UnRavL: A Neuro-Symbolic Framework for Answering Graph Pattern Queries in Knowledge Graphs

Tamara Cucumides[1] **Daniel Daza**[2,3] **Pablo Barceló**[4,5]
Michael Cochez[2,3] **Floris Geerts**[1] **Miguel Romero**[5,6] **Juan L Reutter**[4,6]
[1]Department of Computer Science, University of Antwerp
[2]Computer Science, Vrije Universiteit Amsterdam
[3]Discovery Lab, Elsevier, Amsterdam
[4]Inst. for Math. and Comp. Eng., Universidad Católica de Chile & IMFD Chile
[5]CENIA Chile
[6]Department of Computer Science, Universidad Católica de Chile
{tamara.cucumidesfaundez,floris.geerts}@uantwerp.be
{pbarcelo,mgromero,jreutter}@uc.cl
{m.cochez,d.dazacruz}@vu.nl

Abstract

The challenge of answering graph queries over incomplete knowledge graphs is gaining significant attention in the machine learning community. Neuro-symbolic models have emerged as a promising approach, combining good performance with high interpretability. These models utilize trained architectures to execute atomic queries and integrate modules that mimic symbolic query operators. Most neuro-symbolic query processors, however, are either constrained to *tree-like* graph pattern queries or incur an extensive computational overhead. We introduce a framework for *efficiently* answering *arbitrary* graph pattern queries over incomplete knowledge graphs, encompassing both tree-like and cyclic queries. Our approach employs an approximation scheme that facilitates acyclic traversals for cyclic patterns, thereby embedding additional symbolic bias into the query execution process. Supporting general graph pattern queries is crucial for practical applications but remains a limitation for most current neuro-symbolic models. Our framework addresses this gap and we experimentally demonstrate that it performs competitively on three datasets, effectively handling cyclic queries through our approximation strategy. Additionally, it maintains the performance of existing neuro-symbolic models on anchored tree-like queries and extends their capabilities to queries with existentially quantified variables.

1 Introduction

Knowledge graphs are prevalent in both industry and the scientific community, playing a crucial role in representing organizational knowledge [1, 2]. They model information as nodes (entities) and edges (relations between entities). Many applications using knowledge graphs, however, encounter the issue of *missing* information. As knowledge graphs are created or updated, their information can become stale or conflicting, and some data sources may remain unintegrated. Consequently, knowledge graphs often remain *incomplete*, lacking some entities or relations relevant to the application domain [3].

A particularly important reasoning task on knowledge graphs is *answering queries*. Traditional query answering methods, especially those from the data management and semantic web literature, focus on symbolic queries specified in specific symbolic query languages, and on extracting the information that can be derived from the knowledge *present* in the graph [2, 4, 5]. Given the incomplete nature of knowledge graphs, these methods fail to address the need to reason about unknown information, reducing their usefulness in many application domains [6]. This observation

T. Cucumides et al., UnRavL: A Neuro-Symbolic Framework for Answering Graph Pattern Queries in Knowledge Graphs. *Proceedings of the Third Learning on Graphs Conference (LoG 2024)*, PMLR 269, Virtual Event, November 26–29, 2024.

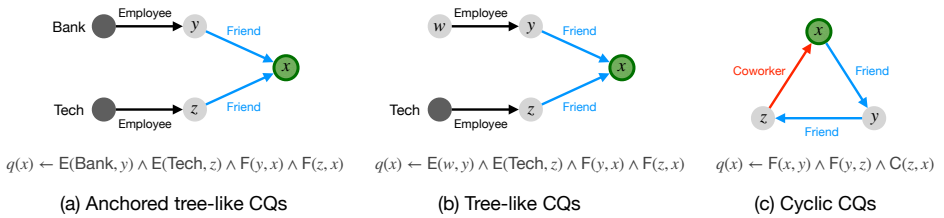


Figure 1: (a) Edges in *anchored tree-like* queries are structured as trees where the leaves are anchors and the root is the target variable (here x); (b) leaves in *tree-like* queries can be anchors or existential (unanchored) variables (here w); (c) arbitrary pattern queries can have cycles.

has spurred the development of numerous machine learning approaches to query answering [7–12], see Ren et al. [3] for a comprehensive recent survey.

Ren et al. argue that most existing approaches evaluate queries using *tree-based query plans*. In these plans, the leaves contain indicator vectors representing specific nodes, known as *query anchors*, within the knowledge graph. A bottom-up vector propagation scheme is then employed: along edges that represent relationships in the knowledge graph, learned relational predicates handle the propagation, while at internal nodes, learned logical operations (such as conjunction) combine multiple input vectors into one. Ultimately, the vector at the root node reflects the likelihood of entities belonging to the query result. However, the reliance on tree-based query plans imposes limitations on the types of queries that can be processed. Yin et al. [13] characterized these permissible queries as (anchored) *tree-form queries*, illustrated in Figure 1(a). This constraint precludes the evaluation of more complex queries, such as unanchored *tree-like* query patterns where leaf nodes may corresponds to variables instead of nodes, shown in Figure 1(b), cyclic query patterns as depicted in Figure 1(c), and queries involving multiple edges between entities. Ren et al. [3] identified the support for such queries as a significant open problem.

Recently, Yin et al. [13] proposed a method – Fuzzy Inference with Truth values (FIT) – for evaluating complex query patterns that involves the fuzzification of logical operations and, crucially, the *grounding* of existentially quantified variables. However, supporting general queries with this approach incurs a computational cost of $\mathcal{O}(n^k)$, where n is the number of entities in the knowledge graph and k is the number of variables.

In this paper we take a different approach for answering general query patterns. More specifically, we wish to evaluate or better, approximate, general query patterns using linear time ($\mathcal{O}(n)$) complexity, hereby enabling the evaluation of complex queries over large knowledge graphs. Our framework consists of two key parts. First, cyclic pattern queries are *unraveled* into tree-like queries that represent all node traversals of the query graph up to a given distance, capturing as much information as possible about the original query. Then, these query unravelings are processed by a neuro-symbolic query processor capable of answering them. More specifically, our contributions are as follows:

(1) We develop UnRavL, a trainable algorithm that processes cyclic queries via an unraveling strategy. This strategy serves as an approximation scheme with strong *theoretical guarantees*: it is *safe*, meaning no false negative query answers are produced, and it is *optimal* in that we provide the best possible approximation using tree-like queries whenever such an approximation exists.

(2) UnRavL is *adaptive* in that it is parameterized by the notion of *depth* of tree-like pattern queries. For any depth, an unraveling exists, and higher depth queries potentially offer better approximations. The choice of depth can be tuned based on available resources, queries, and data.

(3) Our neuro-symbolic processor for unravelings can answer any tree-like query *independently of its anchors*. We achieve this by implementing existential quantification directly in the latent space, which may be of independent interest for developing better processors for tree-like queries.

2 Preliminaries

Knowledge graphs. Knowledge graphs are directed graphs with labeled edges. Formally, a *knowledge graph* is represented as a triple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ where $\mathcal{V}$ is a finite set of *entities*, $\mathcal{R}$ is a

finite set of *edge types*, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ is a finite set of *edges*. An edge (a, R, b) is typically denoted by $R(a, b)$. Additionally, b is considered connected to a via the inverse relation R^{-1}, denoted as (b, R^{-1}, a) or $R^{-1}(b, a)$. The set $\mathcal{R}^{-1}$ represents the inverse relations of $\mathcal{R}$.

Graph pattern queries. A *graph pattern query* q on knowledge graphs is specified by a directed labeled graph, where: (i) each node is labeled with a unique *constant* (entity from $\mathcal{V}$) or variable (x, y, z, etc.); and (ii) each edge is labeled with one edge type from $\mathcal{R}$.

A crucial constraint is that each variable can only be assigned to a single node, while constants can adorn multiple nodes.[1] Furthermore, a single variable must be declared as the *target variable* of q, denoted as $q(x)$. Figure 1 depicts three graph pattern queries, each with x as target variable.

A graph pattern query $q(x)$ is termed *tree-like* if it forms a tree rooted at node x when ignoring edge direction. Specifically, multiple edges between pairs of nodes are not allowed. A tree-like query $q(x)$ must also satisfy that its non-leaf nodes are only labeled by variables. Moreover, q is *anchored* if all the leaves of this tree are constants; otherwise, it is *unanchored*. The *depth* of a tree-like query refers to the depth of the tree, indicating the length of the longest path from the root to any of its leaves. Finally, q is *cyclic* if it contains a cycle, when ignoring edge direction. The query shown in Figure 1(c) is cyclic, while those shown in Figures 1(a) and 1(b) are tree-like and of depth two. In addition, the query in Figure 1(a) is anchored, and the query in Figure 1(b) is unanchored.

Graph pattern queries can also be described using first-order logic formulas with atomic predicates, conjunction, and existential quantification (cf. Appendix A). For illustration, Figure 1 also shows the logical formula of each graph pattern query.

Query answering. Given a graph pattern query $q(x)$ and a knowledge graph $\mathcal{G}$, an *embedding* of q in $\mathcal{G}$ is a mapping μ from variables in q to constants in $\mathcal{V}$ such that each edge e in q maps to an edge $\mu(e)$ in $\mathcal{G}$. Here, $\mu(e)$ is derived by replacing variable nodes with constants as determined by μ. The *answers* of $q(x)$ on $\mathcal{G}$ constitute the set $q(\mathcal{G}) := \{\mu(x) \in \mathcal{V} \mid \mu \text{ embeds } q \text{ in } \mathcal{G}\}$. These are also referred to as the *easy answers*, as they are derived solely from the information available in $\mathcal{G}$. In contrast, neuro-symbolic approaches aim to find *hard answers*, which rely on unknown or missing information in the knowledge graph [3].

3 UnRavL: Query approximation by unraveling

In this section, we present our architecture, named Unravel (UnRavL). Following previous work, our architecture is neuro-symbolic: the task of predicting edges between nodes is carried out with trainable modules, and those modules are linked together with operations that mimic or approximate the symbolic operations mandated by the query. Next, we explain the two main components of UnRavL: how we process tree-like queries and how to extend this architecture to handle arbitrary graph patterns by unraveling queries into tree-like queries.

3.1 Processing tree-like queries

Our architecture builds on GNN-QE [11, 14], a neural-symbolic model that decomposes complex queries into relation projections, handled by graph neural networks, and fuzzy logic operations. We adapted it to support any tree-like query. It employs a bottom-up approach to process queries, iteratively handling subqueries while maintaining a feature-vector representation that signifies the likelihood of each node in the graph belonging to the answer set of the respective subquery.

Similar to the methodology outlined in Zhu et al. [11], the only trainable components of our architecture are the Neural Bellman-Ford Networks (NBFNets) [15], with each relation and its inverse in the knowledge graph associated with a dedicated NBFNet. For a given relation or inverse relation $R \in \mathcal{R} \cup \mathcal{R}^{-1}$, the corresponding NBFNet, denoted as $\mathcal{P}_R$, operates as follows: it takes as input a vector $\mathbf{x} \in [0, 1]^{|\mathcal{V}|}$, which reflects the likelihoods of nodes, and returns the updated likelihoods $\mathcal{P}_R(\mathbf{x}) \in [0, 1]^{|\mathcal{V}|}$ after traversing the graph using edges governed by R. For tree-like queries, we initiate the process from initial vectors in $[0, 1]^{|\mathcal{V}|}$ at the leaves. In previous works, only anchored leaf nodes were considered. In that case, if a is the constant in the leaf node, then the indicator vector

[1] We use a slightly more general definition compared to what is often used in related literature. In our case, multiple nodes in the *query* graph can be denoted with the same constant.

