From Facts to Rules using Relational Databases

Tim Gutberlet[1], Janik Sauerbier[2]

Abstract: In this paper, 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. We specifically analyze how relational database technology performs in solving this problem using indexing, filtering and pre-processing methods. Our research might help to advance approaches for link prediction and can be used to efficiently explain the dynamics of rules given large amounts of target facts or large knowledge graphs.

Keywords: Knowledge graph completion; Link prediction; Knowledge representation and reasoning; Relational databases.

1 Introduction

Knowledge graphs (KGs) are used in different fields from biomedical applications [Br20] over applications in the context of social networks [Wa18] to semantic search applications[BBH16]. We want to contribute by analyzing how to effectively check which rules of a KG imply certain target facts using relational database technology. For the rule learning, we use AnyBURL, a fast bottom up rule learner for KGs.[Me19]

Our research might be used as a primitive for the link prediction problem. Prior research indicates that rule-based approaches and embedding models can be used in combination to improve inference quality.[Me18][Gu18] There is a significant need for a time-efficient solution that is used to check which rules implicate certain facts. Beyond potentially facilitating faster link prediction, our findings can already be helpful to explain the dynamics of rules given large amounts of potential facts for KGs. It might also particularly help in case of large knowledge graphs and memory constraints.

To achieve those goals, we ran several experiments aimed to illustrate how different database setups impact the performance in terms of execution time using filtering, indexing and pre-processing methods. Beyond that, we also performed a quantile performance analysis on the queries and rules.

[1] University of Mannheim, tgutberl@mail.uni-mannheim.de
[2] University of Mannheim, jsauerbi@mail.uni-mannheim.de

2 Preliminaries

A knowledge graph is a set of triples (subject, relation, object) that 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. KGs can be used to describe various domains and to identify rules and patterns in those domains. 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.

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

$$livesIn(X, germany) \leftarrow marriedTo(X, A_1), bornIn(A_1, germany) \tag{2}$$

The grounding of a rule is a set of triples of the KG that fulfil the rule, 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.

3 Problem Statement

The problem we are trying to solve is identifying rules that imply certain target facts which are not part of the KG based on the KG and a previously learned set of rules. This means we are looking for a grounding of the rule with the target triple matching the head of the rule and a subset of triples from the KG matching the body of the rule. To illustrate the problem, think of the following example.

- **Knowledge Graph:**

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

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 has a matching head 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 KG. The KG 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 $A_1 = peter$. Therefore, rule two is part of the solution.

4 Implementation

Our solution was implemented using the open-source object-relational database system PostgreSQL. The basic idea behind our implementation is creating SQL queries for each target fact. The given KG will be stored in one table of the relational database (columns: “sub” for subjects, “pre” for relations and “obj” for objects). The SQL statement consists of *UNION ALL* operations over all sub-queries for the individual rules. The rules are saved in a list and in a loop over all rules, the sub-queries for rules with matching heads are created. To improve this basic idea, we employed several optimizations listed below. This would be the query for the example given in the problem statement:

*(**SELECT** rule_1 **FROM** kg_table t0, kg_table t1 **WHERE** k0.sub = anna **AND** k0.obj = k1.sub **AND** k1.obj = germany **LIMIT** 1) **UNION ALL** (...)*

4.1 Hash Maps for Rule Pre-filtering

To speed up the pre-filtering of the rules, we sorted the given set of rules into a hash map with five keys based on the rule head type. Three of the head types have fixed entities, either as subject and/or as object [e.g., *r1(e1, X)*, *r1(X, e1)* or *r1(e1, e2)*]. Two of the head types have a variable as subject and as object, either the same variable or different variables [e.g., *r1(X, X)*, *r1(X, Y)*]. Within the respective categories, the rules are again sorted into hash maps based on the relation and the fixed entities. This allows for initial filtering of the rules in O(1).

4.2 Tables for each Relation

We created an alternative approach to having one big table for all facts of the KG. This approach uses one table for each relation in the KG therefore decreasing the individual table size. They only contain two columns for the subject and the object. This solution improved the overall performance by reducing the size of the B-trees of the indexes.

