KnowledgeLog: A Datalog Inference Engine for Knowledge Base Reasoning

Yahn-Chung Chen, Paul Luh, and Han Wang

Department of Computer Sciences, University of Wisconsin-Madison

November 29, 2020

1 Introduction

Recent success in information extraction leads to advances in large-scale knowledge base construction such as DBpedia [1], DeepDive [2], Freebase [3], Google knowledge Graph [4] and YAGO [5]. These knowledge bases store information about persons, locations, events, organizations and the relations between them typically in the format of `<subject, relation, object>`. The wealth of information in large-scale knowledge provides tremendous values for downstream machine learning application such as question answering [6], information extraction [7], distant supervision [8] and etc. In this paper, we focus on the problem of knowledge base reasoning. Reasoning over these knowledge bases is an important task because most of the knowledge bases are incomplete. In particular, 60% of person entities miss a place of birth and 58% of the scientists are not linked to their seminal work in DBpedia [9]. In Freebase, 71% of the person entities miss a place of birth and 94% have no facts about their parents [10].

Despite the absent information in knowledge bases, many ontology rules are available to us. Therefore, instead of filling in information by repeatedly extracting information from sources we can take advantage of these ontology rules to expand our knowledge bases. For example, suppose our knowledge base contains the facts "Barack Obama studied at Columbia University" and "Columbia University is located in New York City", then we can conclude the fact that "Barack Obama lived in New York City" although it is not present in the original knowledge base. A more complete knowledge base is crucial for several downstream machine learning applications as mentioned previously such as question answering, distant supervision, and information extraction.

We propose KnowledgeLog[1], an open-source system that enables users to query facts that are not directly stored in the existing knowledge bases but are available through reasoning over ontology rules by taking advantage of an out-of-the-box datalog evaluation engine. KnowledgeLog aims to achieve the following three goals (1) stores and queries large-scale knowledge bases and ontology in an efficient manner (2) retrieves facts not only based on the existing dataset but also indirect fact assertion (3) assigns scores to the resulting facts and prunes out facts with low confidence. Even though there are some existing engines that support datalog evaluation, applying ontology rules with confidence and support to extend the results that user can retrieve is still an uncommon feature. For example, `pyDatalog` [2] is a package written in python that supports evaluating datalog program but fails to utilize probabilistic ontology rules and conclude facts and their associated rankings derived from confidence and support of the ontology rules.

The rest of this paper is organized as follows. Section 2 discussed related work and section 3 describes the architecture design and implementation details and algorithms we used in each component of KnowledgeLog. Section 4 then contains experiments and interesting results of our approach before Section **??** & 6 conclude.

2 Related Work

Knowledge base completion. State-of-the-art methods of knowledge base completion typically operate on the latent representation of the entities and relations in knowledge bases. These methods first require learned representations of the components in the knowledge base usually from neural networks; then the algorithms will use these representations to perform inference in tasks such as knowledge base completion and link prediction. Typical embedding methods include TransE [11] which aims to minimize the re-construction loss while learning the representation. Another interesting method pro-

[1] https://github.com/paul841029/KnowledgeLog/releases/tag/1.0

[2] https://github.com/pcarbonn/pyDatalog

posed in [12] learns representation not only by the data in knowledge bases but also by ontology rules defined elsewhere.

Rule mining in knowledge bases. Our system utilizes existing ontology rules. These rules are obtained through efficient rule mining algorithms such as methods described in [13]. The rule mining algorithms aim to discover Horn rules and their associated confidence and support from existing knowledge bases. The chief technical challenge of these approaches is the absence of counterexample, an essential input for traditional inductive logic programming (ILP) algorithms. In this project, our system does not mine rules from knowledge bases; instead, we take as input the rules mined by existing frameworks and perform inference over them.

