Scalable Multi-Query Optimization for SPARQL

Wangchao Le\textsuperscript{1} Anastasios Kementsietsidis\textsuperscript{2} Songyun Duan\textsuperscript{2} Feifei Li\textsuperscript{1}

\textsuperscript{1}University of Utah \hspace{1cm} \textsuperscript{2}IBM Research

April 6, 2012
Outline

1 Introduction
2 Preliminary
3 Our approach
4 Experiments
5 Conclusions
We are inundated with a large collection of RDF (Resource Description Framework) data.
We are inundated with a large collection of RDF (Resource Description Framework) data.

- DBpedia, Uniprot, Freebase etc

Internally ...

```
<rdf:RDF
   xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:doctors="http://example.org/doctors"
   xmlns:patients="http://example.org/patients">

   <rdf:Description rdf:about="urn:x-states:New York">
      <dcterms:alternative>NY</dcterms:alternative>
   </rdf:Description>

</rdf:RDF>
```
We are inundated with a large collection of RDF (Resource Description Framework) data.
- DBpedia, Uniprot, Freebase etc

Internally ...

```xml
<rdf:RDF
    xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
    xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="urn:x-states:New York">
        <dcterms:alternative>NY</dcterms:alternative>
    </rdf:Description>
</rdf:RDF>
```

Triple format:

```
```

subject  predicate  object
We are inundated with a large collection of RDF (Resource Description Framework) data.

- DBpedia, Uniprot, Freebase etc
- A large graph and encode rich semantics

Internally ...

```xml
<rdf:RDF
  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
  xmlns:dcterms="http://purl.org/dc/terms/">
  <rdf:Description rdf:about="urn:x-states:New York">
    <dcterms:alternative>NY</dcterms:alternative>
  </rdf:Description>
</rdf:RDF>
```

Triple format:

```
```

subject predicate object
We are inundated with a large collection of RDF (Resource Description Framework) data.

- DBpedia, Uniprot, Freebase etc
- A large graph and encode rich semantics

Internally ...

```
<rdf:RDF
  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
  xmlns:dcterms="http://purl.org/dc/terms/">
  <rdf:Description rdf:about="urn:x-states:New York">
    <dcterms:alternative>NY</dcterms:alternative>
  </rdf:Description>
</rdf:RDF>
```

Triple format:
```
```

Query language: SPARQL
Introduction

- We are inundated with a large collection of RDF (Resource Description Framework) data.
  - DBpedia, Uniprot, Freebase etc
  - A large graph and encode rich semantics
- Available engines to manage RDF data?


We are inundated with a large collection of RDF (Resource Description Framework) data.

- DBpedia, Uniprot, Freebase etc
- A large graph and encode rich semantics

Available engines to manage RDF data?

- **RDBMS**: Migrate RDF, e.g., Sesame, JenaSDB etc.
- **Generic RDF stores**: e.g., RDF3X, JenaTDB etc.


Introduction

SPARQL queries

RDF store

Q1
Q2
Q3
Q_{n-1} Q_n
- Observation: queries share common parts
- Multi-query optimization

![Diagram showing RDF store and SPARQL queries](image-url)
A tempting choice: turn to MQO in relational databases

[MQO88][MQO90][MQO00]

- SPARQL $\leftrightarrow$ relational algebra [EPS08][FSR07].
- Exist quite a few relational solutions for RDF store.

A tempting choice: turn to MQO in relational databases [MQO88][MQO90][MQO00]
- SPARQL↔relational algebra [EPS08][FSR07].
- Exist quite a few relational solutions for RDF store.

For SPARQL and RDF, new issues arise in practice.
A tempting choice: turn to MQO in relational databases [MQO88][MQO90][MQO00]

- SPARQL↔relational algebra [EPS08][FSR07].
- Exist quite a few relational solutions for RDF store.

For SPARQL and RDF, new issues arise in practice.
- Convert SPARQL to SQL: not all engines use RDBMS
- Conversion to SQL → a large number of joins
A tempting choice: turn to MQO in relational databases

- SPARQL ↔ relational algebra [EPS08][FSR07].
- Exist quite a few relational solutions for RDF store.

For SPARQL and RDF, new issues arise in practice.

