K.-U. Sattler et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2023),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 1

From Facts to Rules using Relational Databases

Tim Gutberlet[1], Janik Sauerbier[2]

Abstract: Retrieving information from knowledge graphs is a highly relevant problem in various fields. This can for example be achieved by learning and applying rules based on the knowledge graphs. In this paper, we focus on the problem of identifying all rules that entail a certain target fact given a knowledge graph and a set of previously learned rules. We specifically aimed to create and analyze an efficient approach using relational database technology including indexing, filtering and pre-computing methods. This problem is relevant in the context of link prediction and explainability. Our experiments demonstrated the efficacy of our approach and the effect of various optimizations on different datasets like YAGO3-10, WN18RR and FB15k-237 using rules learned by the bottom up rule learner AnyBURL.

Keywords: Knowledge graphs; Relational databases; Link prediction; Explainability.

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]. Learned rules over the KG describe patterns of the KG and allow for knowledge inference. This can can for example be used to predict missing or additional information also known as the link prediction problem.

Identifying which previously learned rules of a KG entail certain target facts is an important problem for two main reasons. Firstly, it is relevant in the context of 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 approach which is used to check which rules entail certain facts in that context. Secondly, it is generally helpful to explain the connections between rules and (potential) facts.

We want to contribute by creating and analyzing an effective approach to this problem using relational database technology. To achieve those goals, we ran several experiments aimed to illustrate the performance of our approach on different KGs (YAGO3-10, WN18RR and FB15k-237). Moreover, we intend to show how different database setups impact the performance in terms of execution time using filtering, indexing and pre-computing methods. For the rule learning, we use AnyBURL, a fast bottom up rule learner for KGs.[Me19]

[1] University of Mannheim, tim.gutberlet@students.uni-mannheim.de
[2] University of Mannheim, janik.sauerbier@students.uni-mannheim.de

2 Preliminaries

A KG is a set of (subject, relation, object)-triples also called facts. There is a set of entities present as subjects and objects and a set of relations in a KG. Here is an example KG with the entities $peter$, $anna$ and $germany$ and the relations $livesIn$ and $marriedTo$.

$$\textbf{KG} = \{marriedTo(anna, peter), bornIn(peter, germany), \dots\}$$

For our purposes, we are concerned about first-order logic Horn rules, which describe patterns of KGs. Here are two example rules. We capitalize the variables and lowercase the constants representing entities.

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

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

These have a head (e.g., $citizenOf(X, germany)$) consisting of one relation and a body (e.g., $bornIn(X, mannheim)$) consisting of one or more relations. A grounding of a rule assigns values to all variables of the rule, resulting in a ground term. The problem we are trying to solve is identifying rules that entail 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 interested in groundings where all body atoms exist in the KG and the head atom matches the target fact. To illustrate the problem, think of the following target fact.

$$\textbf{Target fact} = livesIn(anna, germany)$$

In our example we can start off with excluding rule (1) from further investigation as the target fact needs to match the head of the rule. Rule (2) has a matching head with $X = anna$ and entails the target fact if $\exists A_1(marriedTo(anna, A_1) \wedge bornIn(A_1, germany))$.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 (2) is part of the solution.

3 Proposed Approach

Our approach was implemented using the open-source object-relational database system PostgreSQL. The given KG is stored in one table of the relational database (*kg_table* with columns “sub” for subjects, “rel” for relations and “obj” for objects). The basic idea behind our approach is creating SQL queries which check whether a certain rule entails a certain

target fact. Those individual queries are then combined using *UNION ALL* operations over all rules, where the head matches the target fact. To improve this basic idea, we employed several optimizations listed below. This would be the query for rule (2) and the target fact given in the preliminaries:

SELECT *rule_1* ***FROM*** *kg_table t0, kg_table t1* ***WHERE*** *k0.sub = anna* ***AND*** *k0.rel = marriedTo* ***AND*** *k0.obj = k1.sub* ***AND*** *k1.rel = bornIn* ***AND*** *k1.obj = germany* ***LIMIT*** *1*

3.1 Advanced 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 one or two fixed constants [e.g., *r1(c1, X)*, *r1(X, c1)* or *r1(c1, c2)*]. Two of the head types have two variables, 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 constants. This allows for initial filtering of the rules in O(1).

3.2 Database Structure

Alternative to having one big table for all facts, our approach uses one table for each relation in the KG, with two columns for the subject and the object. This improved the performance by reducing the size of the B-trees of the indexes described in the following paragraph.

To speed up the search and enable direct access to the table contents through the B-trees, we employed unique clustered indexes. We duplicated the knowledge graph tables and used once (subject, object) as key and once (object, subject) as key for the indexes. Within the generated SQL statements, the tables with the order (subject, object) are used in case of a fixed subject. 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 the indexes are used efficiently.