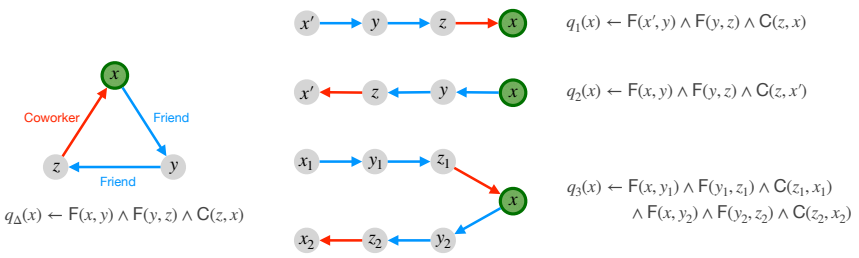


Figure 2: The triangle query q_Δ and tree-like approximations. The approximations represent different traversals of the query graph, starting at node (x). Best viewed in color.

$\mathbf{1}_a \in [0,1]^{|\mathcal{V}|}$ is used as initialization. Here, $\mathbf{1}_a(a) = 1$ and $\mathbf{1}_a(b) = 0$ for all $b \neq a$. Intuitively, NBFNets are used here to complete $\mathcal{G}$ by adding predicted edges.

Queries whose leaves are (existential) variables are not supported by GNN-QE. We handle these queries by encoding them directly in the latent space with the "all-ones" vector $\mathbf{1} \in [0,1]^{|\mathcal{V}|}$, as a way to model a uniform prior over all entities. Once the leaf node vectors are all initialized, we propagate these vectors upwards using the NBFNets corresponding to the relations. Moreover, if $\mathbf{x}_1$ and $\mathbf{x}_2$ are vectors in $[0,1]^{|\mathcal{V}|}$ obtained from processing sub-queries q_1 and q_2, respectively, then the output vector for the conjunction $q_1 \wedge q_2$ (merging of the two query graphs) is obtained as the pointwise multiplication $\mathbf{x}_1 \odot \mathbf{x}_2$. The final vector and evaluation of the query is the vector returned at the root node labeled with the target variable of the query.

Example 3.1. Consider the tree-like query shown in Figure 1(a). Let $\mathcal{P}_{\texttt{Employee}}$ and $\mathcal{P}_{\texttt{Friend}}$ be the trained NBFNets for the relations in our graph. The two leaves in the query are anchored with constants "Tech" and "Bank", and we initialize them with the indicator vectors $\mathbf{1}_{\text{Tech}}$ and $\mathbf{1}_{\text{Bank}}$, respectively. We traverse upward by computing $\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Tech}})$ and $\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Bank}})$. Intuitively, the ith entry of $\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Bank}})$ represents the likelihood of an edge $(\text{Bank}, \texttt{Employee}, n_i)$, with n_i being the ith node in the graph. The traversal continues by applying $\mathcal{P}_{\texttt{Friend}}$ to both $\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Tech}})$ and $\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Bank}})$. Finally, both paths in the tree are joined when reaching the target variable x. This conjunction (join) corresponds to computing

$$\left(\mathcal{P}_{\texttt{Friend}}(\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Tech}}))\right) \odot \left(\mathcal{P}_{\texttt{Friend}}(\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Bank}}))\right)$$

which represents the vector in $[0,1]^{|\mathcal{V}|}$ indicating the likelihood of nodes to be in the query answer.

Next, consider the tree-like query shown in Figure 1(b) in which one of the leaf nodes is unanchored. As mentioned, this corresponds to an initialization with $\mathbf{1}$. In the same way as before, the query is evaluated using $\left(\mathcal{P}_{\texttt{Friend}}(\mathcal{P}_{\texttt{Employee}}(\mathbf{1}))\right) \odot \left(\mathcal{P}_{\texttt{Friend}}(\mathcal{P}_{\texttt{Employee}}(\mathbf{1}_{\text{Bank}}))\right)$, which returns again a vector in $[0,1]^{|\mathcal{V}|}$ encoding likelihoods of nodes to be in the query answer. ◇

3.2 Processing general graph pattern queries

To support queries with cycles, we introduce an approach based on computing *traversals* of a query. The traversals we compute are (not necessarily anchored) tree-like queries, and we deal with them using the evaluation strategy outlined in the previous section.

Example 3.2. Consider the triangle query q_Δ shown in Figure 2 (left). We can associate with q_Δ various tree-like queries, obtained by *traversing* the edges (ignoring direction) in q_Δ, starting from the target variable x. For example, the pattern queries q_1, q_2, and q_3 shown in Figure 2 (right) are examples of traversals of q_Δ. Let a be an answer in $q_\Delta(\mathcal{G})$ for some knowledge graph $\mathcal{G}$. An important property of such traversals is that a will also belong to $q_1(\mathcal{G})$, $q_2(\mathcal{G})$, and $q_3(\mathcal{G})$, simply because there must be a traversal of edges $(a, \texttt{Friend}, b)$, $(b, \texttt{Friend}, c)$, $(c, \texttt{Coworker}, a)$ in $\mathcal{G}$. We say that q_Δ is *contained* in q_1, q_2 and q_3. This property holds true for any answer on any knowledge graph. The opposite is not necessarily true, e.g., $a' \in q_1(\mathcal{G})$ does not necessarily imply that $a' \in q_\Delta(\mathcal{G})$. ◇

Motivated by the example, we construct traversals that *over-approximate* cyclic queries: the traversals retreive all answers of the cyclic query, but may also retrieve additional nodes. While there can be many such traversals, our framework UnRavL only uses the *best traversal*, called the *unraveling* of

Algorithm 1 Unraveling of a general graph pattern query

Require: Depth d, query $q(x)$ with edges $\{R_i(x_i, y_i)\}_{i=1}^n$ and target variable x.
Ensure: Unraveling of q at depth d.
Return Local unraveling of variable x in query graph $\{R_i(x_i, y_i)\}_{i=1}^n$ at depth d, and make x the target variable.

Algorithm 2 Local unraveling

Require: Depth d, query graph $q = \{R_i(x_i, y_i)\}$, variable x and an optional edge e of q.
Ensure: Local unraveling of variable x in q at depth d, without using e.
if $d = 0$ **then**
  return the empty query.
end if
$\mathsf{In}(q, x) \leftarrow \{R(y, x) \mid R(y, x)$ is an edge in $q, R(y, x) \neq e\}$.
$\mathsf{Out}(q, x) \leftarrow \{R(x, y) \mid R(x, y)$ is an edge in $q, R(x, y) \neq e\}$.
$\hat{q} \leftarrow$ empty query.
for each edge f in $\mathsf{In}(q, x) \cup \mathsf{Out}(q, x)$ **do**
  if $f = R(x, y)$ **or** $f = R(y, x)$ with y a constant **then**
    add the edge f to $\hat{q}$.
  else
    add to $\hat{q}$ the edge f', obtained from f by replacing variable y by a fresh variable y'.
    $\alpha \leftarrow$ local unraveling of variable y in q at depth $d - 1$, without using f.
    $\alpha' \leftarrow$ replace in α, variable y by y', and every other variable by a new fresh variable.
    add to $\hat{q}$ every edge in α'.
  end if
end for
Return $\hat{q}$.

the query. The construction is parameterized by a *depth* parameter that poses a trade-off: the greater the depth, the closer the unraveling approximates a query, but the costlier it is to compute.

Algorithm 1 explains how to compute the unraveling. Starting with the target variable of q, we compute its local unraveling by moving through the edges of the query and recursively unraveling along the variables we find. Importantly, further calls to local unraveling also pass on the edge used to arrive at the variable being unraveled, which will not be traversed in that particular call; this avoids traversing the same edge consecutively. The output is always a tree-like query since different recursive calls, when computing the local unraveling of a variable, involve disjoint fresh variables.

For a given query q, let us denote by $\mathsf{UnRavL}_d(q)$ its *unraveling of depth* d, as computed by Algorithm 1. Our architecture UnRavL will, on input query q and graph $\mathcal{G}$, consider its tree-like unraveling $\mathsf{UnRavL}_d(q)$ and use the bottom-up evaluation technique, described earlier, to approximate q.

Example 3.3. Let us compute the unraveling of depth 3 for a slightly more general triangle query $q_\Delta(x)$ consisting of edges $q = \{R(x, y), S(y, z), T(z, x)\}$. For depth $d = 3$, the local unraveling at x adds edges $R(x, y_2)$ and $T(z_1, x)$ to $\hat{q}$, with appropriately renamed variables, originating from $\mathsf{In}(q, x)$ and $\mathsf{Out}(q, x)$. The two recursive calls for the depth $d = 2$ local unravelings of $\{S(y, z), T(z, x)\}$ at y, and of $\{R(x, y), S(y, z)\}$ at z, respectively, add the edges $S(y_2, z_2)$ and $S(y_1, z_1)$ to $\hat{q}$. In this example, we end up with two additional calls for the depth $d = 1$ local unravelings of $\{R(x, y), T(z, x)\}$ at z and at y, respectively, adding edges $T(z_2, x_2)$ and $R(x_1, y_1)$ to $\hat{q}$. The final unraveling is the tree-like query $\hat{q}(x)$ with edges $\{R(x, y_2), S(y_2, z_2), T(z_2, x_2), T(z_1, x), S(y_1, z_1), R(x_1, y_1)\}$. This unraveling corresponds to $q_3(x)$, shown in Figure 2 (right), when R and S are `Friend` and T is `CoWorker`. Hence, for the depth 3 approximation of q_Δ, UnRavL returns the following for the likelihoods of answer nodes.

$$\left(\mathcal{P}_{\texttt{CoWorker}}\left(\mathcal{P}_{\texttt{Friend}}\left(\mathcal{P}_{\texttt{Friend}}(\mathbf{1})\right)\right) \odot \left(\mathcal{P}_{\texttt{Friend}^{-1}}\left(\mathcal{P}_{\texttt{Friend}^{-1}}\left(\mathcal{P}_{\texttt{CoWorker}^{-1}}(\mathbf{1})\right)\right)\right)\right) \qquad \diamond$$

3.3 Theoretical guarantees

In this section we state the most important properties of unravelings, see Appendix B for more details. The first two properties show that our algorithm does produce good approximations of queries.

Proposition 3.1. For any query q, its unraveling $\mathsf{UnRavL}_d(q)$ at depth d satisfies:

- (**Safety**) It is an over-approximation of q: For any knowledge graph $\mathcal{G}$, any answer to q on $\mathcal{G}$ is always an answer to $\mathsf{UnRavL}_d(q)(\mathcal{G})$ on $\mathcal{G}$.
- (**Conservativeness**) If q is a tree-like query of depth d, then $\mathsf{UnRavL}_d(q)$ is equivalent to q. That is, they produce the same answers on any knowledge graph. □

Our next property states that $\mathsf{UnRavL}_d(q)$ is the *best possible tree-like approximation* at depth d of any query q without constants.

Proposition 3.2. For any query q without constants, its unraveling $\mathsf{UnRavL}_d(q)$ at depth d satisfies:

- (**Optimality**) It is the best possible over-approximation: any other tree-like query q' of depth d that satisfies the safety property must always produce the same or more answers that $\mathsf{UnRavL}_d(q)$ (but not less). That is, we have that $\mathsf{UnRavL}_d(q)(\mathcal{G})$ is contained in $q'(\mathcal{G})$ for any graph $\mathcal{G}$. □

We only show optimality for queries without constants because the unraveling of queries with constants have uneven depth, hence opening up room for intricate approximations that end up being incomparable to our unraveling.

Finally, we also have that higher depth unravelings potentially provide better approximations. The choice of depth can be tuned depending on available resources, queries and data at hand.

Proposition 3.3. For any query q and $d > 0$, the set of answers $\mathsf{UnRavL}_d(q)(\mathcal{G})$ is always contained in the set of answers $\mathsf{UnRavL}_{d-1}(q)(\mathcal{G})$ on any knowledge graph $\mathcal{G}$. □

Evaluating unravelings takes $\mathcal{O}(dn)$ time as the size of queries can be treated constant compared to n.