- Convert SPARQL to SQL: not all engines use RDBMS
- Conversion to SQL → a large number of joins
- Store dependent solution
We focus on two types of queries
We focus on two types of queries

**Type 1:** \( Q \) := SELECT RD WHERE GP

**Type 2:** \( Q_{\text{OPT}} \) := SELECT RD WHERE GP (OPTIONAL GP\(_{\text{OPT}}\))+
We focus on two types of queries

**Type 1:** $Q := \text{SELECT RD WHERE GP}$

**Type 2:** $Q_{\text{OPT}} := \text{SELECT RD WHERE GP (OPTIONAL GP}_{\text{OPT}})^+$

(a) triple table $D$

<table>
<thead>
<tr>
<th>subj</th>
<th>pred</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>name</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>p1</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p1</td>
<td>mbox</td>
<td>alice@home</td>
</tr>
<tr>
<td>p1</td>
<td>mbox</td>
<td>alice@work</td>
</tr>
<tr>
<td>p1</td>
<td>www</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>p2</td>
<td>name</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>p2</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p3</td>
<td>name</td>
<td>&quot;Ella&quot;</td>
</tr>
<tr>
<td>p3</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p3</td>
<td>www</td>
<td><a href="http://work/ella">http://work/ella</a></td>
</tr>
<tr>
<td>p4</td>
<td>name</td>
<td>&quot;Tim&quot;</td>
</tr>
<tr>
<td>p4</td>
<td>zip</td>
<td>&quot;11234&quot;</td>
</tr>
</tbody>
</table>

(b) Example query $Q_{\text{OPT}}$

```sql
SELECT ?name
WHERE { ?x name ?name, ?x zip 10001,
}
```

(name)

"Alice"
"Bob"
"Ella"
We focus on two types of queries

**Type 1:** $Q := \text{SELECT RD WHERE GP}$

**Type 2:** $Q_{OPT} := \text{SELECT RD WHERE GP (OPTIONAL GP}_{OPT})^+$

(a) triple table $D$

<table>
<thead>
<tr>
<th>subj</th>
<th>pred</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>name</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>p1</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p1</td>
<td>mbox</td>
<td>alice@home</td>
</tr>
<tr>
<td>p1</td>
<td>mbox</td>
<td>alice@work</td>
</tr>
<tr>
<td>p1</td>
<td>www</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>p2</td>
<td>name</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>p2</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p3</td>
<td>name</td>
<td>&quot;Ella&quot;</td>
</tr>
<tr>
<td>p3</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p3</td>
<td>www</td>
<td><a href="http://work/ella">http://work/ella</a></td>
</tr>
<tr>
<td>p4</td>
<td>name</td>
<td>&quot;Tim&quot;</td>
</tr>
<tr>
<td>p4</td>
<td>zip</td>
<td>&quot;11234&quot;</td>
</tr>
</tbody>
</table>

(b) Example query $Q_{OPT}$

```
SELECT ?name, ?mail, ?hpage
WHERE { ?x name ?name, ?x zip 10001,
    OPTIONAL {?x mbox ?mail }
    OPTIONAL {?x www ?hpage } }
```
We focus on two types of queries

**Type 1:** \[ Q :\text{=} \text{SELECT RD WHERE GP} \]

**Type 2:** \[ Q_{\text{OPT}} :\text{=} \text{SELECT RD WHERE GP (OPTIONAL GP_{OPT})}^{\pm} \]

(a) triple table \( D \)

<table>
<thead>
<tr>
<th>subj</th>
<th>pred</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>name</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>p1</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p1</td>
<td>mbox</td>
<td>alice@home</td>
</tr>
<tr>
<td>p1</td>
<td>mbox</td>
<td>alice@work</td>
</tr>
<tr>
<td>p1</td>
<td>www</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>p2</td>
<td>name</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>p2</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p3</td>
<td>name</td>
<td>&quot;Ella&quot;</td>
</tr>
<tr>
<td>p3</td>
<td>zip</td>
<td>10001</td>
</tr>
<tr>
<td>p3</td>
<td>www</td>
<td><a href="http://work/ella">http://work/ella</a></td>
</tr>
<tr>
<td>p4</td>
<td>name</td>
<td>&quot;Tim&quot;</td>
</tr>
<tr>
<td>p4</td>
<td>zip</td>
<td>&quot;11234&quot;</td>
</tr>
</tbody>
</table>

(b) Example query \( Q_{\text{OPT}} \)

```
SELECT ?name , ?mail , ?hpage
WHERE { ?x name ?name , ?x zip 10001 ,
      OPTIONAL { ?x mbox ?mail }
      OPTIONAL { ?x www ?hpage }}
```

(c) Output \( Q_{\text{OPT}}(D) \)

