Foundations of SPARQL Query Optimization

Michael Schmidt, Michael Meier, Georg Lausen

Albert-Ludwigs-Universität Freiburg
– Database and Information Systems Group –

13th International Conference on Database Theory, March 2010
The RDF Data Format

"RDF"  "J. Due"

A1 → P1
creator

Article  Person
type
title
creator

type

title

"SPARQL"  "J. ACM"

“Triples of knowledge”

\[ D := \{ (A1, \text{type}, \text{Article}), \]
\[ (A1, \text{title}, \text{"RDF"}), \]
\[ (A1, \text{creator}, P1), \]
\[ (P1, \text{type}, \text{Person}), \ldots \} \]
Introduction

SPARQL: A Query Language for RDF

Basic Construct

Triples that may contain variables in positions, e.g. (?article, type, Article)
SPARQL: A Query Language for RDF

Basic Construct

Triples that may contain variables in positions, e.g. (?article, type, Article)

SPARQL Evaluation

Evaluation result described by mapping sets, e.g.
\[
[(?article, type, Article)]_D = \{\{ ?article \mapsto A1 \}, \{ ?article \mapsto A2 \}\}
\]
SPARQL: A Query Language for RDF

Basic Construct

Triples that may contain variables in positions, e.g. (\(?article, type, Article\))

SPARQL Evaluation

Evaluation result described by mapping sets, e.g.

\[
\llbracket (\text{?article, type, Article}) \rrbracket_D = \{\{ \text{?article} \mapsto A1\}, \{ \text{?article} \mapsto A2\}\}
\]

Operators to Build More Expressive Queries (will be explained later)

**SELECT, AND, FILTER, OPTIONAL, UNION**

⇒ no explicit minus operator, but negation can be simulated using a combination of operators **OPTIONAL** and **FILTER**
SPARQL Example

**SPARQL Query (Abstract Syntax)**

Select the titles of all articles and, optionally, the persons that authored the articles:

\[
Q: \\
\text{SELECT } ?\text{title} ?\text{person} \\
\text{WHERE} \{ \\
\quad ((?\text{article}, \text{type}, \text{Article}) \text{ AND} \\
\quad \quad (\text{?article}, \text{title}, ?\text{title})) \\
\quad \text{OPTIONAL } (\text{?article}, \text{creator}, \text{?person}) \\
\} \\
\]

**Solution**

\[
[Q]_D = \{ \{ ?\text{title} \mapsto \text{“RDF”}, ?\text{person} \mapsto \text{P1} \}, \\
\{ ?\text{title} \mapsto \text{“SPARQL”} \} \}
\]
## SPARQL Semantics

### SPARQL Semantics Idea

- Semantics $[.]_D$ maps SPARQL operators into algebraic operations over mapping sets
- SPARQL Algebra operations similar in idea to Relational Algebra
SPARQL Semantics

**SPARQL Semantics Idea**

- Semantics $[[.]]_D$ maps SPARQL operators into algebraic operations over mapping sets
- SPARQL Algebra operations similar in idea to Relational Algebra

**Translation of SPARQL Operators into SPARQL Algebra**

- **SELECT** $\texttt{?x}_1 \ldots \texttt{?x}_n$ **WHERE** $\{A\}$:
  - $\Rightarrow$ mapped to a projection operation $\pi_{\texttt{?x}_1 \ldots \texttt{?x}_n}([A]_D)$
SPARQL Semantics

SPARQL Semantics Idea

- Semantics $\llbracket . \rrbracket_D$ maps SPARQL operators into algebraic operations over mapping sets
- SPARQL Algebra operations similar in idea to Relational Algebra

Translation of SPARQL Operators into SPARQL Algebra

- **SELECT** $?x_1 \ldots $?x_n** WHERE** $\{A\}$:
  $\Rightarrow$ mapped to a projection operation $\pi_{?x_1 \ldots $?x_n}(\llbracket A \rrbracket_D)$

- $A$ **AND** $B$ (also denoted as “$A \cdot B$”):
  $\Rightarrow$ mapped to an algebraic join operation $\llbracket A \rrbracket_D \bowtie \llbracket B \rrbracket_D$
SPARQL Semantics

SPARQL Semantics Idea

- Semantics $\semantics \{ . \}_D$ maps SPARQL operators into algebraic operations over mapping sets
- SPARQL Algebra operations similar in idea to Relational Algebra

Translation of SPARQL Operators into SPARQL Algebra

- **SELECT** $?x_1 \ldots $?x_n WHERE $\{A\}$:
  \[ \Rightarrow \text{mapped to a projection operation } \pi_{?x_1 \ldots ?x_n}(\semantics A_D) \]
- **A AND B** (also denoted as “$A \cdot B$”):
  \[ \Rightarrow \text{mapped to an algebraic join operation } \semantics A_D \bowtie \semantics B_D \]
- **A OPTIONAL B**:
  \[ \Rightarrow \text{mapped to a left outer join } \semantics A_D \bowland \semantics B_D \]
SPARQL Semantics

SPARQL Semantics Idea
- Semantics $[.]_D$ maps SPARQL operators into algebraic operations over mapping sets.
- SPARQL Algebra operations similar in idea to Relational Algebra.