4 Experiments

We empirically investigate the performance of UnRavL[2] for answering complex graph pattern queries. Specifically, we aim to address the following questions:

Q1: How does UnRavL perform on cyclic queries?
Q2: How does the theoretical properties of our approximations translate into practice?
Q3: How does UnRavL perform on tree-like queries? Is it able to answer unanchored queries whilst remaining competitive on anchored queries against other neuro-symbolic approaches?

To answer **Q1** we tested our method on two query sets that include cyclic queries. For anchored cyclic queries we use the *real* EFO_1 introduced in [13], comparing against the FIT method proposed in the same paper. But we also try our method using a new benchmark of three new, non-anchored cyclic queries, which takes us beyond the capabilities of FIT. Here we compare against MPQE [16] and PQE (see details in Section 4.2). No other methods are considered as they don't support cyclic and/or unanchored queries. For **Q2**, we zoom in on results for different depths of unraveling, to see whether longer queries indeed perform better in practice, as suggested by our theoretical work. For **Q3**, we test our method on a mix of anchored queries from the BetaE benchmark [17] and unanchored queries resulting from expanding this benchmark. First, to see the performance regarding unanchored tree-like queries, we use our benchmark of unanchored queries and compare against PQE and MPQE, the only two methods we know are able to handle these queries. Further, to see whether UnRavL remains competitive on anchored, tree-like queries, we compare performance on the BetaE benchmark against GNN-QE [11], since it is the method closest to our tree-like query processor.

4.1 Experimental setup

We test two versions of UnRavL, depending on how we train the underlying tree-like query processor. a-UnRavL uses only anchored queries for training, while UnRavL uses a mix of both anchored and unanchored tree-like queries. We perform our training and evaluation on the commonly used knowledge graphs FB15k-237 [18], FB15k [19] and NELL995 [20] and we use four different query sets. Firstly, and following most previous work, we use the BetaE query set, consisting of 10 types of anchored tree-like queries. Second, we create a new set of 10 queries, based on the BetaE but allowing existential leaves (unanchored queries). Third, we use the *real* EFO_1 query set consisting of 10 query types, featuring unanchored tree-like queries and anchored cyclic queries. Finally, we

[2]The source code for UnRavL is available at https://anonymous.4open.science/r/UnRavL-A6EF/.

Table 1: Results of UnRavL, PQE and MPQE for cyclic queries: lollipop (∃1p2c), triangle (∃3c) and square query (∃4c). Results for UnRavL correspond to the best unraveling depth, while results for PQE corresponds to the metrics when considering the smallest dissociation score for each entity.

		FB15k237			FB15k			NELL995		
Metric	**Model**	**∃1p2c**	**∃3c**	**∃4c**	**∃1p2c**	**∃3c**	**∃4c**	**∃1p2c**	**∃3c**	**∃4c**
hits@1	UnRavL	**0.027**	**0.073**	**0.036**	0.057	**0.092**	**0.057**	0.065	**0.114**	0.067
	PQE	0.011	0.027	0.009	—	—	—	—	—	—
	MPQE	0.023	0.031	0.005	**0.064**	0.063	0.000	**0.074**	0.060	**0.098**
hits@10	UnRavL	**0.151**	**0.292**	**0.187**	**0.227**	**0.305**	**0.372**	0.286	**0.419**	0.257
	PQE	0.127	0.135	0.059	—	—	—	—	—	—
	MPQE	0.147	0.149	0.062	0.225	0.218	0.000	**0.301**	0.313	**0.326**
mrr	UnRavL	0.067	**0.148**	**0.088**	0.098	**0.297**	**0.159**	0.146	**0.211**	0.122
	PQE	0.057	0.077	0.031	—	—	—	—	—	—
	MPQE	**0.069**	0.072	0.025	**0.120**	0.116	0.000	**0.150**	0.142	**0.180**

introduce three extra cyclic- unanchored queries. See Figure 3, Figure 4 and Figure 5 in the Appendix for a graphic depiction of the queries. Although all these queries are used for testing, we train and validate on only certain types to effectively evaluate generalization on unseen query structures. It is also worth noticing that both the BetaE and *real* EFO1 query set include queries with negations and union. The ability of UnRavL to answer these queries comes from its underlying model. Details on training, validation, and results for queries with negation and union are in Appendix D.1 and D.5.

We follow previous work [11] to evaluate the performance of our model, so we use mean reciprocal rank (mrr) and hits@k as metrics. To further understand the intricacies of our model, we also incorporate classification metrics such as precision and recall using different classification thresholds.

4.2 Implementation

The implementation of UnRavL is built on top of GNN-QE [11]. We adapted GNN-QE to support unanchored tree-like queries and further trained it with both anchored and unanchored tree like queries. In line with previous works [8, 10, 11, 17], our model (i.e. the underlying NBFNets) is trained to minimize the binary cross entropy loss. Full details can be found in Appendix D.1.

Our baseline PQE is based on a non-trainable neuro-symbolic algorithm from the *Probabilistic Query Evaluation* literature. PQE starts by building a probabilistic database by assigning a likelihood score to each potential relation using an NBFNets link predictor [15]. Then, treating these scores as probabilities, PQE evaluates the query directly using a probabilistic solver. Storing the probabilistic graph requires $\mathcal{O}(|\mathcal{V}|^2)$ space and it was feasible only for FB15k-237, which is also why this approach is often disregarded as a possible baseline [7, 9]. As for the probabilistic solver, given the #P-hardness of computing the *possible world semantics* of graph pattern queries [21], PQE utilizes a polynomial time evaluation method based on *dissociations* of the original queries [22]. This approach is known to provide good upper approximations of the actual probabilities of query answers. Further details on PQE can be found in Appendix C.

As a second baseline, we consider Message Passing Query Embedding (MPQE) [16]. MPQE is a neural method that learns embeddings of entities in the knowledge graph, together with an *encoder* of queries–a graph neural network that maps a query graph to a vector representation. For a given query, MPQE returns a ranked list of entities by computing the cosine similarity between the query embedding computed by the encoder, and the embeddings of all entities in the graph. Although not studied in the original publication, the query encoder in MPQE supports arbitrary query patterns, including cyclic and unanchored queries. We train MPQE with anchored and unanchored tree-like queries, and we use the performance on the validation set for hyperparameter tuning. Further, we rely on the implementation by Alivanistos et al. [23], which makes use of a CompGCN [24] for the query encoder instead of the original R-GCN. More details can be found in Appendix D.2.

We do not implement (FIT) [13], but rather turn to the paper for the results.

Table 2: Mean reciprocal rank on the *real* EFO1 query set (excluding queries with negation). Results for (a-)UnRavL correspond to the best unraveling depth, and results of FIT are taken from the paper.

KG	Model	2il	3il	2m	3mp	3pm	im	3c	3cm	avg
FB15k-237	FIT	0.342	0.514	0.099	0.119	0.077	0.196	0.294	0.373	0.252
	a-UnRavL	0.379	0.521	**0.101**	**0.137**	**0.092**	**0.217**	**0.314**	**0.401**	**0.270**
	UnRavL	**0.401**	**0.546**	0.087	0.121	0.085	0.201	0.305	0.395	0.268
FB15k	FIT	0.704	0.776	0.735	0.573	0.640	0.794	0.638	0.654	0.689
	a-UnRavL	0.701	0.781	**0.740**	**0.606**	**0.677**	**0.814**	0.696	0.705	**0.715**
	UnRavL	**0.731**	**0.801**	0.720	0.598	0.637	0.801	**0.704**	**0.723**	0.714
NELL995	FIT	**0.533**	0.695	**0.421**	**0.240**	**0.228**	**0.415**	**0.475**	**0.453**	**0.433**
	a-UnRavL	0.501	0.690	0.417	0.231	0.222	0.411	0.452	0.437	0.420
	UnRavL	0.508	**0.701**	0.419	0.229	0.207	0.409	0.439	0.399	0.414

Table 3: Results of UnRavL for the triangle query's unravelings on FB15k-237 dataset, depths 2 to 10

Depth	2	3	4	5	6	7	8	9	10
mrr	0.137	0.147	0.143	0.140	0.143	0.141	0.143	**0.148**	0.147
hits@1	0.066	0.068	0.071	0.068	0.072	0.070	0.068	0.073	**0.075**
hits@10	0.285	**0.301**	0.290	0.290	0.287	0.283	0.292	0.292	0.293

4.3 Results

Q1 - Cyclic queries. Tables 1 and 2 display results for different cyclic queries on FB15k-237, FB15k, and NELL995. For unanchored queries (Table 1), we compare against MPQE and our baseline PQE, the only architectures we know that can handle these queries. We see that PQE is the worst performer, highlighting the advantage of training with different types of queries, and not just links. In general, UnRavL outperforms MPQE on the longer queries ($\exists$3c and $\exists$4c) across all datasets. MPQE is the best performer on the shorter $\exists$1p2c query, specially in the NELL dataset. We also note that when focusing in hits@10, a more comprehensive metric, UnRavL outperforms MPQE on 8 out of 9 query / dataset pairs.

In parallel, Table 2 show results of UnRavL and FIT for the *real* EFO1 query set that contains anchored cyclic queries. UnRavL outperforms FIT in datasets FB15k-237 and FB15k. In NELL, we note a slight disadvantage of UnRavL when compared to FIT. Interestingly, the same pattern occurs when comparing GNN-QE to FIT on tree-like anchored queries (see Table 9 in the Appendix). This result demonstrates that the performance of our framework UnRavL is inherited from its underlying neural model. We remark that although FIT performs slightly better in NELL, UnRavL remains competitive with its linear time complexity and no need for heuristic data preprocessing.

Q2 - Theoretical and experimental results. Tables 3 and 4 show how our approximation scheme performs on the triangle and square query respectively, when considering different depths for the unravelings. Our theoretical results state that performance should not decrease as we increase the depth of the unravelings and the experiments show that indeed both the mean reciprocal rank and hits@k metric tend to remain relatively stable. This suggests that the cases where one needs deep unravelings tend not to show in datasets. On the other hand, there are practical intricacies that are not taken into account in a theoretical setting: the training sets for complex query answering models only consider queries that involve up to 3 successive projections (3p), while our unravelings can be arbitrarily deep, which may lead to successive stacking of errors. In Appendix D.6 we show that Precision and Recall metrics are more affected by the depth of unravelings, although these metric themselves depend on a particular classification threshold.

Q3 - Tree-like queries. Table 5 shows the performance of UnRavL, PQE, and MPQE for general (unanchored) tree-like queries where UnRavL outperforms both of the other approaches. We see also the same pattern in the two unanchored queries (2il/3il) proposed in the *real* EFO1 dataset in Table 2. We note that this performance can be achieved without a noticeable drop in performance for *anchored* tree-like queries (an average drop of 4.24% in mrr, considering all datasets and queries). For more details, see Appendix D.3, where we compare with GNN-QE. We also report our comparison with UnRavL trained using anchored queries only.

Table 4: Results of UnRavL for the square query's unravellings on FB15k-237 dataset, depths 3 to 12

Depth	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
mrr	0.074	**0.088**	0.085	0.087	0.087	0.087	0.087	0.087	**0.088**	0.086
hits@1	0.023	**0.035**	0.028	0.033	0.033	0.034	0.034	0.032	**0.036**	0.030
hits@10	0.166	0.187	0.193	0.197	**0.201**	0.178	0.181	0.188	0.182	0.192

Table 5: Results of UnRavL, PQE and MPQE for unanchored tree-like queries.