<table>
<thead>
<tr>
<th>name</th>
<th>mail</th>
<th>hpage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>alice@home</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>&quot;Alice&quot;</td>
<td>alice@work</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Ella&quot;</td>
<td></td>
<td><a href="http://work/ella">http://work/ella</a></td>
</tr>
</tbody>
</table>
We focus on two types of queries

**Type 1:** $Q := \text{SELECT RD WHERE GP}$

**Type 2:** $Q_{opt} := \text{SELECT RD WHERE GP (OPTIONAL GP}_{opt})^+$

Problem statement.
We focus on two types of queries

**Type 1:** $Q := \text{SELECT RD WHERE GP}$

**Type 2:** $Q_{\text{OPT}} := \text{SELECT RD WHERE GP (OPTIONAL GP}_{\text{OPT}}\text{)}^+$

Problem statement.

- Input: a set $Q$ of **Type 1** queries and a data graph $G$
We focus on two types of queries

**Type 1:** \( Q := \text{SELECT RD WHERE GP} \)

**Type 2:** \( Q_{\text{OPT}} := \text{SELECT RD WHERE GP \ (OPTIONAL \ GP_{\text{OPT}})}^{+} \)

**Problem statement.**

- Input: a set \( Q \) of **Type 1** queries and a data graph \( G \)
- Output: a set of **rewritten** queries, \( Q_{\text{OPT}} \) of **Type 1** and **Type 2** queries
We focus on two types of queries

Type 1: \( Q \) := SELECT RD WHERE GP

Type 2: \( Q_{\text{OPT}} \) := SELECT RD WHERE GP (OPTIONAL GP\text{OPT})

Problem statement.

- Input: a set \( Q \) of Type 1 queries and a data graph \( G \)
- Output: a set of rewritten queries, \( Q_{\text{OPT}} \) of Type 1 and Type 2 queries
- Requirements:
  - soundness and completeness: \( Q_{\text{OPT}}(G) \equiv Q(G) \)
  - cost: \( \frac{T_r(Q)+T_e(Q_{\text{opt}})}{T_e(Q)} \leq 1 \)
Our approach

1. Introduction
2. Preliminary
3. Our approach
4. Experiments
5. Conclusions
Motivating example

(a) Query $Q_1$

(b) Query $Q_2$
Motivating example

(a) Query Q₁

(b) Query Q₂

- : constant
- : variable
Motivating example

(a) Query $Q_1$

(b) Query $Q_2$
Motivating example

(a) Query Q₁

(b) Query Q₂

SELECT *
WHERE { ?x P₁ ?z, ?y P₂ ?z,

}
Motivating example

SELECT *
WHERE { ?x P1 ?z, ?y P2 ?z,
OPTIONAL {?y P3 ?w, ?w P4 v1 }
}

(I) Structure only $Q_{OPT}$
Motivating example

SELECT *
WHERE { ?x P1 ?z, ?y P2 ?z,
    OPTIONAL {?y P3 ?w, ?w P4 v1 }
    OPTIONAL {?t P3 ?x, ?t P5 v1, ?w P4 v1 }
}

(I) Structure only $Q_{OPT}$
Motivating example

SELECT *
WHERE { ?x P1 ?z, ?y P2 ?z,
    OPTIONAL { ?y P3 ?w, ?w P4 v1 }
    OPTIONAL { ?t P3 ?x, ?t P5 v1, ?w P4 v1 }
}

OPTIONALs are evaluated on top of the common substructures
(intermediate results cached by engine).
Motivating example

(a) Query Q₁
(b) Query Q₂

<table>
<thead>
<tr>
<th>pattern p</th>
<th>( \alpha(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?x ) P₁ (?z)</td>
<td>30%</td>
</tr>
<tr>
<td>(?y ) P₂ (?z)</td>
<td>20%</td>
</tr>
<tr>
<td>(?y ) P₃ (?w)</td>
<td>18%</td>
</tr>
<tr>
<td>(?w ) P₄ (v_1)</td>
<td>1%</td>
</tr>
<tr>
<td>(?t ) P₅ (v_1)</td>
<td>2%</td>
</tr>
</tbody>
</table>

*Max common subquery is not selective

(II) Using cost in optimization
Motivating example

(a) Query Q₁
(b) Query Q₂

<table>
<thead>
<tr>
<th>pattern p</th>
<th>(\alpha(p))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?x P₁ ?z)</td>
<td>30%</td>
</tr>
<tr>
<td>(?y P₂ ?z)</td>
<td>20%</td>
</tr>
<tr>
<td>(?y P₃ ?w)</td>
<td>18%</td>
</tr>
<tr>
<td>(?w P₄ v₁)</td>
<td>1%</td>
</tr>
<tr>
<td>(?t P₅ v₁)</td>
<td>2%</td>
</tr>
</tbody>
</table>

