Finding Rules that imply Facts based on a Knowledge Graph using a Relational Database

Tim Gutberlet and Janik Sauerbier

University of Mannheim, Mannheim, Germany
{tgutberl,jsauerbi}@mail.uni-mannheim.de

Abstract. Retrieving information from knowledge graphs is a highly relevant problem in academia and the industry. This includes among other things the knowledge graph completion/link prediction problem and rule-based pattern recognition. We focus on the problem of finding all rules that imply a certain target fact given a knowledge graph and a set of previously learned rules over the knowledge graph. Here we specifically analyze how relational database technologies perform in solving this problem, including a comparison of different indexing, hashing and pre-processing methods. Our research might help to advance approaches for link prediction and can be used to better understand the dynamics of rules given large amounts of target facts.

Keywords: Knowledge graph completion · Link Prediction · Knowledge representation and reasoning · Database design and models.

1 Introduction

Retrieving information from knowledge graphs has been a much-discussed topic in research over the years. Knowledge graphs are used in different fields from biomedical applications [OpenBioLink, HETIONET, Bio2RDF] over supply chain risk analysis to semantic search applications. Understanding patterns and rules as well as predicting links based on these patterns are relevant problems in the context of knowledge graphs in academia and industry. There are already many effective tools available that solve these problems, either as black box knowledge graph embedding models or through learning interpretable symbolic rules. We want to contribute to the field by analyzing on how to effectively find rules that imply certain target facts, which have not been part of the knowledge graph yet. We are not concerned with learning those rules, but just with identifying rules that have been learned previously and explain a certain target fact. Here, we specifically explore relational database technologies.

Many techniques are used to approach the link prediction problem, including various embeddings and rule-based methods. Prior research indicates that rule based approaches and embedding models can be used together to improve inference quality. This can be done by combining embedding models and constraints derived from rules as the objective function of an integer linear programming problem, or by employing other methods focused on combining the results of

different approaches. Another method is the use of rules during the training of embedding models. There is a significant need for a time-efficient database architecture which is used to identify rules that implicate the predictions of the embedding models.

Beyond potentially facilitating faster link prediction in the future, our findings can already be helpful to build tools that help to understand the dynamics of rules given large amounts of potential facts for knowledge graphs. Our research might also be helpful to understand the limits of relational database technologies in the context of knowledge graphs and might help to facilitate the comparison between different database technologies. [. . .]

To achieve those goals, we ran several experiments aimed to illustrate how different database structures impact the performance in terms of execution time using different indexing, hashing and pre-processing methods. Beyond that, we also analyzed how certain queries (aka. the target facts) and certain rule types (e.g., by length of the rule body) performed in comparison to each other.

2 Related Work

- Work on combining embeddings and rule-based approaches. (e.g. Wang, Quan, Bin Wang, and Li Guo. "Knowledge base completion using embeddings and rules."[3] or Zhang, Wen, et al. "Iteratively learning embeddings and rules for knowledge graph reasoning."[4])
- Work specifically focused on building effective databases (e.g. Indexing, Hashing and preprocessing strategies)

3 Preliminaries

Knowledge Graphs and Rules A knowledge graph (KG) is a set of triples (subject, relation, object) which describe a directed graph. The edges are represented by the relations going from subject to object. The nodes are the set of entities which appear as subjects and objects of the triples. The individual triples are also called facts. Knowledge graphs can be used to describe a variety of domains and can store knowledge about them. Possible domains are for example found in the biomedical field or in the context of the Semantic Web. Knowledge graphs can be used to identify rules and patterns in those domains. This can be very helpful to for example analyze how different biomedical processes correlate with certain genes, diseases, or treatments. For our purposes, we are concerned about first-order logic Horn rules. These rules consist of a head and a body, where the head consists of one triple and the body consists of one or more triples. The subjects and objects of the triples can either be specific entities or variables. Here are two example rules. We use the notation relation(subject, object) for the triples and capitalize the variables.

$$citizenOf(X, germany) \leftarrow bornIn(X, mannheim) \tag{1}$$

$$livesIn(X, germany) \leftarrow marriedTo(X, A_1), bornIn(A_1, germany) \quad (2)$$

The grounding of a rule is a set of triples that fulfil the rule. That means the knowledge graph contains triples with matching entities for all variables. The triple set livesIn(tom, germany), marriedTo(tom, bob), bornIn(bob, germany) would be a grounding for the second rule. Rules are not necessarily universally true, and there might be sets of triples that match the body, but there is no matching triple for the head. Based on the ratio of the number of groundings of a rule (matching head + body) and the number of matching bodies without a matching head, we can derive a confidence score for any given rule.