Metric	**Model**	**∃1p**	**∃2p**	**∃3p**	**∃2i**	**∃3i**	**∃ip**	**∃pi**
				FB15k237				
hits@1	UnRavL	**0.028**	**0.046**	**0.057**	**0.123**	**0.203**	**0.030**	**0.090**
	PQE	0.021	0.001	0.011	0.017	0.071	0.027	0.051
	MPQE	0.012	0.041	0.035	0.021	0.027	**0.030**	0.028
mrr	UnRavL	**0.055**	**0.088**	**0.079**	**0.212**	**0.292**	0.076	**0.151**
	PQE	0.031	0.025	0.051	0.034	0.111	**0.107**	0.064
	MPQE	0.029	0.069	0.061	0.041	0.049	0.052	0.046
				FB15k				
hits@1	UnRavL	**0.831**	**0.643**	**0.533**	**0.759**	**0.791**	**0.566**	**0.668**
	MPQE	0.258	0.061	0.060	0.094	0.121	0.081	0.096
mrr	UnRavL	**0.855**	**0.688**	**0.587**	**0.801**	**0.833**	**0.620**	**0.720**
	MPQE	0.362	0.107	0.100	0.153	0.198	0.138	0.145
				NELL995				
hits@1	UnRavL	**0.831**	**0.643**	**0.533**	**0.759**	**0.791**	**0.566**	**0.668**
	MPQE	0.191	0.032	0.046	0.066	0.114	0.031	0.056
mrr	UnRavL	**0.479**	**0.160**	**0.119**	**0.378**	**0.484**	**0.140**	**0.269**
	MPQE	0.270	0.061	0.075	0.132	0.198	0.060	0.097

5 Related work

Neural and neuro-symbolic query answering. The machine learning community has produced a wide body of literature investigating how to answer complex queries over incomplete knowledge graphs. These works build on and extend successful methods for learning embeddings of entities and relations in knowledge graphs [19, 24–29]. We can identify two different approaches to complex query answering. Firstly, *neural* approaches [7, 8, 16, 17, 30–32] answer queries by designing functions (parameterized by neural networks) for mapping queries and entities in an embedding space where similarity indicates the likelihood of an answer. While competitive, the neural networks in these works act as black boxes that turn a query into a vector, thus resulting in a less interpretable system. Secondly, there are so-called *neuro-symbolic* approaches, which combine neural approaches to compute missing links between entities, and symbolic approaches to extract answers from the completed data [9, 11–13, 33–35]. While logical operators are still processed in the latent space, this process enforces a strong bias when computing queries, as the learning process must take into account the symbolic operations occurring downstream. With the exception of MPQE [16] and FIT [13], both neural and neuro-symbolic methods for query answering in the literature are limited to anchored tree-like queries, which is needed to form a computation graph in an embedding space, and thus they cannot be directly applied to cyclic or unanchored queries.

Approximation of conjunctive queries. The literature on tree-like approximations of queries identifies two types of approximations: *underapproximations*, which yield sound but not necessarily complete answers [36], and *overapproximations*, which yield complete but not necessarily sound answers [37]. However, these works focus on a slightly different notion of tree-like, namely, *treewidth-1* queries and are purely theoretical and have not found impact in practice before.

The notion of unraveling. The idea of unraveling the nodes of a graph to form a tree, also known as tree unfolding has been used before in several domains, from temporal logic [38] to database theory [39] to formal language theory [40], and even in machine learning, to formally study methods for graph embeddings [41]. However, up to our best knowledge, there is no previous work on the practical application of these techniques in the context of traditional query answering or in neural

query answering. Hence, our paper also contributes by showing how query approximations obtained by unravelings can be used in practice to provide state of the art querying algorithms.

6 Conclusions

UnRavL enhances neural query processors through an approximation scheme for cyclic queries and a latent encoding of existential quantification for queries without anchors. It outperforms other neural methods for general cyclic patterns and a probabilistic query evaluation baseline, while remaining competitive on existing benchmarks for anchored queries. However, a fully capable neural query processing engine for any graph database query is still elusive [3]. Future work includes developing better exact or approximation schemes for complex queries by, e.g., leveraging meta-path predictors [42–44], or recent query evaluation algorithms [45].

Acknowledgments

Michael Cochez was partially funded by the Graph-Massivizer project, funded by the Horizon Europe programme of the European Union (grant 101093202). Daniel Daza was funded by a research gift from Accenture, PLL.

References

[1] Dieter Fensel, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. *Knowledge Graphs - Methodology, Tools and Selected Use Cases*. Springer, 2020. URL https://doi.org/10.1007/978-3-030-37439-6. 1

[2] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021. URL https://doi.org/10.2200/S01125ED1V01Y202109DSK022. 1

[3] Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. Neural graph reasoning: Complex logical query answering meets graph databases. *CoRR*, abs/2303.14617, 2023. URL https://doi.org/10.48550/arXiv.2303.14617. 1, 2, 3, 10

[4] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50 (5), 2017. URL https://doi.org/10.1145/3104031. 1

[5] Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *The VLDB Journal*, 31, 2022. URL https://doi.org/10.1007/s00778-021-00711-3. 1

[6] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015. URL https://doi.org/10.1109/JPROC.2015.2483592. 1

[7] William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 2030–2041, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/ef50c335cca9f340bde656363ebd02fd-Abstract.html. 2, 7, 9

[8] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=BJgr4kSFDS. 7, 9

[9] Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Mos9F9kDwkz. 7, 9

[10] Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. In *Advances in Neural Information Processing Systems (NEURIPS)*, volume 34, pages 19172–19183, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/a0160709701140704575d499c997b6ca-Abstract.html. 7

[11] Zhaocheng Zhu, Mikhail Galkin, Zuobai Zhang, and Jian Tang. Neural-symbolic models for logical queries on knowledge graphs. In *International Conference on Machine Learning (ICML)*, PMLR, pages 27454–27478, 2022. URL https://proceedings.mlr.press/v162/zhu22c.html. 3, 6, 7, 9, 14, 19

[12] Erik Arakelyan, Pasquale Minervini, Daniel Daza, Michael Cochez, and Isabelle Augenstein. Adapting neural link predictors for data-efficient complex query answering. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/55c518a17bd17dcb69aa14d69d085994-Abstract-Conference.html. 2

[13] Hang Yin, Zihao Wang, and Yangqiu Song. Rethinking complex queries on knowledge graphs with neural link predictors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=1BmveEMNbG. 2, 6, 7, 9, 14

[14] Michael Galkin, Zhaocheng Zhu, Hongyu Ren, and Jian Tang. Inductive logical query answering in knowledge graphs. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/6246e04dcf42baf7c71e3a65d3d93b55-Abstract-Conference.html. 3

[15] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural Bellman-Ford Networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems (NEURIPS)*, 34: 29476–29490, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f6a673f09493afcd8b129a0bcf1cd5bc-Abstract.html. 3, 7, 19

[16] Daniel Daza and Michael Cochez. Message passing query embedding. In *ICML Workshop - Graph Representation Learning and Beyond*, 2020. URL https://arxiv.org/abs/2002.02406. 6, 7, 9

[17] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. In *Advances in Neural Information Processing Systems (NEURIPS)*, volume 33, pages 19716–19726, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e43739bba7cdb577e9e3e4e42447f5a5-Abstract.html. 6, 7, 9, 14

[18] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality (CVSC)*, pages 57–66, 2015. URL https://doi.org/10.18653/v1/W15-4007. 6

[19] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems (NEURIPS)*, volume 26, 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf. 6, 9

[20] Wenhan Xiong, Thien Hoang, and William Yang Wang. DeepPath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 564–573, 2017. URL https://doi.org/10.18653/v1/d17-1060. 6

[21] Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In Leonid Libkin, editor, *PODS*, pages 293–302. ACM, 2007. URL https://doi.org/10.1145/1265530.1265571. 7, 18

[22] Wolfgang Gatterbauer and Dan Suciu. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *VLDB J.*, 26(1):5–30, 2017. URL https://doi.org/10.1007/s00778-016-0434-5. 7, 18

[23] Dimitrios Alivanistos, Max Berrendorf, Michael Cochez, and Mikhail Galkin. Query embedding on hyper-relational knowledge graphs. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=4rLw09TgRw9. 7, 20

[24] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=BylA_C4tPr. 7, 9, 20

[25] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*, 2015. URL http://arxiv.org/abs/1412.6575.

[26] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the International Conference on International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, page 2071–2080, 2016. URL http://proceedings.mlr.press/v48/trouillon16.html.

[27] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

[28] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference (ESWC)*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018. URL https://doi.org/10.1007/978-3-319-93417-4_38.

[29] Komal K. Teru, Etienne G. Denis, and William L. Hamilton. Inductive relation prediction by subgraph reasoning. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 9448–9457, 2020. URL http://proceedings.mlr.press/v119/teru20a.html. 9

[30] Bhushan Kotnis, Carolin Lawrence, and Mathias Niepert. Answering complex queries in knowledge graphs with bidirectional sequence encoders. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 35(6):4968–4977, 2021. URL https://doi.org/10.1609/aaai.v35i6.16630. 9

[31] Xiao Liu, Shiyu Zhao, Kai Su, Yukuo Cen, Jiezhong Qiu, Mengdi Zhang, Wei Wu, Yuxiao Dong, and Jie Tang. Mask and reason: Pre-training knowledge graph transformers for complex logical queries. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1120–1130, 2022. URL https://doi.org/10.1145/3534678.3539472.

[32] Maximilian Pflueger, David J Tena Cucala, and Egor V Kostylev. GNNQ: A neuro-symbolic approach to query answering over incomplete knowledge graphs. In *International Semantic Web Conference (ISWC)*, pages 481–497, 2022. URL https://doi.org/10.1007/978-3-031-19433-7_28. 9

[33] Yushi Bai, Xin Lv, Juanzi Li, and Lei Hou. Answering complex logical queries on knowledge graphs via query computation tree optimization. In *International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 1472–1491, 2023. URL https://proceedings.mlr.press/v202/bai23b.html. 9

[34] Haoran Luo, E Haihong, Yuhao Yang, Gengxian Zhou, Yikai Guo, Tianyu Yao, Zichen Tang, Xueyuan Lin, and Kaiyang Wan. NQE: N-ary query embedding for complex query answering over hyper-relational knowledge graphs. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 37(4):4543–4551, 2023. URL https://doi.org/10.1609/aaai.v37i4.25576.

[35] Xuelu Chen, Ziniu Hu, and Yizhou Sun. Fuzzy logic based logical query answering on knowledge graphs. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 36 (4):3939–3948, 2022. URL https://doi.org/10.1609/aaai.v36i4.20310. 9

[36] Pablo Barceló, Leonid Libkin, and Miguel Romero. Efficient approximations of conjunctive queries. *SIAM J. Comput.*, 43(3):1085–1130, 2014. URL https://doi.org/10.1137/1309117. 9

[37] Pablo Barceló, Miguel Romero, and Thomas Zeume. A more general theory of static approximations for conjunctive queries. *Theory of Computing Systems*, 64(5):916–964, 2020. URL https://doi.org/10.1007/s00224-019-09924-0. 9

[38] Philippe Schnoebelen. The complexity of temporal logic model checking. *Advances in modal logic*, 4(393-436):35, 2002. 9

[39] Peter Buneman, Mary Fernandez, and Dan Suciu. Unql: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9:76–110, 2000. 9

[40] Moshe Y Vardi. Nontraditional applications of automata theory. In *International Symposium on Theoretical Aspects of Computer Software*, pages 575–597. Springer, 1994. 9

[41] Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel. A generalized weisfeiler-lehman graph kernel. *Machine Learning*, 111(7):2601–2629, 2022. 9

[42] Francesco Ferrini, Antonio Longa, Andrea Passerini, and Manfred Jaeger. Meta-path learning for multi-relational graph neural networks. In *Learning on Graphs Conference*, pages 2–1. PMLR, 2024. URL https://proceedings.mlr.press/v231/ferrini24a/ferrini24a.pdf. 10

[43] Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Heterogeneous graph representation learning with relation awareness. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5935–5947, 2023. URL https://doi.org/10.1109/TKDE.2022.3160208.