4.3 Indexing

To speed up the joining of the tables, we duplicated the knowledge graph tables and employed a unique clustered index, once with (subject, object) as key and once with (object, subject) as key. Within the SQL statement, the tables with the order (subject, object) are used in case of a fixed subject and the tables with the order (object, subject) are used in the case of a fixed object or no fixed subject and object. This ensures that an index is used as often as possible. The index speeds up the search and enables the direct access to the table contents through the B-trees.

4.4 Pre-processing of expensive Rules

As we will illustrate in our experiments, certain rules are way more expensive than others. To address that, we integrated a simple pre-processing method. We gather an independent set of n facts and used them as target facts. The queries were amended with the *EXPLAIN ANALYZE* command to get the execution time for the individual rule sub-queries. Afterwards, we ranked all rules by the sum of the execution time over all queries. The top x% of rules were then pre-processed, by calculating all potential rule heads which could form a grounding together with a set of triples from the KG as rule body. We excluded rules with one body triple, as pre-processing didn't improve the performance. We also excluded rules with more than two body triples, as those rules incur a pre-processing time two orders of magnitude higher than rules with two body triples, while they only make up a small fraction of the most expensive and frequent rules.

5 Experiments

The goal of our experiments was to benchmark the performance of our solution and to measure the effects of different optimizations. We also analyzed the execution time for individual queries and rules. Beyond that, we tested our solution on different datasets and rulesets. Our solution is written in Java using the PostgreSQL JDBC driver. The source code and datasets, used in the experiments, can be found at https://github.com/timgutberlet/From-Facts-to-Rules-using-Relational-Databases. 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.

5.1 Ablation Study - Optimizations

Experiment	Avg. Execution Time
All optimizations enabled	10 ms
Hash Maps for Rule Pre-filtering disabled	XX ms
Tables for each relation disabled	140 ms
Indexing disabled	1680 ms
Pre-processing of expensive rules disabled	11 ms

Tab. 1: Ablation Study of Optimizations using YAGO3-10

For these experiments, we used YAGO3-10 as the benchmark dataset.[MBS14] The dataset consists of a training set (1079040 triples), test set and validation set (5000 triples each). We used AnyBURL to learn rules for 50 seconds, only generating rules that do not have empty bodies (107301 Rules). We used the training set as the KG and the validation set as the target triples. For the *Pre-processing of expensive Rules* optimization, we use the test set (n = 5000) to create the rule ranking and pre-process the top 1% rules. The results in Tab. 1 are based on executing all 5000 queries.

The *Tables for each Relation* optimization specifically reduced the execution time for long rules. For rules with one body triple, the execution time was reduced by 12%, for two body triples by 25% and for three body triples by 74%. This is caused by the increased use of the indexes during execution.

5.2 Quantile Performance Analysis

Fig. 1: Query quantile analysis (All optimizations enabled)

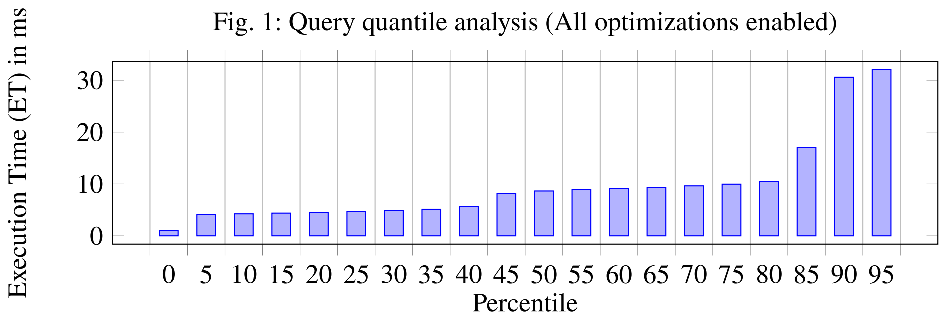
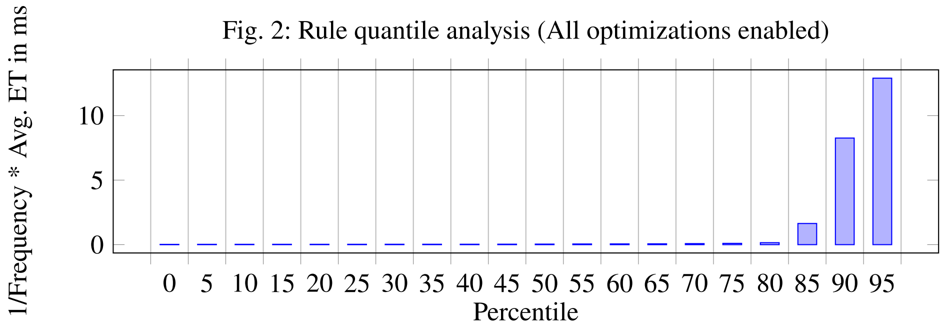


