logical modules to perform logical operations as expected, we add logical regularizers to the neural modules to constrain their behavior as defined in [3, 31]. The regularizers are not only added to the input predicate embeddings but also to the intermediate hidden vectors as well as the output vector to guarantee that all the embeddings are in the same representation and reasoning space. The logic constraint is represented as $\mathcal{L}_{logic}$.

With these logical modules, we can then assemble a neural network for Expression (6). To make the explanation easy to follow, we use a specific example as shown in Figure 2 to explain the network construction process. This reasoning network structure is corresponding to the heterogeneous graph given in the Figure 1. Suppose we are given two vertices v_1 and v_2, our goal is to predict if they could have a valid connection through relation r_x. According to the steps mentioned before, we need to first find the neighbors of both v_1 and v_2, in this example are $\{v_3, v_4, v_5, v_6, v_7\}$. Then we feed these vertex pairs into the corresponding predicate encoders to get the predicate embeddings based on Eq.(5). By sending these predicate embeddings into the $\text{NOT}(\cdot)$ module, we can calculate the negated embeddings, e.g. $\neg\mathbf{e}^{r_1}_{1,4}$. After that, we follow the structure of Eq.(6) to send the target predicate embedding $\mathbf{e}^{r_x}_{1,2}$ together with the negated embeddings into the OR$(\cdot,\cdot)$ module to get the final vector representation of the entire expression in the reasoning space. Since OR$(\cdot,\cdot)$ only takes two inputs at one time, we calculate the joint embedding for more than two predicate embeddings in a recurrent manner. That is, we first send two predicates, e.g. $\neg\mathbf{e}^{r_1}_{1,4}$ and $\neg\mathbf{e}^{r_2}_{1,3}$ in Figure 2, into the OR module and get the hidden vector $\mathbf{e}^{r_1,r_2}$, which represents the result of $\neg\mathbf{e}^{r_1}_{1,4} \vee \neg\mathbf{e}^{r_2}_{1,3}$. The next predicate embedding in the expression and the previous hidden vector $\mathbf{e}^{r_1,r_2}$ will be sent into the same OR neural module. This process is recurrently conducted until we get the final vector representation of the entire logical expression. However, we need to guarantee that the order information will not affect the final output since the logical OR operation need to satisfy the associativity and commutativity laws. This is done by randomly shuffling the order of the expression terms in each iteration. The following equations describe the process shown in Figure 2:

$$\begin{aligned}\neg\mathbf{e}^{r_k}_{i,j} &= \text{NOT}(\mathbf{e}^{r_k}_{i,j}), \forall i,j\\ \mathbf{E} &= \text{OR}\left(\neg\mathbf{e}^{r_1}_{1,4}, \neg\mathbf{e}^{r_2}_{1,3}, \cdots, \neg\mathbf{e}^{r_4}_{2,7}, \mathbf{e}^{r_x}_{1,2}\right)\end{aligned} \tag{7}$$

For expressions that have more predicate embeddings in the expression, we can simply add more recurrent steps and do the same operation as mentioned above. The final output $\mathbf{E}$ is the vector representation of the whole expression in the form of Eq.(6). The next step is to evaluate the distance between $\mathbf{E}$ and the constant true vector $\mathbf{T}$. As stated before, this true vector is randomly initialized and will not be updated during the learning process, as a result, it can be treated as an anchor vector in the reasoning space. Here we apply cosine similarity as the measure:

$$CosineSim(\mathbf{E},\mathbf{T}) = \frac{\mathbf{E}\cdot\mathbf{T}}{||\mathbf{E}||||\mathbf{T}||} \tag{8}$$

This cosine similarity measure is the score function and the output is treated as the ranking score to generate the entity ranking list.

