Property Path Query in SPARQL 1.1

Worarat Krathu    Guohui Xiao

Institute of Information Systems, Vienna University of Technology

July 2012
Overview

Introduction

Limitation of navigational capabilities in SPARQL 1.0
SPARQL 1.1 property path
Experiments on Evaluation and Counting

Complexity

Evaluation Complexity
Counting Complexity

Conclusion
Outline

Introduction
Limitation of navigational capabilities in SPARQL 1.0
SPARQL 1.1 property path
Experiments on Evaluation and Counting

Complexity
Evaluation Complexity
Counting Complexity

Conclusion
Outline

Introduction

Limitation of navigational capabilities in SPARQL 1.0

- SPARQL 1.1 property path
- Experiments on Evaluation and Counting

Complexity

- Evaluation Complexity
- Counting Complexity

Conclusion
SPARQL 1.0 provides limited navigational capabilities

Example Query

```
SELECT ?x
WHERE
{
  ?y :name "Axel" .
}
```
Property Path: in SPARQL 1.0 Query

SPARQL 1.0 provides limited navigational capabilities

Example Query

```sparql
SELECT ?x
WHERE
{
  ?y :name "Axel" .
}
```
Property Path: in SPARQL 1.0 Query

SPARQL 1.0 provides limited navigational capabilities

Example Query

```sparql
SELECT ?x
WHERE
{
  ?x (:knows)* ?y . # Property Path in SPARQL 1.1
  ?y :name "Axel" .
}
```
Outline

Introduction

Limitation of navigational capabilities in SPARQL 1.0
SPARQL 1.1 property path
Experiments on Evaluation and Counting

Complexity

Evaluation Complexity
Counting Complexity

Conclusion
### Property Path Syntax in SPARQL 1.1

<table>
<thead>
<tr>
<th>Syntax Form</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iri</code></td>
<td>An IRI. A path of length one.</td>
</tr>
<tr>
<td><code>^elt</code></td>
<td>Inverse path (object to subject).</td>
</tr>
<tr>
<td><code>!iri</code> or `!(iri_{1}...</td>
<td>iri_{n})`</td>
</tr>
<tr>
<td></td>
<td><code>!eri</code> is short for <code>!(iri)</code>.</td>
</tr>
<tr>
<td><code>!^iri</code> or `!(iri_{1}...</td>
<td>iri_{j}^</td>
</tr>
<tr>
<td></td>
<td><code>!iri</code> is short for <code>!(iri)</code>.</td>
</tr>
<tr>
<td><code>(elt)</code></td>
<td>A group path <code>elt</code>, brackets control precedence.</td>
</tr>
<tr>
<td><code>(elt_{1}) / (elt_{2})</code></td>
<td>A sequence path of <code>elt_{1}</code> followed by <code>elt_{2}</code>.</td>
</tr>
<tr>
<td>`(elt_{1})</td>
<td>(elt_{2})`</td>
</tr>
<tr>
<td><code>elt^{*}</code></td>
<td>A path of zero or more occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt^{+}</code></td>
<td>A path of one or more occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt^{?}</code></td>
<td>A path of zero or one occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt\{n, m\}</code></td>
<td>A path of between <code>n</code> and <code>m</code> occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt\{n\}</code></td>
<td>A path of exactly <code>n</code> occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt\{n, \}</code></td>
<td>A path of <code>n</code> or more occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt\{, n\}</code></td>
<td>A path of between 0 and <code>n</code> occurrences of <code>elt</code>.</td>
</tr>
</tbody>
</table>

`elt`: is a path element, which may itself be composed of path syntax constructs.
A property path is a possible route through a graph between two graph nodes.

Main interesting problems:

- Evaluation – Is there a path from 0 to 6? - Yes!
- Counting – How many different paths between 0 to 6? - 4 paths (i.e. aceg, acfh, bdeg, and bdfh)
Property Path

Property Path (SPARQL 1.1 Spec. [Harris et al., 2012])

A property path is a possible route through a graph between two graph nodes.

Main interesting problems:

- **Evaluation** – Is there a path from 0 to 6? - Yes!
- **Counting** – How many different paths between 0 to 6? - 4 paths (i.e. aceg, acfh, bdeg, and bdfh)
Property Path

Property Path (SPARQL 1.1 Spec. [Harris et al., 2012])
A property path is a possible route through a graph between two graph nodes.