Knowledge Graph Completion As knowledge graphs are often based on real-world data, they are prone to be incomplete. Therefore, one very relevant problem in the context of knowledge graphs is link prediction/knowledge graph completion. The problem is concerned with identifying missing links (aka. missing facts) that might be true in the real world but are not yet part of the knowledge graph. One example would be the generation of friendship suggestions for users of social media sites based on known connections. A variety of black box and interpretable approaches have been used to tackle the problem. These include knowledge graph embedding models and rules-based approaches.

4 Problem statement

The problem we are trying to solve is identifying rules that imply certain target facts that are not part of the knowledge graph. There are various rule learners like AnyBURL available that can learn many rules even on big knowledge graphs in a short amount of time. That's why we are not concerned with learning those rules based on a given knowledge graph. Instead, we want to identify all matching rules for a certain target triple based on a previously learned set of rules. This means that the target triple should match the head of the rule and a subset of triples from the knowledge graph should match the body of the rule. We want to specifically explore in how far we can use traditional relational database technology to solve the problem as efficiently as possible. To illustrate the problem, think of the following example. We are given a knowledge graph, a target triple and a set of rules previously learned over the knowledge graph.

- **Knowledge Graph:**
  $\{\ldots, marriedTo(anna, peter), bornIn(peter, germany), \ldots\}$
- **Target Fact:**
  $livesIn(anna, germany)$
- **Rules:**
  1. $citizenOf(X, germany) \leftarrow bornIn(X, mannheim)$
  2. $livesIn(X, germany) \leftarrow marriedTo(X, A_1), bornIn(A_1, germany)$

Our goal is to identify all rules that imply the target fact. In our example we can start off with excluding rule one from further investigation as the target fact needs to match the head of the rule. Rule two could potentially imply the target fact with $X = anna$. To investigate whether rule two actually implies the target fact, we need to find matching facts for the rule body from the knowledge graph. The knowledge graph contains the facts $marriedTo(anna, peter)$ and $bornIn(peter, germany)$. Together with the target fact, this results in a grounding for the second rule with $X = anna$ and $A_1 = peter$. Therefor, the solution for the example problem is rule two.

5 Implementation

Summary Our tool was implemented using the open-source object-relational database system PostgreSQL. Before accepting any queries, we store the rules into a hash map to allow for easy pre-filtering of the rules. The given knowledge graph will be stored in tables of the relational database. After that, our implementation is ready to accept queries which each contain a single target fact. The result is the set of rules implying the target fact.

Data Model of the Rules Before processing any queries, our tool sorts the given set of rules into a hash map with five keys. The keys are based on the structure of the head of the rule.

1. **Subject Bound:** The head has a fixed entity as the subject and a variable as the object.
2. **Object Bound:** The head has a variable as the subject and a fixed entity as the object.
3. **Both Bound:** The head has a fixed entity as the subject and the object.
4. **Unbound and Unequal:** The head has a variable as the subject and another variable as the object.
5. **Unbound and Equal:** The head has a variable as the subject and the same variable as the object.

Within the respective categories, the rules are again sorted into a hash map by all fixed components (relation + potentially subject and object). This allows for initial filtering of the rules in O(1).

Data Model of the Knowledge Graph The knowledge graph consisting of $(subject, relation, object)$ triples is stored in a table with three columns for each element of the triple (column names: "sub" = subjects, "pre" = relations, "obj" = objects). We tested three different indexing approaches to optimize the execution time of the queries:

1. **Secondary Unclustered Index** on all three columns (sub, pre, obj)
2. **Unique Primary Clustered Index** on all three columns (sub, pre, obj)

3. **Secondary Unclustered Index** on the column combinations (pre), (pre, obj), (sub, pre) and (sub, pre, obj)

Beyond that, we created an alternative approach using materialized views (aka. additional tables) for each relation in the knowledge graph. These views only contain two columns for the subject and the object. Instead of checking in the overall knowledge graph table if a rule applies, we now check if a specific triple exists in its respective materialized view. We tested this alternative approach once without indexes and once with a Unique Unclustered Secondary Index on all columns (subject + object).