4.3 Learning Algorithm

We use pair-wise learning algorithm [26] to train our model. Specifically, during the training process, for each known triplet in the training set, we fix the head entity and their corresponding relation and sample another entity as the tail. We treat expression created by this fake triplet T'_x as the negative sample. The same operation can be done one more time by holding the tail entity unchanged and replace the head entity. One thing need to mention here is that the neighbors to be sampled for creating the logic expression are never changed even when the head or tail entity is replaced, i.e., the only change in Eq.(3) is to replace T_x with T'_x. The expression for the valid triplet, known as the positive sample, is evaluated based on Eq.(8) and we have the score s^+_T, while the score for negative sample is $s^-_{T'}$. The loss function is written as:

$$\mathcal{L}_{gcr} = -\sum_{T\in\mathcal{T},T'\notin\mathcal{T}} \ln\sigma(\alpha(s^+_T - s^-_{T'})) \tag{9}$$

where $\sigma(\cdot)$ is the logistic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$; α is an amplification coefficient, which is set to 10 in our implementation. We can apply an optimization algorithm to minimize $\mathcal{L}_{gcr}$ so as to maximize the distance between positive and negative samples. By integrating the logical regularizers into the graph collaborative reasoning network loss, we get the final loss function:

$$\mathcal{L} = \mathcal{L}_{gcr} + \lambda_l\mathcal{L}_{logic} + \lambda_\Theta||\Theta||^2_2 \tag{10}$$

where λ_l is the coefficient of the logical regularizers; Θ represents all the trainable parameters of the model, including entity embeddings, predicate encoder parameters and the parameters of the neural logical modules; λ_Θ is the ℓ_2-norm regularization weight; We use back propagation [29] to optimize the model parameters. The pseudocode for the entire training algorithm, including neighbor sampling, is given in Appendix A.

Table 1: Statistics of the recommendation datasets.

Dataset	#Users	#Items	#Interaction	Density
Beauty	22,363	12,101	198,502	0.073%
Clothing	39,387	23,033	278,677	0.031%

5 EXPERIMENTS

In this section, we evaluate our proposed model on two types of link prediction tasks—graph link prediction and recommendation. The reason why we choose these two tasks for evaluation are based on two considerations: the uncertainty of the target links and the type of the graph structure.

Knowledge graph is a type of heterogeneous graph that contains multi-type relations among entities, which makes the link prediction task challenging. It requires the model to predict not only if two entities will be connected but also determine which type of relation connects them. The information in knowledge graphs is usually based on objective facts. That means each link can only be grounded as either true or false—not anything in between—since the links represent facts. Recommendation task usually considers a bipartite graph, which takes user and item as two types of nodes. The model needs to predict if a user and an item can be potentially connected so that we can recommend an item to a target user. The challenge is that the data is human generated which contains uncertainty and noise, so that it is usually not suitable to assign a deterministic truth value for a specific pair of nodes.

As we mentioned before, our model can handle the uncertainty for relational reasoning over multi-relational graphs, we choose these two tasks to verify the effectiveness of our graph collaborative reasoning model by answering the following research questions:

- **RQ1**: What is the performance of GCR in terms of graph link prediction and recommendation tasks? Does it outperform state-of-the-art models? (Section 5.4)
- **RQ2**: If and how does the logic regularizer help to improve the performance? (Section 5.5)
- **RQ3**: What is the impact of logical reasoning on few-shot data? (Section 5.6)

5.1 Datasets

For graph link prediction task, we use a well-known dataset **FB15k-237** [33], which is a subset of FB15k by removing the inverse relations in the training set to avoid data leakage. It contains 14,541 entities and 237 relations. The training dataset contains 272,115

Table 2: Baseline models used for either graph link prediction task or recommendation task.

Baseline	TransE	DistMult	ConvE	R-GCN	pLogicNet	pGAT	BPR-MF	NCR	NGCF
KG Completion	✓	✓	✓	✓	✓	✓	✗	✗	✗
Recommendation	✓	✓	✓	✗	✗	✗	✓	✓	✓

edges while the validation and testing sets contain 17,535 and 20,466 edges, respectively. In the experiment, we use the same training, validation and testing data splits as described in [33].

For recommendation task, we use a publicly available Amazon e-commerce dataset [18], which includes the user, item and rating information. The user-item interaction matrix can be viewed as a bipartite graph with two types of nodes, i.e. user and item, and a single relation, which is the purchase relation in e-commerce scenario. This is a sparse dataset which makes personalized recommendation challenging. We take **Beauty** and **Clothing** sub-categories for our experiments to explore both the link prediction performance and how our model performs in few-shot scenarios. Statistics of the datasets are shown in Table 1.