[44] Seongjun Yun, Minbyul Jeong, Sungdong Yoo, Seunghun Lee, Sean S. Yi, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks: Learning meta-path graphs to improve gnns. *Neural Networks*, 153:104–119, 2022. URL https://doi.org/10.1016/j.neunet.2022.05.026. 10

[45] Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013. URL https://doi.org/10.1145/2590989.2590991. 10

[46] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977. URL https://doi.org/10.1145/800105.803397. 16

[47] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. URL https://doi.org/10.2200/S00362ED1V01Y201105DTM016. 18

[48] Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2013. URL https://doi.org/10.1145/2395116.2395119. 18

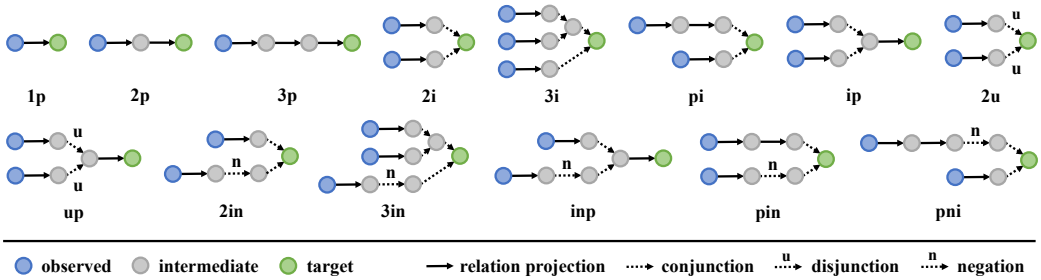


Figure 3: Graph pattern queries introduced by Ren and Leskovec [17] and used in our experiments. (The figure is taken from Zhu et al. [11].)

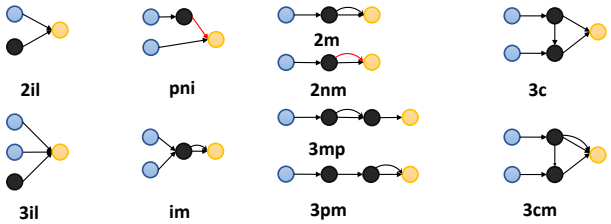


Figure 4: Graph pattern queries introduced by Yin et al. [13] and used in our experiments. (The figure is taken from Yin et al. [13].)

A Connection to logic

We have described queries in terms of their graph representation. Here, we make the connection to logic more precise.

Let Var be a countably infinite set of *variables* and let Con be a countably infinite set of *constants*. Consider the following class of first-order logic formulas (called *(unary) conjunctive queries*) over the set $\mathcal{R}$ of edge types and constants in Con, of the form (we use rule-based notation)

$$q(x) \leftarrow R_1(y_1, z_1) \wedge \cdots \wedge R_m(y_m, z_m),$$

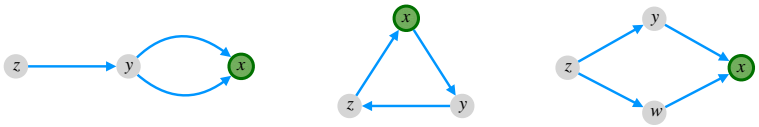
where x is the *target* variable, and each $R_i(y_i, z_i)$ is an *atom* with $R_i \in \mathcal{R}$ and $\{y_i, z_i\} \subseteq \mathsf{Con} \cup \mathsf{Var}$ (y_i, z_i are either variables or constants). The variable set $\mathsf{Var}(q)$ of q is the set of variables appearing in the atoms of q, that is, the variables appearing in $\{y_1, z_1, \ldots, y_m, z_m\}$. Similarly, we denote by $\mathsf{Con}(q)$ the constants appearing in the atoms of q. As usual, we assume $x \in \mathsf{Var}(q)$. The variables in $\mathsf{Var}(q) \setminus \{x\}$ are the *existentially quantified* variables of q. The semantics of these queries is defined using the standard semantics of first-order logic.

As an example, Figure 1(c) shows the logical formula $q(x) \leftarrow \mathrm{Friend}(x, y) \wedge \mathrm{Friend}(y, z) \wedge \mathrm{Coworker}(z, x)$ looking for all persons x that have a friend y and a coworker z who are friends with each other. Here, $\mathsf{Var}(q) = \{x, y, z\}$, x is the target variable, and y and z are both existentially quantified.

We turn logical formulas into a graph representation as follows. We take $\mathsf{Var}(q) \cup \mathsf{ConOcc}(q)$ as nodes, and an edge from node u to node v for every atom $R(u, v)$ in q. Here, $\mathsf{ConOcc}(q)$ is the set of occurrences of constants in q, i.e., if the number of occurrences in different atoms of q of a constant $a \in \mathsf{Con}(q)$ is k, then there are k duplicates of a in $\mathsf{ConOcc}(q)$. The target variable is, of course, taken as the same.

Conversely, it is readily verified that the graph representation of queries corresponds to formulas as described above. Indeed, simply take a conjunction over all edges and existentially quantify all variables except for the target variable.

With this connection in place, we can say that a formula has a certain graph property, e.g., being tree-like, meaning that its graph representation has that property.



(a) Double paths (b) Triangle (c) Square

Figure 5: Examples of cyclic queries.

Example A.1. The cyclic queries shown Figures 5 can be easily seen to correspond to the logical formulas $q_{\mathsf{lp}}(x) \leftarrow R(y,x) \wedge S(y,x) \wedge T(z,y)$, $q_{\Delta}(x) \leftarrow R(x,y) \wedge S(y,z) \wedge T(z,x)$, and $q(x) \leftarrow R(y,x) \wedge S(y,x) \wedge T(z,y)$ and $q_{\square}(x) \leftarrow R(y,x) \wedge S(w,x) \wedge T(z,y) \wedge U(z,w)$, respectively. ◇

B Details of Propositions 3.1, 3.2 and 3.3

We here provide an alternative description for computing the unraveling of a query, which can easily be shown to be equivalent (by induction on d) to the unraveling computed by Algorithm 1.

Unraveling in terms of valid paths. Let $q(x)$ be a graph pattern query with target variable x. A *valid path* of $q(x)$ is a sequence $u_0, e_1, u_1, \ldots, e_k, u_k$, for $k \geq 0$, such that:

- For all $1 \leq i \leq k$, we have that u_i is a node in the graph pattern. Moreover, u_0 is labeled by x, u_i is labeled by a variable, for all $1 \leq i < k$, and u_k is either labeled by a variable or a constant.
- For all $1 \leq i \leq k$, we have that e_i is an edge in q with endpoints u_{i-1} and u_i (that is, e_i is either of the form $R(u_{i-1}, u_i)$ or $R(u_i, u_{i-1})$).
- For all $1 \leq i < k$, we have that $e_i \neq e_{i+1}$.

Intuitively, a valid path is a way of traversing q starting from the target variable x and sequentially moving through the edges of q, ignoring direction. We can visit the same variable or edge several times. The only restriction is that an edge cannot be visited multiple times *consecutively* in the sequence. Hence, once an edge is traversed, we cannot go back via the same edge immediately.

The *length* of a valid path is the number of edges k. Note that the valid path of length 0 is well-defined and corresponds to the sequence x. A valid path is *unanchored* if it ends at a variable of q; otherwise, it is *anchored*. For a valid path P, we denote by $\mathsf{end}(P)$ the node at the end of path P (which is labeled either by a variable or constant).

Example B.1. Consider the triangle query $q_{\Delta}(x)$, shown in Figure 2 (left), consisting of edges $\{\texttt{Friend}(x,y), \texttt{Friend}(y,z), \texttt{Coworker}(z,x)\}$. An example of an unanchored valid path is $x, \texttt{Friend}, y, \texttt{Friend}, z, \texttt{Coworker}, x$, which corresponds to a clockwise traversal of length 3 starting at x. The anticlockwise traversal of length 3 is given by the valid path $x, \texttt{Coworker}, z, \texttt{Friend}, y, \texttt{Friend}, x$. ◇

Let q be a graph pattern query and $d \geq 1$. We need variables corresponding to each unanchored valid path P of q of length $\leq d$. We denote the variable corresponding to such a path P by a fresh variable z_P, although encoding of paths by any fresh variables suffices, as shown next.

Example B.2. We continue with the triangle query $q_{\Delta}(x)$. Let us introduce some variables for valid paths:

variable	path
x	x
z_1	$x, \texttt{Coworker}, z$
y_1	$x, \texttt{Coworker}, z, \texttt{Friend}, y$
x_1	$x, \texttt{Coworker}, z, \texttt{Friend}, y, \texttt{Friend}, x$
y_2	$x, \texttt{Friend}, y$
z_2	$x, \texttt{Friend}, y, \texttt{Friend}, z$
x_2	$x, \texttt{Friend}, y, \texttt{Friend}, z, \texttt{Coworker}, x$

We thus have introduced variables for each unanchored valid path of length at most three in q, starting from x. ◇

The *unraveling* of $q(x)$ of depth $d \geq 1$, as computed by Algorithm 1, is easily shown to be equivalent to the following graph pattern query:

- The nodes of the graph pattern are the valid paths of length $\leq d$.
- Unanchored valid paths P are labeled with the fresh variable z_P. Anchored valid paths P are labeled with the constant labeling $\mathsf{end}(P)$.
- The target variable is $x := z_x$.
- For valid paths P and $P' = P, e', \mathsf{end}(P')$ of q of lengths $\leq d$, if e' is an edge labeled R from $\mathsf{end}(P)$ to $\mathsf{end}(P')$, then we add an edge with label R from P to P'. Otherwise, we add an edge with label R from P' to P.

We recall that we denote the corresponding graph query by $\mathsf{UnRavL}_d(q)$.

Example B.3. It is now easily verified that the graph pattern query $q_3(x)$ shown in Figure 2(c) is the depth three unraveling of $q(x)$, using the variables for paths introduced in the previous example. Indeed, it is the pattern query consisting of the following edges: $\{\texttt{Friend}(x, y_2), \texttt{Friend}(y_2, z_2), \texttt{Coworker}(z_2, x_2), \texttt{Coworker}(z_1, x), \texttt{Friend}(y_1, z_1), \texttt{Friend}(x_1, y_1)\}$. ◇

The idea is that the unraveling of depth d of $q(x)$ is obtained by traversing q in a tree-like fashion, starting from the target variable x and moving from one variable to all of its neighbors through the edges of q. Each time, we add fresh variables to the unraveling, resulting in a tree-like query. The tree traversal has depth d and is always restricted to valid paths (no immediate returns to the same atom). The leaves of the unraveling could be anchors or variables. As mentioned, the traversals and addition of edges by Algorithm 1 precisely follow the description above.

Containment. Before proving properties of the unraveling of a query we recall how containment of two queries can be defined in terms of homomorphisms. We recall that a graph pattern query q is *contained* in another graph pattern query q', denoted by $q \subseteq q'$, if $q(\mathcal{G}) \subseteq q'(\mathcal{G})$, for all knowledge graphs $\mathcal{G}$. That is, the answer of q is always contained in the answer of q', independently of the underlying knowledge graph. While this notion reasons over all knowledge graphs, it admits a simple syntactic characterization based on homomorphisms.

A *homomorphism* from query $q(x)$ to query $q'(x)$ is a mapping h from the nodes of q to the nodes of q' such that:

- $h(x) = x$;
- if u is a node in q labeled by a constant a, then $h(u)$ is also labeled by a in q'; and
- For all edges $R(u, v)$ of q, we have that $R(h(u), h(v))$ is an edge of q'.

Intuitively, a homomorphism is a way of replacing the variables of q by variables of q' such that each edge of q becomes an edge of q'. The target variable of q must be mapped to the target variable of q'. The following is a well-known characterization of containment of graph pattern queries.

