Une nouvelle algèbre pour SPARQL permettant l’optimisation des requêtes contenant des expressions de chemin
Louis Jachiet, Pierre Genevès, Nabil Layaida, Nils Gesbert

To cite this version:

HAL Id: hal-01647638
https://hal.archives-ouvertes.fr/hal-01647638
Submitted on 24 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
ABSTRACT

SPARQL is a language of queries on the graph RDF standardised by the W3C. Since its version 1.1, SPARQL allows the expressions of path (Property Paths) and these expressions pose new challenges to SPARQL engines. In fact, as regular expressions, path expressions can be recursive. Or the optimisation of recursive queries remains a challenge both in the relational world and in the world of the web of data.

We introduced an algebra inspired by the relational algebra and by the SPARQL algebra as well as a translation from SPARQL to this algebra. We then equipped this algebra with a rewriting scheme: given a SPARQL query one can then translate it into our algebra and generate many equivalent terms which are then seen as possible execution plans for the initial SPARQL query.

Finally, we showed that our rewriting schemes considered execution plans that are not considered by existing approaches and we validated this result experimentally: we implemented a SPARQL evaluator based on this algebra and implementing this method. The efficiency of our prototype shows the interest of our approach.

1 INTRODUCTION

In the recent years, we have seen an unprecedented development of heterogeneous data formats and stores. A major challenge consists in querying datasets that are not only increasingly large, but that also come from numerous sources with different data models. It would thus be very desirable to have a common language capable of handling the diversity of data formats while allowing optimization of the querying phase especially across query languages.

SQL has long been viewed as a common language for querying data represented as relational tables. SQL stores are very popular, well optimized, and many of the NoSQL query languages can be translated to SQL. However, for structurally rich data models such as graphs and trees, SQL has not proved to be the ideal candidate, and so far optimization techniques from the relational world hardly carry over languages such as SPARQL [4, 7]. While SQL might not be the perfect candidate, we postulate that it is possible to extend or adapt relational algebra for other purposes and benefit from the massive amount of research invested in it.

2 PROPOSED CONTRIBUTION

We propose a new algebra, \( \mu \)-algebra, inspired by works on the relational algebra, SQL and NoSQL languages (especially SPARQL) along with a prototype implementation of a SPARQL optimizer based on this algebra\(^1\). Our algebra has the following properties:

1. It subsumes the SPARQL Algebra (under the set semantics) with a more general recursion.
2. SPARQL with Property Paths can be efficiently translated to this algebra.
3. We have a type system and rewriting rules for terms of this algebra that allow optimization, notably of terms involving recursion.

In this paper we illustrate the differences and the benefits of our approach on recursive query optimization. While a generic approach often comes at the cost of performance, we experimentally show that this approach actually leads to more efficient evaluation of queries with Property Paths. We also show that our approach produces Query Execution Plans (QEP) that are not considered by other existing approaches.

3 COMPARISON WITH OTHER APPROACHES

The optimized evaluation of SPARQL is a well studied especially for the BGP fragment. The evaluation of recursive queries is also a well studied subject. In this section we propose to compare our approach to various lines of work that have tackled the subject from the more ad-hoc, tailored for SPARQL to very general approaches.

3.1 Competitors

Jena ARQ. Jena ARQ\(^2\) is a sparql query evaluator. Jena evaluates the queries in the order of the query. In our benchmark (see figure 1) we compare thus the two orders (ARQ\(_1\) is quadratic and ARQ\(_2\) linear) but the size of the stack breaks the JVM for n=3000.

The relational algebra. The relational algebra introduced by Codd built the foundations of our work. The relational algebra differs from our work in several points. The two most salient ones being

\(^1\)https://gitlab.inria.fr/jachiet/musparql
\(^2\)https://jena.apache.org/
that the relational algebra works on a fixed domain and that it is not equipped with a fixpoint operator. There have been attempts[1] to extend the relational algebra. With an operator $\alpha$ representing recursive queries (if $R$ is a binary relation $\alpha (R)$ is the transitive closure of $R$). Or with a special join reachability $\alpha_{\text{dp}} \ b$ (equivalent to $a \bowtie \alpha (b)$).

However, if these operators are sufficient to represent SPARQL, they do not allow for a plan space as large as our $\mu$-algebra. For instance, on the query $?a \ knowns \ b lastname Doe \land \exists b \ \text{friendOf} \ b$, then these approaches will need to compute the whole $\text{knowns} \ \text{childOf} \ \text{friendOf}$ in order to compute the transitive closure (which might be very large in comparison with the set of actual solutions when e.g. $\alpha$ is conditioned by some other triple pattern).

SQL. SQL is based on the relational algebra. Both have been extensively studied either for themselves or in the context of SPARQL query evaluation. However using SQL for the optimization of SPARQL has not been very successful[4,7] (even without recursion).

The SQL `99 standard includes Recursive Common Table Expressions (CTE). Recursive CTE are a very broad kind of recursive queries, broader than what is allowed in the alpha-extended. However not all SQL databases support recursive CTE and vendors generally consider CTE as “optimization fences”. We benchmarked several SQL stores in our benchmark comparison but they behavior is quadratic on queries where our prototype has a nice linear behavior.

Waveguide. Waveguide[12] introduced Waveguide Plans (WGP) allowing the optimized evaluation of Property Paths. WGP mix together $\alpha$-plans (which are plans based on the $\alpha$-extended relational algebra) and FA-plans (which are based on automata). Waveguide translates one PP at a time which means the method is not capable of optimizing across multiple TP. For instance given 3 TP: $\exists a \ knowns \ b$, $\exists b \ \text{lastname Doe}$ and $\exists b \ \text{firstname John}$ Waveguide computes the whole $\text{knowns}$ and even on a single triple pattern it does not consider all the plans that our approach considers.

Datalog. Finally, a major line of research to tackle recursive queries is datalog. There has been translations from SPARQL 1.0 to datalog[10]. The optimization and fast execution of datalog on graph data is a challenge due to the expressive power of datalog and its logic-based form[3]. The translation SPARQL 1.1 with Property Paths to datalog seems to raise no particular issue even though we have not found any attempt in the literature therefore we hand-translated recursive queries.

3.2 Benchmark

We benchmarked the query composed of the two triple patterns $?a \ knowns \ ?b$ and $?b \ \text{lastname Doe}$ on the following system: postgresql and sqlite for SQL, datalog for Datalog and Jena ARQ. Waveguide is not publicly available thus not tested. The results are in figure 1 and demonstrate that our approach is the one that is not quadratic in the size of the graph in all cases.

![Figure 1: Evaluation time for the query on a graph of size n](image)

4 CONCLUSION AND FUTURE WORKS

We believe that our algebra represents a step toward the ambitious goal of unifying various traits of the relational algebra with traits of NoSQL languages in a common framework (syntax, semantics, typing, rewriting schemes). Our algebra subsumes the SPARQL Algebra (under the set semantics) with a more general recursion. As a perspective for further work, we plan to investigate how our approach can be improved along several directions: finer-grained cardinality estimation, distributed implementations for evaluating terms of our algebra, and extensions for the compilation of query languages with other data models.

REFERENCES


