# Knowledge Graph Reasoning with Relational Digraph

Yongqi Zhang
4Paradigm Inc.
Beijing, China
zhangyongqi@4paradigm.com

Quanming Yao
EE, Tsinghua University
Beijing, China
qyaoaa@tsinghua.edu.cn

## ABSTRACT

Reasoning on the knowledge graph (KG) aims to infer new facts from existing ones. Methods based on the relational path have shown strong, interpretable, and transferable reasoning ability. However, paths are naturally limited in capturing local evidence in graphs. In this paper, we introduce a novel relational structure, i.e., relational directed graph (r-digraph), which is composed of overlapped relational paths, to capture the KG's local evidence. Since the r-digraphs are more complex than paths, how to efficiently construct and effectively learn from them are challenging. Directly encoding the r-digraphs cannot scale well and capturing query-dependent information is hard in r-digraphs. We propose a variant of graph neural network, i.e., RED-GNN, to address the above challenges. Specifically, RED-GNN makes use of dynamic programming to recursively encodes multiple r-digraphs with shared edges, and utilizes query-dependent attention mechanism to select the strongly correlated edges. We demonstrate that RED-GNN is not only efficient but also can achieve significant performance gains in both inductive and transductive reasoning tasks over existing methods. Besides, the learned attention weights in RED-GNN can exhibit interpretable evidence for KG reasoning. [1]

## CCS CONCEPTS

• **Information systems** → *Web searching and information discovery*; • **Computing methodologies** → **Knowledge representation and reasoning**; **Machine learning algorithms**.

## KEYWORDS

Knowledge graph, Graph embedding, Knowledge graph reasoning, Graph representation learning, Graph neural network

**ACM Reference Format:**
Yongqi Zhang and Quanming Yao. 2022. Knowledge Graph Reasoning with Relational Digraph. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3485447.3512008

[1]The code is available at https://github.com/AutoML-Research/RED-GNN/. Correspondence is to Q. Yao.

## 1 INTRODUCTION

Knowledge graph (KG), which contains the interactions among real-world objects, peoples, concepts, etc., brings much connection between artificial intelligence and human knowledge [2, 16, 18]. The interactions are represented as facts with the triple form *(subject entity, relation, object entity)* to indicate the relation between entities. The real-world KGs are large and highly incomplete [18, 44], thus inferring new facts is challenging. KG reasoning simulates such a process to deduce new facts from existing facts [7]. It has wide application in semantic search [4], recommendation [6], and question answering [1], etc. In this paper, we focus on learning the relational structures for reasoning on the queries in the form of *(subject entity, relation, ?)*.

Over the last decade, triple-based models have gained much attention to learn semantic information in KGs [44]. These models directly reason on triples with the entity and relation embeddings, such as TransE [5], ConvE [10], ComplEx [40], RotatE [36], QuatE [51], AutoSF [54], etc. Since the triples are independently learned, they cannot explicitly capture the *local evidence* [26, 38, 46, 48], i.e., the local structures around the query triples, which can be used as evidence for KG reasoning [26].

Learning on paths can help to better capture local evidences in graphs since they can preserve sequential connections between nodes [13, 28]. Relational path is the first attempt to capture both the semantic and local evidence for reasoning [23]. DeepPath [46], MINERVA [9] and M-walk [35] use reinforcement learning (RL) to sample relational paths that have strong correlation with the queries. Due to the sparse property of KGs, the RL approaches are hard to train on large-scale KGs [7]. PathCon [43] samples all the relational paths between the entities and use attention mechanism to weight the different paths, but is expensive for the entity query tasks. The rule-based methods, such as RuleN [25], Neural LP [48], DRUM [33] and RNNLogic [29], generalize the relational paths as logical rules, which learn to infer relations by logical composition of relations, and can provide interpretable insights. Besides, the logical rules can handle inductive reasoning where there exist unseen entities in the inference, which are common in real-world applications [33, 48, 50].

Subgraphs can naturally be more informative than paths in capturing the local evidence [2, 11, 45]. Their effectiveness has been empirically verified in, e.g., graph-based recommendation [50, 56] and node representation learning [14]. With the success of graph neural network (GNN) [12, 20] in modeling graph-structured data, GNN has been introduced to capture the subgraph structures in KG. R-GCN [34], CompGCN [41] and KE-GCN [49] propose to update the representations of entities by aggregating all the neighbors in each layer. However, they cannot distinguish the structural role of different neighbors and cannot be interpretable. DPMPN [47] learns to reduce the size of subgraph for reasoning on large-scale

KGs by pruning the irrelevant entities for a given query rather than learning the specific local structures. p-GAT [15] jointly learns a Graph Attention Network and a Markov Logic Network to bridge the gap between embeddings and rules. However, a set of rules must be pre-defined and the expensive EM-algorithm should be used for optimization. Recently, GraIL [38] proposes to predict relation from the local enclosing subgraph and shows the inductive ability of subgraph. However, it suffers from both effectiveness and efficiency problems due to the limitation of the enclosing subgraph.

Inspired by the interpretable and transferable path-based methods and the structure preserving subgraph methods, we introduce a novel relational structure into KG, called r-digraph, to combine the best of both worlds. The r-digraphs generalize relational paths to subgraphs by preserving the overlapped relational paths and the structures of relations for reasoning. Different from the relational paths that have simple structures, how to efficiently construct and learn from the r-digraphs are challenging since the construction process is expensive [38, 43]. Inspired by saving computation costs in overlapping sub-problems using dynamic programming, we propose RED-GNN, a variant of GNN [12], to recursively encode multiple **RE**lational **D**igraphs (r-digraphs) with shared edges and select the important edges through query-dependent attention weights. Empirically, RED-GNN shows significant gains over the state-of-the-art reasoning methods in both inductive and transductive benchmarks. Besides, the training and inference processes are efficient, the number of model parameters are small, and the learned structures are interpretable.

## 2 RELATED WORKS

A knowledge graph is in the form of $\mathcal{K} = \{\mathcal{V}, \mathcal{R}, \mathcal{F}\}$, where $\mathcal{V}$, $\mathcal{R}$ and $\mathcal{F} = \{(e_s, r, e_o) | e_s, e_o \in \mathcal{V}, r \in \mathcal{R}\}$ are the sets of entities, relations and fact triples, respectively. Let $e_q$ be the query entity, $r_q$ be query relation, and $e_a$ be answer entity. Given the query $(e_q, r_q, ?)$, the reasoning task is to predict the missing answer entity $e_a$. Generally, all the entities in $\mathcal{V}$ are candidates for $e_a$ [7, 9, 48].

The key for KG reasoning is to capture the *local evidence*, i.e., entities and relations around the query triples $(e_q, r_q, e_a)$. We regard $(e_q, r_q, e_a)$ as missing link, thus the local evidence does not contain this triplet in. Examples of such local evidence explored in the literature are relational paths [9, 24, 43, 46] and subgraphs [34, 38, 41]. In this part, we introduce the path-based methods and GNN-based methods that leverage structures in $\mathcal{F}$ for reasoning.

### 2.1 Path-based methods

Relational path (Definition 1) is a set of triples that are sequentially connected. The path-based methods learn to predict the triple $(e_q, r_q, e_a)$ by a set of relational paths as local evidence. DeepPath [46] learns to generate the relational path from $e_q$ to $e_a$ by reinforcement learning (RL). To improve the efficiency, MINERVA [9] and M-walk [35] learn to generate multiple paths starting from $e_q$ by RL. The scores are indicated by the arrival frequency on different $e_a$'s. Due to the complex structure of KG, the reward is very sparse, making it hard to train the RL models [7]. PathCon [43] samples all the paths connecting the two entities to predict the relation between them, which is expensive for the reasoning task $(e_q, r_q, e_a)$.

Definition 1 (Relational path [24, 46, 53]). *The relational path with length $L$ is a set of $L$ triples $(e_0, r_1, e_1)$, $(e_1, r_2, e_2)$, ..., $(e_{L-1}, r_L, e_L)$, that are connected head-to-tail sequentially.*

Instead of directly using paths, the rule-based methods learn logical rules as a generalized form of relational paths. The logical rules are formed with the composition of a set of relations to infer a specific relation. This can provide better interpretation and can be transfered to unseen entities. The rules are learned by either the mining methods like RuleN [25], EM algorithm like RNNLogic [29], or end-to-end training, such as Neural LP [48] and DRUM [33], to generate highly correlated relational paths between $e_q$ and $e_a$. The rules can provide logical interpretation and transfer to unseen entities. However, the rules only capture the sequential evidences, thus cannot learn complex patterns such as the subgraph structures.

### 2.2 GNN-based methods

As mentioned in Section 1, the subgraphs can preserve richer information than the relational paths as more degree are allowed in the subgraph. GNN has shown strong power in modeling the graph structured data [2]. This inspires recent works, such as R-GCN [34], CompGCN [41], and KE-GCN [49], extending GNN on KG to aggregate the entities' and relations' representations under the message passing framework [12] as

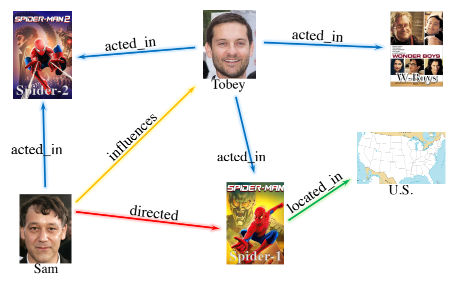
$$\boldsymbol{h}_{e_o}^{\ell} = \delta\Big(\boldsymbol{W}^{\ell} \cdot \sum\nolimits_{(e_s, r, e_o) \in \mathcal{F}} \phi\big(\boldsymbol{h}_{e_s}^{\ell-1}, \boldsymbol{h}_r^{\ell}\big)\Big), \tag{1}$$

which aggregates the message $\phi(\cdot,\cdot)$ on the 1-hop neighbor edges $(e_s, r, e_o) \in \mathcal{F}$ of entity $e_o$ with dimension $d$. $\boldsymbol{W}^{\ell} \in \mathbb{R}^{d \times d}$ is a weighting matrix, $\delta$ is the activation function and $\boldsymbol{h}_r^{\ell}$ is the relation representation in the $\ell$-th layer. After $L$ layers' aggregation, the representations $\boldsymbol{h}_e^L$ capturing the local structures of entities $e \in \mathcal{V}$ jointly work with a decoder scoring function to measure the triples. Since the message passing function (1) aggregates information of all the neighbors and is independent with the query, R-GCN and CompGCN cannot capture the explicit local evidence for specific queries and are not interpretable.

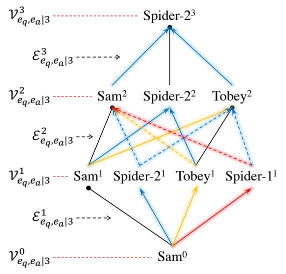
Instead of using all the neighborhoods, DPMPN [47] designs one GNN to aggregate the embeddings of entities, and another GNN to dynamically expand and prune the inference subgraph from the query entity $e_q$. A query-dependent attention is applied over the sampled entities for pruning. This approach shows interpretable reasoning process by the attention flow on the pruned subgraph, but still requires embeddings to guide the pruning, thus cannot be generalized to unseen entities. Besides, it cannot capture the explicit local evidence supporting a given query triple.