5.2 Baselines

We select several representative models for graph link prediction and recommendation to evaluate the performance of our proposed method. For graph link prediction, we use translation-based, tensor factorization-based, neural network-based as well as logic-based baselines for performance comparison.

- **TransE** [2]: A classical translation-based knowledge graph embedding algorithm. The scoring function for each triplet is given as $||\mathbf{h}+\mathbf{r}-\mathbf{t}||_p$, where $\mathbf{h}, \mathbf{r}, \mathbf{t}$ are entity and relation embeddings and $||\cdot||_p$ is the p-norm of the output vector.
- **DistMult** [42]: This is a tensor factorization-based knowledge graph embedding algorithm, which is a bilinear diagonal model.
- **ConvE** [5]: This approach uses 2D-convolutional operation over embeddings to capture the information from the triplets, which is one of the state-of-the-art models on graph link prediction.
- **R-GCN** [30]: This is a graph neural network based method, which extends Graph Convolutional Network (GCN) [15] to handle multi-relational link prediction tasks.
- **pLogicNet** [24]: The Probabilistic Logic Network, which is a logic-based relational reasoning model. It defines the joint distribution of all possible triplets trough Markov Logic Network (MLN) with logic rules, so that the optimization process can be efficient.
- **pGAT** [12]: This is a state-of-the-art MLN-based relational reasoning model, which combines MLN with graph attention network for link prediction.

For recommendation task, we also use the **TransE**, **DistMult** and **ConvE** knowledge graph embedding models as baselines since these models can also handle recommendation tasks. Other than that, we also use three recommendation models to explore if the GCR relational reasoning model can outperform those models that are specifically designed for recommendation, including:

- **BPR-MF** [26]: This is a pair-wise ranking model for recommendation. We implement the prediction function under the BPR framework by following [16], which considers user, item and global bias terms for matrix factorization.
- **NCR** [3]: This is a state-of-the-art reasoning-based recommendation framework. It utilizes neural logic reasoning to model recommendation tasks.
- **NGCF** [40]: This is an extension of GCN for recommendation task. It allows for multi-hop user-item information aggregation via message passing to enhance the user and item embeddings for recommendation.

We use Table 2 to show which baseline model can be used for which link prediction task. For reproducibility, we present the details of the experimental setup for training and evaluating our model and baselines in Appendix B.

5.3 Evaluation Protocol

5.3.1 ***Link Prediction.*** In the evaluation step, for each triplet, we first hold the head entity and replace the tail entity with ones that the head entity is not connected to. Then we do the same operation to hold the tail entity and replace the head entity. We call these generated non-existent triplets as negative samples. For each triplet and its corresponding negative samples, we calculate their evaluation metrics. The final results are averaged over all the triplets. We follow existing works [2, 42] and use the filtered setting for evaluation. We report Mean Reciprocal Rank (MRR) and top-K Hit rate (Hit@K) evaluation metrics in our results.

5.3.2 ***Recommendation.*** In recommendation task, for each user-item interaction, we only sample items for each user that the user has never interacted with. Then these negative samples together with the target triplets constitute a user ranking list. Then we calculate the corresponding ranking score for each user and report the final scores by averaging over all the users. Here we use Normalized Discounted Cumulative Gain (NDCG@K) and Hit rate (Hit@K) metrics in our recommendation evaluation.

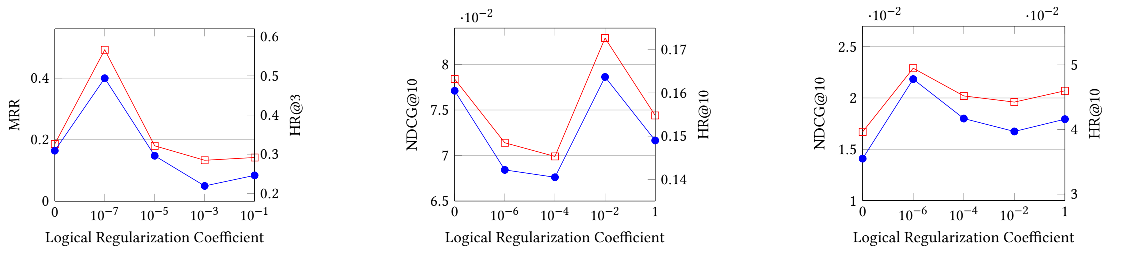
5.4 Overall Performance of GCR (RQ1)

We report the overall performance for graph link prediction and recommendation tasks in Table 3.