*Max common subquery is not selective

(II) Using cost in optimization
Motivating example

SELECT *
WHERE { ?w P4 v1,
    OPTIONAL { ?x1 P1 ?z1, ?y1 P2 ?z1, ?y1 P3 ?w }
    OPTIONAL { ?x2 P1 ?z2, ?y2 P2 ?z2, ?t2 P3 ?x2, ?t2 P5 v1 }
}

(II) Using cost in optimization
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

They often do not share one common subquery
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

- Similar queries can be optimized together

Diagram:
- Partition input queries
  - Group 1
  - Group 2
  - \( \cdots \)
  - Group \( k \)
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

- Similar queries can be optimized together
- Finding structure similarity is expensive
- Group by predicates
- Distance: Jaccard similarity of predicate sets
Our approach

\[ Q = \{q_1, q_2, \ldots, q_n\} \]

Paritition input queries

Group 1

Group 2

\cdots

Group \(k\)

Rewriting

Rewriting

\cdots

Rewriting

• Similar queries can be optimized together
• Finding structure similarity is expensive
• Group by predicates
• Distance: Jaccard similarity of predicate sets
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

Paritition input queries

- Group 1
  - Rewriting

- Group 2
  - Rewriting

- \( \cdots \)
  - Rewriting

- Group \( k \)

\( \Rightarrow \) Recursively rewrite a subset of type 1 queries (hierarchically) \( \Rightarrow \) a set of type 2 queries
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

- Paritition input queries
  - Group 1
  - Group 2
  - \ldots
  - Group \( k \)
    - Rewriting
    - Rewriting
    - \ldots
    - Rewriting

  - Recursively rewrite a subset of type 1 queries (hierarchically) \( \rightarrow \) a set of type 2 queries
    - finding common edge subgraphs
    - optimizations to avoid bad efficiency
    - cost: guard against bad rewritings
    - approx. by the min selectivity in common subquery
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

Paritition input queries

Group 1 \quad \text{Group 2} \quad \cdots \quad \text{Group } k

Rewriting \quad \text{Rewriting} \quad \cdots \quad \text{Rewriting}

- Recursively rewrite a subset of type 1 queries (hierarchically) → a set of type 2 queries
  - finding common edge subgraphs
  - optimizations to avoid bad efficiency
  - cost: guard against bad rewritings
  - approx. by the min selectivity in common subquery

<table>
<thead>
<tr>
<th>pattern p</th>
<th>( \alpha(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?x P_1 ?z)</td>
<td>30%</td>
</tr>
<tr>
<td>(?y P_2 ?z)</td>
<td>20%</td>
</tr>
<tr>
<td>(?y P_3 ?w)</td>
<td>18%</td>
</tr>
<tr>
<td>(?w P_4 v_1)</td>
<td>1%</td>
</tr>
<tr>
<td>(?t P_5 v_1)</td>
<td>2%</td>
</tr>
</tbody>
</table>
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

Partition input queries

Group 1 → Rewriting → Execution

Group 2 → Rewriting → Execution

\ldots

Group \( k \) → Rewriting → Execution

- Similar queries can be optimized together
- Finding structure similarity is expensive
- Group by predicates
- Distance: Jaccard similarity of predicate sets

Rewriting

- Recursively rewrite a subset of type 1 queries
- Finding common edge subgraphs
- Optimizations to avoid bad efficiency
- Cost: guard against bad rewritings
- Approx. by the min selectivity in common subquery

Execution (hierarchically) → a set of type 2 queries
Our approach

\[ Q = \{ q_1, q_2, \ldots, q_n \} \]

Partition input queries

Group 1
Rewriting

Group 2
Rewriting

\ldots

Group \( k \)
Rewriting

Execution

Result distribution

\[ r(q_1) \quad r(q_2) \quad r(q_n) \]
Our approach

- Related issues
Our approach

- Related issues
  - Distributing results, *i.e.*, **Type 2** query $\rightarrow$ **Type 1** queries

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>

RD of a **Type 1** query: *e.g.*, $X$ and $Z$

↑↓

columns from results of the **Type 2** rewriting
Our approach

- Related issues
  - Distributing results, *i.e.*, **Type 2 query** → **Type 1 queries**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>