Markov logic network [14] is a set of pairs (F_i, w_i) where F_i is a first-order logic formula and w_i is its associated weight and a finite set of constants. One constructs a Markov Logic Network by grounding all atoms with its appropriate domain and creates edges between atoms together appearing in a single grounded formula. Due to the potential large network size, an approximate inference for an assignment over the network is obtained typically through MCMC over the minimal subset of the ground network essential to answering the query.

3 Architecture

The KnowledgeLog includes the following components: (1) **Rules aggregator** is an application for processing and retrieving rules from an ontology rules database. It can retrieve rules related to the user-provided query and send the relevant rules to the result ranking engine. (2) **Result ranking engine** is where the algorithms and metrics be applied to select rules for execution. With rules and confidence from rules aggregator and the data retrieved from the Fuseki Server, the result ranking engine can decide the best order for executing rules and recalculating rank for each fact. (3) **Apache Jena Fusek** [15] is a SPARQL server. It is tightly integrated with Jena TDB RDF triple store to provide a persistent storage layer, and supports a RESTful API for Jena text query. In Figure 1, the architecture of KnowledgeLog is displayed to demonstrate the flow of query processing.

3.1 Rule Aggregator

Ontology rules are the key components for KnowledgeLog to extend the knowledge it has. By defining the ontology rule between predicates, the direct and indirect causation can be built by either applying stand-alone rule or bundle rules. However, since the bundle rules can include giant amount of rules and predicates which may run out of the memory on KnowledgeLog, a level-order traversal is applied as the selection process to pick up rules relating to user query and prune others with low confidence.

No.	Rules	Conf. score
1	notableWork <= writer	0.4734
2	notableWork <= author	0.5991
3	notableWork <= creator & series	0.6116
4	writer <= director	0.4275
5	creator <= executiveProducer	0.4032

Table 1: Table 1. The predicate `notableWork` can be directly derived from `writer`, `author`, `creator`, `series`, and it can also indirectly derived from `director` and `executiveProducer`. If the depth for level-order traversal is set to be 1, then only rule 1, 2 and 3 will be chosen.

In the table 1, it is shown that the predicate, `notableWork`, can not only be derived from rules 1, 2, and 3, but also indirectly derived from rules 4 and 5, because `creator` and `writer` are respective in the body of rule 1 and 3. To deal with both direct and indirect rules, a tree structure is introduced to capture this property by assigned the head of the rule (`notableWork`) as a parent node and the body of rule (`writer`, `author`, `creator`, `series`) as children nodes, where the distance between nodes is used to determine whether a rule should be considered while querying. On top of that, when there is a rule with low confidence, this rule is most likely to be false and had better to be pruned out during the selection process

3.2 Result Ranking Engine

Result ranking engine is the key component that gives rank to each data tuple, and it is also what makes KnowledgeLog different from existing datalog system. Result ranking engine is an intermediate layer that integrates rule aggregator, pyDatalog Engine and Jena Fuseki Server. It can determine the best order for rule execution, relocate data tuples based on the ontology rules, and also assign each of data tuple a new confidence score according to the credibility of rules.

In the previous section, it is shown that predicates can be derived from both direct and indirect rules, and consequently lead to some dependencies among direct and indirect rules based on their distance to the predicate queried by user. In table 1, if the first rule is executed before the fourth rule, it‘s apparent that the first rule have