For the graph link prediction task, from the results, we see that our GCR model significantly outperforms all the baselines on MRR and Hit@1. The good performance on MRR and Hit@1 indicates that our model can generate high-quality predictions by ranking the correct target at top positions. Although Hit@3 is not better than pGAT, the performance is still competitive. According to the results, we observe that logic-based methods can consistently outperform the other non-logical models. This indicates the effectiveness of applying logic to graph link prediction tasks.

Table 3: Link prediction performance on three datasets with metrics NDCG (N) and Hit Ratio (HR). We use underline (number) to show the best result among the baselines, and use bold font to mark the best result of the whole column. We use star (*) to indicate that the performance is significantly better than all baselines. The significance is at 0.05 level based on paired *t*-test. The last row shows the relative improvement of our model against the best baseline performance.

	FB15k-237			Beauty				Clothing			
	MRR	Hit@1	Hit@3	NDCG@5	NDCG@10	Hit@5	Hit@10	NDCG@5	NDCG@10	Hit@5	Hit@10
TransE	0.326	0.229	0.363	0.0063	0.0086	0.0096	0.0165	0.0025	0.0035	0.0040	0.0069
DistMult	0.241	0.155	0.263	0.0105	0.0139	0.0171	0.0278	0.0036	0.0046	0.0055	0.0086
ConvE	0.325	0.237	0.356	0.0064	0.0084	0.0099	0.0162	0.0030	0.0042	0.0047	0.0083
R-GCN	0.248	0.153	0.258	–	–	–	–	–	–	–	–
pLogicNet	0.332	0.237	0.367	–	–	–	–	–	–	–	–
pGAT	0.457	0.377	**0.494**	–	–	–	–	–	–	–	–
BPRMF	–	–	–	0.0274	0.0348	0.0428	0.0658	0.0086	0.0109	0.0129	0.0200
NCR	–	–	–	0.0369	0.0453	0.0664	0.0767	0.0109	0.0132	0.0143	0.0246
NGCF	–	–	–	0.0453	0.0576	0.0715	0.1057	0.0133	0.0173	0.0219	0.0331
GCR	**0.492***	**0.490***	0.493	**0.0606***	**0.0829***	**0.0940***	**0.1637***	**0.0159***	**0.0229***	**0.0262***	**0.0478***
Improvment	7.66%	29.97%	–	33.77%	43.92%	31.47%	54.87%	19.55%	32.37%	19.63%	44.41%



(a) FB15k-237 (b) Beauty (c) Clothing

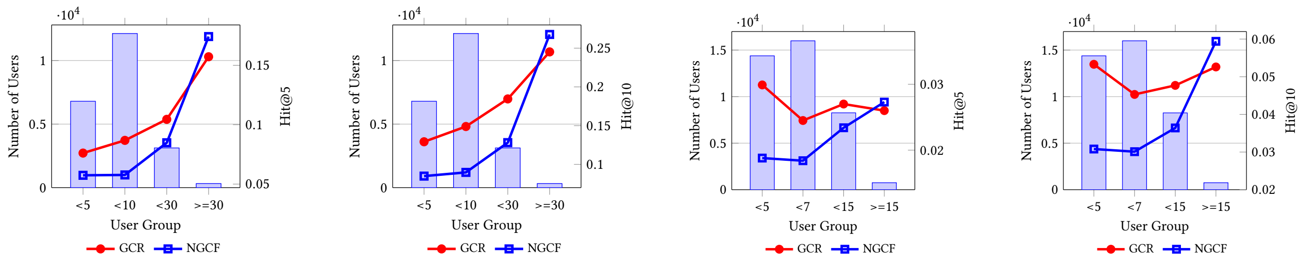
Figure 3: MRR/NDCG@10 (red squared line) and HR@3/HR@10 (blue circled line) on three datasets according to the increment of the logical regularization coefficient λ_r.

