Predicting the Cost of Online Reasoning on Knowledge Graphs: Some Heuristics

Varsha R Mouli, Unmesh Joshi, Ceriel Jacobs, and Jacopo Urbani

Department of Computer Science
Vrije Universiteit Amsterdam, The Netherlands
v.ravichandramouli@student.vu.nl,u.n.joshi@vu.nl,
{ceriel,jacopo}@cs.vu.nl

1 Introduction

Knowledge Graphs (KGs) are large repositories of knowledge where pairs of entities are connected to each other with named relations, e.g., *typeOf(VU,University)*. Modern KGs contain a wealth of implicit knowledge that is useful for a variety of usecases, e.g., query answering, inconsistency detection, or KG completion. This knowledge can be extracted in the form of derived triples by computing rule-based reasoning, but this task is challenging due to the large size of modern KGs. So far, the most common approach for this task consists of materializing all possible derivations in an offline fashion, i.e., before the user can query the KG [4, 3]. However, reasoning can be performed in an online fashion as well. Online reasoning is ideal when the queries require different rule-sets, and/or when we must avoid a full materialization because of some erroneous input. Moreover, materialization simply might not be possible due to the lack of resources.

In an online setting, we assume that there is a user interacting with the KG. Therefore, it is crucial that either reasoning is computed quickly or that the user is alerted immediately that the computation will take a long time. In this context, query-driven reasoning algorithms are ideal because they limit the computation to only derivations that are useful to answer the input query. However, to the best of our knowledge there are no mechanisms for predicting the runtime of these algorithms before their execution.

In this paper, we investigate on the applicability of two popular query-driven rule evaluation procedures developed for the Datalog language [1]: Query-Subquery (QSQ), a top-down procedure, and Magic Set (MS) which proceeds bottom-up [1]. In theory, both algorithms attempt at limiting the scope of the computation with the same principle. However, the adopted strategy is significantly different. For instance, MS performs an initial rewriting while QSQ operates directly on the original ruleset. This rewriting introduces some startup cost, which might pay off in some cases, but not in all. As a result, there are some queries for which one algorithm is much faster than the other.

Unfortunately, both procedures trigger a complex recursive computation so it is hard to determine beforehand which algorithm is likely to be faster. In this paper, we offer a preliminary investigation on how effective some simple heuristics are in order to quickly make such prediction.

2 Heuristics for Predicting the Cost of Online Reasoning

We focus on reasoning that can be encoded using a Datalog program [1]. Here, reasoning is invoked to answer a Datalog query γ, which consists of a single atom, on a database D and with a ruleset P. Given a rule $r = \alpha \leftarrow \beta_1, \ldots, \beta_n$, we define $r(D) = \{\alpha\sigma | \beta_1\sigma, \ldots, \beta_n\sigma \in D\}$ where σ is a postfix operator with a mapping from variables to constants, and let $P(D) = \bigcup_{r \in P} r(D)$ be the set of facts derived by the rule r and all the rules in P respectively. Further, we set $P^0(D) = D$ and recursively define $P^{i+1} = P(P^i(D) \cup D)$ for $i \geq 0$. The *materialization* of D with P is the union $P^\infty(D) = \bigcup_{i \geq 0} P^i(D)$ and $Ans^\gamma_P(D) = \{\gamma\sigma \mid \gamma\sigma \in P^\infty(D)\}$ is the desired set of answers that must be computed for answering γ.

QuerySubQuery (QSQ) and MagicSets (MS) [1] are two procedures to compute Ans which work with the same principle: Starting with an input query, they rewrite it in a number of subqueries (stored in some temporary relations, which we call $input*$) and incrementally compute relevant derivations (stored in other temporary relations, which we call $ans*$). The main difference between the two procedures is that QSQ uses some external ad-hoc relations while MS creates a new ruleset P_γ with new predicates and rules and delegates its execution to a Datalog engine (see [1] for more details).

In our experiments, we found that QSQ is faster than MS for queries that trigger reasoning that can be computed quickly. This is because QSQ requires no rewriting and thus has a much lower overhead than MS. If this is not the case, then MS is faster because we can offload the computation of reasoning to highly efficient engines. This brings us to our research question: Can we estimate the runtime of reasoning beforehand so that we can pick the right algorithm and obtain significantly lower runtimes?

First, we must determine which are the factors that impact the cost (i.e., runtime) of reasoning. These are the number of rule executions, the cost of each rule execution, and the operations for storing the derivations and other intermediate data. An exact estimation of these factors is not possible without actually computing reasoning. However, we propose the following five indicators to give us some hints on their real values.

• *EF1:* Let $D_q = \{\alpha \mid \alpha \in D \wedge Ans^\gamma_P(D \setminus \{\alpha\}) \subset Ans^\gamma_P(D)\}$ be the set of facts that are relevant for answering γ. In the best case, any query-driven algorithm would need to store at least the subset $Inp_q \subseteq P^\infty(D_q) \setminus D_q$ which represents the minimal set of derivations necessary to compute all answers. These derivations are stored in the relations $ans*$ and this indicator estimates their total size.

• *EF2:* Assume that, while computing Ans^γ_P, we must execute the rule $\alpha \leftarrow \beta_1, \ldots, \beta_n$ to a partially augmented database $P^i(D) \subseteq P^\infty(D)$, and let $I_r = \bigcup_{i=1}^n \{\sigma \mid \beta_i\sigma \in P^i(D)\}$. A large part of the runtime for executing this rule is determined by the size of I_r. This is because we need to join the substitutions for the body atoms ($\beta_1, \ldots, \beta_n$) to calculate suitable substitutions for the head of the rule (α). This indicator estimates the size of I_r.

