Query Planning for Evaluating SPARQL Property Paths

Nikolay Yakovets, Parke Godfrey, and Jarek Gryz

Department of Computer Science and Engineering, York University, Canada
{hush,godfrey,jarek}@cse.yorku.ca

Abstract. The extension of SPARQL in version 1.1 with *property paths* offers a type of *regular path query* for RDF graph databases. Such queries are difficult to optimize and evaluate efficiently, however. We have embarked on a project, WAVEGUIDE, to build a cost-based optimizer for SPARQL queries with property paths. WAVEGUIDE builds a query plan—which we call a *waveplan* (WP)—which *guides* the query evaluation. There are numerous choices in the construction of a plan, and a number of optimization methods, so the space of plans for a query can be quite large. Execution costs of plans for the same query can vary by orders of magnitude. A WP's costs can be estimated, which opens the way to cost-based optimization. We demonstrate that the plan space of WAVEGUIDE properly subsumes existing techniques and that the new plans it adds are relevant.

1 Introduction & Motivation

Graph data is becoming rapidly prevalent with the rise of the Semantic Web, social networks, and data-driven exploration in life sciences. There is a need for natural, expressive ways to query over these graphs. Standards are coming into place for this. The Resource Description Framework (RDF) [9] provides a data model for graph data. An RDF *store* is a set of *triples* that describes a directed, edge-labeled multi-graph. A triple, $\langle \mathsf{s}, \mathsf{r}, \mathsf{o} \rangle$, denotes an *edge* from *node* "s" (the *subject*) to node "o" (the *object*), with the edge labeled by "r" (the *role*, also called *label* or *predicate*). The SPARQL query language [8] correspondingly provides a formal means to query over RDF stores. A query defines sub-graph match criteria; its evaluation over an RDF store returns all embedded sub-graphs or variable bindings meeting the criteria. For example, the query

$$\text{?friend :friendOf Charles .} \qquad (\mathcal{Q}_1)$$

evaluates to a list of people (nodes binding to variable "?friend") who are friends of (role ":friendOf") "Charles" (a named node, so a constant).

In its latest version, 1.1, SPARQL's expressiveness is extended with *property paths* [5]. This effectively introduces the concept of *regular path queries* (RPQs)—well studied before the advent of RDF and SPARQL—into the query language. Instead of specifying the path of interest *explicitly* between nodes, one may now specify it *implicitly* via a *regular expression*. For example, the query

?friend :friendOf+ Charles . ($\mathcal{Q}_2$)

evaluates to a list of people who are friends of "Charles", or friends of people who are friends of "Charles", and so forth (that is, a *transitive closure* over ":friendOf").

While SPARQL provides the expressiveness we desire, such queries are more challenging to optimize well. Query $\mathcal{Q}_1$ could be evaluated just by extracting the triples with "r = :friendOf" and "o = Charles". For even a slightly more complicated query, however, it may not be straightforward to find a *plan* to evaluate it efficiently. $\mathcal{Q}_2$ is more challenging in requiring transitive closure over ":friendOf" with respect to the graph.

Property path evaluation is a tale of two methods: two quite different approaches appear in the literature. For RPQs, the seminal work [7] which introduced the G+ query language shows how to use a *finite state machine* effectively as a plan to guide the *graph walk* for the query's evaluation. We call this approach FA. Subsequent work on RPQs has followed this idea.

SPARQL with property paths is much more recent. Systems for SPARQL query evaluation have followed the second approach, based primarily on the seminal work of [6]. These extend the *relational algebra* to accommodate the translation of a property path's regular expression, and then use dynamic programming over the (extended) relational-algebra parse to devise a plan. Added is an "α" operator, which provides the transitive closure over a relation to accommodate regular expressions' Kleene star. Thus, we call this approach α-RA, the relational algebra extended by "α" [1].

Which approach is better? We shall show that the effective "plan spaces" that result from FA and α-RA are incomparable. Sometimes, for a given query and graph, an FA plan will be the better choice. Other times, an α-RA plan will be. Our goal is to formalize the notion of plan space for both, to be able to choose the best plan. We shall show that a richer plan space can be had that properly subsumes FA and α-RA, and offers more plans existing in neither ("mixed" plans), which sometimes are the best plans.

We have designed a system called Waveguide [12] with the goal to provide viable cost-based query optimization and evaluation for SPARQL over RDF stores that is on par with the state of the art for relational database systems. In [12], we address the first but critical step of this endeavor, defining a *plan space*—the space of query plans, *waveplans* (WPs)—for SPARQL queries. We focus on single-path, property-path queries, essentially the RPQ fragment of SPARQL 1.1. We consider a set semantics—the distinct directive in each query—and thus do not consider aggregation.

In [12], our contributions are as follows.

1. *plan space.*
   (a) *Summarize* the state of the art for evaluation of RPQs and SPARQL property paths.
   (b) *Establish* why none suffices.
   (c) *Devise* Waveguide's plan space, and *demonstrate* that it *subsumes* the state of the art, and extends beyond it.