Translation of SPARQL Operators into SPARQL Algebra
- **SELECT** $?x_1 \ldots ?x_n$ **WHERE** $\{A\}$:
  $\Rightarrow$ mapped to a projection operation $\pi_{?x_1 \ldots ?x_n}(\llbracket A \rrbracket_D)$
- $A$ **AND** $B$ (also denoted as “$A \cdot B$”):
  $\Rightarrow$ mapped to an algebraic join operation $\llbracket A \rrbracket_D \bowtie \llbracket B \rrbracket_D$
- $A$ **OPTIONAL** $B$:
  $\Rightarrow$ mapped to a left outer join $\llbracket A \rrbracket_D \bowtie \llbracket B \rrbracket_D$
- $A$ **FILTER** $R$:
  $\Rightarrow$ mapped to an algebraic selection $\sigma_R(\llbracket A \rrbracket_D)$
Introduction

SPARQL Semantics

**SPARQL Semantics Idea**

- Semantics \([.]_D\) maps SPARQL operators into algebraic operations over mapping sets
- SPARQL Algebra operations similar in idea to Relational Algebra

**Translation of SPARQL Operators into SPARQL Algebra**

- **SELECT ?x_1 \ldots ?x_n WHERE \{A\}:**
  - \(\Rightarrow\) mapped to a projection operation \(\pi_{?x_1 \ldots ?x_n}(\llbracket A\rrbracket_D)\)
- **A AND B (also denoted as "A \cdot B"):**
  - \(\Rightarrow\) mapped to an algebraic join operation \(\llbracket A\rrbracket_D \bowtie \llbracket B\rrbracket_D\)
- **A OPTIONAL B:**
  - \(\Rightarrow\) mapped to a left outer join \(\llbracket A\rrbracket_D \bowtie \llbracket B\rrbracket_D\)
- **A FILTER \(R\):**
  - \(\Rightarrow\) mapped to an algebraic selection \(\sigma_R(\llbracket A\rrbracket_D)\)
- **A UNION B:**
  - \(\Rightarrow\) mapped to a set-theoretic union \(\llbracket A\rrbracket_D \cup \llbracket B\rrbracket_D\)
Contributions and Outline

Key Contributions of the Paper

- Comprehensive study of SPARQL evaluation complexity
- Novel techniques for SPARQL query optimization
  - Algebraic query optimization
  - Semantic query optimization
The SPARQL Evaluation Problem

**Evaluation Problem**

**Input:** SPARQL query $Q$, candidate mapping $\mu$, and RDF document $D$

**Question:** Is $\mu \in \llbracket Q \rrbracket_D$?

**Answer:** yes or no

**Motivation**

- Default problem to understand the complexity of query evaluation and the expressiveness/limitations of query languages
- Study of the problem for different language fragments
- Results allow to understand relations between SPARQL fragments and established query languages (such as Relational Algebra)
**Complete Classification**

**Operator Shortcuts**
- $A := \text{And}$, $O := \text{Optional}$, $F := \text{Filter}$, $U := \text{Union}$
- $\pi := \text{Select}$

**Central Results$^a$**
- Already $\text{Optional}$ alone leads to $\text{PSPACE}$-hardness
- Combinations $\text{And}/\text{Union}$ and $\text{And}/\text{Select}$ cause $\text{NP}$-hardness
- Operator $\text{Filter}$ never affects complexity

---

$^a$Results in blue from J. Pérez, M. Arenas, C. Gutierrez: Semantics and Complexity of SPARQL. In ISWC 2006.
The Source of Complexity

Operator Shortcuts

- $A := \text{And}$, $O := \text{Optional}$
- $F := \text{Filter}$, $U := \text{Union}$
- $\pi := \text{Select}$
- $\leq n := \text{expression contains at most } n \text{ nested Optional subexpressions}$

Central Result

- Nesting depth of Optional expressions fixes class in the polynomial hierarchy
Algebraic Rewritings for SPARQL

Central Results: Various Algebraic Equivalences over SPARQL Algebra

- Comprehensive investigation of algebraic laws
  - Associativity, Commutativity, Distributivity
  - Idempotence and Inverse

- Study of rewriting rules known from Relational Algebra
  - Filter Pushing
  - Projection Pushing

- SPARQL-specific rewritings
  - Rules concerning unbound variables
  - Simulated negation

- Difference between bag and set semantics

⇒ in total about 30 new equivalences over SPARQL Algebra
SPARQL Algebra vs. Relational Algebra

Question

Given the close connection between Relational and SPARQL Algebra, do the rewritings trivially follow from Relational Algebra rewritings?
SPARQL Algebra vs. Relational Algebra

Question
Given the close connection between Relational and SPARQL Algebra, do the rewritings trivially follow from Relational Algebra rewritings?