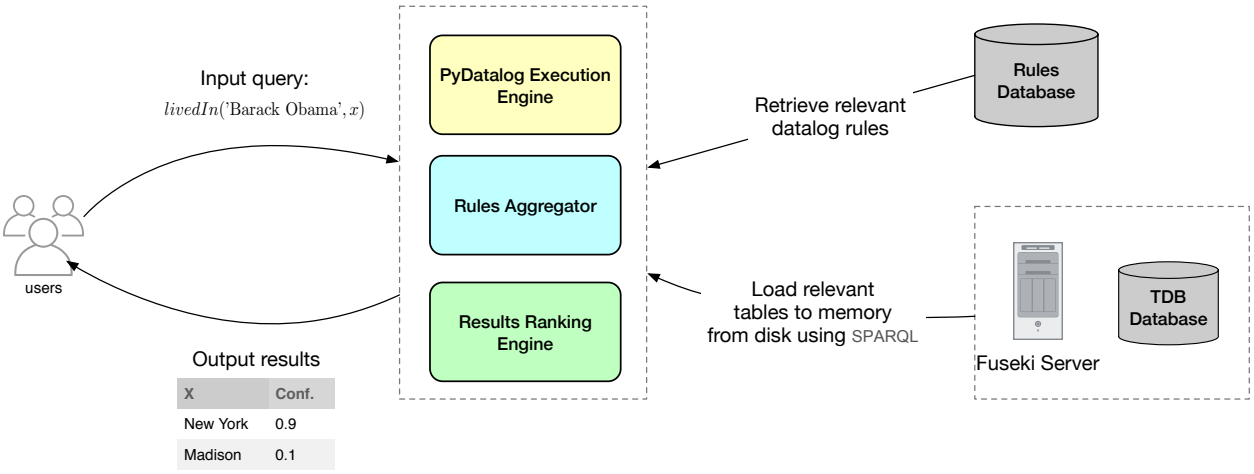


Figure 1: The architecture diagram of the KnowledgeLog system

to be executed again as there may be some data tuples relocated from `director` to `writer`, which should also be added to `notableWork` thereafter. The reason why rule 1 encounters the recurrent execution issue is because the `writer` in `notableWork`'s body is an intensional relations (IDB) whose values are determined by other predicated in the right-hand-side of the body, so the data tuple in the IDB may change while rule is executing. To prevent the influence of IDB and recurrent execution, a execution framework in Algorithm 1 is proposed to solve this issue.

In the execution framework (Algorithm 1), the extensional relations (EDB) are first picked up and added into a set called `computed`. Since EDB occurs only in the right-hand-side of the rules, such relations are intuitively the input of the Datalog and will not be modified even other rules are executed. After the `computed` is set up, all rules whose body are all in `computed` will be added to a set named `cand`, which represents the set of rules that is going to be executed in next iteration. In the execution process, all rules in `cand` will first be executed, and then the data tuples in the body of rules will be stored in `tempTable`, where a scoring algorithm, that will be introduced later, is applied to recalculate a confidence score for each of them. Once all rules in `cand` is executed, the head of these rules will be stored in `computed` to represent that the predicates in head of rule are no longer IDBs, and can be used as EDBs for further calculation. Other than that, the rules in `cand` will be updated based on the new predicates just added to `computed`. This execution process will continue until there is no rule that hasn't been executed, and then switch to the pyDatalog engine for running over the `tempTable` to retrieve data tuples matched the head of user query.

Scoring algorithm in Algorithm 2 is how a new confidence score be calculated and assigned to a new data tuple, this algorithm is applied while a rule is processed in the execution framework. Before diving into the calculation, the assumption made here is that every the fact original in the knowledge base (data pulled from Jena Fueski server) is true, and should be given a confidence score 1.00. This assumption provides KnowledgeLog not only the starting point but also a baseline to evaluate how credible a new fact is. While a rule is executing, the facts belong to corresponding predicates in the body of rule are retrieved with their confidence, and then be joined to generate new facts for the predicate in the head. At the meantime, the confidence scores from right-hand-side of rule will be multiplied together with the confidence of rule, and then attach to the new generated facts.

Algorithm 1: Execution framework

Input : Query q, Datalog Rules $\mathcal{R}$, Database Instance $\mathcal{I}$.
Output: Tuples with associated confidence score.

```
computed :={EDB}
cand :={all rules whose atoms in body are all in computed}
tempTable = ∅
while there are rules that haven't been executed do
    execute all rules in cand and store results in tempTable;
    store all head of rules in cand to computed;
    update cand with new predicate stored in computed;
end
return tuple in the tempTable matched the head of query;
```

3.3 Fuseki Server

The deployment of Jena Fuseki server is based on a docker image which is integrated with the docker volume to facilitate data loading and persistence. Once the server is set up, the result ranking engine from client side can easily send a HTTP request to the server to retrieve data with corresponding SPARQL query.