For the recommendation task, our model consistently outperforms all the baselines on all the evaluation metrics. From the reported results, we have the following observations:

- Knowledge graph embedding models have relatively worse performance than those recommendation models on the recommendation task. One reason is that the KG embedding models treat each triplet independently while recommendation needs to consider users and items from a collaborative learning perspective. This could limit the KG models to gain a good performance on recommendation tasks. Another reason is that the recommendation data presents more uncertainty than KG data since the recommendation data is recorded from user behaviors while the KG data is mostly fact-based, which is a challenge for the KG embedding methods.
- Among the recommendation baseline models, NGCF outperforms all other baseline methods. This indicates that it is beneficial to incorporate neighborhood information over graphs to make recommendation predictions.
- GCR outperforms NCR. This is because NCR only takes user historical interactions to generate logic expressions. However, GCR not only considers the items that the user interacted with, but also considers which other users interacted with these items. By leveraging the rich information from both user- and item-side, GCR can have a better recommendation quality than NCR.
- GCR consistently outperforms all the baselines. In particular, GCR improves over the strongest baseline NGCF on both datasets by at least 19.55% on NDCG@5. For Hit@10, our model can achieve even 44.41% improvement on the Clothing dataset. We realize that our model can have higher improvements over baselines when the dataset is more sparse. The Beauty dataset has a density 0.073% while the Clothing dataset is 0.031%. This result is reasonable because NGCF needs to aggregate neighborhood information to enhance user and item embedding representations. A very sparse dataset means that the average interactions over each user is limited so that the model cannot aggregate enough neighbor information to promote the representation quality. However, our GCR, by modeling link prediction from logical reasoning perspective, can help to improve the recommendation performance on sparse dataset. We conducted paired t-test and the p-value < 0.05, which shows that our model has statistical significant improvements over the strongest baseline.

5.5 Impact of Logical Regularization (RQ2)

In this section, we answer the question that if the logical regularization helps the learning process. We conduct experiments by tuning the logical regularization coefficient λ_l in $[0, 10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}]$ for *FB15k-237* and $[0, 10^{-6}, 10^{-4}, 10^{-2}, 1]$ for *Beauty* and *Clothing*. We show how performance changes w.r.t MRR, Hit Rate and NDCG in Figure 3. We have two major observations from the results:



(a) Beauty Hit@5 **(b) Beauty Hit@10** **(c) Clothing Hit@5** **(d) Clothing Hit@10**

Figure 4: Performance comparision between GCR and NGCF on Beauty and Clothing datasets. The histograms represent the total number of users in each group, the lines indicate the performance trend with the growing number of per user interactions.

- The results show that logical regularization do help to improve the performance when comparing the results of non-logic model ($\lambda_l = 0$) and logic-regularized models ($\lambda_l \neq 0$). However, how strong the regularization should be added to the neural network need to be carefully adjusted, similar to the observations in [3].
- Sparser data needs a relatively smaller logical regularization coefficient. For the Beauty and Clothing datasets, which are bipartite graphs, their densities are 0.073% and 0.031%, respectively. For FB15k-237, which is a multi-relational graph, the density is $\frac{|\mathcal{T}|}{|\mathcal{V}|\times|\mathcal{V}-1|\times|\mathcal{R}|} \times 100\% \approx 0.0006\%$. This is because we not only need to decide if an entity pair will be connected but also need to decide the type of relation between them, which is different from the recommendation bipartite graphs. For the most sparse data FB15k-237, the best logic regularization weight is 10^{-7}, while the best weight for the most dense dataset among the three is 10^{-2}. The reason for the observation is that there is a trade-off between the prediction loss and the logical loss. The model needs to learn useful information from limited data to generate good predictions. For the sparse FB15k-237 dataset, the model is very sensitive to large logical regularization weights because the logical loss will dominate the total loss when training data is insufficient for the prediction loss. However, for Clothing dataset, which is about 50 times denser than FB15k-237, we see that the model is not that sensitive to large logical regularization weights. Even with a higher regularization weight, the model still achieves better performance than non-logic model that $\lambda_l = 0$.

