SPARQLing Kleene – Fast Property Paths in RDF-3X

Andrey Gubichev, TU Munich
Stephan Seufert, MPI
Srikanta Bedathur, IIIT-Delhi

June 23, 2013
Motivation

- RDF data is a graph
- SPARQL 1.1 has introduced the property paths
- `select * where {Munich yago:isLocatedIn* ?place }`
- What entities are reached from Munich via `yago:isLocatedIn`?
Motivation

- RDF data is a graph
- SPARQL 1.1 has introduced the property paths
  
  ```sparql
  select * where {Munich yago:isLocatedIn* ?place }
  ```

- What entities are reached from Munich via yago:isLocatedIn?
- We could use joins and unions over the triple store to answer it
- Can we do better with a bit of indexing?
Semantics of Property Paths

- Originally, one could also count the number of paths between start and end point
- However, this semantics leads to \#P-hard problems (M.Arenas, WWW’12)
- Now, W3C standard only allows to check for reachability, not counting paths
Previous Work: RDF-3X

- a triple store
- extensive indexing
- join ordering with Dynamic Programming
- accurate cardinality estimation for common types of queries
- T. Neumann et al, SIGMOD 2009
Previous Work: Reachability Index FERRARI

- FERRARI index: based on tree interval labeling, assigns exact and approximate labels to nodes (ICDE’2013)
- Runtime: use index plus limited DFS
- FERRARI:
  - indexes 100 Mln triples of YAGO in 90 sec
  - takes 210 Mb
  - answers a reachability query for (start,end) in microseconds
- (all the numbers: off-the-shelf laptop)
Our Contribution

How to use FERRARI in RDF-3X

- Query optimization
- Runtime technique to speed up query execution
QO: Getting the Logical Operator

Property path triple may correspond to:

- a filter (if one of subject or object is constant)
  ```
  select * where {Munich yago:isLocatedIn* ?place }
  ```
- a scan, if one of subject or object is not bound
  ```
  select * where {?city yago:isLocatedIn* ?place }
  ```
- a join, otherwise
  ```
  select * where {?city yago:isLocatedIn* ?place.
  ?city hasName "Munich".
  ?place type ?type. }
  ```

In the last case, there is one more join opportunity (reflected in the Query Graph)
QO: Plan generation

In order to use Dynamic Programming, we extend the cost model

- Estimated cardinality of the scan is provided by the index immediately
- Cardinality estimation for the join: independence assumption + index information
select ?city ?p ?type where {
  ?city hasName "Munich".
}

\[ R (c, o) \]

\[ MJ_{c_1 = c_2} \]

\[ \times^R (c, o) \]

index scan \[ PS (c_1, \text{name}, \text{Munich}) \]

index scan \[ POS (c_2, \text{population}, ?p) \]

individual triple patterns are very unselective

we can pass gap information between different index scans, so that most part of the data can be skipped (indirectly)
Runtime: A typical execution plan

```
select ?city ?p ?type where {
  ?city hasName "Munich".
  ?city locatedIn*/type ?type.
}
```

```
\bowtie R (\exists c, o)
\bowtie MJ c_1 = c_2
  \bowtie index scan PS (?c_1, name, Munich)
  \bowtie index scan POS (?c_2, population, ?p)
  \bowtie index scanMJ
```

- Individual triple patterns are very unselective
- We can pass gap information between different index scans, so that most part of the data can be skipped (indirectly)
- (With some restrictions) this idea extends to Reachability Joins
Sideways Information Passing for Property Paths

**Build phase:** construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

![Diagram with logical expression and data structures]
Sideways Information Passing for Property Paths

**Build phase:** construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

$$\forall_{x_1=x_2}^{RJ}$$
Sideways Information Passing for Property Paths

**Build phase:** construct domain filters for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloome filter to the right index scan; it can skip values

\[
\mathcal{R}_J \times^{x_1 = x_2} \\
\]

\[
\begin{array}{c}
\phantom{0}3 \\
\phantom{0}4 \\
\end{array} \\
\begin{array}{c}
\phantom{0}x_1 \\
\phantom{0}x_2 \\
\end{array} \\
\]

**FERRARI Index**

<table>
<thead>
<tr>
<th>ID</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>4</td>
<td>[8, 8], [9, 9]</td>
</tr>
</tbody>
</table>

**Domain for \( ?o \)**

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>011000</td>
</tr>
</tbody>
</table>

hash function: \( v \mod 7 \)
Sideways Information Passing for Property Paths

**Build phase:** construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

\[
\forall_{x_1 = x_2} \times_{RJ}^{x_1} \times_{RJ}^{x_2}
\]

**FERRARI Index**

<table>
<thead>
<tr>
<th>ID</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>4</td>
<td>[8, 8], [9, 9]</td>
</tr>
</tbody>
</table>

Domain for \(?o\)

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>011000</td>
</tr>
</tbody>
</table>

hash function: \(v \mod 7\)
Sideways Information Passing for Property Paths

**Build phase:** construct domain filters for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

![Diagram of relational algebra expression]

### FERRARI Index

<table>
<thead>
<tr>
<th>ID</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>4</td>
<td>[8, 8], [9, 9]</td>
</tr>
</tbody>
</table>

### Domain for ?o

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>011000</td>
</tr>
</tbody>
</table>

Hash function: \( v \mod 7 \)
Sideways Information Passing for Property Paths

**Build phase:** construct *domain filters* for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

---

FERRARI Index

<table>
<thead>
<tr>
<th>ID</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>4</td>
<td>[8, 8], [9, 9]</td>
</tr>
</tbody>
</table>

Domain for ?\(o\)

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>011000</td>
</tr>
</tbody>
</table>

Hash function: \(v \mod 7\)
Sideways Information Passing for Property Paths

**Build phase:** construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

FERRARI Index

<table>
<thead>
<tr>
<th>ID</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>4</td>
<td>[8, 8], [9, 9]</td>
</tr>
</tbody>
</table>

Domain for $\ominus$

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>011000</td>
</tr>
</tbody>
</table>

hash function: $v \mod 7$
Sideways Information Passing for Property Paths

**Build phase:** construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom filter (1024 bytes)</th>
</tr>
</thead>
</table>

**Probe phase:** pass the bloom filter to the right index scan; it can skip values

$RJ_{x_1=x_2}$

$x_1$

3

4

$x_2$

1

3

4

6

8

FERRARI Index

<table>
<thead>
<tr>
<th>ID</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>4</td>
<td>[8, 8], [9, 9]</td>
</tr>
</tbody>
</table>

Domain for $o$

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>Bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>011000</td>
</tr>
</tbody>
</table>

hash function: $v \mod 7$
How to formulate interesting queries to test property path support? What are the hard things?

- Choosing the right build part
- Compare cardinalities of different property paths
- Compare cardinalities of property paths vs index scans

We suggested some queries and evaluated our solution (against Virtuoso)
Conclusions

We have:

- Support for property paths in RDF-3X
- Full-fledged system: query optimization, sideways information passing
- Choke points and queries and evaluation

Future Work:

- Updates