• *EF3 and EF4:* EF3 estimates the number of rules executions. EF4 counts the distinct number of rules that should be executed.

• *EF5:* Let P_γ be the rewritten program that the MS procedure returns for the query γ. In this case, the set of subqueries is defined as $Q = \{p(\mathbf{t}) \mid p(\mathbf{t}) \in P^\infty_\gamma(D) \wedge p \text{ starts with 'input'}\}$. Subqueries are produced both by QSQ and MS and must be

Depth	QSQ	MS	**Est**
2	47.13	21.08	0.32
4	48.25	21.32	0.35
8	47.32	21.82	0.38

(a) Avg. runtime: Reasoning vs. Estim. (sec.)

Ind.	QSQ	MS	**Pred.**
EF1	18.6	11.97	9.3
EF2	40.02	4.36	30.6
EF3	41.32	5.06	33.76
EF4	9.8	2.4	2.1
EF5	41.32	5.06	33.76

(b) Runtime using Indicators (min.)

Fig. 1. Runtime of the estimation procedure (a), and impact on reasoning runtime (b).

stored in the same way as inferred facts in order to ensure termination. This indicator estimates the cardinality of Q.

How can we estimate these indicators and use them to answer the query? We simulate the execution of QSQ up to a certain depth (termination criterion specifying maximum iteration count). This mimics the execution of QSQ without actually performing any derivation, but simply by counting the number of facts that would need to be joined during the application of the rules (EF2), the number of potential derivations (EF1) (calculated as cartesian product), and the number of subqueries (EF5) triggered by the algorithm. Moreover, the procedure counts the number of (distinct) rule executions (EF3 and EF4). While this procedure performs only a very rough estimation and stops at a certain depth, it has the advantage that it is fast.

Then, the actual prediction is made considering the output value of one of these indicators. In particular, two threshold values are used: One value is used as a sort of minimum eligible criterion: If the indicator returns a value below this number, then the prediction is not made due to lack of confidence. Another threshold value is used to pick either QSQ and MS: If the value is lower than this second threshold, then QSQ is chosen. Otherwise, it chooses MS. For this preliminary study, values for these thresholds are calculated manually after analyzing some example queries.

3 Preliminary Evaluation and Conclusions

We used the system VLog [4], which contains optimized implementations of QSQ and MS, on a machine with a high-end CPU and 64GB RAM. As input, we considered LUBM-1K, a popular benchmark dataset with 133M triples [2]. As ruleset, we used the *LUBM_L* ruleset used in [4, 3]. To test our predictions on a large number of queries, we performed a full materialization, and wrote a procedure to extract 300 test queries which return differed answers on the original KG and on the materialized one (thus, they must trigger reasoning). These queries range from being very specific (i.e., the query is a fact that needs to be proven) to being more generic (i.e. they return many answers).

Cost of estimating. If the estimation takes more time than the actual reasoning, then it is no longer useful. We report in Tab. 1a, the average execution time taken for the test queries using QSQ and MS against the average time taken for estimating the indicators using different values for the depth. The results show that that computing the estimated values is significantly lower than performing reasoning.

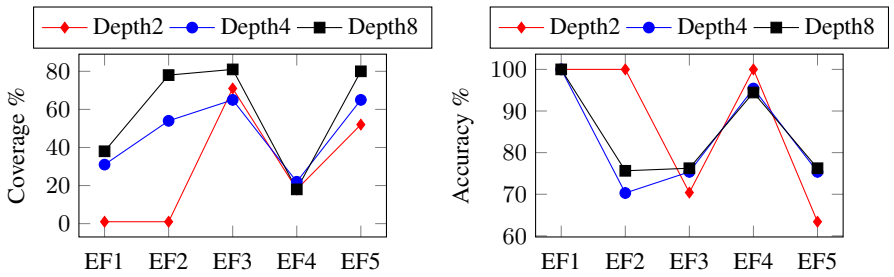

Fig. 2. Coverage and Accuracy of the test queries using the LUBM-1K dataset.

Impact on reasoning. Tab. 1b reports the time which would take if we only execute QSQ, MS, or if instead we decide the algorithm at runtime. Notice that these experiments were performed only on the queries that were eligible for prediction (i.e., indicator value greater than first threshold). Each line reports the runtime for each indicator. From the table, we observe that only EF1 and EF4 lead to an increase in the performance. For all other indicators, the strategy of always selecting MS would lead to better runtimes. However, notice that using any indicator is better than using only QSQ.

Accuracy and Coverage. Fig. 2 reports the accuracy and the coverage of the various indicators. We see that the accuracy is highest for EF1 (100% across all depths) followed by EF4. However, notice that in this case the coverage is low, which means that only few queries qualify. In contrast, the other indicators have a higher coverage. For instance, EF5 with depth 8 has a coverage of 80% and accuracy of 70-80%. This means that we can apply it on 80% of the queries and in general it guesses correctly with a similar rate. Therefore, it is a much better candidate to estimate the cost of reasoning.

Outlook. Our preliminary results show that heuristics like the indicators presented above are helpful to estimate the cost of reasoning, and hence can be used to improve the runtime of query answering. Even though our results vary considerably, both in terms of accuracy and coverage, we believe that they illustrate the potential of this method. Future work is necessary to better understand the strengths of these heuristics, and to investigate whether more sophisticated combinations lead to better predictions.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
2. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3:158–182, 2005.
3. Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RDFox: A Highly-scalable RDF Store. In *Proceedings of ISWC*, pages 3–20, 2015.
4. Jacopo Urbani, Ceriel Jacobs, and Markus Krötzsch. Column-Oriented Datalog Materialization for Large Knowledge Graphs. In *Proceedings of AAAI*, pages 258–264, 2016.