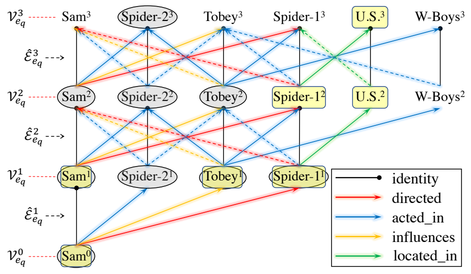
p-GAT [15] and pLogicNet [30] jointly learns an embedding-based model and a Markov logic network (MLN) by variational-EM algorithm. The embedding model generates evidence for MLN to update and MLN expends the training data for embedding model to train. MLN brings the gap between embedding models and logic rules, but it requires a pre-defined set of rules for initialization and the training is expensive with the variational-EM algorithm.

Recently, GraIL [38] proposes to extract the enclosing subgraph $\mathcal{G}_{(e_q, e_a)}$ between the query entity $e_q$ and answer entity $e_a$. To learn the enclosing subgraph, the relational GNN [34] with query-dependent attention is applied over the edges in $\mathcal{G}_{(e_q, e_a)}$ to control the importance of edges for different queries, but the learned



(a) Knowledge graph.


$\mathcal{V}^3_{e_q,e_a|3}$
Spider-2³
$\mathcal{E}^3_{e_q,e_a|3}$
$\mathcal{V}^2_{e_q,e_a|3}$
Sam²
Spider-2²
Tobey²
$\mathcal{E}^2_{e_q,e_a|3}$
$\mathcal{V}^1_{e_q,e_a|3}$
Sam¹
Spider-2¹
Tobey¹
Spider-1¹
$\mathcal{E}^1_{e_q,e_a|3}$
$\mathcal{V}^0_{e_q,e_a|3}$
Sam⁰
(b) $\mathcal{G}_{\text{Sam,Spider-2}|3}$.



(c) Recursive encoding.

**Figure 1: Graphical illustration. In (c), the subgraph formed by the gray ellipses is $\mathcal{G}_{Sam,\ Spider\text{-}2|3}$ and the subgraph formed by the yellow rectangles is $\mathcal{G}_{Sam,U.S.|3}$. Dashed edges mean the reverse relations of corresponding color (best viewed in color).**

attention weights are not interpretable (see Appendix B). After $L$ layers' aggregation, the graph-level representations aggregate all the entities $e \in \mathcal{V}$ in the subgraph and are used to score the triple $(e_q, r_q, e_a)$. Since the subgraphs need to be explicitly extracted and scored for different triples, the computation cost is very high.

## 3 RELATIONAL DIGRAPH (R-DIGRAPH)

The relational paths have shown strong transferable and interpretable reasoning ability on KGs [9, 33, 48]. However, they are limited in capturing more complex dependencies in KG since the nodes in paths are only sequentially connected. GNN-based methods can learn different subgraph structures. But none of the existing methods can efficiently learn the subgraph structures that are both interpretable and inductive like rules. Hence, we are motivated to define a new kind of structure, to explore the important local evidence.

Before defining r-digraph, we first introduce a special type of directed graph in Definition 2.

Definition 2 (Layered *st*-graph [3]). *The layered st-graph is a directed graph with exactly one source node (s) and one sink node (t). All the edges are directed, connecting nodes between consecutive layers and pointing from $l$-th layer to $l+1$-th layer.*

Here, we adopt the general approaches to augment the triples with reverse and identity relations [33, 41]. Then, all the relational paths with length less than or equal to $L$ between $e_q$ and $e_a$ can be represented as relational paths $e_q \rightarrow_{r^1} \cdot \rightarrow_{r^2} \cdots \rightarrow_{r^L} e_a$ with length $L$. In this way, they can be formed as paths in a layered *st*-graph, with the single source entity $e_q$ and sink entity $e_a$. Such a structure preserves all the relational paths between $e_q$ and $e_a$ up to length $L$, and maintains the subgraph structures. Based on this observation, we define r-digraph in Definition 3.

Definition 3 (r-digraph). *The r-digraph $\mathcal{G}_{e_q,e_a|L}$ is a layered st-graph with the source entity $e_q$ and the sink entity $e_a$. The entities in the same layer are different with each other. Any path pointing from $e_q$ to $e_a$ in the r-digraph is a relational path $e_q \rightarrow_{r^1} \cdot \rightarrow_{r^2} \cdots \rightarrow_{r^L} e_a$ with length $L$, where $r^\ell$ connects an entity in the $\ell{-}1$-layer to an entity in $\ell$-layer. We define $\mathcal{G}_{e_q,e_a|L} = \emptyset$ if there is no relational path connecting $e_q$ and $e_a$ with length $L$.*

Figure 1(b) provides an example of r-digraph $\mathcal{G}_{\text{Sam,Spider-2}|3}$, which is used to infer the new triple *(Sam, directed, Spider-2)*, in the exemplar KG in Figure 1(a). Inspired by the reasoning ability of relational paths [9, 33, 48], we aim to leverage the r-digraph for KG reasoning. However, different from the relational paths which have simple structures to learn with sequential models [9, 46], how to efficiently construct and how to effectively learn from the r-digraphs are challenging.

In the paper, $\mathcal{E}^\ell_{e_q,e_a|L}$ are the edges and $\mathcal{V}^\ell_{e_q,e_a|L} = \{e_o | (e_s, r, e_o) \in \mathcal{E}^\ell_{e_q,e_a|L}\}$ are the entities in the $\ell$-th layer of the r-digraph

$$\mathcal{G}_{e_q,e_a|L} = \mathcal{E}^1_{e_q,e_a|L} \otimes \cdots \otimes \mathcal{E}^L_{e_q,e_a|L},$$

with $\otimes$ denoting the layer-wise connection. We define the union operator as $\mathcal{G}_{e_{q_1},e_{a_1}|L} \cup \mathcal{G}_{e_{q_2},e_{a_2}|L} = (\mathcal{E}^1_{e_{q_1},e_{a_1}|L} \cup \mathcal{E}^1_{e_{q_2},e_{a_2}|L}) \otimes \cdots \otimes (\mathcal{E}^L_{e_{q_1},e_{a_1}|L} \cup \mathcal{E}^L_{e_{q_2},e_{a_2}|L})$. Given an entity $e$, we denote $\hat{\mathcal{E}}^\ell_e$, $\check{\mathcal{E}}^\ell_e$ and $\mathcal{V}^\ell_e$ as the sets of out-edges, in-edges and entities, respectively, visible in $\ell$ steps walking from $e$. $\mathcal{E}^\ell_{e_q,e_a|L}$, $\mathcal{V}^\ell_{e_q,e_a|L}$ and $\hat{\mathcal{E}}^\ell_{e_q}$, $\mathcal{V}^\ell_{e_q}$ are graphically shown in Figure 1.

## 4 THE PROPOSED MODEL

Here, we show how GNNs can be tailor-made to efficiently and effectively learn from the r-digraphs. Extracting subgraph structures and then learning the subgraph representation is a common practice for subgraph encoding in the literature, such as GraphSage [14] and GraIL [38]. Given a query triple $(e_q, r_q, e_a)$, the subgraph encoding generally contains three processes:

(i). extract the neighborhoods of both $e_q$ and $e_a$;
(ii). take the intersection to construct the subgraph;
(iii). run message passing and use the graph-level representation as the subgraph encoding.

When working on the r-digraph $\mathcal{G}_{e_q,e_a|L}$, the same approach can be customized in Algorithm 1. First, we get the neighborhoods of both $e_q$ and $e_a$ in steps 2-5. Second, we take the intersection of neighborhoods from $e_q$ and $e_a$ in steps 6-8 to induce the r-digraph $\mathcal{G}_{e_q,e_a|L}$ layer-wisely. Third, if the r-digraph is empty, we set the representation as $\mathbf{0}$ in step 9. Otherwise, the message passing is conducted layer-by-layer in steps 10-12. Since $e_a$ is the single sink entity, the final layer representation $\boldsymbol{h}^L_{e_a}(e_q, r_q)$ is used as subgraph representation to encode the r-digraph $\mathcal{G}_{e_q,e_a|L}$. We name this simple solution as *RED-Simp*.

However, Algorithm 1 is very expensive. First, we need to conduct the two directional sampling in steps 3 and 4, and take the intersection to extract the r-digraph. Second, given a query $(e_q, r_q, ?)$, we need to do this algorithm for $|\mathcal{V}|$ different triples

**Algorithm 1** RED-Simp: Message passing on single r-digraph

```
1: initialize $\boldsymbol{h}^0_{e_q}(e_q, r_q) = \mathbf{0}$ and the entity sets $\mathcal{V}^0_{e_q} = \{e_q\}$, $\mathcal{V}^0_{e_a} = \{e_a\}$;
2: for $\ell = 1, 2, \dots, L$ do
3:     get the $\ell$-hop out-edges $\hat{\mathcal{E}}^\ell_{e_q} = \{(e_s, r, e_o) \in \mathcal{F} | e_s \in \mathcal{V}^{\ell-1}_{e_q}\}$
       and entities $\mathcal{V}^\ell_{e_q} = \{e_o | (e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}\}$ of $e_q$;
4:     get the $\ell$-hop in-edges $\check{\mathcal{E}}^\ell_{e_a} = \{(e_s, r, e_o) \in \mathcal{F} | e_o \in \mathcal{V}^{\ell-1}_{e_a}\}$
       and entities $\mathcal{V}^\ell_{e_a} = \{e_s | (e_s, r, e_o) \in \check{\mathcal{E}}^\ell_{e_a}\}$ of $e_a$;
5: end for
6: for $\ell = 0, 1, \dots, L$ do
7:     intersection $\mathcal{E}^\ell_{e_q, e_a|L} = \hat{\mathcal{E}}^\ell_{e_q} \cap \check{\mathcal{E}}^{L-\ell}_{e_a}$ and $\mathcal{V}^\ell_{e_q, e_a|L} = \mathcal{V}^\ell_{e_q} \cap \mathcal{V}^{L-\ell}_{e_a}$;
8: end for
9: if $\mathcal{V}^L_{e_q, e_a|L} = \emptyset$ return $\boldsymbol{h}^L_{e_a}(e_q, r_q) = \mathbf{0}$;
10: for $\ell = 1, 2, \dots, L$ do
11:     message passing for entities $e_o \in \mathcal{V}^\ell_{e_q, e_a|L}$:
        $\boldsymbol{h}^\ell_{e_o}(e_q, r_q) = \delta\Big(\boldsymbol{W}^\ell \cdot \sum_{(e_s, r, e_o) \in \mathcal{E}^\ell_{e_q, e_a|L}} \phi\big(\boldsymbol{h}^{\ell-1}_{e_s}(e_q, r_q), \boldsymbol{h}^\ell_r\big)\Big)$;
12: end for
13: return $\boldsymbol{h}^L_{e_a}(e_q, r_q)$.
```

with different answering entities $e_a \in \mathcal{V}$. It needs $O\big(|\mathcal{V}| \cdot (\min(\bar{D}^L, |\mathcal{F}|L) + d\bar{E}L)\big)$ time (see Section 4.3) to predict a given query $(e_q, r_q, ?)$, where $\bar{D}$ is the average degree of entities in $\mathcal{V}$ and $\bar{E}$ is the average number of edges in $\mathcal{E}^\ell_{e_q, e_a|L}$'s. These limitations also exist in PathCon [43] and GraIL [38]. To improve efficiency, we propose to encode multiple r-digraphs recursively in Section 4.1.