3.3 Pre-computing of Expensive Rules

As we will illustrate in our experiments, certain rules result in way more cost in terms of execution time than others. To address that, we pre-computed a specific portion of those rules. To identify the most expensive rules, we gathered an independent set of n target facts from the KG and ran their queries with the *EXPLAIN ANALYZE* command to get the execution time for the individual rule sub-queries. Afterwards, we ranked all rules which appeared in the results by their average execution time multiplied with their number of appearances in the results. The top x% of rules were then pre-computed, by calculating all potential assignments for variables in the rule head that form a grounding together with a

	27k rules	107k rules	167k rules	#entities	#relations	#facts of KG
YAGO3-10	4 ms	10 ms	14 ms	123,182	37	1,079,040
WN18RR	35 ms	65 ms	77 ms	40,943	11	86,835
FB15k-237	3 ms	8 ms	11 ms	14,505	237	272,115

Tab. 1: Performance results (average execution time per target fact)

set of facts from the KG. We excluded rules with one body atom, as pre-computing didn't improve their performance. Moreover, we excluded rules with more than two body atoms, as those rules incur a pre-computing time two orders of magnitude higher than rules with two body atoms, while they only make up a small fraction of the most expensive and frequent rules.

4 Experiments

The goal of our experiments was to benchmark the performance of our approach on different datasets and rulesets, as well as to measure the effects of different optimizations. Beyond that, we analyzed the execution time for individual rules. The implementation of our approach was written in Java using the PostgreSQL JDBC driver. The source code and datasets, used for 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. The rule learning with AnyBURL was mostly done with the standard configuration of AnyBURL-22 available at https://web.informatik.uni-mannheim.de/AnyBURL/. We only extended the limit of body atoms from one to three for acyclic rules to match the cyclic rules, as we do not intent to discriminate between them.

4.1 Overall Performance

As illustrated in Tab. 1, our approach works for different datasets [MBS14][De18][TC15] and rulesets learned by AnyBURL in 10s (27k rules), 50s (107k rules) and 100s (167k rules). We used the training sets as the KGs and the test sets as the target triples (3k-20k triples). The validation sets (3k-20k triples) were used as target triples to create the rule rankings to pre-compute the top 1% most expensive rules. Our approach is particularly well suited for datasets with many relations, like the Freebase dataset, as it allows more tables with smaller B-trees of the indexes.

Experiment	Avg. execution time
All optimizations enabled	10 ms
Advanced rule pre-filtering disabled	55 ms
Tables for each relation disabled	133 ms
Indexing disabled	1720 ms
Pre-computing of expensive rules disabled	16 ms

Tab. 2: Ablation study of optimizations using YAGO3-10 & 107k rules

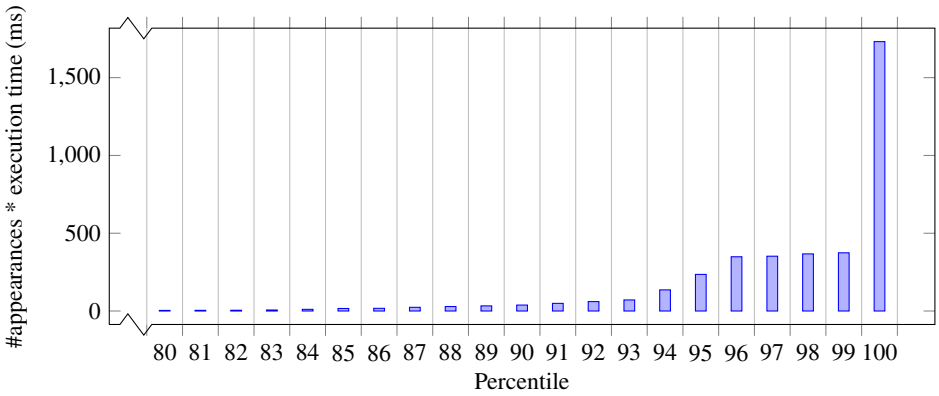

Fig. 1: Quantile performance analysis of rules using YAGO3-10 & 107k rules

4.2 Ablation Study - Optimizations

For the ablation study in Tab. 2, we used YAGO3-10 as the benchmark dataset and the AnyBURL 50s ruleset (107k rules). The "tables for each relation" optimization specifically reduced the execution time for long rules. For rules with one body atom, the execution time was reduced by 12%, for two body atoms by 25% and for three body atoms by 74%. This is caused by the increased use of the indexes during execution for longer rules.

4.3 Rule Quantile Performance Analysis

The quantile analysis in Fig. 1 is based on the number of appearances multiplied with the average execution of the rules after using the validation set as target triples as described in 3.3. It indicates that only few rules incur a major share of the cost. This ranking is used to determine the top x% for the "pre-computing of expensive rules" optimization. In the given example (YAGO3-10 & 107k rules), we achieved an 98.8% execution time reduction for the pre-computed top 1% rules. The pre-compution took five minutes. This reduced our average execution time from 16 ms to 10 ms, as illustrated in Tab. 2.

5 Conclusions

We demonstrated an effective approach 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-computing methods. Potential next steps include a further analysis of our approach 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.

[De18] Dettmers, Tim; Minervini, Pasquale; Stenetorp, Pontus; Riedel, Sebastian: Convolutional 2d knowledge graph embeddings. In: Proceedings of the AAAI conference on artificial intelligence. volume 32, 2018.

[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.

[TC15] Toutanova, Kristina; Chen, Danqi: Observed versus latent features for knowledge base and text inference. In: Proceedings of the 3rd workshop on continuous vector space models and their compositionality. pp. 57–66, 2015.

[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.