Algorithm 2: Calculate score for a tuple

Input : A datalog rule r $(S := A, B)$, Database Instance $\mathcal{I}$,
Output: Tuples with associated confidence score.

```
for t ∈ r(I) do
    if t ∈ {m ⋈ n | m ∈ A, n ∈ B} then
        score(t):=score(r)×score(m)×score(n)
end
```

4 Evaluation

The main purpose of this project is to evaluate the performance of KnowledgeLog. In this part, We measured the performance of KnowledgeLog based on rule threshold, rule depth, and data size.

4.1 Experiment Setup

Hardware. All experiments were run on a personal laptop with 16GB of RAM, 2 CPUs (3.1 GHz Intel Core i7).

Dataset. We use the English version of the DBpedia 3.8 data set as our main Knowledge base data source. The DBpedia data set uses a large multi-domain ontology which has been derived from Wikipedia. It contains 11 million facts and 650 different predicates.

Logical rules are extracted with Partial Completeness Assumption (PCA) confidence by applying AMIE+ to the rules in DBpedia 3.8 data set [13].

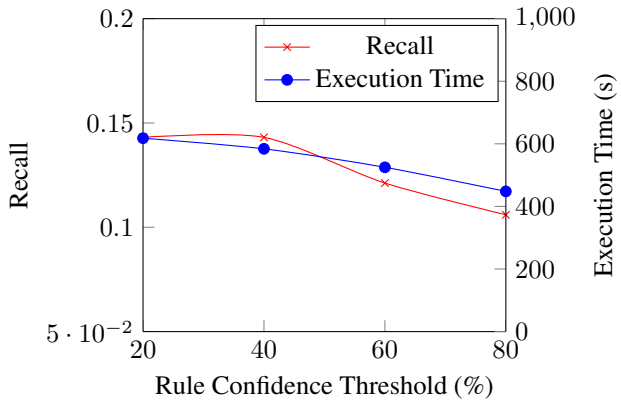
Evaluation Metrics. If a tuple `<subject, relation, object>` is in the source data set, we can see this tuple as a fact. If a predicted tuple appears in the source data set, we know that it's corrected predicted. However, if the predicted tuple not inside the data set, we are not able to say that the predicted tuple is wrong. As a result, without labeling, we calculate the recall to evaluate the quality of KnowledgeLog.

To calculate the recall, we deleted the specified relation in the source data set and then use the remained data set to do KnowledgeLog. No matter how much the confidence we calculate for a predicted tuple is, if the tuple is inside the source data set, we seen it as corrected predicted.

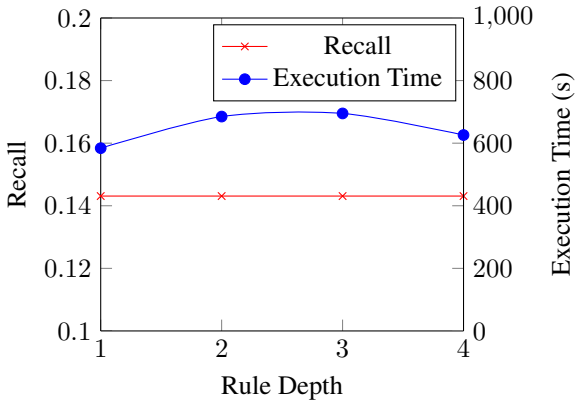
Settings. We compared the quality and time of each different hyper-parameters such as the depth for level-order traversal (Rule Depth), the threshold of minimum rule confidence(Rule Confidence Threshold), and percentage of data size (Data Size). We show the experimental result in **4.2**.