Answer
There are important discrepancies between both algebras, most notably:

**Relational Algebra**
- Defined over relations with fixed schema
- Joins over NULL values rejecting

**SPARQL Algebra**
- Defined over mapping sets with loose schema
- Joins over unbound variables accepting

⇒ profound implications that complicate SPARQL Algebra rewritings
# SPARQL Algebra vs. Relational Algebra

## Relational Algebra (SQL)

<table>
<thead>
<tr>
<th>$R_{RA}$</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_{RA}$</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NULL</td>
<td>3</td>
</tr>
</tbody>
</table>

$R_{RA} \bowtie S_{RA} = \emptyset$

## SPARQL Algebra

$R_{SA} := \{\{?A \mapsto 1, ?B \mapsto 2\}\}$

$S_{SA} := \{\{?C \mapsto 3\}\}$

$R_{SA} \bowtie S_{SA} = \{\{?A \mapsto 1, ?B \mapsto 2, ?C \mapsto 3\}\}$
Example

Candidate SPARQL Algebra Equivalence: Filter Pushing

\[(FP) \quad \sigma_{?y=d}(A \Join B) \equiv \sigma_{?y=d}(A) \Join B\]

Problem

- Assume the evaluation result of \(A\) contains \(\mu : = \{ ?x \mapsto c \}\)
- \(\mu\) never contributes to the result of the right-side expression
- **But:** \(\mu\) may contribute to the result of the left-side expression, e.g. when joined with a mapping \(\nu : = \{ ?y \mapsto d \}\) from \(B\)
Approximating Bound Variables

Approach

- Classify variables appearing in SPARQL Algebra expressions
  - *Possible variables*: overestimation for variables that might be bound in every result mapping
  - *Certain variables*: underestimation for variables that are definitely bound in every result mappings
- Classification independent from input document
Approximating Bound Variables

Approach

- Classify variables appearing in SPARQL Algebra expressions
  - *Possible variables*: overestimation for variables that might be bound in every result mapping
  - *Certain variables*: underestimation for variables that are definitely bound in every result mappings
- Classification independent from input document

Filter Pushing Revisited

If ?y is a *certain* variable of A or ?y is not a *possible* variable of B, then

\[
(FP) \quad \sigma_{?y=d}(A \bowtie B) \equiv \sigma_{?y=d}(A) \bowtie B
\]
Motivation: Semantic Query Optimization

- Constraints are vital to assert data integrity and avoid anomalies
  - Keys and functional dependencies
  - Foreign keys
  - User-defined constraints (e.g., subclass relationships)
- Semantic query optimization
  - Find queries that are equivalent on each database instance satisfying the constraints
  - Queries often exhibit non-trivial rewritings in the presence of constraints
  - Rich theory of constraint-based optimization in the Relational context has been developed
RDF Constraints in SPARQL

Representing RDF Constraints

**Idea:** encode RDF constraints as First-order Logic sentences over a ternary predicate $D(subject, predicate, object)$.

Upcoming Question

Is SPARQL a candidate for a constraint language, i.e. is it expressive enough to ...

- test if a (first-order) constraint holds on some database?
- express user-defined constraints?
RDF Constraints in SPARQL

Representing RDF Constraints

**Idea:** encode RDF constraints as First-order Logic sentences over a ternary predicate \( D(\text{subject}, \text{predicate}, \text{object}) \).

Upcoming Question

Is SPARQL a candidate for a constraint language, i.e. is it expressive enough to ...

- ... test if a (first-order) constraint holds on some database?
- ... express user-defined constraints?

For a slight extension of SPARQL, called SPARQL\(^C\), the following holds:

**Theorem**

Let \( \varphi \) be a (first-order) RDF constraint and \( D \) be an RDF document. There is a SPARQL\(^C\) query \( Q \) s.t. \( \llbracket Q \rrbracket_D = \emptyset \iff D \models \varphi. \)
Exploiting RDF Constraints for Query Optimization

- Exploit close connection between SPARQL AND-only queries and Conjunctive Queries, an important subclass of Relational Algebra.
- Fall back on established semantic optimization techniques developed in the Relational context, namely the classical Chase algorithm.
Central Results and Properties

- Provably **sound** approach to semantic query optimization for **AND**-only queries
- Also works for **AND**-connected blocks inside more complex queries
- **Completeness** depends mainly on the termination of the Chase algorithm (like in the Relation context)
- Can easily be coupled with a cost estimation function
- Complemented by a rule-based optimization approach for the remaining SPARQL operators (e.g., **OPTIONAL**)
Conclusion

Closing Remarks

- Research on SPARQL still in its infancy
- Results in the paper improve on two critical research subjects
  - **Complexity results**
    ⇒ deepen understanding of the SPARQL query language, its fragments, and interrelations to established query languages
  - **Algebraic and semantic optimization approaches**
    ⇒ lay the theoretical foundations for building efficient, scalable SPARQL engines
- An important step towards efficient Semantic Web data processing
Thank you for your attention!