RD of a **Type 1 query**: e.g., X and Z

↑↓
columns from results of the **Type 2 rewriting**

- Soundness and completeness
Related issues

Distributing results, \textit{i.e.}, \textbf{Type 2 query} $\rightarrow$ \textbf{Type 1 queries}

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>g</td>
</tr>
</tbody>
</table>

RD of a \textbf{Type 1} query: e.g., $X$ and $Z$

\[\uparrow\downarrow\]

columns from results of the \textbf{Type 2} rewriting

Soundness and completeness

Extensibility of the solution: more general queries

- handle variable predicates
- \textit{OPTIONAL} queries
Experiments

- Implementation highlights
  - C++
  - 64-bit Linux, 2GHz Xeon(R) CPU, 4GB memory
Experiments

- Implementation highlights
  - C++
  - 64-bit Linux, 2GHz Xeon(R) CPU, 4GB memory
- Dataset
  - Extend LUBM benchmark generator: randomness in structure, variances of sel.
Experiments

- Implementation highlights
  - C++
  - 64-bit Linux, 2GHz Xeon(R) CPU, 4GB memory

- Dataset
  - Extend LUBM benchmark generator: randomness in structure, variances of sel.

- RDF stores: Jena TDB 0.85 etc
Experiments

- Implementation highlights
  - C++
  - 64-bit Linux, 2GHz Xeon(R) CPU, 4GB memory
- Dataset
  - Extend LUBM benchmark generator: randomness in structure, variances of sel.
- RDF stores: Jena TDB 0.85 etc
- Queries
  - Ensure randomness in structure, e.g., star, chain and circle
Experiments

- Implementation highlights
  - C++
  - 64-bit Linux, 2GHz Xeon(R) CPU, 4GB memory

- Dataset
  - Extend LUBM benchmark generator:
    randomness in structure, variances of sel.

- RDF stores: Jena TDB 0.85 etc

- Queries

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset size</td>
<td>D</td>
<td>4M</td>
<td>3M to 9M</td>
</tr>
<tr>
<td>Number of queries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size of query (num of triple patterns)</td>
<td></td>
<td>60 to 160</td>
<td>60 to 160</td>
</tr>
<tr>
<td>Number of seed queries</td>
<td>κ</td>
<td>6</td>
<td>5 to 10</td>
</tr>
<tr>
<td>Size of seed queries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max selectivity of patterns in Q</td>
<td>α_{max}(Q)</td>
<td>random</td>
<td>0.1% to 4%</td>
</tr>
<tr>
<td>Min selectivity of patterns in Q</td>
<td>α_{min}(Q)</td>
<td>1%</td>
<td>0.1% to 4%</td>
</tr>
</tbody>
</table>
Experiments

- Implementation highlights
  - C++
  - 64-bit Linux, 2GHz Xeon(R) CPU, 4GB memory

- Dataset
  - Extend LUBM benchmark generator: randomness in structure, variances of sel.
  - RDF stores: Jena TDB 0.85 etc

- Queries

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset size</td>
<td>(D)</td>
<td>4M</td>
<td>3M to 9M</td>
</tr>
<tr>
<td>Number of queries</td>
<td>(</td>
<td>Q</td>
<td>)</td>
</tr>
<tr>
<td>Size of query (num of triple patterns)</td>
<td>(</td>
<td>Q</td>
<td>)</td>
</tr>
<tr>
<td>Number of seed queries</td>
<td>(\kappa)</td>
<td>6</td>
<td>5 to 10</td>
</tr>
<tr>
<td>Size of seed queries</td>
<td>(</td>
<td>q_{cmn}</td>
<td>)</td>
</tr>
<tr>
<td>Max selectivity of patterns in Q</td>
<td>(\alpha_{max}(Q))</td>
<td>random</td>
<td>0.1% to 4%</td>
</tr>
<tr>
<td>Min selectivity of patterns in Q</td>
<td>(\alpha_{min}(Q))</td>
<td>1%</td>
<td>0.1% to 4%</td>
</tr>
</tbody>
</table>

- Rewriting w/ structure: **MQO-S**; rewriting w/ structure and cost: **MQO**
Experiments

- **Time on rewriting**
  - **MQO-S-C**: structure based rewriting
  - **MQO-C**: rewriting integrating with cost
Experiments

- Time on rewriting
  - MQO-S-C: structure based rewriting
  - MQO-C: rewriting integrating with cost

![Graph showing time on rewriting](image)

*Costly/bad rewritings are rejected → more rounds of comparisons.
Experiments

- Time on distributing results
  - MQO-S-P: parsing results from MQO-S
  - MQO-P: parsing results with MQO
Experiments

- Time on distributing results
  - MQO-S-P: parsing results from MQO-S
  - MQO-P: parsing results with MQO

*Non-selective common subqueries increase the set of results.*
Experiments

- Time on distributing results
  - MQO-S-P: parsing results from MQO-S
  - MQO-P: parsing results with MQO

*Non-selective common subqueries increase the set of results.

