Une nouvelle algèbre pour SPARQL permettant l'optimisation des requêtes contenant des expressions de chemin

Louis Jachiet, Pierre Genevès, Nabil Layaïda, Nils Gesbert

Une nouvelle algèbre pour SPARQL permettant l'optimisation des requêtes contenant des expressions de chemin

Louis Jachiet
Univ. Grenoble Alpes, CNRS, Grenoble INP*, LIG,
38000 Grenoble, France
louis.jachiet@inria.fr

Pierre Genevès
Univ. Grenoble Alpes, CNRS, Grenoble INP*, LIG,
38000 Grenoble, France
pierre.geneves@cnrs.fr

Nabil Layaïda
Univ. Grenoble Alpes, CNRS, Grenoble INP*, LIG,
38000 Grenoble, France
nabil.layaida@inria.fr

Nils Gesbert
Univ. Grenoble Alpes, CNRS, Grenoble INP*, LIG,
38000 Grenoble, France
nils.gesbert@grenoble-inp.fr

ABSTRACT

SPARQL est un langage de requêtes sur les graphes RDF standardisé par le W3C. Depuis sa version 1.1, SPARQL autorise les expressions de chemin (Property Paths) et ces expressions posent de nouveaux défis aux moteurs SPARQL. En effet, en tant qu'expressions régulières, les expressions de chemin peuvent être récursives. Or l'optimisation des requêtes récursives reste un défi tant dans le monde relationnel que dans celui du web des données.

Nous avons introduit une algèbre inspirée par l'algèbre relationnelle et par l'algèbre SPARQL ainsi qu'une traduction depuis SPARQL vers cette algèbre. Nous avons ensuite équipé cette algèbre d'un schéma de réécriture : étant donné une requête SPARQL on peut alors la traduire dans notre algèbre puis générer de nombreux termes équivalents qui sont alors vus comme de possibles plans d'exécution de la requête SPARQL initiale.

Enfin, nous avons montré que nos schémas de réécritures considèrent des plans d'exécution qui ne sont pas considérés par les méthodes existantes et avons validé ce résultat expérimentalement : nous avons implémenté un évaluateur de requêtes SPARQL basé sur cette algèbre et mettant en place cette méthode. L'efficacité de notre prototype montre l'intérêt de notre approche.

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 such 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, μ-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

*Institute of Engineering Univ. Grenoble Alpes

[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 α representing recursive queries (if R is a binary relation $\alpha(R)$ is the transitive closure of R). Or with a special join reachability $a \bowtie_R b$ (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 μ-algebra. For instance, on the query $?a\,(knows/childOf/friendOf)*\,?b$, then these approaches will need to compute the whole $knows/childOf/friendOf$ in order to compute the transitive closure (which might be very large in comparison with the set of actual solutions when e.g. $?a$ 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 been 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 α-plans (which are plans based on the α-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: $(?a\ knows*\ ?b)$, $(?b\ lastname\ Doe)$ and $(?b\ firstname\ John)$ Waveguide computes the whole $knows*$ 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 $?a\ knows*\ ?b$ and $?b\ lastname\ Doe$ on the following system: *postgresql* and *sqlite* for SQL, *datalog*[3] and *dlv* 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.

[3] http://www.ccs.neu.edu/home/ramsdell/tools/datalog/

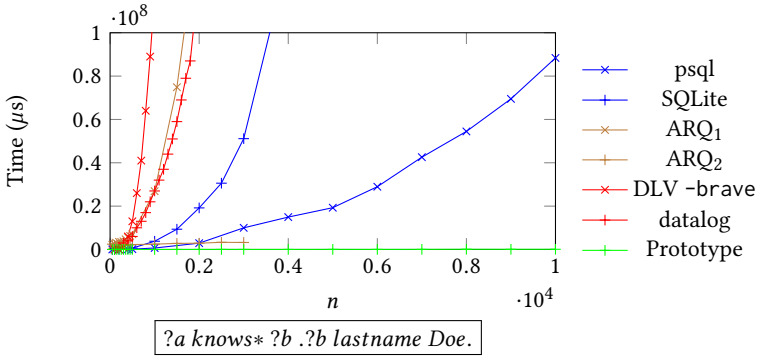

Figure 1: Evaluation time for the query on a graph of size n

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

[1] R. Agrawal. 1988. Alpha: an extension of relational algebra to express a class of recursive queries. *IEEE Transactions on Software Engineering* 14, 7 (July 1988), 879–885. https://doi.org/10.1109/32.42731

[2] Faisal Alkhateeb, Jean-François Baget, and Jérôme Euzenat. 2009. Extending SPARQL with Regular Expression Patterns (for Querying RDF). *Web Semant.* 7, 2 (April 2009), 57–73. https://doi.org/10.1016/j.websem.2009.02.002

[3] S. Ceri, G. Gottlob, and L. Tanca. 1989. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering* 1, 1 (Mar 1989), 146–166. https://doi.org/10.1109/69.43410

[4] Orri Erling and Ivan Mikhailov. 2009. *RDF Support in the Virtuoso DBMS.* Springer Berlin Heidelberg, Berlin, Heidelberg, 7–24. https://doi.org/10.1007/978-3-642-02184-8_2

[5] Andrey Gubichev, Srikanta J. Bedathur, and Stephan Seufert. 2013. Sparqling Kleene: Fast Property Paths in RDF-3X. In *First International Workshop on Graph Data Management Experiences and Systems (GRADES '13).* ACM, New York, NY, USA, Article 14, 7 pages. https://doi.org/10.1145/2484425.2484443

[6] Katja Losemann and Wim Martens. 2013. The Complexity of Regular Expressions and Property Paths in SPARQL. *ACM Trans. Database Syst.* 38, 4 (Dec. 2013), 24:1–24:39. https://doi.org/10.1145/2494529

[7] T. Neumann and G. Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *2011 IEEE 27th International Conference on Data Engineering.* 984–994. https://doi.org/10.1109/ICDE.2011.5767868

[8] Carlos Ordonez. 2010. Optimization of Linear Recursive Queries in SQL. *IEEE Transactions on Knowledge and Data Engineering* 22, 2 (Feb. 2010), 264–277. https://doi.org/10.1109/TKDE.2009.83

[9] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2006. Semantics and Complexity of SPARQL. *CoRR* abs/cs/0605124 (2006). http://arxiv.org/abs/cs/0605124

[10] Simon Schenk. 2007. *A SPARQL Semantics Based on Datalog.* Springer Berlin Heidelberg, Berlin, Heidelberg, 160–174. https://doi.org/10.1007/978-3-540-74565-5_14

[11] Michael Schmidt, Michael Meier, and Georg Lausen. 2010. Foundations of SPARQL Query Optimization. In *Proceedings of the 13th International Conference on Database Theory (ICDT '10).* ACM, New York, NY, USA, 4–33. https://doi.org/10.1145/1804669.1804675

[12] Nikolay Yakovets, Parke Godfrey, and Jarek Gryz. 2016. Query Planning for Evaluating SPARQL Property Paths. In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016.* http://ceur-ws.org/Vol-1644/paper38.pdf