Main interesting problems:
- Evaluation – Is there a path from 0 to 6? - Yes!
- Counting – How many different paths between 0 to 6? - 4 paths (i.e. aceg, acfh, bdeg, and bdfh)
Property Path

A property path is a possible route through a graph between two graph nodes.

Main interesting problems:
▶ Evaluation – Is there a path from 0 to 6? - Yes !
▶ Counting – How many different paths between 0 to 6? - 4 paths (i.e. aceg, acfh, bdeg, and bdfh)
A property path is a possible route through a graph between two graph nodes.

Main interesting problems:
- Evaluation – Is there a path from 0 to 6? - Yes !
- Counting – How many different paths between 0 to 6? - 4 paths (i.e. acesg, acfh, bdeg, and bdfh)
A property path is a possible route through a graph between two graph nodes.

Main interesting problems:
- Evaluation – Is there a path from 0 to 6? - Yes!
- Counting – How many different paths between 0 to 6? - 4 paths (i.e. aceg, acfh, bdeg, and bdfh)
A property path is a possible route through a graph between two graph nodes.

Main interesting problems:
- Evaluation – Is there a path from 0 to 6? - Yes!
- Counting – How many different paths between 0 to 6? - 4 paths (i.e. aceg, acfh, bdeg, and bdfh)
Outline

Introduction
    Limitation of navigational capabilities in SPARQL 1.0
    SPARQL 1.1 property path
    Experiments on Evaluation and Counting

Complexity
    Evaluation Complexity
    Counting Complexity

Conclusion
Experiments on Evaluation

[Figure: Evaluation time for Jena and RDF::Query.]

[Figure: Evaluation time for Sesame.]

The language defined by an expression $r$, denoted by $L(r)$, is inductively defined as follows: $L(\varepsilon) = \{\varepsilon\}; L(a) = \{a\}$ for any letter $a$. A regular path expression is a regular expression that has the power to branch out in the graph. Path expressions are defined as follows:

1. $L(\varepsilon) = \{\varepsilon\}$
2. $L(a) = \{a\}$ for any letter $a$
3. $L(s \cdot r) = L(s) \cdot L(r)$
4. $L(s + r) = L(s) \cup L(r)$
5. $L(\neg s) = \overline{L(s)}$
6. $L(\exists a \cdot s) = \{x \mid \exists y \in L(s) \cdot x \cdot y = a\}$

A regular expression $r$ is a regular path expression if and only if it matches a regular expression $\sum \cdot L(r)$. The following discussion is mainly interested in the following problems:

- Example 1: The polynomial-time algorithm for RDF::Query is annotated with the binary relation that we compute for it. Finally, the relation for the expression (Fig. 2(a)) is annotated with the binary relation $\sum \cdot L(r)$. We show that the double exponential behavior we observed in the literature several times, e.g., as Lemma 1 in [34], on p. 7 in [2], and in [3].

\[
L(\varepsilon) = \{\varepsilon\}; L(a) = \{a\}; L(s \cdot r) = L(s) \cdot L(r); L(s + r) = L(s) \cup L(r); L(\neg s) = \overline{L(s)}; L(\exists a \cdot s) = \{x \mid \exists y \in L(s) \cdot x \cdot y = a\}.
\]
Experiments on Counting

[ Losemann and Martens, 2012]

SELECT * WHERE { :a0 (p)* :a1 }

Figure: Time in seconds for processing the queries w.r.t. the clique size n (time axis in log-scale)

Figure: a) Clique with 4 nodes, b) RDF graph representing a clique with 4 nodes
Outline

Introduction

Limitation of navigational capabilities in SPARQL 1.0
SPARQL 1.1 property path
Experiments on Evaluation and Counting

Complexity

Evaluation Complexity
Counting Complexity

Conclusion
Outline

Introduction
- Limitation of navigational capabilities in SPARQL 1.0
- SPARQL 1.1 property path
- Experiments on Evaluation and Counting

Complexity
- Evaluation Complexity
- Counting Complexity

Conclusion
Property Path Evaluation

- In the experiment, the evaluation algorithm is shown as double exponential behavior.
- This depends on which semantics that algorithm relies on:
  - Regular path
  - Simple walk (or simple path and simple cycle): a path that does not visit the same node twice, but is allowed to return to its first node (cycle).
- Under the semantics of regular path, the evaluation can be improved to polynomial-time.
- Under the simple path, the evaluation is intractable.
In the experiment, the evaluation algorithm is shown as double exponential behavior. This depends on which semantics that algorithm relies on:

- Regular path
- Simple walk (or simple path and simple cycle): a path that does not visit the same node twice, but is allowed to return to its first node (cycle).