4.2 Experimental Result

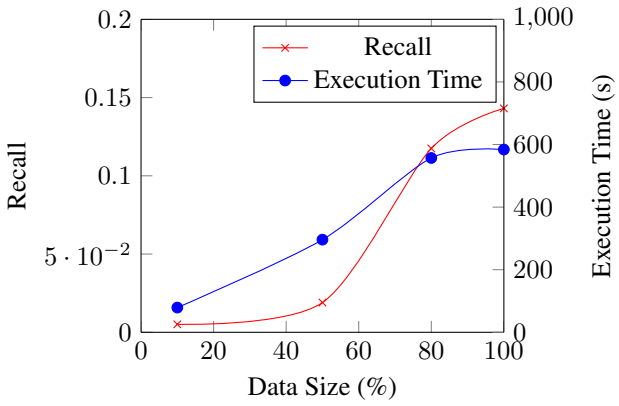
The following chart is the experimental result based on each rule threshold. We did evaluation by using the 100% dataset from dbpedia.3.8 with rule_depth=1 over the predicate `notableWork`. We can easily see that the execution speed is faster and the recall is lower while the rule confidence threshold is larger. Though recall is lower when the threshold is larger, we suppose that the precision might be higher at the same time because higher threshold let more reliable rules remain but may trimmed out some somewhat influential rules.



The following chart is the experimental result based on each rule depth. We did evaluation by using the 100% dataset from dbpedia.3.8 with rule_threshold=0.4 over the predicate `notableWork`. The rule depth did not affect the recall. Therefore, we do not consider rule depth is important. However, choosing the lower level of depth might reduce execution time.



The following chart is the experimental result based on each the sampled percentage of data set. We did evaluation by using the 100% dataset from dbpedia.3.8 with rule_threshold=0.4 over the predicate `notableWork`. Data size is the factor largely affect the quality of KnowledgeLog. The larger the data size is, the more reliably predicted results we can make. It's important to find the balance between recall and execution time based on data size.



5 Future Work

The KnowledgeLoghas shown to do the knowledge base reasoning well. However, there is still room for improvement.

Our engine currently cannot work correctly if there are cycles in our tree structured rules. However, cycles in the rules are inevitable. To avoid this problem, we trimmed some edge, which defines a rule, in the tree to make it a directed ascyclic graph. However, this solution might lead us missing out some important rules and thus decrease the performance. We are interested in finding a way to predict the result safely without deleting any rules.

6 Conclusion

In this paper, we present KnowledgeLog a system that enables users to query indirect facts in knowledge bases with ontology rules and associated confidence. We described the architecture and algorithms that make the system capable of processing large-scale knowledge bases on disk while the main inference process is in memory. We also conducted experiments that demonstrated interesting trade-off in performance when using different set of hyper-parameters. Last but not least, we pointed out several directions that we will further investigate to strengthen the project in the future.

References

[1] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[2] Feng Niu, Ce Zhang, Christopher Ré, and Jude W Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. *VLDS*, 12:25–28, 2012.

[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[4] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[5] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[6] Yanchao Hao, Hao Liu, Shizhu He, Kang Liu, and Jun Zhao. Pattern-revising enhanced simple question answering over knowledge bases. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3272–3282, 2018.

[7] Lan Du, Anish Kumar, Mark Johnson, and Massimiliano Ciaramita. Using entity information from a knowledge base to improve relation extraction. In *Proceedings of the Australasian Language Technology Association Workshop 2015*, pages 31–38, 2015.

[8] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[9] Denis Krompaß, Stephan Baier, and Volker Tresp. Type-constrained representation learning in knowledge graphs. In *International semantic web conference*, pages 640–655. Springer, 2015.

[10] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526. ACM, 2014.

[11] 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*, pages 2787–2795, 2013.

[12] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 192–202, 2016.

[13] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with amie$$+$$+. *The VLDB Journal*, 24(6):707–730, December 2015.

[14] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[15] Apache Software Foundation. Apache jena, 2000.