*Both rewriting and parsing are efficiently doable
Experiments

- Varying num of queries in a batch
  - No-MQO: no optimization
  - MQO-S: optimization based on structural rewriting
  - MQO: integrating cost
Experiments

- Varying num of queries in a batch
  - No-MQO: no optimization
  - MQO-S: optimization based on structural rewriting
  - MQO: integrating cost

*Both reduce the num of queries to be executed*
Experiments

- Varying num of queries in a batch
  - No-MQO: no optimization
  - MQO-S: optimization based on structural rewriting
  - MQO: integrating cost

![Graph showing time in seconds for varying query sizes with different optimization methods.]
Experiments

- Varying min. selectivity in seed queries
  - **No-MQO**: no optimization
  - **MQO-S**: optimization based on structural rewriting
  - **MQO**: integrating cost
Experiments

- Varying min. selectivity in seed queries
  - **No-MQO**: no optimization
  - **MQO-S**: optimization based on structural rewriting
  - **MQO**: integrating cost

*MQO: reject more bad rewritings; MQO-S: not sensitive*
Experiments

- Varying min. selectivity in seed queries
  - **No-MQO**: no optimization
  - **MQO-S**: optimization based on structural rewriting
  - **MQO**: integrating cost

![Graph showing the relationship between \( \alpha_{min}(q_{cmn}) \) and time (seconds) for No-MQO, MQO-S, and MQO.](image)
• Varying seed size
  MQO-S: optimization based on structural rewriting
  MQO: integrating cost
  percentage = $\frac{T_e(\text{common subquery})}{T_e(Q_{opt})} \times 100\%$
Experiments

- Varying seed size
  MQO-S: optimization based on structural rewriting
  MQO: integrating cost
  percentage = \( \frac{T_e(\text{common subquery})}{T_e(Q_{opt})} \times 100\% \)

*MQO-S: up to 25% time on optional*
Conclusions

- In dealing RDF data on the Web, store independency is important.
- Combining SPARQL language and graph algorithms can achieve MQO, i.e., by rewriting queries.
- Cost must be taken in consideration during rewriting.
Thank You

Q and A
Our approach

- Partition input queries
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting

Distance: Jaccard similarity on predicates
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting

Distance: Jaccard similarity on predicates
- Represent each query as a set of predicates.
- Measure the similarity of a pair of queries by set similarity
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting

Distance: Jaccard similarity on predicates
- Represent each query as a set of predicates.
- Measure the similarity of a pair of queries by set similarity
- Grouping: k-means
Our approach

- Partition input queries
  - Object: similar queries can be optimized together in rewriting

Distance: Jaccard similarity on predicates
- Represent each query as a set of predicates.
- Measure the similarity of a pair of queries by set similarity
- Grouping: k-means
Hierarchical rewriting and clustering (inside a group)
Our approach

- Hierarchical rewriting and clustering (inside a group)
  Rewrite pairs of queries bottom up
Our approach

- Hierarchical rewriting and clustering (inside a group)
  Rewrite pairs of queries bottom up
Our approach

- Hierarchical rewriting and clustering (inside a group)
  
  Rewrite pairs of queries bottom up
  Pair up queries with max Jaccard similarity
Hierarchical rewriting and clustering (inside a group)

Rewrite pairs of queries bottom up
Pair up queries with max Jaccard similarity
Our approach

- Hierarchical rewriting and clustering (inside a group)
  
  Rewrite pairs of queries bottom up
  Pair up queries with max Jaccard similarity

- Rewriting → finding maximal common triple patterns
- In the language of graph . . .
Our approach

- Hierarchical rewriting and clustering (inside a group)
  - Rewrite pairs of queries bottom up
  - Pair up queries with max Jaccard similarity

- Rewriting → finding maximal common triple patterns
- In the language of graph ...

