Anytime Query Answering in RDF through Evolutionary Algorithms

Eyal Oren  Christophe Guéret  Stefan Schlobach

Vrije Universiteit Amsterdam

ISWC 2008
Problem: query answering over large RDF graphs

- find variable assignments, such that the data graph entails the substituted query graph

Requirements:

- approximate (ranking more important than perfect)
- anytime (streaming answers improving over time)
- parallel (scale computation with added nodes)

Evolutionary approach:

- guess complete variable assignment
- verify solution on the data graph
- improve assignment and repeat
Example

Listing 1: RDF data

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix opus: <http://datavelopment.net/opus#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

dblp:ullman89 rdf:type opus:Book .
dblp:ullman88 opus:author dblp:ullman .
dblp:ullman foaf:homepage ... .
```

Listing 2: SPARQL query

```sparql
SELECT ?title WHERE {
  ?publication rdf:type opus:Book .
}
```

Listing 3: Expected answer

```python
(?
```
Traditional approach

... WHERE { ?publication rdf:type opus:Book; rdfs:label ?title . }

   ?publication... 
dblp:ullman88
dblp:ullman89

   ?publication... | ?title
   dblp:ullman88 | "Principles of ..."

3. Join both result tables, project the result
   (?title, "Principles of ...")

- Solving clauses: iterate over all records
- Solving joins: nested loops, etc.
Our approach

... WHERE { ?publication rdf:type opus:Book; rdfs:label ?title . }

1. Assign some value to each query variable
   (?publication, dblp:ullman88)
   (?title, opus:Book)

2. Verify solution (substitute into query, verify on graph)

<table>
<thead>
<tr>
<th>triple</th>
<th>correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp:ullman88 rdf:type opus:Book</td>
<td>yes</td>
</tr>
<tr>
<td>dblp:ullman88 rdfs:label opus:Book</td>
<td>no</td>
</tr>
</tbody>
</table>

3. If the solution is OK, stop. Otherwise, try again.

- The loop may be stopped at any time
- A result may satisfy a part of the query
- Uses membership testing instead of lookup
- Implementation uses dictionary encoding
Requirements

- A good way to evolve answers
  - results should improve with each iteration
  - choice: evolutionary algorithm
- A fast method to verify answers
  - we will try many solutions
  - verification should not require iteration
  - choice: Bloom filters
Evolutionary algorithms
Binary Bloom filters

- Compact representation: eg. set of $n = 8$ bits

- Supports two operations
  - $\text{INSERT(key)}$: insert a key into the filter
  - $\text{CONTAINS(key)}$: test for the presence of a key

- Use eg. $k = 3$ hash functions to compute a set of bits from a key

\[
\begin{align*}
\text{hash1(“Hello World”)}&=8 \\
\text{hash2(“Hello World”)}&=6 \\
\text{hash3(“Hello World”)}&=3
\end{align*}
\]
Binary Bloom filters

- **insert(“Hello World”)**
  - Current
    - OR
    - “Hello World”
    - =
  - New

- **Bit-wise OR operation**
  - Always successful

- **contains(“Bonjour”)**
  - Current
    - AND
    - “Bonjour”
    - =
  - Result

- **Bit-wise AND operation**
  - Positive result may be a collision:
    \[ p_{\text{error}} = \left(1 - e^{-\frac{kn}{m}}\right)^k \]
Basic operations

- Parsing: construct Bloom filter TRIPLE, and construct domain of candidate values
- Initial population: for each variable $v_i$, select some value $d_i$ from domain
- Verify solution: substitute all selected values, verify each clause on the Bloom filter
- Fitness: number of satisfied clauses
- Evolution: one-point cross-over, random mutation
Optimising representation

- Prevent useless assignments: construct more domains \((s, p, o)\) to select values depending on variable position in the query
- Improve granularity of fitness function: construct more Bloom filters \((spo, sp, po, so)\) to reward partial solutions
Graph parsing

- Each triple is inserted into 4 Bloom filters
  
  dblp:ullman88  rdf:type  opus:Book
Graph parsing

- Each triple is inserted into 4 Bloom filters

  \[\text{dblp:ullman88} \quad \text{rdf:type} \quad \text{opus:Book}\]

  \[\text{ullman_type_Book}\]

  \[\text{SPO}\]
Each triple is inserted into 4 Bloom filters

dblp:ullman88

dblp:ullman

rdf:type