5.6 Impact of Sparsity Levels (RQ3)

The sparsity issue brought by data incompleteness may limit the embedding quality of prediction models. When the data is insufficient, it is difficult for models to capture the relations between entity pairs, and thus influence the quality of the generated predictions. This issue would especially affect the link prediction models since they usually relies on collective information for model learning. In this section, we explore whether logical reasoning models can help to improve the prediction performance when the data is sparse. With this consideration, we conduct an experiment by evaluating the model performance over different data groups that have different sparsity. For better visualization of the results, we perform the experiments on the two bipartite graphs.

In particular, we split the users in the testing set into different groups based on their total number of interactions in the training data. Take the Beauty dataset as an example, users are divided into four groups, corresponding to the users whose number of interactions is in $[1, 5)$, $[5, 10)$, $[10, 30)$ and $[30, \infty)$, respectively. We compare our model with the strong baseline NGCF and report the results with respect to Hit@5 and Hit@10 in Figure 4. Since similar trend is also observed on the NDCG metric, we do not plot the NDCG results to keep the figure clarity.

From the experiments, we see that our GCR model has significantly better performance than NGCF on sparse user groups. When the user has more interactions, the performance of NGCF can be better than ours. This observation can be explained by the underlying modeling mechanism of NGCG and GCR. NGCF needs to take the neighborhood information to enrich the node embeddings. For the users with very few interactions, it would be challenging for NGCF to capture the user similarities. Although the GCR model also relies on the neighborhood information, it benefits from two special advantages. First, the model can leverage both neighbour node and neighbour link information, and second, the logic component helps to model the logical relationship among the limited neighbourhood entities rather than merely relying on the associative node similarity information for prediction. The good performance on sparse user groups show that our logical reasoning-based model helps to improve the recommendation quality on sparse data. This is an important advantage of our model, since users with fewer interactions are the majority, as shown in Figure 4.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose to model link prediction as a reasoning problem over graphs. Specifically, we propose a Graph Collaborative Reasoning (GCR) approach, which takes the neighborhood link information to predict the connections in a latent reasoning space. Experiments on two representative link prediction tasks—graph link prediction and recommendation—show the effectiveness of the model, especially for link prediction on sparse data.

We believe enabling the ability of reasoning over graphs is important for future cognitive intelligent systems. This work is just one of our first steps towards this goal, and there is still much room for future improvements. In this paper, we only used the one-hop neighborhood links, while in the future we will extend to multi-hop reasoning over graphs based on the GCR framework to model hierarchical data structure. Besides the knowledge graph and recommendation tasks considered in this work, graph collaborative reasoning may also help other intelligent tasks such as question answering, molecular graph modeling, entity search and conversational systems, which we will explore in the future.

REFERENCES

[1] Siddhant Arora. 2020. A Survey on Graph Neural Networks for Knowledge Graph Completion. *arXiv preprint arXiv:2007.12374* (2020).

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*. 2787–2795.

[3] Hanxiong Chen, Shaoyun Shi, Yunqi Li, and Yongfeng Zhang. 2021. Neural Collaborative Reasoning. In *Proceedings of the 30th Web Conference (WWW)*.

[4] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. 2016. Lifted Rule Injection for Relation Embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 1389–1399.

[5] T Dettmers, P Minervini, P Stenetorp, and S Riedel. 2018. Convolutional 2D knowledge graph embeddings. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, Vol. 32. AAI Publications, 1811–1818.

[6] Boyang Ding, Quan Wang, Bin Wang, and Li Guo. 2018. Improving Knowledge Graph Embedding Using Simple Constraints. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 110–121.

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

[8] Shu Guo, Lin Li, Zhen Hui, Lingshuai Meng, Bingnan Ma, Wei Liu, Lihong Wang, Haibin Zhai, and Hong Zhang. 2020. Knowledge Graph Embedding Preserving Soft Logical Regularity. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 425–434.

[9] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2016. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 192–202.

[10] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2018. Knowledge graph embedding with iterative guidance from soft rules. *AAAI* (2018).