Under the semantics of regular path, the evaluation can be improved to polynomial-time.

Under the simple path, the evaluation is intractable.
Property Path Evaluation

- In the experiment, the evaluation algorithm is shown as double exponential behavior.
- This depends on which semantics that algorithm relies on:
  - Regular path
  - Simple walk (or simple path and simple cycle): a path that does not visit the same node twice, but is allowed to return to its first node (cycle).
- Under the semantics of regular path, the evaluation can be improved to polynomial-time.
- Under the simple path, the evaluation is intractable.
Property Path Evaluation

- In the experiment, the evaluation algorithm is shown as double exponential behavior.
- This depends on which semantics that algorithm relies on:
  - Regular path
  - Simple walk (or simple path and simple cycle): a path that does not visit the same node twice, but is allowed to return to its first node (cycle).
- Under the semantics of regular path, the evaluation can be improved to polynomial-time.
- Under the simple path, the evaluation is intractable.
Simple Path vs. Regular Path

Simple path
A simple path in a graph is a sequence of nodes such that each node in a path occurs exactly once.

Regular path (path)
A path in a graph is a sequence of nodes such that from each of its nodes there is an edge to the next node in the sequence.

Find paths from $x$ to $z$.

<table>
<thead>
<tr>
<th>Path</th>
<th>$a_4$</th>
<th>$a_1, a_2, a_3$</th>
<th>$a_1, a_2, a_5, a_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simple?</td>
<td>✓</td>
<td>✓</td>
<td>× (x visited twice)</td>
</tr>
</tbody>
</table>

Property Path Query in SPARQL 1.1
Evaluation under Regular Path Semantics

(a) Part of a run on the expression \((b + c)^*b^3.5\) and the graph in Fig. 2(b).

(b) An edge-labeled graph.

Under the semantics of regular path, the evaluation can be done in polynomial-time by using dynamic programming approach.

The Complexity of Evaluation

The complexity of evaluation under regular path semantics is in polynomial time (PTIME).

Figure: Illustration of polynomial-time dynamic programming algorithm
Evaluation under Regular Path Semantics

(a) Part of a run on the expression \((b + c)^*b^{3,5}\) and the graph in Fig. 2(b).

(b) An edge-labeled graph.

**Figure:** Illustration of polynomial-time dynamic programming algorithm

- Under the semantics of regular path, the evaluation can be done in polynomial-time
  - by using dynamic programming approach

The Complexity of Evaluation

The complexity of evaluation under regular path semantics is in polynomial time (PTIME).
Evaluation under Regular Path Semantics

(a) Part of a run on the expression \((b + c)^3\) and the graph in Fig. 2(b).

(b) An edge-labeled graph.

Figure: Illustration of polynomial-time dynamic programming algorithm

- Under the semantics of regular path, the evaluation can be done in polynomial-time
  - by using dynamic programming approach

The Complexity of Evaluation

The complexity of evaluation under regular path semantics is in polynomial time (PTIME).
Definition: ZeroOrMorePath

An arbitrary length path \( P = (X \ (path)^* Y) \) is all solutions from \( X \) to \( Y \) by repeated use of \( path \) such that any nodes in the graph are traversed once only. \( ZeroOrMorePath \) includes \( X \).

Definition: OneOrMorePath

An arbitrary length path \( P = (X \ (path)^+ Y) \) is all solutions from \( X \) to \( Y \) by repeated use of \( path \) such that any nodes in the graph are traversed once only. This does not include \( X \), unless repeated evaluation of the path from \( X \) returns to \( X \).