## 4.1 Recursive r-digraph encoding

In Algorithm 1, when evaluating $(e_q, r_q, e_a)$ with different $e_a \in \mathcal{V}$ but the same query $(e_q, r_q, ?)$, the neighboring edges $\hat{\mathcal{E}}^\ell_{e_q}, \ell = 1 \dots L$ of $e_q$ are shared. We have the following observation:

Proposition 1. *The set of edges $\hat{\mathcal{E}}^\ell_{e_q}$ visible from $e_q$ by $\ell$-steps equals $\cup_{e_a \in \mathcal{V}} \mathcal{E}^\ell_{e_q, e_a|L}$, namely $\hat{\mathcal{E}}^\ell_{e_q}$ is the union of the $\ell$-th layer edges in the r-digraphs between $e_q$ and all the entities $e_a \in \mathcal{V}$.*

Proposition 1 indicates that the $\ell$-th layer edges $\mathcal{E}^\ell_{e_q, e_a|L}$ with different answer entities $e_a$ share the same set of edges in $\hat{\mathcal{E}}^\ell_{e_q}$. A common approach to save computation cost in overlapping subproblems is dynamic programming. This has been used to aggregate node representations on large-scale graphs [14] or to propagate the question representation on KGs [52]. Inspired by the efficiency gained by dynamic programming, we recursively construct the r-digraph between $e_q$ and any entity $e_o$ as

$$\mathcal{G}_{e_q, e_o|\ell} = \cup_{(e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}} \mathcal{G}_{e_q, e_s|\ell-1} \otimes \left\{(e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}\right\}. \quad (2)$$

Once the representations of $\mathcal{G}_{e_q, e_s|\ell-1}$ for all the entities $e_s \in \mathcal{V}^{\ell-1}_{e_q}$ in the $\ell$−1-th layer are ready, we can encode $\mathcal{G}_{e_q, e_o|\ell}$ by combining $\mathcal{G}_{e_q, e_s|\ell-1}$ with the shared edges $(e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}$ in the $\ell$-th layer. Based on Proposition 1 and Eq.(2), we are motivated to recursively encode multiple r-digraphs with the shared edges in $\hat{\mathcal{E}}^\ell_{e_q}$ layer by layer. The full process is in Algorithm 2.

Initially, only $e_q$ is visible in $\mathcal{V}^0_{e_q}$. In the $\ell$-th layer, we collect the edges $\hat{\mathcal{E}}^\ell_{e_q}$ and entities $\mathcal{V}^\ell_{e_q}$ in step 3. Then, the message passing

**Algorithm 2** RED-GNN: recursive r-digraph encoding.

```
1: initialize $\boldsymbol{h}^0_{e_q}(e_q, r_q) = \mathbf{0}$ and the entity set $\mathcal{V}^0_{e_q} = \{e_q\}$;
2: for $\ell = 1 \dots L$ do
3:     collect the $\ell$-hop edges $\hat{\mathcal{E}}^\ell_{e_q} = \{(e_s, r, e_o) \in \mathcal{F} | e_s \in \mathcal{V}^{\ell-1}_{e_q}\}$
       and entities $\mathcal{V}^\ell_{e_q} = \{e_o | (e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}\}$;
4:     message passing for entities $e_o \in \mathcal{V}^\ell_{e_q}$:
       $\boldsymbol{h}^\ell_{e_o}(e_q, r_q) = \delta\Big(\boldsymbol{W}^\ell \cdot \sum_{(e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}} \phi\big(\boldsymbol{h}^{\ell-1}_{e_s}(e_q, r_q), \boldsymbol{h}^\ell_r\big)\Big)$;
5: end for
6: assign $\boldsymbol{h}^L_{e_a}(e_q, r_q) = \mathbf{0}$ for all $e_a \notin \mathcal{V}^L_{e_q}$;
7: return $\boldsymbol{h}^L_{e_a}(e_q, r_q)$ for all $e_a \in \mathcal{V}$.
```

is constrained to obtain the representations for $e_o \in \mathcal{V}^\ell_{e_q}$ through the edges in $\mathcal{E}^\ell_{e_q}$. Finally, the representations $\boldsymbol{h}^L_{e_a}(e_q, r_q)$, encoding $\mathcal{G}_{e_q, e_a|L}$ for all the entities $e_a \in \mathcal{V}$, are returned in step 7. The recursive encoding can be more efficient with shared edges in $\hat{\mathcal{E}}^\ell_{e_q}$ and fewer loops. It learns the same representations as Algorithm 1 as guaranteed by Proposition 2.

Proposition 2. *Given the same triple $(e_q, r_q, e_a)$, the structures encoded in $\boldsymbol{h}^L_{e_a}(e_q, r_q)$ by Algorithm 1 and Algorithm 2 are identical.*

## 4.2 Interpretable reasoning with r-digraph

As the construction of $\mathcal{G}_{e_q, e_a|L}$ is independent with the query relation $r_q$, how to encode $r_q$ is another problem to address. Given different triples sharing the same r-digraph, e.g. *(Sam, starred, Spider-2)* and *(Sam, directed, Spider-2)*, the local evidence we use for reasoning is different. To capture the query-dependent knowledge from the r-digraphs and discover interpretable local evidence, we use the attention mechanism [42] and encode $r_q$ into the attention weight to control the importance of different edges in $\mathcal{G}_{e_q, e_a|L}$. The message passing function is specified as

$$\boldsymbol{h}^\ell_{e_o}(e_q, r_q) = \delta\Big(\boldsymbol{W}^\ell \cdot \sum\nolimits_{(e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}} \alpha^\ell_{e_s, r, e_o | r_q}\big(\boldsymbol{h}^{\ell-1}_{e_s}(e_q, r_q) + \boldsymbol{h}^\ell_r\big)\Big), \quad (3)$$

where the attention weight $\alpha^\ell_{e_s, r, e_o|r_q}$ on edge $(e_s, r, e_o)$ is

$$\alpha^\ell_{e_s, r, e_o|r_q} = \sigma\left((\boldsymbol{w}^\ell_\alpha)^\top \mathrm{ReLU}\left(\boldsymbol{W}^\ell_\alpha \cdot \big(\boldsymbol{h}^{\ell-1}_{e_s}(e_q, r_q) \oplus \boldsymbol{h}^\ell_r \oplus \boldsymbol{h}^\ell_{r_q}\big)\right)\right), \quad (4)$$

with $\boldsymbol{w}^\ell_\alpha \in \mathbb{R}^{d_\alpha}$, $\boldsymbol{W}^\ell_\alpha \in \mathbb{R}^{d_\alpha \times 3d}$, and $\oplus$ is the concatenation operator. Sigmoid function $\sigma$ is used rather than softmax attention [42] to ensure multiple edges can be selected in the same neighborhood.

After $L$ layers' aggregation by (3), the representations $\boldsymbol{h}^L_{e_a}(e_q, r_q)$ can encode essential information for scoring $(e_q, r_q, e_a)$. Hence, we design a simple scoring function

$$f(e_q, r_q, e_a) = \boldsymbol{w}^\top \boldsymbol{h}^L_{e_a}(e_q, r_q), \quad (5)$$

with $\boldsymbol{w} \in \mathbb{R}^d$. We associate the multi-class log-loss [22] with each training triple $(e_q, r_q, e_a)$, i.e.,

$$\sum\nolimits_{(e_q, r_q, e_a) \in \mathcal{T}_{\text{tra}}} \Big(-f(e_q, r_q, e_a) + \log\big(\sum\nolimits_{\forall e \in \mathcal{V}} e^{f(e_q, r_q, e)}\big)\Big). \quad (6)$$

The first part in (6) is the score of the positive triple $(e_q, r_q, e_a)$ in $\mathcal{T}_{\text{tra}}$, the set of training queries, and the second part contains the scores of all triples with the same query $(e_q, r_q, ?)$. The model parameters $\boldsymbol{\Theta} = \big\{\{\boldsymbol{W}^\ell\}, \{\boldsymbol{w}^\ell_\alpha\}, \{\boldsymbol{W}^\ell_\alpha\}, \{\boldsymbol{h}^\ell_r\}, \boldsymbol{w}\big\}$ are randomly initialized and are optimized by minimizing (6) with stochastic gradient descent [19].

We provide Theorem 1 to show that if a set of relational paths are strongly correlated with the query triple, they can be identified by the attention weights in RED-GNN, thus being interpretable. This theorem is also empirically illustrated in Section 5.4.

Theorem 1. *Given a triple $(e_q, r_q, e_a)$, let $\mathcal{P}$ be a set of relational paths $e_q \rightarrow_{r_i^1} \cdot \rightarrow_{r_i^2} \cdots \rightarrow_{r_i^L} e_a$, that are generated by a set of rules between $e_q$ and $e_a$ with the form*

$$r_i^1(X, Z_1) \wedge r_i^2(Z_1, Z_2) \wedge \cdots \wedge r_i^L(Z_{L-1}, Y) \rightarrow r_q(X, Y),$$

*where $X, Y, Z_1, \ldots, Z_{L-1}$ are variables that are bounded by unique entities. Denote $\mathcal{G}_{\mathcal{P}}$ as the r-digraph constructed by $\mathcal{P}$. There exists a parameter setting $\Theta$ and a threshold $\theta \in (0,1)$ for RED-GNN that $\mathcal{G}_{\mathcal{P}}$ can equals to the r-digraph, whose edges have attention weights $\alpha^{\ell}_{e_s,r,e_o|r_q} > \theta$, in $\mathcal{G}_{e_q,e_a|L}$.*

## 4.3 Inference complexity

In this part, we compare the inference complexity of different GNN-based methods. When reasoning on the query $(e_q, r_q, ?)$, we need to evaluate $|\mathcal{V}|$ triples with the different answer entity $e \in \mathcal{V}$. We assume the average degree as $\bar{D}$ and average number of edges in each layer of r-digraph as $\bar{E}$.

