SPARQL Cost-based Query Optimization

Edna Ruckhaus, Dr. María Esther Vidal, Eduardo Ruiz
Dept. Computer Science and Information Technology
Universidad Simón Bolívar
Caracas, Venezuela

Workshop “Query Optimization for the Semantic Web”
Madrid, Spain, May 2007
Our Objective

• Define techniques to execute SPARQL queries efficiently:
  – We consider basic and optional patterns, union of patterns, filters
  – Other systems translate SPARQL to Datalog (Polleres, 2006).
    • We add efficient physical operators
    • We introduce cost-based optimization techniques

http://www.ldc.usb.ve/~ruckhaus/oneql
Outline

• Motivation
• Our Approach
  – DOB
  – Physical Operators
  – Cost-Based Query Optimization
  – SPARQL to DOB
• Initial Experimental Results
• Conclusions and Future Work
Ontology Query

Agent that discovers and selects Web sources

Large amount of inferred knowledge

Explicit knowledge

Domain Ontologies

Annotated Web Sources

Inference Service
Query Languages for the Semantic Web

SPARQL

- **RDF** query language
  - Triple-based
- **Select-from-where** syntax
  - Different triple/graph patterns
- The output of a SPARQL query is
  - The **subset** of the input set of triples that satisfy the patterns
The objective is to find a plan to efficiently evaluate the input query.
Answering Web ontology queries efficiently

- Different strategies to retrieve and combine RDF triples
- Different plans to evaluate each strategy
  - Induce operations that are not relevant to the result
  - Traverse the same input multiple times

<table>
<thead>
<tr>
<th>Size of knowledge</th>
<th>Evaluation strategy</th>
<th>Order of Strategy (plan)</th>
</tr>
</thead>
</table>

Query evaluation cost

Different plans may have different cost
An example
Different Evaluation Plans

“All the different types of Faculties that have written conference papers on the Semantic Web”

- **Plan I**
  1. Subclasses of Faculty class
  2. For each subclass, its instances
  3. For each instance, if the instance has written conference papers on the Semantic Web, return the instance and the class it belongs to

- **Plan II**
  1. Instances and the most specific classes they belong to
  2. Subclasses of the Faculty class
  3. Instances that have written conference papers on the Semantic Web
  4. For each instance in 3. that belongs to a class in 2. return the instance and class
Answering Web ontology queries efficiently

Cost of evaluation plan depends on cost of computing explicit and inferred knowledge.

Cost-based optimization techniques for relational databases. (Based on statistics on the data)

Good for basic knowledge

Does not scale for inferred knowledge

Sampling techniques are good for estimating statistical characteristics where data is not known a priori
Our general approach

- We represent an ontology as a **Deductive Ontology Base** (Datalog plus built-ins) composed of a set of **extensional ground predicates** (EOB) and a **set of intensional predicates** (IOB) representing I
Our general approach

Cost model is a combination of:

1. **Predicate cost**-based reordering strategy similar to relational databases
   - Depends on cost and cardinality of each predicate
     - Cost and cardinality of basic knowledge
     - Cost and cardinality of inferred knowledge → not known a priori

2. **Adaptive Sampling** Technique (Lipton et al, 1990) for cost and cardinality of predicates that represent inferred knowledge

3. The objective function is to minimize the **number of intermediate inferences** to answer the query.
Previous work

- **DOB** formalism
- Ontologies:
  - OWL Lite, XmlOnto
- Queries
  - DOB conjunctive queries
  - RDQL queries
  - SPARQL Select queries with basic patterns
- Logical operator: Join
- Physical operators:
  - Nested-Loop, Block Nested-Loop and Hash
- Query optimization
  - Cost-based (Hybrid cost model)
  - Cost-based + Magic Sets rewriting
The OneQL system
Our formalism
Deductive Ontology Base

- A **DOB query** is a rule
  \[ q:Q(X) \leftarrow \exists Y \, B(X,Y) \]
- Predicates in the body **B** of a rule or query
  \[ a_1 \, op_1 \, ... \, op_{n-1} \, a_n \]
- **DOB algebra operators** \( op_i \):  
  - **Join** \( \Join \)
  - **Left Outer Join** \( \LeftJoin \)
  - **Union** \( \cup \)
- Each logical operator corresponds to one or more **physical operators**
  - The **cost function** for each physical operator has been defined
A **valuation** is a function $\gamma: V \rightarrow C$

The **cardinality** of $\text{areSubClasses}(S, \text{Faculty})$ corresponds to the number of valid instantiations, the **valuations that evaluate to true in the Minimal Perfect Model I of O**

$$\gamma(\text{areSubClasses}(S, \text{Faculty})) = \{\text{Lecturer, PostDoc, Professor, VisitingProfessor, AssistantProfessor, AssociateProfessor, FullProfessor, Chair, Dean, Article, Publication,...}\}$$

$$\text{Card}(\text{areSubClasses}(S, \text{Faculty})) = 9$$

The **cardinality** of a predicate $p$ is the **number of valid instantiations of $p$**
Given an ontology O and a predicate p, a proof tree may be defined for p.
The valuations needed to define all the valid instantiations in the proof tree correspond to the intermediate inferred facts of p.