Theorem B.1 ([46])**.** A graph pattern query q is contained in a graph pattern query q' if and only if there is a homomorphism from q' to q. □

Proof of Proposition 3.1. We next provide the proof of Proposition 3.1.

▷ *Safety*

We start by showing that $\mathsf{UnRavL}_d(q)$ over-approximates q, that is, q is contained in $\mathsf{UnRavL}_d(q)$. By Theorem B.1 it suffices to show that there exists a homomorphism h from $\mathsf{UnRavL}_d(q)$ to q.

Consider the mapping h from the nodes of $\mathsf{UnRavL}_d(q)$ to the nodes of q that maps each valid path P in $\mathsf{UnRavL}_d(q)$ to its final element $\mathsf{end}(P)$. We claim that h is a homomorphism. Indeed, note first that the target variable of $\mathsf{UnRavL}_d(q)$ is mapped via h to the target variable x of q, as the former corresponds to the valid path x. Second, if a valid path P is labeled by a constant a, then $\mathsf{end}(P)$ is labeled by a by definition. In particular, $h(P) = \mathsf{end}(P)$ is labeled by a. Finally, take and edge $R(P, P')$ in $\mathsf{UnRavL}_d(q)$. By definition, we have two cases: either $P' = P, e', \mathsf{end}(P')$ or $P = P', e, \mathsf{end}(P)$. In the first case, we know that e' is an edge of the form $R(\mathsf{end}(P), \mathsf{end}(P'))$ in q. In particular, $R(h(P), h(P'))$ is an edge in q. In the second case, we know that e is an edge of the form $R(\mathsf{end}(P), \mathsf{end}(P'))$ in q. Again, we have that $R(h(P), h(P'))$ is an edge in q. We conclude that h is a homomorphism.

▷ *Conservativeness*

By Theorem B.1, it suffices to show that in case $q(x)$ is tree-like, we have a homomorphism from q to $\mathsf{UnRavL}_d(q)$, where d is the depth of q. This follows directly by observing that q is exactly $\mathsf{UnRavL}_d(q)$, modulo renaming of variables. Formally, we can define the "identity" homomorphism as follows. For each node u of q, we define $h(u)$ to be the unique path in q from the tree root to the node u. Note that this is actually a valid path and hence h is well-defined. By construction, h is clearly a homomorphism.

Proof of Proposition 3.2. We next provide the proof of Proposition 3.2.

We define a *path* of $q(x)$ as a valid path without the third condition in the definition. That is, in a path we can traverse consecutively the same edge.

As for valid paths, we denote by $\mathsf{end}(P)$ the node at the end of the path P. Note that every path P that is not valid define a unique valid path $\mathsf{valid}(P)$ such that $\mathsf{end}(P) = \mathsf{end}(\mathsf{valid}(P))$. Indeed, whenever we have a subsequence in the path P of form y, e, z, e, y violating the third condition, we can replace it by y. By iteratively, applying this simplification, we always obtain a unique valid path $\mathsf{valid}(P)$ with $\mathsf{end}(P) = \mathsf{end}(\mathsf{valid}(P))$.

Now suppose $q'(x)$ is a tree-like over-approximation of $q(x)$ of depth at most d. By Theorem B.1, there exists a homomorphism h from q' to q. We shall define a homomorphism g from q' to $\mathsf{UnRavL}_d(q)$, which implies $\mathsf{UnRavL}_d(q) \subseteq q'$ as required.

For a node w in q', we define the path P_w in q as follows. Take the unique path from the root x to w in q'. As h is a homomorphism, the image of this path via h produces a path in q, which we denote P_w. Note that P_w is actually a path as q does not have constants. (If q has constants, then P_w could have internal nodes in the sequence that are constants, violating the first condition of valid paths.)

Consider the mapping g from the nodes of q' to the nodes of $\mathsf{UnRavL}_d(q)$ defined as follows. For each node w in q', we assign $g(w) = \mathsf{valid}(P_w)$. Note that since the depth of q' is at most d, then the length of P_w is at most d, and consequently, $\mathsf{valid}(P_w)$ is a valid path of q of length at most d. Therefore, g is well-defined.

We claim that g is a homomorphism. Note that, by construction, the target variable of q' is mapped via g to the target variable of $\mathsf{UnRavL}_d(q)$. Now, take an edge $R(w, w')$ in q' and suppose w is the parent of w' in q' (the other case is analogous). Note that either $\mathsf{valid}(P_{w'})$ extends $\mathsf{valid}(P_w)$, that is, $\mathsf{valid}(P_{w'}) = \mathsf{valid}(P_w), e', z'$, or $\mathsf{valid}(P_w)$ extends $\mathsf{valid}(P_{w'})$. Indeed, if the last edge e' of $P_{w'}$ is different from the last edge of $\mathsf{valid}(P_w)$ then $\mathsf{valid}(P_{w'}) = \mathsf{valid}(P_w), e', z'$. Otherwise, if the last edge e' of $P_{w'}$ coincides with the last edge of $\mathsf{valid}(P_w)$, then $\mathsf{valid}(P_{w'})$ is the subsequence of $\mathsf{valid}(P_w)$ that ends just before traversing the last edge e' of $\mathsf{valid}(P_w)$. In particular, $\mathsf{valid}(P_w) = \mathsf{valid}(P_{w'}), e', z$. Suppose $\mathsf{valid}(P_{w'}) = \mathsf{valid}(P_w), e', z'$ (the other case is analogous). Note that $e' = R(h(w), h(w')) = R(\mathsf{end}(\mathsf{valid}(P_w)), \mathsf{end}(\mathsf{valid}(P_{w'})))$ and $z' = h(w')$. By construction of $\mathsf{UnRavL}_d(q)$, we have an edge $R(\mathsf{valid}(P_w), \mathsf{valid}(P_{w'}))$ in $\mathsf{UnRavL}_d(q)$. It follows that $R(g(w), g(w')) = R(\mathsf{valid}(P_w), \mathsf{valid}(P_{w'}))$ belongs to $\mathsf{UnRavL}_d(q)$ as required.

This concludes the proof. □

Proof of Proposition 3.3. We verify that $\mathsf{UnRavL}_d(q) \subseteq \mathsf{UnRavL}_{d-1}(q)$ using Theorem B.1. This follows from the fact that $\mathsf{UnRavL}_{d-1}(q)$ is a subtree of $\mathsf{UnRavL}_d(q)$. Hence we can consider the identity homomorphism that maps each valid path P in $\mathsf{UnRavL}_{d-1}(q)$ to itself. □

C PQE: A new probabilistic query answering baseline

Possible world semantics. A *probabilistic knowledge graph* $(\mathcal{G}, \omega)$ is a knowledge graph $\mathcal{G}$ in which ω assigns a probability $\omega_R(a, b)$ to each edge $R(a, b)$ in $\mathcal{G}$. The standard semantics of graph pattern queries (and for any query in any query language for that matter) is defined in terms of the *possible world semantics*. A *possible world* $\mathcal{W}$ *of* $\mathcal{G}$ is any subset of edges in $\mathcal{G}$. The probability $\mathbb{P}_\omega(\mathcal{W})$ of this world corresponds to

$$\prod_{R(a,b)\in\mathcal{W}} \omega_R(a, b) \cdot \prod_{R(a,b)\in\mathcal{G}\setminus\mathcal{W}} \big(1 - \omega_R(a, b)\big).$$

Table 6: Queries categorized in terms of being hierarchical (H) and/or tree-like.

Tree-like+H	Tree-like+not H	Not tree-like+H	Not tree-like+ not H
1p,2p,2i,3i,ip, ∃1p,∃2p,∃2i,∃3i,∃ip	3p, pi, ∃3p,∃pi	∃1p2c	∃3c,∃4c

That is, $\mathbb{P}_\omega(\mathcal{W})$ is the probability of hitting every edge in $\mathcal{W}$ (according to ω), and missing every edge not in $\mathcal{W}$. We refer to the book by Suciu et al. [47] for further details and background.

Query answering in this context corresponds to computing the probability of each possible world where a particular answer is valid. That is, the evaluation of a query q over $(\mathcal{G}, \omega)$ associates to every answer a the probability

$$\mathbb{P}(q, \mathcal{G}, \omega, a) := \sum_{\mathcal{W} \subseteq \mathcal{G}, a \in q(\mathcal{W})} \mathbb{P}_\omega(\mathcal{W}).$$

This semantics is widely accepted as *the standard* for probabilistic query answering. However, due to the exponential number of possible worlds, evaluating general queries using this method is $\#P$-hard [21, 48].

Hierarchical queries and dissociations. To cope with the intractability of computing the possible world semantics, previous work either identified subclasses of queries where evaluation is in PTIME, or constructed approximations.

For tractability, it was shown by Dalvi and Suciu [21] that PTIME evaluation is possible for so-called self-join-free hierarchical queries. Let $\mathsf{N}(q, x) := \mathsf{In}(q, x) \cup \mathsf{Out}(q, x)$, that is, the set of edges in q adjacent to x. A graph pattern query q is *hierarchical* if for each pair of non-target variables x, y in q, either $\mathsf{N}(q, x) \subseteq \mathsf{N}(q, y)$, or $\mathsf{N}(q, y) \subseteq \mathsf{N}(q, x)$, or $\mathsf{N}(q, x) \cap \mathsf{N}(q, y) = \emptyset$. If a relation type never occurs more than once in a query, then it is called self-join free.

For approximation, for queries that are not hierarchical, Gatterbauer and Suciu [22] developed an approximation technique. In a nutshell, they showed that every (self-join-free) query can be *upper bounded* by a finite set of hierarchical queries, called *dissociations*. In particular, they show that

$$\mathbb{P}(q, \mathcal{G}, \omega, a) \leq \mathbb{P}(\delta, \mathcal{G}, \omega, a)$$

for any dissociation δ of q. Gatterbauer and Suciu [22] propose to take $\min_{\delta \in \mathsf{Dis}(q)}\{\mathbb{P}(\delta, \mathcal{G}, \omega, a)\}$, with $\mathsf{Dis}(q)$ the finite set of dissociations of q, as PTIME-computable approximation of $\mathbb{P}(q, \mathcal{G}, \omega, a)$.

PQE.. In our new baseline PQE we use NBFNets to turn $\mathcal{G}$ into a probabilistic knowledge graph. More precisely, we define for each relation type $R \in \mathcal{R}$, $\omega_R(R(a, b)) := \mathcal{P}_R(\mathbf{1}_a)_b$ with $\mathcal{P}_R$ is the operator learned by an NBFNet. We store all $\mathcal{O}(|\mathcal{V}|^2)$ probabilistic edges.

For query evaluation, PQE does the following:

- If $q(x)$ is tree-like we perform a bottom-up computation, as shown in Algorithm 3. We here also cover tree-like queries with negation and disjunction. We remark that, if $q(x)$ is also hierarchical (also self-join-free, no negation, disjunction), then PQE actually computes the possible world semantics. Indeed, PQE coincides with the PTIME algorithm [21] for tree-like hierarchical queries
- If $q(x)$ is not tree-like but is hierarchical, we evaluate it according to Dalvi and Suciu [21]. In our cases, this only applies to the lollipop query (∃1p2c).
- Finally, if $q(x)$ is not-tree like and not hierarchical we compute its dissociations, evaluate these according to their PTIME evaluation algorithm, and take the minimal value, according to the dissociation-based approximation approach of Gatterbauer and Suciu [22].

Table 6 shows which of the queries used in our experiments satisfy the conditions of being tree-like and/or hierarchical, but without negation or disjunction.

The query patterns with disjunction (2u, up, ∃2u, ∃up) and negation (2in, 3in, inp, pni, pin, ∃2in, ∃3in, ∃inp, ∃pni, ∃pin) are also evaluated in a bottom-up fashion, similarly as GNN-QE, as explained in Algorithm 3. Dissociations are computed for all non-hierarchical queries. We remark that the maximal number of dissociations was five (for the 4c query).