- RED-Simp (Algorithm 1). For construction, the main cost, which is $O\big(\min(\bar{D}^L, |\mathcal{F}|L)\big)$ with the worst case cost $O(|\mathcal{F}|L)$, comes from extracting the neighborhoods of $e_q$ and $e_a$. Along with encoding, the total cost is $O\big(|\mathcal{V}| \cdot (\min(\bar{D}^L, |\mathcal{F}|L) + d\bar{E}L)\big)$.
- RED-GNN (Algorithm 2). Since the full computation is conducted on $\hat{\mathcal{E}}^{\ell}_{e_q}$ and $\mathcal{V}^{\ell}_{e_q}$, thus the cost is the number of edges times the dimension $d$, i.e. $O(d \cdot \min(\bar{D}^L, |\mathcal{F}|L))$ and $d \ll |\mathcal{V}|$.
- CompGCN. All the representations are aggregated in one pass with cost $O(d|\mathcal{F}|L)$. Then the scores for all the entities are computed with cost $O(|\mathcal{V}|d)$. The total cost is $O(d|\mathcal{F}|L + d|\mathcal{V}|)$.
- DPMPN. Denote the dimension and layer in the non-attentive GNN as $d_1, L_1$, the average sampled edges in the pruning procedure as $\bar{D}$. The main cost comes from the non-attentive GNN with cost $O(d_1|\mathcal{F}|L_1)$. The cost on sampled subgraph is $O(d\bar{D}^L)$. Thus, the total computation cost is $O(d_1|\mathcal{F}|L_1 + d\bar{D}^L)$.
- GraIL. Denote $\bar{V}$ as the average number of entities in the subgraphs. The complexity in constructing the enclosing subgraph is $O(\bar{E}\log\bar{V})$ with $\bar{E}$ edges and $\bar{V}$ entities. The cost of the GNN module is $O(d\bar{E}L)$. Then the overall cost is $O\big(|\mathcal{V}|(\bar{E}\log\bar{V} + d\bar{E}L)\big)$.

In comparison, we have RED-GNN ≈ CompGCN < DPMPN < RED-Simp < GraIL in terms of inference complexity. The empirical evaluation is provided in Section 5.3.

## 5 EXPERIMENTS

All the experiments are written in Python with PyTorch framework [27] and run on an RTX 2080Ti GPU with 11GB memory.

## 5.1 Inductive reasoning

Inductive reasoning is a hot research topic [14, 33, 38, 48] as there are emerging new entities in the real-world applications, such as new users, new items and new concepts [50]. Being able to reason on unseen entities requires the model to capture the semantic and local evidence ignoring the identity of entities.

**Setup.** We follow the general inductive setting [33, 38, 48] where there are new entities in testing and the relations are the same as those in training. Specifically, the training and testing contain two KGs $\mathcal{K}_{\text{tra}} = \{\mathcal{V}_{\text{tra}}, \mathcal{R}, \mathcal{F}_{\text{tra}}\}$ and $\mathcal{K}_{\text{tst}} = \{\mathcal{V}_{\text{tst}}, \mathcal{R}, \mathcal{F}_{\text{tst}}\}$, with the same set of relations but disjoint sets of entities. Three sets of triples $\mathcal{T}_{\text{tra}}/\mathcal{T}_{\text{val}}/\mathcal{T}_{\text{tst}}$, augmented with reverse relations, are provided. $\mathcal{F}_{\text{tra}}$ is used to predict $\mathcal{T}_{\text{tra}}$ and $\mathcal{T}_{\text{val}}$ for training and validation, respectively. In testing, $\mathcal{F}_{\text{tst}}$ is used to predict $\mathcal{T}_{\text{tst}}$. Same as [33, 38, 48], we use the filtered ranking metrics, i.e., mean reciprocal rank (MRR), Hit@1 and Hit@10 [5], to indicate better performance with larger values.

**Baselines.** Since training and testing contain disjoint sets of entities, all the methods requiring the entity embeddings [5, 15, 36, 41, 47, 51] cannot be applied here. We mainly compare with four methods: 1) RuleN [25], the discrete rule induction method; 2) Neural-LP [48], the first differentiable method for rule learning; 3) DRUM [33], an improved work of Neural-LP [48]; and 4) GraIL [38], which designs the enclosing subgraph for inductive reasoning. MINERVA [9], PathCon [43] and RNNLogic [29] can potentially work on this setting but there lacks the customized source code for the inductive setting, thus not compared.

**Hyper-parameters.** For RED-GNN, we tune the learning rate in $[10^{-4}, 10^{-2}]$, weight decay in $[10^{-5}, 10^{-2}]$, dropout rate in [0,0.3], batch size in $\{5, 10, 20, 50, 100\}$, dimension $d$ in {32,48,64,96}, $d_\alpha$ for attention in {3,5}, layer $L$ in {3,4,5}, and activation function $\delta$ in {identity, tanh, ReLU}. Adam [19] is used as the optimizer. The best hyper-parameter settings are selected by the MRR metric on $\mathcal{T}_{\text{val}}$ with maximum training epochs of 50. For RuleN, we use their implementation with default setting. For Neural LP and DRUM, we tune the learning rate in $[10^{-4}, 10^{-2}]$, dropout rate in [0,0.3], batch size in {20,50,100}, dimension $d$ in {64,128}, layer $L$ of RNN in $\{1, 2\}$, and number of steps in $\{2, 3, 4, 5\}$. For GraIL we tune the learning rate in $[10^{-5}, 10^{-2}]$, weight decay in $[10^{-6}, 10^{-3}]$, batch size in $\{8, 16, 32\}$, dropout rate in [0,0.4], edge_dropout rate in $[0, 0.6]$, GNN aggregator among {sum, MLP, GRU} and hop numbers among $\{2, 3, 4\}$. The training epochs are all set as 50.

**Tie policy.** In evaluation, the tie policy is important. Specifically, when there are triples with the same rank, choosing the largest rank and smallest rank in a tie will lead to rather different results [37]. Considering that we give the same score 0 for triples where $\mathcal{G}_{e_q,e_a|L} = \emptyset$, there will be a concern of the tie policy. Hence, we use the average rank among the triples in tie as suggested [31].

**Datasets.** We use the benchmark dataset in [38], created on WN18RR [10], FB15k237 [39] and NELL-995 [46]. Each dataset includes four versions with different groups of triples. Please refer to [38] for more details.

**Results.** The performance is shown in Table 1. First, GraIL is the worst among all the methods since the enclosing subgraphs do not learn well of the relational structures that can be generalized to unseen entities (more details in Appendix B). Second, there is not absolute winner among the rule-based methods as different rules adapt differently to these datasets. In comparison, RED-GNN outperforms the baselines across all the benchmarks. Based on Thereom 1, the attention weights can help to adaptively learn

**Table 1: Inductive reasoning. Best performance is indicated by the bold face numbers.**

| | | WN18RR | | | | FB15k-237 | | | | NELL-995 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | V1 | V2 | V3 | V4 | V1 | V2 | V3 | V4 | V1 | V2 | V3 | V4 |
| MRR | RuleN | .668 | .645 | .368 | .624 | .363 | .433 | .439 | .429 | .615 | .385 | .381 | .333 |
| | Neural LP | .649 | .635 | .361 | .628 | .325 | .389 | .400 | .396 | .610 | .361 | .367 | .261 |
| | DRUM | .666 | .646 | .380 | .627 | .333 | .395 | .402 | .410 | .628 | .365 | .375 | .273 |
| | GraIL | .627 | .625 | .323 | .553 | .279 | .276 | .251 | .227 | .481 | .297 | .322 | .262 |
| | **RED-GNN** | **.701** | **.690** | **.427** | **.651** | **.369** | **.469** | **.445** | **.442** | **.637** | **.419** | **.436** | **.363** |
| Hit@1 (%) | RuleN | 63.5 | 61.1 | 34.7 | 59.2 | 30.9 | 34.7 | 34.5 | 33.8 | **54.5** | 30.4 | 30.3 | 24.8 |
| | Neural LP | 59.2 | 57.5 | 30.4 | 58.3 | 24.3 | 28.6 | 30.9 | 28.9 | 50.0 | 24.9 | 26.7 | 13.7 |
| | DRUM | 61.3 | 59.5 | 33.0 | 58.6 | 24.7 | 28.4 | 30.8 | 30.9 | 50.0 | 27.1 | 26.2 | 16.3 |
| | GraIL | 55.4 | 54.2 | 27.8 | 44.3 | 20.5 | 20.2 | 16.5 | 14.3 | 42.5 | 19.9 | 22.4 | 15.3 |
| | **RED-GNN** | **65.3** | **63.3** | **36.8** | **60.6** | **30.2** | **38.1** | **35.1** | **34.0** | 52.5 | **31.9** | **34.5** | **25.9** |
| Hit@10 (%) | RuleN | 73.0 | 69.4 | 40.7 | 68.1 | 44.6 | 59.9 | 60.0 | 60.5 | 76.0 | 51.4 | 53.1 | 48.4 |
| | Neural LP | 77.2 | 74.9 | 47.6 | 70.6 | 46.8 | 58.6 | 57.1 | 59.3 | **87.1** | 56.4 | 57.6 | 53.9 |
| | DRUM | 77.7 | 74.7 | 47.7 | 70.2 | 47.4 | 59.5 | 57.1 | 59.3 | 87.3 | 54.0 | 57.7 | 53.1 |
| | GraIL | 76.0 | 77.6 | 40.9 | 68.7 | 42.9 | 42.4 | 42.4 | 38.9 | 56.5 | 49.6 | 51.8 | 50.6 |
| | **RED-GNN** | **79.9** | **78.0** | **52.4** | **72.1** | **48.3** | **62.9** | **60.3** | **62.1** | 86.6 | **60.1** | **59.4** | **55.6** |

**Table 2: Transductive reasoning. Best performance is indicated by the bold face numbers. '-' means unavailable results and results for methods with '*' are copied from the original papers.**

| type | models | Family | | | UMLS | | | WN18RR | | | FB15k-237 | | | NELL-995 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 | MRR | Hit@1 | Hit@10 |
| triple | ConvE* | - | - | - | .94 | 92. | 96. | .43 | 39. | 49. | .325 | 23.7 | 50.1 | - | - | - |
| | RotatE | .921 | 86.6 | 98.8 | .925 | 86.3 | 99.3 | .477 | 42.8 | 57.1 | .337 | 24.1 | 53.3 | .508 | 44.8 | 60.8 |
| | QuatE | .941 | 89.6 | 99.1 | .944 | 90.5 | 99.3 | .480 | 44.0 | 55.1 | .350 | 25.6 | 53.8 | .533 | 46.6 | 64.3 |
| path | MINERVA | .885 | 82.5 | 96.1 | .825 | 72.8 | 96.8 | .448 | 41.3 | 51.3 | .293 | 21.7 | 45.6 | .513 | 41.3 | 63.7 |
| | Neural LP | .924 | 87.1 | 99.4 | .745 | 62.7 | 91.8 | .435 | 37.1 | 56.6 | .252 | 18.9 | 37.5 | out of memory | | |
| | DRUM | .934 | 88.1 | 99.6 | .813 | 67.4 | 97.6 | .486 | 42.5 | 58.6 | .343 | 25.5 | 51.6 | out of memory | | |
| | RNNLogic* | – | – | – | .842 | 77.2 | 96.5 | .483 | 44.6 | 55.8 | .344 | 25.2 | 53.0 | – | – | – |
| GNN | pLogicNet* | – | – | – | .842 | 77.2 | 96.5 | .441 | 39.8 | 53.7 | .332 | 23.7 | 52.8 | – | – | – |
| | CompGCN | .933 | 88.3 | 99.1 | .927 | 86.7 | **99.4** | .479 | 44.3 | 54.6 | .355 | 26.4 | 53.5 | out of memory | | |
| | DPMPN | .981 | 97.4 | 98.1 | .930 | 89.9 | 98.2 | .482 | 44.4 | 55.8 | .369 | **28.6** | 53.0 | .513 | 45.2 | 61.5 |
| | **RED-GNN** | **.992** | **98.8** | **99.7** | **.964** | **94.6** | 99.0 | **.533** | **48.5** | **62.4** | **.374** | 28.3 | **55.8** | **.543** | **47.6** | **65.1** |

correlated relational paths for different datasets, and preserve the structural patterns at the same time. In some cases, the Hit@10 of RED-GNN is slightly worse than the rule-based methods since it may overfit to the top-ranked samples.