**Figure:** Simple Path in SPARQL 1.1 Last Call
Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems

Path evaluation in SPARQL 1.1 → Even length simple path → Path via a node in digraph → Directed subgraph homeomorphism → 3-SAT
Path evaluation in SPARQL 1.1
[ Losemann and Martens, 2012 ]

Path evaluation under simple walk (simple path and simple cycle) semantics is **NP-complete** for the expression \((aa)^*\) and for the expression \((aa)^+\)

Let 0 and 1 be distinct symbols in $\Sigma$. FIXED REGULAR PATH$(R)$, in which is either (1) $(00)^*$, or (2) $0^*10^*$ is NP-complete.

Proof. of (1)

- EVEN PATH is shown to be NP-complete
- We can reduce even path to FIXED REGULAR PATH$(R)$, where $R = (00)^*$ as follows
Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems

Path evaluation in SPARQL 1.1 → Even length simple path → Path via a node in digraph → Directed subgraph homeomorphism → 3-SAT


Given a digraph $D = (V, A)$ and $s, t \in V$, it is **NP-complete** to decide whether there is an even-length simple path from $s$ to $t$
Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems

Path evaluation in SPARQL 1.1 → Even length simple path → Path via a node in digraph → Directed subgraph homeomorphism → 3-SAT

Path via a node in digraph [Lapaugh and Papadimitriou, 1984]
Path via a node problem, is NP-complete: Given a digraph $D = (V, A)$ and $s, t, m \in V$, is there a simple path from $s$ to $t$ via $m$?
Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems

Directed subgraph homeomorphism [Fortune et al., 1980]

For each $P$ not in $C$ the *fixed subgraph homeomorphism problem* with pattern $P$ is **NP-complete**

$C = \cdots$ or $\cdots$

In the reduction of Directed subgraph homeomorphism to Path via a node problem, we use $P = \leftarrow \leftarrow$
Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems

Path evaluation in SPARQL 1.1 → Even length simple path → Path via a node in digraph → Directed subgraph homeomorphism → 3-SAT

3-SAT

3-SAT is well-know NP-complete.

The reduction of 3-SAT to Directed subgraph homeomorphism is very complicated [Fortune et al., 1980].
Outline

Introduction
- Limitation of navigational capabilities in SPARQL 1.0
- SPARQL 1.1 property path
- Experiments on Evaluation and Counting

Complexity
- Evaluation Complexity
- Counting Complexity

Conclusion
9.3 Cycles and Duplicates

SPARQL property paths treat the RDF triples as a directed, possibly cyclic, graph with named edges. Evaluation of a property path expression can lead to duplicates in the results. The property paths are equivalent to their translation into triple patterns and SPARQL UNION graph patterns, with the addition of operators for negated property paths, zero-length paths and arbitrary length paths. Any variables introduced in the equivalent pattern are not part of the results and are not already used elsewhere. They are hidden by implicit projection of the results to just the variables given in the query.

```
@prefix : <http://example/> .
:z :p :y .
:y :p :x .
```

```
PREFIX : <http://example/> SELECT *
{ :z :p* ?o }
```

giving results of:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The order of results in these examples is not significant.
Example of Counting

SELECT * WHERE { :0 (p)* :1 }
on clique(n) = { (: i p : j) | 0 ≤ i, j ≤ n, i ≠ j }

Solution: \{ \{ 0, 0, \ldots, 0 \} \}
duplicates of \( S_n \) times

<table>
<thead>
<tr>
<th>length</th>
<th>p*</th>
<th>path</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p</td>
<td>01</td>
<td>( P_3^0 = 1 )</td>
</tr>
<tr>
<td>2</td>
<td>pp</td>
<td>021, 031, 041</td>
<td>( P_3^1 = 3 )</td>
</tr>
<tr>
<td>3</td>
<td>ppp</td>
<td>0231, 0241, 0321, 0341, 0421, 0431</td>
<td>( P_3^2 = 6 )</td>
</tr>
<tr>
<td>4</td>
<td>pppp</td>
<td>02341, 02431, 03241, 03421, 04231, 04321</td>
<td>( P_3^3 = 6 )</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td></td>
<td>( S_5 = 16 )</td>
</tr>
</tbody>
</table>