We note that the PQE evaluation algorithm for tree-like queries uses $p \veebar q := p + q - pq$ for $p, q \in [0, 1]$ to encode disjunction and existential quantification.

Algorithm 3 Evaluation method of PQE for tree-like queries (extended with negation and disjunction). The algorithm uses the disjunction operation

Require: tree-like query $q(x)$, probabilistic knowledge graph $(\mathcal{G}, \omega)$.
Ensure: $\mathsf{score}_\omega(q, \mathcal{G})(e)$ for entity e

if $q(x)$ is of the form $R(a, x)$ (resp. $R(x, a)$) **then**
  return $\omega_R(a, e)$ (resp. $\omega_R(e, a)$).
end if
if $q(x)$ is of the form $R(y, x)$ (resp. $R(y, a)$) **then**
  return $\bigvee_{a \in \mathcal{V}} \omega_R(a, e)$ (resp. $\bigvee_{a \in \mathcal{V}} \omega_R(e, a)$).
end if
if $q(x)$ can be decomposed in two $q_1(x)$ and $q_2(x)$ with disjoint variable nodes (except for the target variable x **then**
  return $\mathsf{score}_\omega(q_1, \mathcal{G})(e) \cdot \mathsf{score}_\omega(q_1, \mathcal{G})(e)$.
end if
if $q(x)$ can be decomposed in a query $q_1(y)$ not containing x and and edge $R(y, x)$ (resp. $R(x, y)$) **then**
  return $\bigvee_{a \in \mathcal{V}} \mathsf{score}_\omega(q_1, \mathcal{G})(a) \omega_R(a, e)$ (resp. $\bigvee_{a \in \mathcal{V}} \mathsf{score}_\omega(q_1, \mathcal{G})(a) \omega_R(e, a)$.
end if
if $q(x)$ is of the form $\neg q_1(x)$ **then**
  return $1 - \mathsf{score}_\omega(q_1, \mathcal{G})(e)$.
end if
if $q(x)$ is of the form $q_1(x) \vee q_2(x)$ **then**
  return $\mathsf{score}_\omega(q_1, \mathcal{G})(e) \,\bigvee\, \mathsf{score}_\omega(q_2, \mathcal{G})(e)$.
end if

Table 7: Learning hyperparameters of NBFNet for PQE framework for FB15k237. The hyperparameters are kept as in the original NBFNet paper.

	Hyperparameter	FB15k-237
GNN	#layer	6
	hidden dim.	32
MLP	#layer	2
	hidden dim.	64
Batch	#positive	256
	#negative/#positive	32
Learning	optimizer	Adam
	learning rate	5e-3
	#epoch	20
	adv. temperature	0.5

D Experimental details

D.1 Implementation and training details

NBFNet. Table 7 shows the training hyperparameters used to train model NBFNet for PQE. The parameters are kept as in the original paper [15]. The best checkpoint for the model is selected based on the performance in the validation set, using MRR as the selection criteria.

GNN-QE.. Table 8 shows the training hyperparameters used to train model GNN-QE within our framework. The parameters are kept as in the original paper [11]. The best checkpoint for the model is selected based on the performance in the validation set, using MRR as the selection criteria.

UnRavL is trained using both the BetaE query set and unanchored BetaE query set but only on a subset of its queries. In particular, we train with 10 query types (1p/2p/3p/2i/3i/2in/3in/inp/pni/pin) and their unanchored counterparts. Later on, the model is evaluated on this 20 training query types, plus 8 query types that have never been seen during training (ip/pi/2u/up and their unanchored counterparts).

To train and evaluate we used a GPU Titan RTX 24.2 GB. We also provide the configuration files (.yaml) in the source code.

Table 8: Learning hyperparameters of GNN-QE in UnRavL framework for FB15k237, FB15k and NELL.

Hyperparameter	FB15k-237	FB15k	NELL
Batch size	24	24	6
Optimizer	Adam	Adam	Adam
Learning rate	5e-3	5e-3	5e-3
Adv. temperature	0.2	0.2	0.2
#Batches	10.000	10.000	18.000

Table 9: Results of evaluation of GNN-QE and UnRavL over original test queries (anchored tree-like) over FB15k-237, FB15k and NELL995. Metrics of GNN-QE and FIT are taken from their original papers. FIT does not report hits@1.

Metric	Model	1p	2p	3p	2i	3i	ip	pi	2in	3in	inp	pin	pni	2u	up
FB15k-237															
hits@1	GNN-QE	0.328	**0.082**	**0.065**	**0.277**	0.446	0.123	0.224	**0.041**	**0.081**	**0.041**	**0.025**	**0.027**	**0.098**	**0.076**
	UnRavL	**0.343**	0.055	0.018	**0.277**	**0.479**	**0.138**	**0.227**	0.016	0.074	0.037	0.017	0.013	0.087	0.058
mrr	GNN-QE	0.428	**0.147**	0.118	**0.383**	0.541	0.189	**0.311**	0.100	**0.168**	**0.093**	**0.072**	**0.078**	**0.162**	**0.134**
	FIT	**0.467**	0.146	**0.128**	0.375	0.516	**0.219**	0.301	**0.140**	**0.200**	0.102	0.095	-	**0.180**	0.131
	UnRavL	0.431	0.093	0.071	0.364	**0.595**	0.190	0.301	0.048	0.128	0.085	0.045	0.051	0.134	0.095
FB15k															
hits@1	GNN-QE	**0.861**	0.635	0.525	0.748	**0.801**	**0.651**	0.636	**0.354**	**0.331**	**0.338**	0.186	0.218	0.671	0.530
	UnRavL	0.831	**0.643**	**0.533**	**0.759**	0.791	0.566	**0.668**	0.308	0.297	0.315	**0.189**	**0.219**	**0.704**	**0.540**
	MPQE	0.258	0.061	0.060	0.094	0.121	0.081	0.096	-	-	-	-	-	-	-
mrr	GNN-QE	0.885	**0.693**	**0.587**	0.797	**0.835**	0.704	0.699	**0.447**	0.417	**0.420**	0.301	**0.343**	0.741	**0.610**
	FIT	**0.894**	0.656	0.569	0.791	**0.835**	**0.718**	**0.731**	0.402	0.389	0.348	0.281	-	0.739	0.590
	UnRavL	0.855	0.688	**0.587**	**0.801**	0.833	0.620	0.720	0.430	**0.418**	0.403	**0.302**	0.340	**0.747**	0.600
	MPQE	0.362	0.107	0.100	0.153	0.198	0.138	0.145	-	-	-	-	-	-	-
NELL995															
hits@1	GNN-QE	**0.861**	0.635	0.525	0.748	**0.801**	**0.651**	**0.636**	**0.354**	**0.331**	**0.338**	0.186	0.218	0.671	0.530
	UnRavL	0.831	**0.643**	**0.533**	**0.759**	0.791	0.566	0.668	0.308	0.297	0.315	**0.189**	**0.219**	**0.704**	**0.540**
	MPQE	0.191	0.032	0.046	0.066	0.114	0.031	0.056	-	-	-	-	-	-	-
mrr	GNN-QE	0.533	0.189	0.149	0.424	0.525	0.189	0.308	**0.159**	0.126	0.099	**0.146**	**0.114**	0.063	0.063
	FIT	**0.608**	**0.238**	**0.212**	**0.443**	**0.541**	**0.266**	**0.317**	0.126	**0.164**	**0.153**	0.083	-	**0.203**	**0.176**
	UnRavL	0.479	0.160	0.119	0.378	0.484	0.140	0.269	0.086	0.128	0.111	0.055	0.053	0.141	0.107
	MPQE	0.270	0.061	0.075	0.132	0.198	0.060	0.097	-	-	-	-	-	-	-

D.2 Message Passing Query Embedding

Our experiments with MPQE are based on StarQE [23], a method for answering tree-like queries on *hyper-relational* knowledge graphs, where edges can have *qualifiers* that provide additional information about a triple. When ignoring qualifiers, StarQE can be seen as an adaptation of MPQE that employs a CompGCN model [24] for encoding queries. We rely on the implementation of StarQE available online.[3]

D.3 The effect of training with general tree-like queries

One characteristic of our framework is that we extend current neuro-symbolic methods to deal with tree-like queries with and without anchors. In Tables 9 and 10 we report metrics over anchored and unanchored tree-like queries for the same model when including the later in the training set.

For anchored tree-like queries we compare against GNN-QE (see Table 9). The results are taken from the original paper . We note in general that UnRavL tends to inherit the peformance of GNN-QE to deal with anchored queries, with some exceptions where the performance slightly decay or increase.

For unanchored tree-like queries, in Table 10 we compare UnRavL with a-UnRavL that has the same architecture as UnRavL but that is trained only using anchored tree-like queries. As expected, when unanchored tree-like queries are included in the training set, the performance on such queries tend to increase.

[3] https://github.com/DimitrisAlivas/StarQE

Table 10: Comparison of results of evaluation of UnRavL over test set of unanchored queries when including (UnRavL) or not including (a-UnRavL) unanchored queries in the training phase.

Metric	Model	1p	2p	3p	2i	3i	ip	pi	2in	3in	inp	pin	pni	2u	up
FB15k-237															
hits@1	a-UnRavL	0.001	0.042	0.028	0.119	0.189	**0.060**	**0.104**	0.005	**0.057**	0.025	0.020	0.001	0.018	0.016
	UnRavL	**0.028**	**0.046**	**0.057**	**0.123**	**0.203**	0.030	0.090	**0.011**	0.055	**0.028**	**0.019**	**0.005**	**0.030**	**0.023**
	MPQE	0.012	0.041	0.035	0.021	0.027	0.030	0.028	-	-	-	-	-	-	-
mrr	a-UnRavL	0.012	0.081	0.063	0.206	0.286	**0.108**	**0.165**	0.024	**0.108**	0.059	0.044	0.019	0.040	0.049
	UnRavL	**0.055**	**0.088**	**0.079**	**0.212**	**0.292**	0.076	0.151	**0.029**	0.107	**0.061**	**0.045**	**0.027**	**0.065**	**0.053**
	MPQE	0.029	0.069	0.061	0.041	0.049	0.052	0.046	-	-	-	-	-	-	-
FB15k															
hits@1	a-UnRavL	0.075	0.315	0.387	0.552	0.620	0.511	0.470	0.054	0.178	0.162	0.074	0.050	0.084	0.347
	UnRavL	**0.171**	**0.353**	**0.426**	**0.628**	**0.661**	**0.523**	**0.511**	**0.103**	**0.209**	**0.181**	**0.094**	**0.081**	**0.209**	**0.373**
	MPQE	0.083	0.057	0.047	0.053	0.069	0.083	0.041	-	-	-	-	-	-	-
mrr	a-UnRavL	0.112	0.387	0.457	**0.685**	0.686	0.571	0.533	0.109	0.283	0.240	0.152	0.102	0.131	0.421
	UnRavL	**0.228**	**0.434**	**0.496**	0.613	**0.718**	**0.579**	**0.572**	**0.190**	**0.325**	**0.269**	**0.189**	**0.158**	**0.292**	**0.455**
	MPQE	0.148	0.092	0.084	0.094	0.115	0.126	0.074	-	-	-	-	-	-	-
NELL															
hits@1	a-UnRavL	0.004	0.022	0.033	0.130	0.183	0.057	0.082	0.066	0.021	0.025	0.004	0.004	0.005	**0.017**
	UnRavL	**0.010**	0.029	0.038	**0.142**	**0.190**	0.056	**0.084**	**0.097**	**0.022**	**0.027**	**0.006**	**0.006**	**0.012**	0.012
	MPQE	0.008	**0.034**	**0.050**	0.034	0.063	**0.061**	0.038	-	-	-	-	-	-	-
mrr	a-UnRavL	0.007	0.052	0.069	0.211	0.276	**0.110**	**0.139**	0.023	0.075	0.059	0.025	0.018	0.012	0.043
	UnRavL	**0.028**	**0.067**	**0.075**	**0.228**	**0.290**	0.107	0.137	**0.034**	**0.080**	**0.066**	**0.033**	**0.024**	**0.033**	**0.047**
	MPQE	0.023	0.056	0.075	**0.075**	0.125	0.082	0.073	-	-	-	-	-	-	-

Table 11: Results for cyclic queries considering the best unravelings in datasets FB15k and NELL995.