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[12] L Vivek Harsha Vardhan, Guo Jia, and Stanley Kok. 2020. Probabilistic Logic Graph Attention Networks for Reasoning. In *Companion Proceedings of the Web Conference 2020*. 669–673.

[13] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 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.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR '17)*.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[17] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[18] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. ACM.

[19] Pasquale Minervini, Luca Costabello, Emir Muñoz, Vít Nováček, and Pierre-Yves Vandenbussche. 2017. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 668–683.

[20] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. 2019. Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4710–4723.

[21] Tu Dinh Nguyen, Dat Quoc Nguyen, Dinh Phung, et al. 2018. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 327–333.

[22] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[23] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Icml*.

[24] Meng Qu and Jian Tang. 2019. Probabilistic logic neural networks for reasoning. *Advances in neural information processing systems* 32 (2019), 7712–7722.

[25] Hongyu Ren and Jure Leskovec. 2020. Beta Embeddings for Multi-Hop Logical Reasoning in Knowledge Graphs. *arXiv preprint arXiv:2010.11465* (2020).

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

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

[28] Andrea Rossi, Donatella Firmani, Antonio Matinata, Paolo Merialdo, and Denilson Barbosa. 2020. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. *arXiv preprint arXiv:2002.00819* (2020).

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[30] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.

[31] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1365–1374.

[32] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197* (2019).

[33] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*. 1499–1509.

[34] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML).

[35] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2018. Graph Convolutional Matrix Completion. (2018).

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[37] Mengya Wang, Erhu Rong, Hankui Zhuo, and Huiling Zhu. 2018. Embedding knowledge graphs based on transitivity and asymmetry of rules. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 141–153.

[38] Quan Wang, Bin Wang, and Li Guo. 2015. Knowledge base completion using embeddings and rules. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[39] Shen Wang, Xiaokai Wei, Cicero dos Santos, Zhiguo Wang, Ramesh Nallapati, Andrew Arnold, Bing Xiang, and S Yu Philip. 2020. H2KGAT: Hierarchical Hyperbolic Knowledge Graph Attention Network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 4952–4962.

[40] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[41] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes.. In *AAAI*, Vol. 14. Citeseer, 1112–1119.

[42] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

[43] Shihui Yang, Jidong Tian, Honglun Zhang, Junchi Yan, Hao He, and Yaohui Jin. 2019. TransMS: Knowledge Graph Embedding for Complex Relations by Multidirectional Semantics.. In *IJCAI*. 1935–1942.

[44] Wen Zhang, Bibek Paudel, Liang Wang, Jiaoyan Chen, Hai Zhu, Wei Zhang, Abraham Bernstein, and Huajun Chen. 2019. Iteratively learning embeddings and rules for knowledge graph reasoning. In *The World Wide Web Conference*. 2366–2377.

[45] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. 2020. Efficient Probabilistic Logic Reasoning with Graph Neural Networks. In *ICLR*.