Fig. 2: Rule quantile analysis (All optimizations enabled)



The analysis indicates that only few rules and few queries incur a major share of the cost. The *Pre-processing of expensive Rules* optimization helped to reduce the cost for very expensive rules. The execution time for a specific rule is influenced by the length of the rule, as well as the target triple and the structure of the knowledge graph.

5.3 Further Datasets and Rulesets

Our solution works for different datasets and rulesets learned by AnyBURL in the given time. The solution is particularly well suited for datasets with many relations, like the Freebase dataset, as it allows for small B-trees of the indexes.

	AnyBURL - 10s	AnyBURL - 50s	AnyBURL - 100s
YAGO3-10	XX ms	10 ms	XX ms
WN18RR	XX ms	XX ms	XX ms
FB15k-237	XX ms	XX ms	XX ms

Tab. 2: Average Execution Time for different Datasets and Rulesets

6 Conclusions

We demonstrated an effective solution for finding all rules that imply a certain target fact given a knowledge graph and a set of previously learned rules. Our experiments specifically demonstrated the effect of filtering, indexing and pre-processing methods. Potential next steps include a further analysis of our solution on various datasets, a comparison of different database technologies (particularly triplestores), exploring the use of multithreading and creating a dedicated solution only using the main memory.

Acknowledgement. This paper would not have been possible without the exceptional support of our supervisor, Prof. Dr. Rainer Gemulla.

Bibliography

[BBH16] Bast, Hannah; Buchhold, Björn; Haussmann, Elmar: Semantic Search on Text and Knowledge Bases. Foundations and Trends® in Information Retrieval, 10(2-3):119–271, 2016.

[Br20] Breit, Anna; Ott, Simon; Agibetov, Asan; Samwald, Matthias: OpenBioLink: a benchmarking framework for large-scale biomedical link prediction. Bioinformatics, 36(13):4097–4098, 2020.

[Gu18] Guo, Shu; Wang, Quan; Wang, Lihong; Wang, Bin; Guo, Li: Knowledge Graph Embedding With Iterative Guidance From Soft Rules. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1), 2018.

[MBS14] Mahdisoltani, Farzaneh; Biega, Joanna; Suchanek, Fabian: Yago3: A knowledge base from multilingual wikipedias. In: 7th biennial conference on innovative data systems research. CIDR Conference, 2014.

[Me18] Meilicke, Christian; Fink, Manuel; Wang, Yanjie; Ruffinelli, Daniel; Gemulla, Rainer; Stuckenschmidt, Heiner: Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion. In: The Semantic Web – ISWC 2018. pp. 3–20, 2018.

[Me19] Meilicke, Christian; Chekol, Melisachew Wudage; Ruffinelli, Daniel; Stuckenschmidt, Heiner: Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI'19. pp. 3137–3143, 2019.

[Wa18] Wang, Zhouxia; Chen, Tianshui; Ren, Jimmy; Yu, Weihao; Cheng, Hui; Lin, Liang: Deep Reasoning with Knowledge Graph for Social Relationship Understanding. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. IJCAI'18, p. 1021–1028, 2018.