Table: Compute \( S_5 \). Recall that \( P^k_n = n(n-1) \ldots (n-k+1) \)

\[ S_{n+1} = P^0_{n-1} + P^1_{n-1} + \ldots + P^{n-1}_{n-1} > P^{n-1}_{n-1} = (n-1)! \]
Example of Counting

\[ \text{SELECT * WHERE } \{ \text{:0 (p)* :1} \} \]
on \( \text{clique}(n) = \{ (\text{:i p :j}) | 0 \leq i, j \leq n, i \neq j \} \)

Solution: \( \{ \{ \}, \{ \}, \ldots, \{ \} \} \)
duplicates of \( S_n \) times

<table>
<thead>
<tr>
<th>length</th>
<th>( p^* )</th>
<th>path</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p</td>
<td>01</td>
<td>( P_3^0 = 1 )</td>
</tr>
<tr>
<td>2</td>
<td>pp</td>
<td>021, 031, 041</td>
<td>( P_3^1 = 3 )</td>
</tr>
<tr>
<td>3</td>
<td>ppp</td>
<td>0231, 0241, 0321, 0341, 0421, 0431</td>
<td>( P_3^2 = 6 )</td>
</tr>
<tr>
<td>4</td>
<td>pppp</td>
<td>02341, 02431, 03241, 03421, 04231, 04321</td>
<td>( P_3^3 = 6 )</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td></td>
<td>( S_5 = 16 )</td>
</tr>
</tbody>
</table>

Table: Compute \( S_5 \). Recall that \( P_n^k = n(n-1) \ldots (n-k+1) \)

\[ S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)! \]
### Example of Counting

**SELECT * WHERE { :0 (p)* :1 }**  
on *clique*(\(n\)) = \{ (: i \ p \ : j) \mid 0 \leq i, j \leq n, i \neq j \}  

**Solution:** \{\{ [], [], \ldots, [] \} \}  

**Duplicates of \(S_n\) times**

<table>
<thead>
<tr>
<th>length</th>
<th>(p^*)</th>
<th>path</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(p)</td>
<td>01</td>
<td>(P_3^0 = 1)</td>
</tr>
<tr>
<td>2</td>
<td>(pp)</td>
<td>021, 031, 041</td>
<td>(P_3^1 = 3)</td>
</tr>
<tr>
<td>3</td>
<td>(ppp)</td>
<td>0231, 0241, 0321, 0341, 0421, 0431</td>
<td>(P_3^2 = 6)</td>
</tr>
<tr>
<td>4</td>
<td>(pppp)</td>
<td>02341, 02431, 03241, 03421, 04231, 04321</td>
<td>(P_3^3 = 6)</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td></td>
<td>(S_5 = 16)</td>
</tr>
</tbody>
</table>

**Table:** Compute \(S_5\). Recall that \(P_n^k = n(n - 1) \ldots (n - k + 1)\)

\[
S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n - 1)!
\]
Example of Counting

```sparql
SELECT * WHERE { :0 (p)* :1 }
on clique(n) = \{ (: i p : j) \mid 0 \leq i, j \leq n, i \neq j \}
```

Solution: \{\{ \}, \{ \}, \ldots, \{ \} \}

duplicates of \( S_n \) times

<table>
<thead>
<tr>
<th>length</th>
<th>p*</th>
<th>path</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p</td>
<td>01</td>
<td>( P^0_3 = 1 )</td>
</tr>
<tr>
<td>2</td>
<td>pp</td>
<td>021, 031, 041</td>
<td>( P^1_3 = 3 )</td>
</tr>
<tr>
<td>3</td>
<td>ppp</td>
<td>0231, 0241, 0321, 0341, 0421, 0431</td>
<td>( P^2_3 = 6 )</td>
</tr>
<tr>
<td>4</td>
<td>pppp</td>
<td>02341, 02431, 03241, 03421, 04231, 04321</td>
<td>( P^3_3 = 6 )</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td></td>
<td>( S_5 = 16 )</td>
</tr>
</tbody>
</table>

Table: Compute \( S_5 \). Recall that \( P^n_k = n(n-1) \ldots (n-k+1) \)

\[
S_{n+1} = P^0_{n-1} + P^1_{n-1} + \ldots + P^{n-1}_{n-1} > P^{n-1}_{n-1} = (n-1)!
\]
Example of Counting