A TRAINING ALGORITHM PSEUDO-CODE

Algorithm 1: GCR Training Algorithm

```
Input :Graph $\mathcal{G}(\mathcal{V},\mathcal{R},\mathcal{T})$; triples
       $\mathcal{T}(h,r,t)\forall h,t \in \mathcal{V}, \forall r \in \mathcal{R}$; predicate function
       $P_r, \forall r \in \mathcal{R}$; epochs $K$; neighbor sample function $N$;
       negative sample function $S$; scoring function Sim;
       Graph Collaborative Reasoning network GCR;
       anchor vector $\mathbf{T}$; optimization algorithm OPTIM;
       model parameters $\Theta$; logic regularizer weight $\lambda_l$; $\ell_2$
       regularizer weight $\lambda_\Theta$; amplification coefficient $\alpha$
1  Initialize node vectors $\mathbf{x}_v, \forall v \in \mathcal{V}$;
2  Initialize predicate modules $P_r, \forall r \in \mathcal{R}$;
3  for epoch $k \leftarrow 1$ to $K$ do
4      $\mathcal{L} \leftarrow 0$;
5      $\mathbf{e}_T^{k-1} \leftarrow P_r(\mathbf{x}_h, \mathbf{x}_t), \quad \forall T \in \mathcal{T}, h,r,t \in T$;
6      $\mathbf{e}_T^{k} \leftarrow \mathbf{e}_T^{k-1}/||\mathbf{e}_T^{k-1}||_2$;
7      for $T \in \mathcal{T}$ do
8          $T' \leftarrow S(T)$          ▹ sample a fake triplet for $T$;
9          $\mathbf{E} \leftarrow$ GCR($\mathbf{e}_T^k$, $\{\mathbf{e}_{T_N}^k, \forall T_N \in \mathcal{N}(T)\}$);
10         $\mathbf{E}' \leftarrow$ GCR($\mathbf{e}_{T'}^k$, $\{\mathbf{e}_{T_N}^k, \forall T_N \in \mathcal{N}(T)\}$);
11         $s_T^+ \leftarrow$ Sim($\mathbf{E}, \mathbf{T}$), $s_{T'}^- \leftarrow$ Sim($\mathbf{E}', \mathbf{T}$);
12         $\mathcal{L}_{gcr} \leftarrow -\ln \sigma(\alpha(s_T^+ - s_{T'}^-))$;
13         $\mathcal{L}_{logic} \leftarrow \sum_i r_i$  ▹ logic constraints for logical laws;
14         $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{gcr} + \lambda_l \mathcal{L}_{logic} + \lambda_\Theta ||\Theta||_2^2$;
15     end
16     OPTIM($\mathcal{L}$)    ▹ optimize all parameters for round $k$;
17 end
```

B EXPERIMENTAL SETTINGS

B.1 Link Prediction

In the training stage, we first need to find the neighbors of the head and tail entity of the given triplet. Instead of using all the neighbor nodes to assemble the logical expression, we sample the neighbors uniformly, by following [11], in each iteration to predict the target triplet. In our implementation, we sample at most $n \in \{5, 10, 20\}$ neighbors for each entity in the given triplet. In other words, for each target triplet, the total number of neighbor triplets can be up to $2n$ (n from the head entity and n from the tail entity). To train the model, for each target triplet, we sample 1 negative triplet for pair-wise learning as mentioned in Eq.(9).

We set all vector embedding size to 64. The number of layers for predicate encoder networks and logical module networks is set to 3. The network parameters are initialized with normal distribution with mean 0 and standard deviation is 0.01. Dropout and ℓ_2 regularization are adopted to avoid over-fitting. We set the dropout rate to 0.2 and the weight for ℓ_2 regularizer λ_Θ is selected from 10^{-5} to 10^{-7}. The logical regularizer weight λ_l is selected in the range 10^{-1} to 10^{-7}. We use Adam [14] as the optimization algorithm with learning rate initialized to 0.001 and learning rate decay is adopted during the training process. Early-stopping is used and the best model for reporting the results is selected based on the best performance on the validation set.

B.2 Recommendation

For each user-item interaction in training set, we randomly sample the neighbors for both user and item nodes to construct the logical expression. We set the total number of neighbors for each user or item to 5, i.e. there will be at most 10 neighbor user-item interactions in the logical expression. We set the embedding size to 64 and the number of layers for network modules is 2. ℓ_2 penalty weight λ_Θ is 10^{-5} for both datasets. The logical regularization weight λ_l is 10^{-6}. Learning rate is fixed at 0.001. Other settings are the same as the previous subsection.

For TransE, DistMult and ConvE, we set the embedding size to 100, while the embedding size and hidden size for BPR-MF and NCR are 64. ℓ_2 weight for all baselines are 10^{-5}. For ConvE, the number of channel is set to 32 and the kernel size is 3. For NCR, we use the open source implementation[3], more specifically, we apply the BPR-ranking loss to train the model and the neural logic modules have two layers with LeakyReLU as the activation function. Since NCR only considers nodes on user side, we only sample neighbor nodes on the user side. For NGCF, we also use the open source implementation in [40] to run the experiments.

C ACKNOWLEDGEMENT

This work was supported in part by NSF IIS-1910154, IIS-2007907, IIS-2046457 and CCF-2124155. Any opinions and findings in this material are those of the authors and do not necessarily reflect those of the sponsors.

[3]https://github.com/rutgerswiselab/NCR