(d) *Model* the *cost factors* that determine the efficiency of plans, and *present* the powerful optimizations offered by WAVEGUIDE plans.

2. *performance study.*
   (a) Provide a *microbenchmark* over a pertinent RPQ template with realistic queries over real RDF stores / graphs.
   (b) *Substantiate* the optimizations of our approach.
   (c) *Justify* the *necessity* of the richer plan space.

2 Approach

Work on property-path evaluation has been remiss in not drawing the connection to RPQs. How do the FA and α-RA approaches compare? Does one subsume the other? Or are they *incomparable*? If so, a combined approach might be superior. A generalized approach might offer new plans that neither FA nor α-RA can produce with superior performance.

Both the FA and α-RA approaches effectively provide evaluation plans for property-path queries. However, the *plan spaces* that are implicit in these approaches have not been considered. In FA, choosing a different (but still correct) automaton for the plan might offer a significantly more efficient plan. In systems taking the α-RA approach, planning is done over the α-RA expression tree that results from the property path's translation, but no planning specific to the semantics of property paths takes place.

In [12], we show the approaches of FA and α-RA are, indeed, incomparable. We next describe WAVEGUIDE, which is a generalized approach to query planning and evaluation for SPARQL property-path queries, and which properly subsumes the FA and α-RA approaches. We discuss WAVEGUIDE's evaluation model, present WAVEGUIDE's plans, and we show that WAVEGUIDE's plan space properly subsumes FA $\cup$ α-RA.

WAVEGUIDE's evaluation strategy is based on an *iterative search algorithm*, and variations thereof. In WAVEGUIDE, we perform path search efficiently while *simultaneously* recognizing the path expressions. WAVEGUIDE's input is a graph database G and a *waveplan* P_Q which *guides* a number of search *wavefronts* that explore the given graph. We introduce the term *wavefront* to refer to a part of the plan that evaluates breadth-first during the evaluation. This graph exploration, driven by an iterative search procedure, is inspired by the semi-naïve bottom-up strategy used in the evaluation of linear recursive expressions based on *fixpoint*, as is done for the α operator for α-RA.

The key idea is, given a *seed*—a set of nodes in the graph from which this wavefront begins its search—as a start, to expand repeatedly the search wavefronts until no new answers are produced; i.e., we reach a fixpoint. Each search wavefront is guided by an *automaton* in the plan, a finite state machine based on an NFA. This is akin to the FA approach. Different, though, from NFAs which are used as recognizers of regular expressions on strings, wavefront automata have features directed to the evaluation of regular expressions over graphs.

In [12], we present a *cost framework* for WAVEGUIDE search, *search cost factors* that can magnify the cost (properties of the graph and of resulting pre-paths computed during evaluation), and *optimization methods* that are enabled by WPs which address the search factors, in turn.

We prototype a system [11] that implements WAVEGUIDE methodology to benchmark waveplans to study their performance. In this WAVEGUIDE system, resource-intensive tasks are delegated to POSTGRESQL via SQL and *procedural* SQL routines.

We test our implementation of WAVEGUIDE by running a collection of realistic path queries over real-world datasets YAGO2s [10] and DBPedia [3]. We generate path queries based on data patterns we identified in real-world graphs. The goal of these experiments is to verify the gains offered by WAVEGUIDE optimizations, and show that they correspond to our cost framework and analysis. Further, we design a microbenchmark to analyze the performance of the WAVEGUIDE prototype in comparison with two RDF stores: VIRTUOSO [4] and JENA TDB [2].

References

1. R. Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *Software Engineering, IEEE Transactions on*, 14(7):879–885, 1988.
2. J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83. ACM, 2004.
3. The DBpedia knowledge base. http://dbpedia.org/.
4. O. Erling and I. Mikhailov. Virtuoso: RDF Support in Native RDBMS. *Semantic Web Information Management*, 1:501, 2010.
5. S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C Recommendation. http://www.w3.org/TR/sparql11-query/, 2013.
6. K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 101–112. ACM, 2012.
7. A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.
8. E. Prud'Hommeaux, A. Seaborne, et al. SPARQL query language for RDF. *W3C Recommendation*, 15, 2008.
9. W3C: Resource Description Framework (RDF). http://www.w3.org/TR/rdf-concepts/, 2004.
10. YAGO2s: A high-quality knowledge base. http://yago-knowledge.org/resource/. Max Planck Institut Informatik.
11. N. Yakovets, P. Godfrey, and J. Gryz. WAVEGUIDE: evaluating SPARQL property path queries. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 525–528, 2015.
12. N. Yakovets, P. Godfrey, and J. Gryz. Evaluation of SPARQL Property Paths via Recursive SQL. In *Proceedings of the annual ACM SIGMOD conference (SIGMOD'16)*, San Francisco, USA, June 2016. ACM.