## 5.2 Transductive reasoning

Transductive reasoning, also known as KG completion [40, 44], is another general setting in the literature. It evaluates the models' ability to learn the patterns on an incomplete KG.

**Setup.** In this setting, a KG $\mathcal{K} = \{\mathcal{V}, \mathcal{R}, \mathcal{F}\}$ and the query triples $\mathcal{T}_{val}/\mathcal{T}_{\text{tst}}$, augmented with reverse relations, are given. For the triple-based method, triples in $\mathcal{F}$ are used for training, and $\mathcal{T}_{val}/\mathcal{T}_{\text{tst}}$ are used for inference. For the others, 3/4 of the triples in $\mathcal{F}$ are used to extract paths/subgraphs to predict the remaining 1/4 triples in training, and the full set $\mathcal{F}$ is then used to predict $\mathcal{T}_{val}/\mathcal{T}_{\text{tst}}$ in inference [33, 48]. We use the same filtered ranking metrics with the same tie policy in Section 5.1, namely MRR, Hit@1 and Hit@10.

**Baselines.** We compare RED-GNN with the triple-based methods ConvE [10], RotatE [36] and QuatE [51]; the path-based methods MINERVA [9], Neural LP [48], DRUM [33] and RNNLogic [29]; MLN-based method pLogicNet [30] and the GNN-based methods CompGCN [41] and DPMPN [47]. RuleN [25] is not compared here since it has been shown to be worse than DRUM [33] and RNNLogic [29] in this setting. p-GAT is not compared as their results are evaluated on a problematic framework [37]. GraIL [38] is not compared since it is computationally intractable on large graphs (see Section 5.3).

**Hyper-parameters.** The tuning ranges of hyper-parameters of RED-GNN are the same as those in the inductive reasoning. For RotatE and QuatE, we tune the dimensions in $\{100, 200, 500, 1000\}$, batch size in $\{256, 512, 1024, 2048\}$, weight decay in $[10^{-5}, 10^{-2}]$, number of negative samples in $\{64, 128, 256, 512, 1024\}$, with training iterations of 100000. For MINERVA, Neural LP, DRUM and DPMPN, we use their default setting provided. For CompGCN, we choose the decoder as ConvE, the operator as circular correlation as suggested [41], and tune the learning rate in $[10^{-4}, 10^{-2}]$, layer in $\{1, 2, 3\}$ and dropout rate in $[0, 0.3]$ with training epochs of 300.

**Datasets.** Five [2] datasets are used including Family [21], UMLS [21], WN18RR [10], FB15k237 [39] and NELL-995 [46]. We provide the statistics of entities, relations and split of triples in Table 3.

**Results.** As in Table 2, the triple-based methods are better than the path-based ones on Family and UMLS, and is comparable

[2] Data from https://github.com/alisadeghian/DRUM/tree/master/datasets and https://github.com/thunlp/OpenKE/tree/OpenKE-PyTorch/benchmarks/NELL-995.

**Table 3: Statistics of transductive reasoning datasets. Note that NELL-995* is different as the version in [9] since the training triples contains valid and test triples there.**

| | $\vert\mathcal{V}\vert$ | $\vert\mathcal{R}\vert$ | $\vert\mathcal{F}\vert$ | $\vert\mathcal{T}_{\text{val}}\vert$ | $\vert\mathcal{T}_{\text{tst}}\vert$ |
|---|---|---|---|---|---|
| Family | 3,007 | 12 | 23,483 | 2,038 | 2,835 |
| UMLS | 135 | 46 | 5,327 | 569 | 633 |
| WN18RR | 40,943 | 11 | 86,835 | 3,034 | 3,134 |
| FB15k-237 | 14,541 | 237 | 272,115 | 17,535 | 20,466 |
| NELL-995* | 74,536 | 200 | 149,678 | 543 | 2,818 |

with DRUM and RNNLogic on WN18RR, FB15k-237. The entity embeddings can implicitly preserve local information around entities, while the path-based methods may loss the structural patterns. CompGCN performs similar as the triple-based methods since it mainly relies on the aggregated embeddings and the decoder scoring function. Neural LP, DRUM and CompGCN run out of memory on NELL-995 with 74$k$ entities due to the use of full adjacency matrix. For DPMPN, the entities in the pruned subgraph is more informative than that in CompGCN, thus has better performance. For RED-GNN, it is better than all the baselines indicated by the MRR metric. These demonstrate that the r-digraph can not only transfer well to unseen entities, but also capture the important patterns in incomplete KGs without using entity embeddings.

## 5.3 Complexity analysis

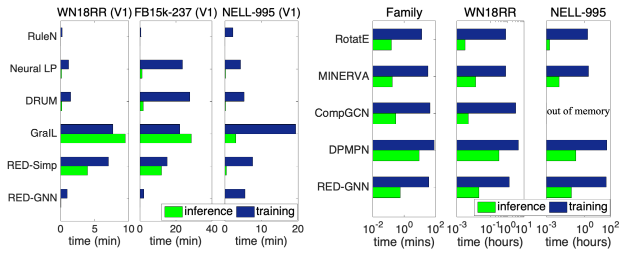
We compare the complexity in terms of running time and parameter size of different methods in this part. We show the training time and the inference time on $\mathcal{T}_{\text{tst}}$ for each method in Figure 2(a), learning curves in Figure 2(b), and model parameters in Figure 2(c).

**Inductive reasoning.** We compare RuleN, Neural LP, DRUM, GraIL and RED-GNN on WN18RR (V1), FB15k-237 (V1) and NELL-995 (V1). Both training and inference are very efficient in RuleN. Neural LP and DRUM have similar cost but are more expensive than RED-GNN by using the full adjacency matrix. For GraIL, both training and inference are very expensive since they require bidirectional sampling to extract the subgraph and then compute for each triple. As for model parameters, RED-GNN and GraIL have similar amount of parameters, less than Neural-LP and DRUM. Overall, RED-GNN is more efficient than the differentiable methods Neural LP, DRUM and GraIL.

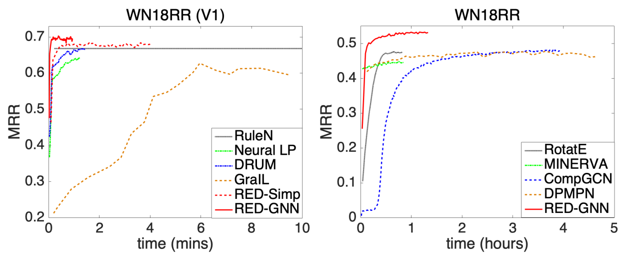
**Transductive reasoning.** We compare RotatE, MINERVA, CompGCN, DPMPN and RED-GNN on Family, WN18RR and NELL-995. Due to the simple training framework on triples, RotatE is the fastest method. MINERVA is more efficient than GNN-based methods since the sampled paths can be efficiently modeled in sequence. CompGCN, with fewer layers, has similar cost with RED-GNN as its computation is on the whole graph. For DPMPN, the pruning is expensive and it has two GNNs working together, thus it is more expensive than RED-GNN. GraIL is hundreds of times more expensive than RED-GNN, thus intractable on the larger KGs in this setting. Since RED-GNN does not learn entity embeddings, it has much less parameters compared with the other four methods.

## 5.4 Case study: learned r-digraphs

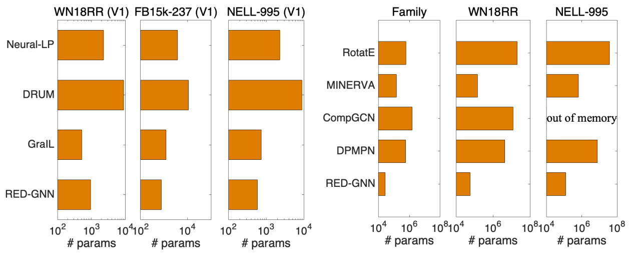
We visualize some exemplar learned r-digraphs by RED-GNN with $L = 3$ on the Family and UMLS datasets. Given the triple $(e_q, r_q, e_a)$,



(a) Running time.



(b) Learning curve.



(c) Model parameters.

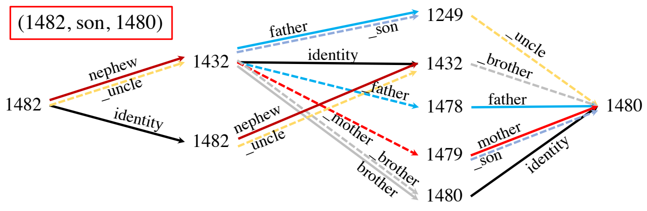
**Figure 2: Running time analysis and learning curve. Left: inductive setting; Right: transductive setting.**

we remove the edges in $\mathcal{G}_{e_q,e_a|L}$ whose attention weights are less than 0.5, and extract the remaining parts. Figure 3(a) shows one triple that DRUM fails. As shown, inferring id-1482 as the son of id-1480 requires the knowledge that id-1480 is the only brother of the uncle id-1432 from the local structure. Figure 3(b) shows the example with the same $e_q$ and $e_a$ sharing the same digraph $\mathcal{G}_{e_q,e_a|L}$. As shown, RED-GNN can learn distinctive structures for different query relations, which caters to Theorem 1. The examples in Figure 3 demonstrate that RED-GNN is interpretable. We provide more examples and the visualization algorithm in Appendix A.

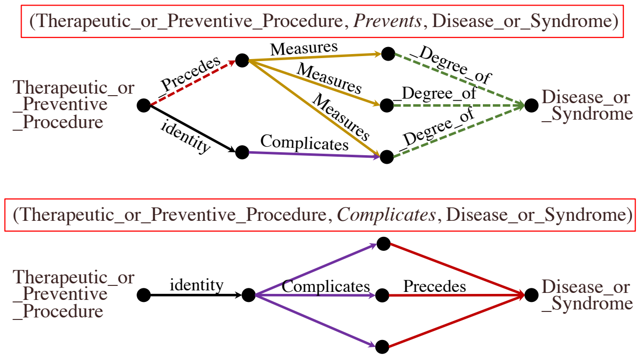
## 5.5 Ablation study

**Variants of RED-GNN.** Table 4 shows the performance of different variants. First, we study the impact of removing $r_q$ in attention (denoted as Attn-w.o.-$r_q$). Specifically, we remove $\boldsymbol{h}^{\ell}_{r_q}$ from the attention weight $\alpha^{\ell}_{e_s,r,e_o|r_q}$ in (4) and change the scoring function to $\boldsymbol{w}^{\top}(\boldsymbol{h}^{L}_{e_a}(e_q, r_q) \oplus \boldsymbol{h}_{r_q})$, with $\boldsymbol{w} \in \mathbb{R}^{2d}$. Since the attention $\alpha^{\ell}_{e_s,r,e_o|r_q}$ aims to figure out the important edges in $\mathcal{G}_{e_q,e_a|L}$, the learned structure will be less informative without the control of the query relation $r_q$, thus has poor performance.