The cost of a predicate p is the number of intermediate inferred facts for p.

- `areSubClasses(S, 'Faculty')` (3,980)
- `areSubClasses('Lecturer', 'Faculty')` (40)
- `areSubClasses('FullProfessor', 'Faculty')` (646)
- `subClassOf('Lecturer', 'Faculty')` (1)
- `subClassOf('FullProfessor', 'Professor')` (1)
- `subClassOf('Professor', 'Faculty')` (40)
- `subClassOf('Professor', 'Faculty')` (1)
DOB Semantics

- Correspondence \textit{valuation} $\gamma$ to mapping $\mu$

- Given \textit{predicates} $R_1$, $R_2$, \textit{patterns} $P_1$, $P_2$, \textit{sets of variables} $V_1$, $V_2$, \textit{valid instantiations} $\gamma(R_1)$, $\gamma(R_2)$ and \textit{operators} for DOB valuations $\text{op}_\gamma$

Evaluation of DOB operators:

- $(R_1 \times R_2)_I = \gamma(R_1) \times \gamma(R_2)$
- $(R_1 \mathcal{M} R_2)_I = \gamma(R_1) \mathcal{M} \gamma(R_2)$
- $(R_1 \cup R_2)_I = \gamma(R_1) \cup \gamma(R_2)$

- $\gamma(R_1) \mathcal{M} \gamma(R_2) = (\gamma(R_1) \mathcal{M} \gamma(R_2)) \cup \{(X1,\text{NULL}) | X1 \in \gamma(R1) \land \neg \exists X2 \in \gamma(R2) \land Y \in V_1 \cap V_2 \land X1.Y = X2.Y\}$
DOB physical operators

• Based on concepts of matching instantiations and join arguments
• Cost metric is the number of intermediate inferred triples
• Physical Operators:
  – Join
    • Nested-Loop Join
    • Block Nested-Loop Join
    • Hash Join
  – Left Outer Join
    • Hash Left Join
  – Union
    • Union
DOB physical operators

Hash Left
Outer Join

P₁. OPTIONAL {P₂}

Hashing function

Pattern P₁

Optional pattern P₂

Hash table

Output

null

Result
DOB physical operators

Hash Left Outer Join

- Create **direct access table RH** with set of **valid instantiations** \( \gamma(R_2) \), according to values of join arguments in \( R_2 \)
- For **each valid instantiation** \( \gamma(R_1) \)
  - Retrieve “directly” **matching instantiation** \( \gamma(R_2) \) in RH through the key value
  - If matching instantiation found return \( (\gamma(R_1), \gamma(R_2)) \)
  else return \( \gamma(R_1) \)

\[ \text{Cost}(R_1 \Join R_2) = \text{cost}(R_1) + \text{cost}(R_2) \]
Query Optimization Algorithm

- **Dynamic Programming** algorithm that traverses the space of plans
  - The space is traversed by **iterations**
    - First best orders of size one,
    - Second best orders of size two
    - ... Orders of size n, where n is the size of the input query.
  - For optional patterns $R_1 \sqsubseteq R_2$, $R_1$ **takes precedence** over $R_2$ in any ordering.
Hybrid Cost Model - Adaptive Sampling

- Identify a sample of the population such that the mean $\mu$ of a function $c$ applied to the population $u$, may be estimated with a certain error $d$ and confidence level $p$.
- **Adaptive** - Size of sample is re-estimated as elements are selected from the population
  - *Sampling efficiency vs. Precision*
- Function $c$:
  - $c$ - cost
  - $c$ - cardinality
Adaptive Sampling - Example
Estimating cardinality

\[
\text{card(areSubClasses}(S, 'Faculty'))) = \frac{1}{\text{nKeys}(S1)} \times \text{card(areSubClasses}(S, S1))
\]

- The population is all the valid instantiations for areSubClasses(S, S1)
- The population is partitioned according to any of its arguments, e.g. S
- There are N possible values for S, e.g. the number of classes in the ontology, 17
- We randomly sample m partitions (3) of the form areSubClasses('classX', S1), for instance areSubClasses('Professor', S1), areSubClasses('article', S1)...

\[
\text{card(areSubClasses}(S, S1)) = \frac{1}{3} \sum_{i=1}^{3} \text{card(areSubClasses}(c_i, S1)) \times 17 = 22.66
\]

\[
\text{card(areSubClasses}(S, 'Faculty'))) = \frac{1}{4} \times 22.66 = 5.66
\]
Extension Hybrid cost model