dblp:ullman

opus:Book

ullman_type

 ullman_type.Book

ullman_type.Book

ullman_type

ullman_type

SPO

SP

Domains of candidates are constructed

s: {dblp:ullman88, b1, ullman}
p: {rdf:type, rdfs:label, dblp:author, homepage}
o: {opus:Book, "Principles of ...", b1, dblp:ullman}
(and pairwise intersections)
Graph parsing

- Each triple is inserted into 4 Bloom filters
  - `dblp:ullman88`
  - `rdf:type`
  - `opus:Book`

Domains of candidates are constructed

- `s`: `{dblp:ullman88, b1, ullman}`
- `p`: `{rdf:type, rdfs:label, dblp:author, homepage}`
- `o`: `{opus:Book, “Principles of...”, b1, dblp:ullman}`

(and pairwise intersections)
Graph parsing

- Each triple is inserted into 4 Bloom filters
  - dblp:ullman88
  - rdf:type
  - opus:Book

Domains of candidates are constructed:
- \( s: \{ \text{dblp:ullman88, b1, ullman} \} \)
- \( p: \{ \text{rdf:type, rdfs:label, dblp:author, homepage} \} \)
- \( o: \{ \text{opus:Book, "Principles of ...", b1, dblp:ullman} \} \)
  (and pairwise intersections)
Graph parsing

- Each triple is inserted into 4 Bloom filters
  - `dblp:ullman88`
  - `rdf:type`
  - `opus:Book`

- Domains of candidates are constructed

  - `s: {dblp:ullman88, b1, ullman}`
  - `p: {rdf:type, rdfs:label, dblp:author, homepage}`
  - `o: {opus:Book, "Principles of ...", b1, dblp:ullman}`
  (and pairwise intersections)
Query parsing

... WHERE { ?publication rdf:type opus:Book; rdfs:label ?title . }

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Filter name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ?publication rdf:type opus:Book</td>
<td>spo</td>
</tr>
<tr>
<td>2 ?publication rdf:type</td>
<td>sp</td>
</tr>
<tr>
<td>3 rdf:type opus:Book</td>
<td>pe</td>
</tr>
<tr>
<td>4 ?publication opus:Book</td>
<td>so</td>
</tr>
<tr>
<td>5 ?publication rdfs:label ?title</td>
<td>spo</td>
</tr>
<tr>
<td>6 ?publication rdfs:label</td>
<td>sp</td>
</tr>
<tr>
<td>7 rdfs:label ?title</td>
<td>po</td>
</tr>
<tr>
<td>8 ?publication ?title</td>
<td>so</td>
</tr>
</tbody>
</table>

Table: Translation of SPARQL query into constraints

<table>
<thead>
<tr>
<th>?publication</th>
<th>rdf:type</th>
<th>opus:Book</th>
<th>rdfs:label</th>
<th>?title</th>
</tr>
</thead>
</table>

Figure: Encoding template for individuals
Fitness evaluation

(a) candidate solution

<table>
<thead>
<tr>
<th>dblp:ullman</th>
<th>rdf:type</th>
<th>opus:Book</th>
<th>rdfs:label</th>
<th>&quot;Principles...&quot;</th>
</tr>
</thead>
</table>

(b) evaluating against Bloom filters

<table>
<thead>
<tr>
<th>constraint</th>
<th>filter</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. dblp:ullman rdf:type opus:Book</td>
<td>spo</td>
<td>false</td>
</tr>
<tr>
<td>2. dblp:ullman rdf:type</td>
<td>sp</td>
<td>false</td>
</tr>
<tr>
<td>4. dblp:ullman opus:Book</td>
<td>so</td>
<td>false</td>
</tr>
<tr>
<td>5. dblp:ullman rdfs:label &quot;Principles...&quot;</td>
<td>spo</td>
<td>false</td>
</tr>
<tr>
<td>6. dblp:ullman rdfs:label</td>
<td>sp</td>
<td>false</td>
</tr>
<tr>
<td>7. rdfs:label &quot;Principles...&quot;</td>
<td>po</td>
<td>true</td>
</tr>
<tr>
<td>8. dblp:ullman &quot;Principles...&quot;</td>
<td>so</td>
<td>false</td>
</tr>
</tbody>
</table>

(c) constraint violations

<table>
<thead>
<tr>
<th>variables</th>
<th>?publication</th>
<th>?title</th>
</tr>
</thead>
<tbody>
<tr>
<td>violation</td>
<td>1 2 4 5 6 8 5 8</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation results

(a) DBLP5k

(b) DBLP500k

(c) LUBM

(d) FOAF

Figure: Best fitness over different datasets
Improvements

- C. Gueřet et al., SUM 2008
- Improved evolutionary operators (mutation, fitness)
- Improved implementation (memory, speed)

![Graph (a) DBLP](image1.png)
![Graph (b) FOAF](image2.png)
Conclusion

- Evolutionary technique for RDF query answering
  - encoding, operators, fitness function
  - fast verification: Bloom filters
  - good evolution: not trivial
- Ongoing:
  - diverse answers: taboo search
  - rest of SPARQL: formal translation
- Future:
  - dealing with unordered domain: SAWing weights
  - change evolutionary technique (eg swarm optimisation)
  - evaluation data for top-k queries: fitness vs. usefulness
- Benefits: anytime, approximate, parallel