Second, we replace Algorithm 2 in RED-GNN with the simple solution in Algorithm 1, i.e. RED-Simp. Due to the efficiency issue, the loss function (6), which requires to compute the scores over



(a) Family.


(Therapeutic_or_Preventive_Procedure, *Prevents*, Disease_or_Syndrome)
Therapeutic_or_Preventive_Procedure
Precedes
identity
Measures
Measures
Measures
Complicates
_Degree_of
Degree_of
Degree_of
Disease_or_Syndrome
(Therapeutic_or_Preventive_Procedure, *Complicates*, Disease_or_Syndrome)
Therapeutic_or_Preventive_Procedure
identity
Complicates
Precedes
Disease_or_Syndrome

(b) UMLS.

**Figure 3: Visualization of the learned structures. Dashed lines mean inverse relations. The query triples are indicated by the red rectangles. Due to space limitation, entities in UMLS dataset are shown as black circles (best viewed in color).**

**Table 4: Comparison of different variants of RED-GNN.**

| methods | WN18RR (V1) MRR | WN18RR (V1) H@10 | FB15k-237 (V1) MRR | FB15k-237 (V1) H@10 | NELL (V1) MRR | NELL (V1) H@10 |
|---|---|---|---|---|---|---|
| Attn-w.o.-$r_q$ | .659 | 78.3 | .268 | 37.6 | .517 | 73.4 |
| RED-Simp | .683 | 79.6 | .311 | 45.3 | .563 | 75.8 |
| RED-GNN | .701 | 79.9 | .369 | 48.3 | .637 | 86.6 |

many negative triples, cannot be used. Hence, we use the margin ranking loss with one negative sample as in GraIL [38]. As in Table 4, the performance of RED-Simp is weak than RED-GNN since the multi-class log loss is better than loss functions with negative sampling [22, 32, 55]. RED-Simp still outperforms GraIL in Table 1 since the r-digraphs are better structures for reasoning. The running time of RED-Simp is in Figure 2(a) and 2(b). Algorithm 1 is much more expensive than Algorithm 2 but is cheaper than GraIL since GraIL needs the Dijkstra algorithm to label the entities.

**Depth of models.** In Figure 4, we show the influence of testing MRR with different layers or steps $L$ in the left $y$-axis. The coverage (in %) of testing triples $(e_q, r_q, e_a)$ where $e_a$ is visible from $e_q$ in $L$ steps, i.e., $e_a \in \mathcal{V}^L_{e_q}$, is shown in the right $y$-axis. Intuitively, when $L$ increases, more triples will be covered, paths or subgraphs between $e_q$ and $e_a$ then contain richer information, but will be harder to learn. As shown, the performance of DRUM, Neural LP and MINERVA decreases for $L \geq 4$. CompGCN runs out of memory when $L > 3$ and it is also hard to capture complex structures with $L = 3$. When $L$ is too small, e.g., $L \leq 2$, RED-GNN has poor performance mainly dues to limited information encoded in such small r-digraphs. RED-GNN achieves the best performance for $L \geq 3$ where the r-digraphs can contain richer information and the important information for reasoning can be effectively learned by (3). Since the computation cost significantly increases with $L$, we tune $L \in \{3, 4, 5\}$ to balance the efficiency and effectiveness in practice.

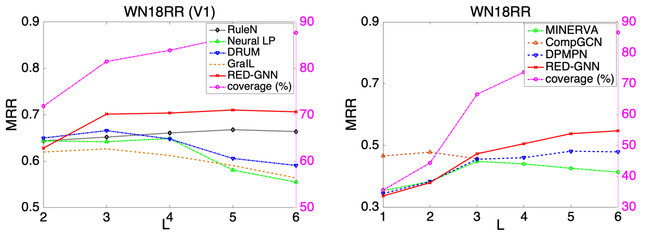


**Figure 4: The MRR performance with different $L$ and coverage of triples within $L$ steps.**

**Table 5: The per-distance evaluation for MRR on WN18RR v.s. the length of shortest path.**

| distance | 1 | 2 | 3 | 4 | 5 | >5 |
|---|---|---|---|---|---|---|
| ratios (%) | 34.9 | 9.3 | 21.5 | 7.5 | 8.9 | 17.9 |
| CompGCN | .993 | .327 | .337 | .062 | .061 | .016 |
| DPMPN | .982 | .381 | .333 | .102 | .057 | .001 |
| RED-GNN | .993 | .563 | .536 | .186 | .089 | .005 |

**Per-distance performance.** Note that given a r-digraph with $L$ layers, the information between two nodes that are not reachable with $L$ hops cannot be propagated by Algorithm 2. This may raise a concern about the predicting ability of RED-GNN, especially for triples not reachable in $L$ steps. We demonstrate this is not a problem here. Specifically, given a triple $(e_q, r_q, e_a)$, we compute the shortest distance from $e_q$ to $e_a$. Then, the MRR performance is grouped in different distances. We compare CompGCN ($L = 2$), DPMPN ($L = 5$) and RED-GNN ($L = 5$) in Table 5. All the models have worse performance on triples with larger distance and cannot well model the triples that are far away. RED-GNN has the best per-distance performance within distance 5.

## 6 CONCLUSION

In this paper, we introduce a novel relational structure, i.e., r-digraph, as a generalized structure of relational paths for KG reasoning. Individually computing on each r-digraph is expensive for the reasoning task $(e_q, r_q, ?)$. Hence, inspired by solving overlapping sub-problems by dynamic programming, we propose RED-GNN as a variant of GNN, to efficiently construct and effectively learn the r-digraph. We show that RED-GNN achieves the state-of-the-art performance in both KG with inductive and transductive KG reasoning benchmarks. The training and inference of RED-GNN are very efficient compared with the other GNN-based baselines. Besides, interpretable structures for reasoning can be learned by RED-GNN.

Lastly, a concurrent work NBFNet [57] proposes to recursively encode all the paths between the query entity to multiple answer entities, which is similar to RED-GNN. However, NBFNet uses the full adjacency matrix for propagation, which is very expensive and requires 128GB GPU memory. In the future work, we can leverage the pruning technique in [47] or distributed programming in [8] to apply RED-GNN on KG with extremely large scale.

## REFERENCES

[1] A. Abujabal, R. Saha Roy, M. Yahya, and G. Weikum. 2018. Never-ending learning for open-domain question answering over knowledge bases. In *The WebConf.* 1053–1062.

[2] P. Battaglia, J. B Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, et al. 2018. *Relational inductive biases, deep learning, and graph networks.* Technical Report. arXiv:1806.01261.

[3] G. D. Battista, P. Eades, R. Tamassia, and I. G Tollis. 1998. *Graph drawing: algorithms for the visualization of graphs.* Prentice Hall PTR.

[4] J. Berant and P. Liang. 2014. Semantic parsing via paraphrasing. In *ACL*. 1415–1425.

[5] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS*. 2787–2795.

[6] Y. Cao, X. Wang, X. He, Z. Hu, and T. Chua. 2019. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *The WebConf.* 151–161.

[7] X. Chen, S. Jia, and Y. Xiang. 2020. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications* 141 (2020), 112948.

[8] W. W Cohen, H. Sun, R A. Hofer, and M. Siegler. 2019. Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base. In *ICLR*.

[9] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*.

[10] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. 2017. Convolutional 2D knowledge graph embeddings. In *AAAI*.

[11] Y. Ding, Q. Yao, H. Zhao, and T. Zhang. 2021. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *SIGKDD*. 279–288.

[12] J. Gilmer, S. S Schoenholz, P. F Riley, O. Vinyals, and G. E Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.

[13] A. Grover and J. Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.

[14] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[15] L V. Harsha V., G. Jia, and S. Kok. 2020. Probabilistic logic graph attention networks for reasoning. In *Companion of WebConf.* 669–673.

[16] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al. 2021. Knowledge graphs. *CSUR* 54, 4 (2021), 1–37.

[17] K. Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[18] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. Yu. 2020. A survey on knowledge graphs: Representation, acquisition and applications. *TKDE* (2020).

[19] D. P Kingma and J. Ba. 2014. *Adam: A method for stochastic optimization.* Technical Report. arXiv:1412.6980.

[20] T. Kipf and M. Welling. 2016. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[21] S. Kok and P. Domingos. 2007. Statistical predicate invention. In *ICML*. 433–440.

[22] T. Lacroix, N. Usunier, and G. Obozinski. 2018. Canonical Tensor Decomposition for Knowledge Base Completion. In *ICML*. 2863–2872.

[23] N. Lao and W. W Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine learning* 81, 1 (2010), 53–67.

[24] N. Lao, T. Mitchell, and W. Cohen. 2011. Random walk inference and learning in a large scale knowledge base. In *EMNLP*. 529–539.

[25] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla, and H. Stuckenschmidt. 2018. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *ISWC*. Springer, 3–20.

[26] M. Niepert. 2016. Discriminative gaifman models. In *NIPS*, Vol. 29. 3405–3413.

[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. 2017. Automatic differentiation in PyTorch. In *ICLR*.

[28] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.

[29] M. Qu, J. Chen, L. Xhonneux, Y. Bengio, and J. Tang. 2021. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *ICLR*.

[30] M. Qu and J. Tang. 2019. Probabilistic Logic Neural Networks for Reasoning. *NeurIPS* 32, 7712–7722.

[31] A. Rossi, D. Firmani, A. Matinata, P. Merialdo, and D. Barbosa. 2021. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. *TKDD* (2021).

[32] D. Ruffinelli, S. Broscheit, and R. Gemulla. 2020. You can teach an old dog new tricks! on training knowledge graph embeddings. In *ICLR*.

[33] A. Sadeghian, M. Armandpour, P. Ding, and D Wang. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In *NeurIPS*. 15347–15357.

[34] M. Schlichtkrull, T. N Kipf, P. Bloem, Rianne Van D., I. Titov, and M. Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*. Springer, 593–607.

[35] Y. Shen, J. Chen, Y. Huang, P.and Guo, and J. Gao. 2018. M-walk: Learning to walk over graphs using monte carlo tree search. In *NeurIPS*.

[36] Z. Sun, Z. Deng, J. Nie, and J. Tang. 2019. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*.

[37] Z. Sun, S. Vashishth, S. Sanyal, P. Talukdar, and Y. Yang. 2020. A Re-evaluation of Knowledge Graph Completion Methods. In *ACL*. 5516–5522.

[38] K. K Teru, E. Denis, and W. Hamilton. 2020. Inductive Relation Prediction by Subgraph Reasoning. In *ICML*.

[39] K. Toutanova and D. Chen. 2015. Observed versus latent features for knowledge base and text inference. In *PWCVSMC*. 57–66.

[40] T. Trouillon, C. R Dance, É. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. 2017. Knowledge graph completion via complex tensor factorization. *JMLR* 18, 1 (2017), 4735–4772.