Our approach

- Hierarchical rewriting and clustering (inside a group)
  - Rewrite pairs of queries bottom up
  - Pair up queries with max Jaccard similarity

- Rewriting → finding maximal common triple patterns
- In the language of graph . . .

  - maximal common connected edge subgraphs
Our approach

- Hierarchical rewriting and clustering (inside a group)
  
  Rewrite pairs of queries bottom up
  Pair up queries with max Jaccard similarity

- Rewriting $\rightarrow$ finding maximal common triple patterns
- In the language of graph . . .


  - maximal common connected edge subgraphs
    $\rightarrow$ maximal common connected *induced* sugraphs in linegraphs
Our approach

- Hierarchical rewriting and clustering (inside a group)
  
  Rewrite pairs of queries bottom up
  Pair up queries with max Jaccard similarity

- Rewriting $\rightarrow$ finding maximal common triple patterns

- In the language of graph $\ldots$


  - maximal common connected edge subgraphs
    $\rightarrow$ maximal common connected *induced* subgraphs in linegraphs
    $\rightarrow$ maximal cliques in the product graph
Our approach

(a) Query $Q_1$  
(b) Query $Q_2$  
(c) Query $Q_3$  
(d) Query $Q_4$
Our approach

- Linegraph: invert vertices and edges
Our approach

- Linegraph: invert vertices and edges
- sub—sub: $\ell_0$, sub—obj: $\ell_1$, obj—sub: $\ell_2$, obj—obj: $\ell_3$
Our approach

- Linegraph: invert vertices and edges
  - sub–sub: $\ell_0$, sub–obj: $\ell_1$, obj–sub: $\ell_2$, obj–obj: $\ell_3$

- Product graph: simultaneous walk
Our approach

- **Linegraph:** invert vertices and edges
- **sub-sub:** $\ell_0$, **sub–obj:** $\ell_1$, **obj–sub:** $\ell_2$, **obj–obj:** $\ell_3$

- **Product graph:** simultaneous walk

(a) Query $Q_1$  (b) Query $Q_2$  (c) Query $Q_3$  (d) Query $Q_4$

(a) $\mathcal{L}(Q_1)$  (b) $\mathcal{L}(Q_2)$  (c) $\mathcal{L}(Q_3)$  (d) $\mathcal{L}(Q_4)$
Our approach

- **Linegraph**: invert vertices and edges
- **sub–sub:** $\ell_0$, **sub–obj:** $\ell_1$, **obj–sub:** $\ell_2$, **obj–obj:** $\ell_3$

- **Product graph**: simultaneous walk

---

The triangle (clique) highlights the common subgraph composed by

---

graph query pattern 1 graph query pattern 2

---

The triangle (clique) highlights the common subgraph composed by ■ x ●
Our approach

- Linegraph: invert vertices and edges
- sub–sub: $\ell_0$, sub–obj: $\ell_1$, obj–sub: $\ell_2$, obj–obj: $\ell_3$

- product graph: simultaneous walk
- blowup in size, esp. > 2 queries affect clique detection

The triangle (clique) highlights the common subgraph composed by graph query pattern 1 and graph query pattern 2.
Our approach

- Linegraph: invert vertices and edges
- sub–sub: \( \ell_0 \), sub–obj: \( \ell_1 \), obj–sub: \( \ell_2 \), obj–obj: \( \ell_3 \)

- product graph: simultaneous walk
- blowup in size, esp. > 2 queries affect clique detection
- optimize the product graph

The triangle (clique) highlights the common subgraph composed by □ × ●
Our approach

- Linegraph: invert vertices and edges
- \( \text{sub-sub: } \ell_0, \text{sub-obj: } \ell_1, \text{obj-sub: } \ell_2, \text{obj-obj: } \ell_3 \)

- product graph: simultaneous walk
- blowup in size, esp. > 2 queries affect clique detection
- optimize the product graph

- prune non-common predicates
- check the constants
Our approach

- Linegraph: invert vertices and edges
- sub–sub: $\ell_0$, sub–obj: $\ell_1$, obj–sub: $\ell_2$, obj–obj: $\ell_3$

- product graph: simultaneous walk
- blowup in size, esp. $>2$ queries affect clique detection
- optimize the product graph

- prune non-common predicates
- check the constants
- prune vertices with non-common neighborhoods
Our approach

- Linegraph: invert vertices and edges
- \( \text{sub-sub:}\ell_0, \text{sub-obj:}\ell_1, \text{obj-sub:}\ell_2, \text{obj-obj:}\ell_3 \)