Metric	Model	∃1p2c	∃3c	∃4c
Cyclic queries FB15k				
mrr	UnRavL	0.098	**0.297**	0.159
	MPQE	**0.120**	0.116	0.000
hits@1	UnRavL	0.057	**0.092**	0.057
	MPQE	**0.064**	0.063	0.000
hits@10	UnRavL	**0.227**	**0.375**	0.302
	MPQE	0.225	0.218	0.000

Metric	Model	∃1p2c	∃3c	∃4c
Cyclic queries NELL995				
mrr	UnRavL	0.146	**0.211**	0.122
	MPQE	**0.150**	0.142	**0.180**
hits@1	UnRavL	0.065	**0.114**	0.067
	MPQE	**0.074**	0.060	**0.098**
hits@10	UnRavL	0.286	**0.419**	0.257
	MPQE	0.301	0.313	0.326

D.4 Results of cyclic queries for FB15k and NELL995

Table 11 presents the results for cyclic queries in datasets FB15k and NELL995.

This results further show that UnRavL is a performant framework to deal with complex queries.

D.5 Results on queries with negation and union

The ability of UnRavL to go beyond graph pattern queries and answering queries with negation and/or disjunction comes from the neuro-symbolic model underneath the framework, in this case GNN-QE.

As for the queries in the benchmarks query set that include negation and union are tree-like, there's no need to approximate them: GNN-QE can already answer them. However, when building UnRavL, we enhance GNN-QE to deal with unanchored queries, we can now present results of UnRavL in 7 new query types including negation or union and existential leaves. Table 13 show the results of UnRavL on those queries.

For completeness, we also address query type 2nm (see Figure 4). While the theoretical guarantees of our unravellings does not include negation, we can still perform the tree approximation. Results for this query on the three knowledge graphs are in Table 12.

D.6 Further results for FB15k-237

Unravelings depth. While the unraveling depth does not impact much on ranking metrics such as mrr and hits@1, when we look at more *global* metrics, such as precision and recall, some tendencies appear. We can see in Table 14 the results of UnRavL for the triangle and square queries when considering different depths and different classification thresholds.

Table 12: Results of UnRavL, a-UnRavL and FIT in query type **2nm** of the *real* EFO1 query set in FB15k-237, FB15k and NELL995

		FB15k237	FB15k	NELL995
Metric	**Model**	**2nm**	**2nm**	**2nm**
mrr	FIT	0.127	0.391	**0.125**
	UnRavL	**0.139**	**0.407**	0.118
	a-UnRavL	0.131	0.399	0.121

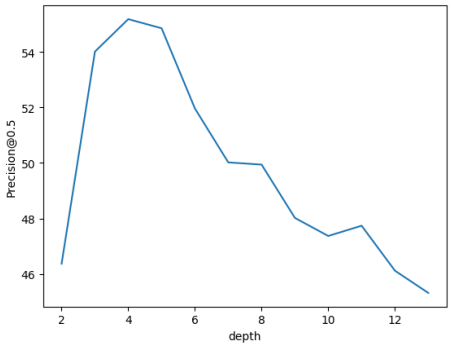


Figure 6: Precision using 0.5 classification threshold for different depths of unravelings for the triangle query (FB15k-237).

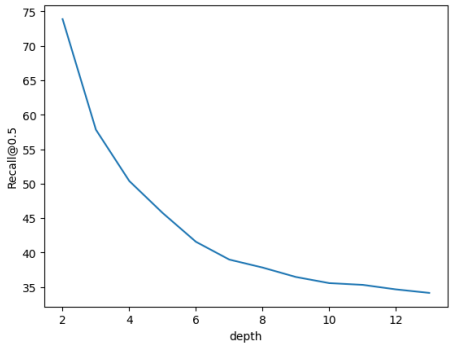


Figure 7: Recall using 0.5 classification threshold for different depths of unravelings for the triangle query (FB15k-237).

It can be noted that the best performance occurs at different depths for different thresholds. Moreover, each metric follows a different tendency:

- **Precision**: Looking at precision against the unravelings depth, we note that it starts growing, it peaks and then starts decaying. Figure 6 shows this for the triangle query with classification threshold 0.5. In this case, the peak occurs at depth $d = 0.4$
- **Recall**: In most cases, recall tends to peak with shorter depths and then starts decaying. Figure 7 shows this for the triangle query with classification threshold 0.5
- **Hard-Recall**: Recall, when only considering *hard answers*, maintains the same tendency as the general recall: it peaks with shorter unravelings and then starts decaying.

Table 13: Results of UnRavL, PQE and MPQE for unanchored tree-like queries on FB15k-237, FB15k, and NELL995.

Metric	**Model**	**∃1p**	**∃2p**	**∃3p**	**∃2i**	**∃3i**	**∃ip**	**∃pi**	**∃2in**	**∃3in**	**∃inp**	**∃pin**	**∃pni**	**∃2u**	**∃up**
General tree-like queries FB15k-237															
hits@1	UnRavL	**0.028**	**0.046**	**0.057**	**0.123**	**0.203**	**0.030**	**0.090**	**0.011**	**0.055**	**0.028**	**0.019**	**0.005**	**0.030**	**0.023**
	PQE	0.021	0.001	0.011	0.017	0.071	0.027	0.051	0.007	0.036	0.022	0.010	0.003	0.021	0.011
	MPQE	0.012	0.041	0.035	0.021	0.027	**0.030**	0.028	-	-	-	-	-	-	-
mrr	UnRavL	**0.055**	**0.088**	**0.079**	**0.212**	**0.292**	0.076	**0.151**	0.029	**0.107**	**0.061**	**0.045**	**0.027**	**0.065**	**0.053**
	PQE	0.031	0.025	0.051	0.034	0.111	**0.107**	0.064	**0.051**	0.057	**0.061**	0.035	**0.027**	0.032	0.029
	MPQE	0.029	0.069	0.061	0.041	0.049	0.052	0.046	-	-	-	-	-	-	-
FB15k															
hits@1	UnRavL	**0.831**	**0.643**	**0.533**	**0.759**	**0.791**	**0.566**	**0.668**	0.308	0.297	0.315	0.189	0.219	0.704	0.540
	MPQE	0.258	0.061	0.060	0.094	0.121	0.081	0.096	-	-	-	-	-	-	-
mrr	UnRavL	**0.855**	**0.688**	**0.587**	**0.801**	**0.833**	**0.620**	**0.720**	0.430	0.418	0.403	0.302	0.340	0.747	0.600
	MPQE	0.362	0.107	0.100	0.153	0.198	0.138	0.145	-	-	-	-	-	-	-
NELL995															
hits@1	UnRavL	**0.831**	**0.643**	**0.533**	**0.759**	**0.791**	**0.566**	**0.668**	0.308	0.297	0.315	0.189	0.219	0.704	0.540
	MPQE	0.191	0.032	0.046	0.066	0.114	0.031	0.056	-	-	-	-	-	-	-
mrr	UnRavL	**0.479**	**0.160**	**0.119**	**0.378**	**0.484**	**0.140**	**0.269**	0.086	0.128	0.111	0.055	0.053	0.141	0.107
	MPQE	0.270	0.061	0.075	0.132	0.198	0.060	0.097	-	-	-	-	-	-	-

Table 14: Precision and recall (%) for UnRavL at classification thresholds 0.3, 0.5, 0.7 for the triangle (unravellings depth 2 to 8) and square (unravellings depth 3 to 9) queries. For recall we consider both easy and hard answers (see Recall in table) and only hard answers (Hard-Recall)

T	Metric	Triangle query (3c)							Square Query (4c)						
		2	3	4	5	6	7	8	3	4	5	6	7	8	9
0.3	Precision	34.9	50.7	53.4	54.8	**55.2**	53.1	52.2	58.2	**61.6**	60.8	55.4	51.4	47.4	43.4
	Recall	**90.8**	82.1	73.2	66.9	61.7	58.0	55.3	**75.5**	60.7	46.5	36.7	30.6	26.8	24.9
	Hard-Recall	**66.3**	46.8	36.4	30.7	28.0	25.7	24.5	**33.8**	25.5	18.3	12.2	8.9	7.1	6.6
0.5	Precision	46.3	54.0	**55.1**	54.8	51.9	50.0	49.9	**48.7**	47.6	42.6	37.6	34.8	32.6	32.3
	Recall	**73.9**	57.8	50.3	45.7	41.5	38.9	37.8	**38.4**	22.6	17.5	15.2	14.0	13.4	13.0
	Hard-Recall	**43.2**	26.0	18.8	16.1	14.1	13.4	12.9	**13.1**	5.6	3.0	1.8	1.6	1.1	1.0
0.7	Precision	38.4	42.3	44.1	**45.0**	43.9	44.4	43.9	**30.5**	23.8	23.8	23.1	22.8	23.0	23.0
	Recall	**45.9**	31.2	27.7	26.6	25.2	24.8	24.7	**8.4**	6.4	6.3	6.4	6.4	6.3	6.3
	Hard-Recall	**25.2**	12.6	9.9	8.5	7.6	7.7	7.4	**0.7**	0.3	0.4	0.3	0.1	0.1	0.0

Table 15: Statistic of unanchored query set for each dataset: FB15k-237, FB15k and NELL.

	Split	∃1p	∃2p	∃3p	∃2i	∃3i	∃ip	∃pi	∃2in	∃3in	∃inp	∃pin	∃pni	∃2u	∃up
FB15k-237	train	474	13139	62826	117688	147721	11644	12790	5719	12286	13358	-	-	-	-
	valid	288	2514	4213	3763	3368	2272	4255	3330	4080	1970	4135	4396	2905	1613
	test	295	2475	4234	3792	3471	2259	4219	3272	3941	1975	3903	4312	2901	1604
FB15k	train	2690	51378	172943	231273	271760	22250	23759	11016	23337	25222	-	-	-	-
	valid	1182	5246	7481	6499	5072	3592	6908	5825	6506	3173	6620	7093	4916	2559
	test	1240	5302	7433	6527	5252	3558	6902	5815	6442	2979	6306	7125	4906	2565
NELL	train	400	8713	35045	50076	37010	4458	3015	1035	4218	5026	-	-	-	-
	valid	346	2118	3239	2731	2342	1721	3193	2643	2884	1410	2772	3421	2428	1249
	test	342	2050	3192	1596	897	474	1725	1568	1097	373	1542	1841	922	241

D.7 Query sets

In this paper we present two new query sets for knowledge graphs FB15k, FB15k-237 and NELL995.

General tree-like queries. We built a new set of general tree-like queries, these are tree-like queries that include existential leaves. We generate this query set by removing anchors for the queries in the original betaE query set. To denote existential quantified leaves, we use -5. For example. take query $Q(x) \leftarrow R(u, x)$. In betaE format this query would be (u, R). Let's say we remove the anchor u, then the query would be $Q(x) \leftarrow \exists y.R(y, x)$ and, in betaE format, $(-5, R)$.

Complete statistics for these queries can be found in Table 15

Cyclic queries. Table 16 show statistics for the unanchored cyclic queries test set.

Table 16: Statistic of cyclic query set for each dataset: FB15k-237, FB15k and NELL.

Dataset	2c	3c	4c
FB15k-237	1452	1047	909
FB15k	1500	4364	260
NELL995	1500	623	1952