[41] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. 2019. Composition-based multi-relational graph convolutional networks. In *ICLR*.

[42] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. Graph attention networks. In *ICLR*.

[43] H. Wang, H. Ren, and J. Leskovec. 2021. Relational Message Passing for Knowledge Graph Completion. In *SIGKDD*. 1697–1707.

[44] Q. Wang, Z. Mao, B. Wang, and L. Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *TKDE* 29, 12 (2017), 2724–2743.

[45] W. Xiao, H. Zhao, V. W Zheng, and Y. Song. 2021. Neural PathSim for Inductive Similarity Search in Heterogeneous Information Networks. In *CIKM*. 2201–2210.

[46] W. Xiong, T. Hoang, and W. Wang. 2017. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In *EMNLP*. 564–573.

[47] X. Xu, W. Feng, Y. Jiang, X. Xie, Z. Sun, and Z. Deng. 2019. Dynamically Pruned Message Passing Networks for Large-Scale Knowledge Graph Reasoning. In *ICLR*.

[48] F. Yang, Z. Yang, and W. Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*. 2319–2328.

[49] D. Yu, Y. Yang, R. Zhang, and Y. Wu. 2021. Knowledge Embedding Based Graph Convolutional Network. In *The WebConf.* 1619–1628.

[50] M. Zhang and Y. Chen. 2019. Inductive Matrix Completion Based on Graph Neural Networks. In *ICLR*.

[51] S. Zhang, Y. Tay, L. Yao, and Q. Liu. 2019. Quaternion knowledge graph embeddings. In *NeurIPS*.

[52] Y. Zhang, H. Dai, Z. Kozareva, A. J Smola, and L. Song. 2018. Variational reasoning for question answering with knowledge graph. In *AAAI*.

[53] Y. Zhang, Q. Yao, and L. Chen. 2020. Interstellar: Searching Recurrent Architecture for Knowledge Graph Embedding. In *NeurIPS*, Vol. 33.

[54] Y. Zhang, Q. Yao, W. Dai, and L. Chen. 2020. AutoSF: Searching scoring functions for knowledge graph embedding. In *ICDE*. IEEE, 433–444.

[55] Y. Zhang, Q. Yao, Y. Shao, and L. Chen. 2019. NSCaching: simple and efficient negative sampling for knowledge graph embedding. In *ICDE*. IEEE, 614–625.

[56] H. Zhao, Q. Yao, J. Li, Y. Song, and K. L. Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *SIGKDD*. 635–644.

[57] Z. Zhu, Z. Zhang, L. Xhonneux, and J. Tang. 2021. Neural Bellman-Ford Networks: A General Graph Neural Network Framework for Link Prediction, In NeurIPS. *arXiv e-prints*, arXiv–2106.

## A VISUALIZATION

We provide the visualization of the r-digraph learned between entities in Algorithm 3. The key point is to backtrack the neighborhood edges of $e_a$ that has attention weight larger than a pre-defined threshold $\theta > 0$, i.e., steps 3-6. The remaining structures $\mathcal{G}_{e_q,e_a|L}(\theta)$ are returned as the structure selected by the attention weights.

**Algorithm 3** Visualization.

```
Require: entities $\mathcal{V}$, triples $\mathcal{F}$, query triple $(e_q, r_q, e_a)$, depth $L$, parameters $\Theta$, threshold $\theta$.
1: Run Algorithm 2 to obtain the attention weights $\alpha^{\ell}_{e_s,r,e_o|r_q}$, $\ell = 1 \dots L$ on the edges in each layer.
2: Initialize $\mathcal{V}^L_{e_q,e_a|L}(\theta) = \{e_a\}$.
3: for $\ell = L, L-1 \dots 1$ do
4:     collect the edges $\mathcal{E}^{\ell}_{e_q,e_a|L}(\theta) = \{(e_s, r, e_o)|e_o \in \mathcal{V}^L_{e_q,e_a|L}, \alpha^{\ell}_{e_s,r,e_o|r_q} \geq \theta\}$.
5:     collect the entities $\mathcal{V}^{\ell-1}_{e_q,e_a|L}(\theta) = \{e_s|(e_s, r, e_o) \in \mathcal{E}^{\ell}_{e_q,e_a|L}(\theta)\}$.
6: end for
7: return $\mathcal{G}_{e_q,e_a|L}(\theta) = \mathcal{E}^1_{e_q,e_a|L}(\theta) \otimes \mathcal{E}^2_{e_q,e_a|L}(\theta) \cdots \otimes \mathcal{E}^L_{e_q,e_a|L}(\theta)$.
```

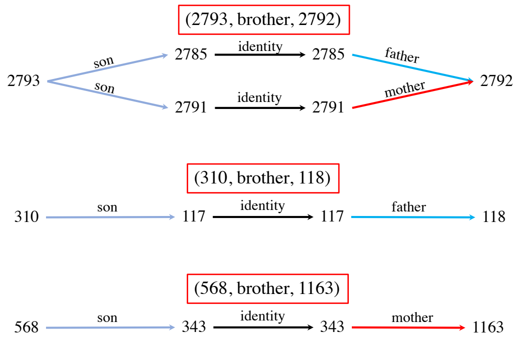


**Figure 5: Visualization of the learned structure on Family.**

We provide some additional visualized results on Family and UMLS datasets in Figure 5 and Figure 6,. There are consistent and semantically friendly patterns across different query triples.

## B PROBLEM ANALYSIS ON GRAIL

As mentioned in the main text, efficiency is the preliminary issue of GraIL [38]. In this part, we mainly analyze it in terms of effectiveness.

**Enclosing subgraph in [38] v.s. r-digraph.** When constructing the enclosing subgraph, the bidirectional information between $e_q$ and $e_a$ is preserved. It contains all the relational paths from $e_q$ to $e_a$ and all the paths from $e_a$ to $e_q$. This is the key difference between enclosing subgraph and r-digraph. In this view, the limitations of the enclosing subgraph can be summarized as follows.

- The entities in the enclosing subgraph need distance as labels. But in r-digraph, the distance is implicitly encoded in layers.
- The order of relations, inside the enclosing subgraph is mixed, making it hard to learn the order of relations, e.g. the difference between *brother*∧*mother*→*uncle* and *mother*∧*brother*→*aunt*.

The enclosing subgraph, even with more edges than the r-digraph, does not show valuable inductive bias for KG reasoning.

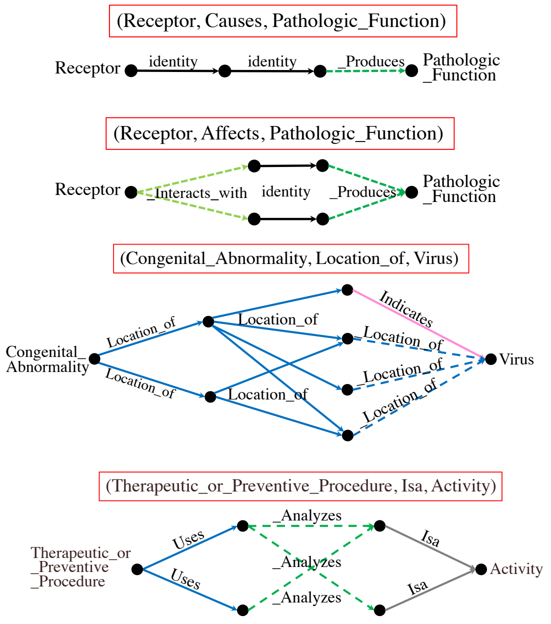


**Figure 6: Visualization of the learned structure on UMLS.**

**Non-interpretable.** We show the computation graphs of GraIL and RED-GNN in the yellow space and green space respectively in Figure 7. Even though attention is applied in the GNN framework, how the weights cooperate in different layers is unclear. Thus they did not provide interpretation and nor can we come up a way to interpret the reasoning results in GraIL.

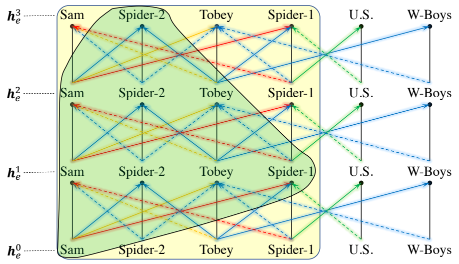


**Figure 7: The computation graph of GraIL (yellow space) and RED-GNN (green space).**

**Empirical evidence.** To show that GraIL fails to learn the relational structures, we conduct a tiny ablation study here. Specifically, we change the message function in GraIL from

$$\boldsymbol{a}_t^k = \sum\nolimits_{r=1}^{R} \sum\nolimits_{s \in \mathcal{N}_r(t)} \alpha^k_{r,r_t,s,t} \boldsymbol{W}_r^k \boldsymbol{h}_s^{k-1},$$

$$\text{to } \boldsymbol{a}_t^k = \sum\nolimits_{r=1}^{R} \sum\nolimits_{s \in \mathcal{N}_r(t)} \alpha^k_{r_t,s,t} \boldsymbol{W}^k \boldsymbol{h}_s^{k-1},$$

where $\alpha^k_{r_t,s,t} = \text{MLP}(\boldsymbol{h}_s^{k-1}, \boldsymbol{h}_t^{k-1}, \boldsymbol{e}^a_{r_t})$. This removes the relation when aggregating edges to only focus on graph structures.

We name this as GraIL no rel. The results are shown in Table 6. We observe that the performance decay is really marginal if relation is removed in the edges. This means that GraIL mainly learns on the graph structures and the relations play little role in. Thus, we claim that GraIL fails to learn the relational dependency in the ill-defined enclosing subgraph.

We also change the aggregation function in RED-GNN from (3) to

$$\boldsymbol{h}_{e_o}^{\ell}(e_q,r_q) = \delta\Big(\boldsymbol{W}^{\ell} \cdot \sum\nolimits_{(e_s,r,e_o)\in\hat{\mathcal{E}}_{e_q}^{\ell}} \alpha_{e_s,r_q}^{\ell}(\boldsymbol{h}_{e_s}^{\ell-1}(e_q,r_q) + \boldsymbol{v}^{\ell})\Big),$$

where $\boldsymbol{v}^{\ell} \in \mathbb{R}^d$ is shared in the same layer, to remove the relation on edges. We denote this variant as RED-no rel. Note that RED-no rel is different from the variant Attn-w.o.-$r_q$ in Table 4. As in Table 6, the performance drops dramatically. RED-GNN mainly relies on the relational dependencies in edges as evidence for reasoning.

**Table 6: Comparison of GraIL with and without relation information. Evaluated by MRR.**

| methods | WN18RR (V1) | FB15k-237 (V1) | NELL-995 (V1) |
|---|---|---|---|
| GraIL no rel | 0.620 | 0.255 | 0.475 |
| GraIL | 0.627 | 0.279 | 0.481 |
| RED-no rel | 0.557 | 0.198 | 0.384 |
| RED-GNN | 0.701 | 0.369 | 0.637 |

## C PROOFS

### C.1 Proposition 1

We prove this proposition from the bi-directions such that $\cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell} \subseteq \hat{\mathcal{E}}_{e_q}^{\ell}$ and $\hat{\mathcal{E}}_{e_q}^{\ell} \subseteq \cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell}$.