Estimating cardinality of Datalog built-ins

1. Datalog built-ins correspond to SPARQL built-in unary and binary functions and operators.
2. P is a unary or binary built-in:
   - Select an argument of P and partition P accordingly.
   - \( \text{card}(P) = Y \times n \), where \( n \) is the number of partitions, \( Y \) is the average partition cardinality.
   - \( n \) is the number of the argument’s datatype instances in the ontology.
   - \( \text{cost}(P) = \text{card}(P) \)
SPARQL Syntax and Semantics

Syntax
• SPARQL graph patterns defined recursively in terms of the “.”, “UNION”, “OPTIONAL” and “FILTER” language constructors.

Semantics
• Semantics defined by Gutierrez, et. al. based on an interpretation function $\parallel . \parallel_D$
  - From pattern expressions to sets of mappings.
  - Defined recursively in terms of Join, Left Outer Join and Union of sets of mappings
SPARQL-DOB translation

• **Translation** function $\lambda: P \rightarrow D$, $P$ set of valid graph patterns, $D$ set of DOB queries

• Considers cases for **triples that encode ontologies written in OWL**:
  
  – Triple where **predicate is an RDF property**, e.g. `rdf:subClassOf`, `owl:inverseOf`
  
  – Triple where **predicate is rdf:type**
    
    • Special case where **object is an RDF class**, e.g. `rdfs:Class`, `rdfs:Property`

• Translates **SPARQL built-in conditions** into **Datalog built-ins**.
SPARQL-DOB translation

- **Base cases:**
  - $\lambda(\text{BuiltInPredPattern}) = \lambda(\text{builtinPred})(\lambda(\text{subject}), \lambda(\text{object}))$
  - $\lambda(?C \text{ rdfs:subClassOf ex:Faculty}) = \text{areSubClasses}(?C, \text{ex:Faculty})$
  - $\lambda(\text{RegularPattern}) = \text{areStatements}(\lambda(\text{subject}), \lambda(\text{predicate}) \lambda(\text{object}))$
  - $\lambda(?X \text{ ex:writes ex:Article}) = \text{areStatements}(?X, \text{ex:writes, ex:Article})$

- **Inductive cases:**
  - $\lambda(\text{P}_1. \text{OPTIONAL \{P}_2\}) = \lambda(\text{P}_1) \sqcap \lambda(\text{P}_2)$
Initial Experimental Results

- **Ontologies:**
  - **Real-world**: Galen, EHR_RM
  - Synthetic

- **Experiments:**
  - Real-world - Sun-Fire V440 (1593 MHz) with 16GB RAM.
  - Synthetic - SunBlade 150 (650MHz), 1 GB RAM

- System implemented in SWI-Prolog 5.6.1

- **Cost metrics**
  - Number of intermediate inferred facts
  - Evaluation time

- **Sampling parameters:**
  - $p = 0.7$
  - $d = 0.2$
  - $k = 7$
Experimental results

Predictive capacity real-world ontologies

190-9000 domain basic facts
50-400 queries
4 subgoals
Cost of each ordering estimated and evaluated

- High correlation.
- Problems uniformity and independence assumption for some predicates.
- Estimates same order of magnitude than evaluation cost.

Correlation Galen: 0.92
Correlation EHR: 0.98
Experimental Results
Galen and EHR_RM

- Twenty-five queries
- All the orders were evaluated and compared the worst ordering against the one identified by the optimizer

Ratio of optimal cost versus the worst is less than 10%, i.e., the cost of the plan identified by the optimizer is less than the 10% of the cost of the worst ordering
Experimental Results

Cost Improvement

- Optimizer-generated ordering is better than median ordering.
- Optimizer-generated ordering is in top 10% with respect to the worst ordering.
- Cost improvement may depend on the “shape” of the query.

20 queries
4 subgoals. Evaluated 24 orderings

Cost Optimal Ordering/Cost Worst Ordering - Galen
Experimental results

Predictive capacity synthetic ontologies

- 2200 domain basic facts
- 10 ontologies
- 10 chain, 10 star queries
- 3 subgoals
- Cost each ordering estimated and evaluated

Correlation synthetic: 0.92

High correlation.
Conclusions and Future Work

• Additional **physical operators** have been defined for **optional** patterns and **union** of patterns.
• The **optimization algorithm** has been **extended** taking into account the **non-commutativity** of the “. OPTIONAL” operator.
• The **cost model** has been extended to handle **SPARQL built-in sampling**.
• In the future:
  - **Extend** the definition to **nested patterns**.
  - **Compare OneQL** with other **rule-based query engines**.

35