Queries The queries for our initial data model of the knowledge graph are built as follows. The basic idea is to check every possible rule that wasn't filtered out before. The SQL statement is an $UNIONALL$ operation over all subqueries for the individual rules. The SQL statement structure of the subquery is implemented as one $SELECT$ statement. N is the number of body triples of the checked rule and $rule\_id$ is a unique identifier for the checked rule. kg_table is the name of the table containing all triples of the knowledge graph.

*(**SELECT** $rule\_id_1$ **FROM** kg_table t0, kg_table t1, . . . , kg_table tN **WHERE** t0.sub = c0 **AND** t0.obj = c1 **AND** t1.sub = c2 **AND** t1.obj = c3 **AND** t0.obj = t1.sub . . . **AND** tN.sub = cN-1 and tN.obj = cN **AND** tN-1.obj = tN.sub **LIMIT** 1) **UNION ALL** (**SELECT** $rule\_id_2$ . . .) . . .*

For the first alternative setup using one materialized view per relation, the SQL statement is structured very similar. The only difference is lies in the fact that the table kg_table is now split up by relation. The table kg_table_{rX} contains all triples of the relevant relation at the respective position in the rule body.

*(**SELECT** $rule\_id_1$ **FROM** kg_table_{r1} t0, kg_table_{r2} t1, . . . , kg_table_{rN} tN **WHERE** t0.sub = c0 **AND** t0.obj = c1 **AND** t1.sub = c2 **AND** t1.obj = c3 **AND** t0.obj = t1.sub . . . **AND** tN.sub = cN-1 and tN.obj = cN **AND** tN-1.obj = tN.sub **LIMIT** 1) **UNION ALL** (**SELECT** $rule\_id_2$ . . .) . . .*

6 Experiments

For our experiments, we used YAGO3-10 as a benchmark dataset. The dataset was divided into a training dataset (1079040 triples), test dataset (5000 triples) and query dataset (5000 triples). We used the AnyBURL implementation for the bottom up rule-based learning approach learning rules for 50 seconds, only generating rules that do not have empty bodies (107301 Rules). To have an independent set of queries for which we want to test if they fire, we used the query dataset.

Our approach is written in Java and requires a PostgreSQL JDBC Driver installed. The source code and datasets, used in the experiments, can be found at

https://github.com/timgutberlet/Online-Rule-Search-Database. We conducted all experiments on a Fujitsu Esprimo P957 (construction year 2017) with 32 GB RAM, 512 GB SSD, an Intel i7-7700 @ 3.6 GHz CPU and Ubuntu 22.04.1 LTS as an operating system.

AnyBURL offers a similar feature for link prediction, which we will use to compare our optimisation features with. For the Comparison with AnyBURL given the limitation of AnyBURL to only find the following rule types:

$$h(X, Y) \leftarrow r(X, Y) \tag{3}$$

$$h(X, Y) \leftarrow r_1(X, Z) \wedge r_2(Z, Y) \tag{4}$$

$$h(X, e_1) \leftarrow r(X, e_2) \tag{5}$$

Using the given rule set and filtering it by these easy rule types, this leaves 19271 Rules from the original 106480 rules. As we have identified, our approach of [...] is most efficient for rule sets that only contain easy rule types, like in the AnyBURL example. Testing the approach of AnyBURL it takes on average 11 ms per query to find all rules, if there are any, that determine if the query fires. For our approach, it takes on average 3 ms. To find out the performance influences of different optimisation features, we deactivated each feature on an isolated level and looked at how this worsened performance. [...]

However, to show the relevance of our approach, we also tested the performance of our optimisation strategies, without limiting the rule types in any sense.

Conclusions / Assumptions: Yago3 - 10 has relatively low amount of 37 different relations. Working with datasets with a higher count of relations could possibly improve the approach of using materialized views for every relation, as the knowledgegraph would be split even more fine-grained.

- XXX
- XXX

7 Conclusions

Acknowledgements ...

References

1. Betz, Patrick, Christian Meilicke, and Heiner Stuckenschmidt. "Adversarial explanations for knowledge graph embeddings." International Joint Conferences on Artificial Intelligence, 2022.
2. Mayer, Marta Cialdea, and Fiora Pirri. "First order abduction via tableau and sequent calculi." Logic Journal of the IGPL 1.1 (1993): 99-117.
3. Wang, Quan, Bin Wang, and Li Guo. "Knowledge base completion using embeddings and rules." Twenty-fourth international joint conference on artificial intelligence. 2015.
4. Zhang, Wen, et al. "Iteratively learning embeddings and rules for knowledge graph reasoning." The World Wide Web Conference. 2019.