Proof. First, we prove that $\cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell} \subseteq \hat{\mathcal{E}}_{e_q}^{\ell}$.

As in step-7 of Algorithm 1, $\mathcal{E}_{e_q,e_a|L}^{\ell}$ is defined as $\mathcal{E}_{e_q,e_a|L}^{\ell} = \{(e_s,r,e_o) \in \hat{\mathcal{E}}_{e_q}^{\ell} | e_o \in \mathcal{V}_{e_q,e_a|L}^{\ell}\}$. Thus we have $\mathcal{E}_{e_q,e_a|L}^{\ell} \subseteq \hat{\mathcal{E}}_{e_q}^{\ell}$ and also $\cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell} \subseteq \hat{\mathcal{E}}_{e_q}^{\ell}$.

Second, we prove that $\hat{\mathcal{E}}_{e_q}^{\ell} \subseteq \cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell}$.

We use proof by contradiction here. Assume that $\hat{\mathcal{E}}_{e_q}^{\ell} \subseteq \cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell}$ is wrong, then there must exist $(e_s,r,e_o) \in \hat{\mathcal{E}}_{e_q}^{\ell}$ and $(e_s,r,e_o) \notin \mathcal{E}_{e_q,e_a|L}^{\ell} : \forall e_a \in \mathcal{V}$. In other words, there exists an edge $(e_s,r,e_o)$ that can be visited in $\ell$ steps walking from $e_q$, but not in any r-digraph $(e_s,r,e_o) \notin \mathcal{E}_{e_q,e_a|L}^{\ell}$. Based on Def. 3 of r-digraph, then, $(e_s,r,e_o)$ will be an edge that is $\ell$ steps away from $e_q$ but does not belong to the $\ell$-th triple of any relational paths $e_q{\rightarrow}_{r^1}{\rightarrow}_{r^2}\cdots{\rightarrow}_{r^L}e_a$, this is impossible. Therefore, we have $\hat{\mathcal{E}}_{e_q}^{\ell} \subseteq \cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell}$.

Based on above two directions, we prove that $\hat{\mathcal{E}}_{e_q}^{\ell} = \cup_{e_a\in\mathcal{V}}\mathcal{E}_{e_q,e_a|L}^{\ell}$. □

### C.2 Proposition 2

Given a query triple $(e_q,r_q,e_a)$ and $L$, when $\mathcal{G}_{e_q,r_q|L} = \emptyset$, it is obvious that $\boldsymbol{h}_{e_a}^{L}(e_q,r_q) = \boldsymbol{0}$ in both Algorithm 1 and Algorithm 2. For $\mathcal{G}_{e_q,r_q|L} \neq \emptyset$, we prove by induction. We denote $\hat{\boldsymbol{h}}_e^{\ell}(e_q,r_q)$ as the representations learned by Algorithm 1 and $\check{\boldsymbol{h}}_e^{\ell}(e_q,r_q)$ as the representations learned by Algorithm 2. Then, we show that $\hat{\boldsymbol{h}}_e^{\ell}(e_q,r_q) = \check{\boldsymbol{h}}_e^{\ell}(e_q,r_q)$ for all $\ell = 1 \dots L$ and $e \in V_{e_q,e_a|L}^{\ell}$.

Proof.

- When $\ell = 1$, $\hat{\boldsymbol{h}}_e^1(e_q,r_q) = \delta(\boldsymbol{W}^1 \cdot \sum_{(e_q,r,e)\in\mathcal{E}_{e_q,e_a|L}^1} \phi(\boldsymbol{0},\boldsymbol{h}_r^1))$, and $\check{\boldsymbol{h}}_e^1(e_q,r_q) = \delta(\boldsymbol{W}^1 \cdot \sum_{(e_q,r,e)\in\hat{\mathcal{E}}_{e_q}^1} \phi(\boldsymbol{0},\boldsymbol{h}_r^1))$, for $e \in V_{e_q,e_a|L}^1$. It is obvious that $\{(e_q,r,e) \in \mathcal{E}_{e_q,e_a|L}^1\} \equiv \{(e_q,r,e) \in \hat{\mathcal{E}}_{e_q}^1 | e \in V_{e_q,e_a|L}^1\}$. Thus, we have $\hat{\boldsymbol{h}}_e^1(e_q,r_q) = \check{\boldsymbol{h}}_e^1(e_q,r_q)$ for all $e \in V_{e_q,e_a|L}^1$.
- Assume that $\hat{\boldsymbol{h}}_e^{\ell-1}(e_q,r_q) = \check{\boldsymbol{h}}_e^{\ell-1}(e_q,r_q)$ for all $e \in V_{e_q,e_a|L}^{\ell-1}$, we prove that $\hat{\boldsymbol{h}}_e^{\ell}(e_q,r_q) = \check{\boldsymbol{h}}_e^{\ell}(e_q,r_q)$ for all $e \in V_{e_q,e_a|L}^{\ell}$.
  For Algorithm 1, we have
  $$\hat{\boldsymbol{h}}_e^{\ell}(e_q,r_q) = \delta(\boldsymbol{W}^{\ell} \cdot \sum\nolimits_{(e_s,r,e)\in\mathcal{E}_{e_q,e_a|L}^{\ell}} \phi(\hat{\boldsymbol{h}}_{e_s}^{\ell-1}(e_q,r_q),\boldsymbol{h}_r^{\ell})). \quad (7)$$
  For Algorithm 2, we have
  $$\check{\boldsymbol{h}}_e^{\ell}(e_q,r_q) = \delta(\boldsymbol{W}^{\ell} \cdot \sum\nolimits_{(e_s,r,e)\in\hat{\mathcal{E}}_{e_q}^{\ell}} \phi(\check{\boldsymbol{h}}_{e_s}^{\ell-1}(e_q,r_q),\boldsymbol{h}_r^{\ell})). \quad (8)$$
  As in step 7 of Algorithm 1, $\mathcal{E}_{e_q,e_a|L}^{\ell} = \{(e_s,r,e_o) \in \hat{\mathcal{E}}_{e_q}^{\ell} | e_o \in \mathcal{V}_{e_q,e_a|L}^{\ell}\}$. Then, for $e \in V_{e_q,e_a|L}^{\ell}$, the ranges of summation in (7) and (8) are the same, i.e., $\{(e_s,r,e) \in \mathcal{E}_{e_q,e_a|L}^{\ell}\} \equiv \{(e_s,r,e) \in \hat{\mathcal{E}}_{e_q}^{\ell} | e \in V_{e_q,e_a|L}^{\ell}\}$ and $e_s$ here are all belonging to $V_{e_q,e_a|L}^{\ell-1}$. Hence, based on the assumption, we have $\hat{\boldsymbol{h}}_e^{\ell}(e_q,r_q) = \check{\boldsymbol{h}}_e^{\ell}(e_q,r_q)$ or all $e \in V_{e_q,e_a|L}^{\ell}$.

By induction, we can have $\hat{\boldsymbol{h}}_e^{\ell}(e_q,r_q) = \check{\boldsymbol{h}}_e^{\ell}(e_q,r_q)$ for all $\ell = 1 \dots L$ and $e \in V_{e_q,e_a|L}^{\ell}$. Therefore, the representation $\hat{\boldsymbol{h}}_{e_a}^{\ell}(e_q,r_q)$ and $\check{\boldsymbol{h}}_{e_a}^{\ell}(e_q,r_q)$ learned by Algorithm 1 and Algorithm 2, respectively, are identical. □

### C.3 Theorem 1

Proof. Based on Definition 3, any set $\mathcal{P}$ of relational paths $e_q \rightarrow_{r_i^1}\rightarrow_{r_i^2} \cdots \rightarrow_{r_i^L} e_a$ are contained in the r-digraph $\mathcal{G}_{e_q,e_a|L}$.

Denote $G_{\mathcal{P}}$ as the r-digraph constructed by $\mathcal{P}$. In each layer, the attention weight is computed as

$$\begin{aligned}\alpha_{e_s,r,e_o|r_q}^{\ell} &= \sigma\Big((\boldsymbol{w}_{\alpha}^{\ell})^{\top}\text{ReLU}\big(\boldsymbol{W}_{\alpha}^{\ell} \cdot (\boldsymbol{h}_{e_s}^{\ell-1}(e_q,r_q) \oplus \boldsymbol{h}_r^{\ell} \oplus \boldsymbol{h}_{r_q}^{\ell})\big)\Big)\\ &= \text{MLP}^{\ell}(e_s,r,e_q,r_q).\end{aligned}$$

Then, we prove that $\mathcal{G}_{\mathcal{P}}$ can be extracted from the $L$-th layer and recursively to the first layer.

- In the $L$-th layer, denote $\mathcal{T}_{+}^{L}$ as the set of the triples $(e_s,r_i^L,e_a)$, whose attention weights are $\alpha_{e_s,r_i^L,e_a|r_q}^{L} = MLP^L(e_s,r_i^L,r_q,e_q)$, in the $L$-th layer of $\mathcal{G}_{\mathcal{P}}$. Based on the universal approximation theorem [17], there exists a set of parameters $\boldsymbol{w}_{\alpha}^{L}, \boldsymbol{W}_{\alpha}^{L}, \boldsymbol{h}_r^L$ that can learn a decision boundary $\theta$ that $(e_s,r_i^L,e_a) \in \mathcal{T}_{+}^{L}$ if $\alpha_{e_s,r_i^L,r_q}^{L} > \theta$ and otherwise $(e_s,r_i^L,e_a) \notin \mathcal{T}_{+}^{L}$. Then the $L$-th layer of $\mathcal{G}_{\mathcal{P}}$ can be extracted.
- Similarly, denote $\mathcal{T}_{+}^{L-1}$ as the set of the triples $(e_s,r_i^{L-1},e_o)$ that connects with the remaining entities in the $L-1$-th layer. Then, there also exists a set of parameters $\boldsymbol{w}_{\alpha}^{L-1}, \boldsymbol{W}_{\alpha}^{L-1}, \boldsymbol{h}_r^{L-1}$ that can learn a decision boundary $\theta$ that $(e_s,r_i^{L-1},e_o) \in \mathcal{T}_{+}^{L-1}$ if $\alpha_{e_s,r_i^{L-1},e_o|r_q}^{L-1} > \theta$ and otherwise not in. Besides,

$\boldsymbol{w}_\alpha^{L-1}, \boldsymbol{W}_\alpha^{L-1}, \boldsymbol{h}_r^{L-1}$ and $\boldsymbol{w}_\alpha^{L}, \boldsymbol{W}_\alpha^{L}, \boldsymbol{h}_r^{L}$ are independent with each other. Thus, we can extract the $L-1$-tfh layer of $\mathcal{G}_{\mathcal{P}}$.

- Finally, with recursive execution, $\mathcal{G}_{\mathcal{P}}$ can be extracted as the subgraph in $\mathcal{G}_{e_q, e_a|L}$ with attention weights $\alpha^{\ell}_{e_s, r, e_o|r_q} > \theta$.

□