- product graph: simultaneous walk
- blowup in size, esp. \( > 2 \) queries affect clique detection
- optimize the product graph

- prune non-common predicates
- check the constants
- prune vertices with non-common neighborhoods

\[ \mathcal{L}(G_{P_p}): \]
\[ \ell_3 \quad \ell_3 \]
\[ P_1 \quad P_2 \]

\[ S: \]
\[ P_3 \quad P_4 \]
Our approach

- Find maximal cliques in the product graph [CLQ02][CLQ03]


Our approach

- Find maximal cliques in the product graph [CLQ02][CLQ03]


Our approach

- Find maximal cliques in the product graph [CLQ02][CLQ03]
  


- Integrate cost into rewriting
Our approach

- Find maximal cliques in the product graph [CLQ02][CLQ03]
  
  

- Integrate cost into rewriting
  
  - **Structure**: maximize size of the common subquery in a rewriting
  - Evaluation on cost: guard against bad rewritings
Our approach

- Find maximal cliques in the product graph [CLQ02][CLQ03]
  
  

- Integrate cost into rewriting
  
  - **Structure**: maximize size of the common subquery in a rewriting
  
  - Evaluation on cost: guard against bad rewritings
  
  - Measure: min selectivity in the common subquery for approximation
Our approach

- Find maximal cliques in the product graph \[\text{[CLQ02][CLQ03]}\]

- Integrate cost into rewriting
  - **Structure**: maximize size of the common subquery in a rewriting
  - Evaluation on cost: guard against bad rewritings
  - Measure: min selectivity in the common subquery for approximation
  - **Cost**: discard bad rewritings, keep good ones in hierarchical rewriting
Our approach

- Find maximal cliques in the product graph [CLQ02][CLQ03]
  
  

- Integrate cost into rewriting
  
  **Structure**: maximize size of the common subquery in a rewriting
  
  **Evaluation on cost**: guard against bad rewritings
  
  **Measure**: min selectivity in the common subquery for approximation
  
  **Cost**: discard bad rewritings, keep good ones in hierarchical rewriting
Our approach

- Find maximal cliques in the product graph \[\text{[CLQ02][CLQ03]}\]
  
  

- Integrate cost into rewriting
  
  - **Structure**: maximize size of the common subquery in a rewriting
  - **Evaluation on cost**: guard against bad rewritings
  - **Measure**: min selectivity in the common subquery for approximation
  - **Cost**: discard bad rewritings, keep good ones in hierarchical rewriting

![Diagram showing the rewritings process with selectivity drops](image-url)
Our approach

- Related issues
Our approach

- Related issues
  - Distributing results, *i.e.*, **Type 2 query**→**Type 1 queries**

<table>
<thead>
<tr>
<th>name</th>
<th>mail</th>
<th>hpage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>alice@home</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>alice@work</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>&quot;Ella&quot;</td>
<td></td>
<td><a href="http://work/ella">http://work/ella</a></td>
</tr>
</tbody>
</table>

RD of a **Type 1 query**

↑↓

columns from results of the **Type 2 rewriting**
Our approach

- Related issues
  - Distributing results, i.e., **Type 2 query** → **Type 1 queries**

<table>
<thead>
<tr>
<th>name</th>
<th>mail</th>
<th>hpage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>alice@home</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>&quot;Alice&quot;</td>
<td>alice@work</td>
<td><a href="http://home/alice">http://home/alice</a></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Ella&quot;</td>
<td></td>
<td><a href="http://work/ella">http://work/ella</a></td>
</tr>
</tbody>
</table>

RD of a **Type 1 query**

↑↓

columns from results of the **Type 2 rewriting**

- Soundness and completeness
Our approach

- Related issues
  - Distributing results, \textit{i.e.}, \textbf{Type 2} query $\rightarrow$ \textbf{Type 1} queries
    
    \begin{tabular}{|l|l|l|}
    \hline
    name & mail & hpage \\
    \hline
    "Alice" & alice@home & http://home/alice \\
    "Alice" & alice@work & http://home/alice \\
    "Bob" & & \\
    "Ella" & & http://work/ella \\
    \hline
    \end{tabular}

    RD of a \textbf{Type 1} query
    \begin{align*}
    \uparrow \downarrow
    \end{align*}

    columns from results of the \textbf{Type 2} rewriting

  - Soundness and completeness
  - Extensibility of the solution: more general queries
    \begin{itemize}
    \item handle variable predicates
    \item nested OPTIONALs
    \end{itemize}