SELECT * WHERE { :0 (p)* :1 }
on clique(n) = \{ (: i p : j) | 0 \leq i, j \leq n, i \neq j \}

Solution: \{ \{ , , \ldots , \} \}
duplicates of \( S_n \) times

<table>
<thead>
<tr>
<th>length</th>
<th>p*</th>
<th>path</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p</td>
<td>01</td>
<td>( P_3^0 = 1 )</td>
</tr>
<tr>
<td>2</td>
<td>pp</td>
<td>021, 031, 041</td>
<td>( P_3^1 = 3 )</td>
</tr>
<tr>
<td>3</td>
<td>ppp</td>
<td>0231, 0241, 0321, 0341, 0421, 0431</td>
<td>( P_3^2 = 6 )</td>
</tr>
<tr>
<td>4</td>
<td>pppp</td>
<td>02341, 02431, 03241, 03421, 04231, 04321</td>
<td>( P_3^3 = 6 )</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td></td>
<td>( S_5 = 16 )</td>
</tr>
</tbody>
</table>

Table: Compute \( S_5 \). Recall that \( P_n^k = n(n-1) \ldots (n-k+1) \)

\[
S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)!
\]
Nesting of $\ast$

$s = 1$: SELECT * WHERE { :a0 (p)$\ast$ :a1 }
$s = 2$: SELECT * WHERE { :a0 ((p)$\ast$)$\ast$ :a1 }
$s = 3$: SELECT * WHERE { :a0 (((p)$\ast$)$\ast$)$\ast$ :a1 }

<table>
<thead>
<tr>
<th>s</th>
<th>n</th>
<th>COUNTCLIQUE($p_s, n$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1806</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>56931605</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$&gt; 10^{23}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>n</th>
<th>COUNTCLIQUE($p_s, n$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>418576</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>$&gt; 10^{23}$</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$&gt; 10^{93}$</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>28278702465</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>$&gt; 10^{53}$</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>$&gt; 10^{269}$</td>
</tr>
</tbody>
</table>

**Figure:** Number of occurrences of the mapping in the answer to property-path triple ($a_1, p_s, a_n$) over RDF graph $clique(n)$ [Arenas et al., 2012]
Data Complexity of Counting

Data complexity [Losemann and Martens, 2012, Arenas et al., 2012]
Counting in SPARQL 1.1 Draft is \( \#P \)-complete for the expressions \( a^* \) and \( a^+ \).

\#P Complexity Class

- The class of function problems of the form "compute \( f(x) \)," where \( f \) is the number of accepting paths of an NP machine.
- The canonical \( \#P \)-complete problem is \( \#\text{SAT} \).
- More difficult than NP, thus intractible

Proof of \( \#P \)-completeness

- \( \#P \)-membership. The non-deterministic TM simply guesses a path of a certain length and tests whether it matches
- \( \#P \)-hardness. Reductions from \( \#\text{DNF} \)
An Existential Semantics to the Rescue

- the core of this problem the necessity of counting different paths.
- Existential Semantics used in Graph DB, XML is tractable
- Possible solution: Discarding duplicates from the standard
- `SELECT DISTINCT * WHERE { ... }`

Figure: Experiment with Existential Semantics
Outline

Introduction
  - Limitation of navigational capabilities in SPARQL 1.0
  - SPARQL 1.1 property path
  - Experiments on Evaluation and Counting

Complexity
  - Evaluation Complexity
  - Counting Complexity

Conclusion
Conclusion

- The property path in SPARQL 1.1 query can make it intractable
- Two requirements makes property queries difficult
  - Simple path requirement
  - Duplicates in path counting
- Possible solutions:
  - Avoid simple path requirement (like XPATH)
  - Existential Semantics
  - Only count paths for some specific number of occurrence (e.g. shortest paths)
Conclusion

- The property path in SPARQL 1.1 query can make it intractable
- Two requirements makes property queries difficult
  - Simple path requirement
  - Duplicates in path counting
- Possible solutions:
  - Avoid simple path requirement (like XPATH)
  - Existential Semantics
  - Only count paths for some specific number of occurrence (e.g. shortest paths)
Conclusion

- The property path in SPARQL 1.1 query can make it intractable
- Two requirements makes property queries difficult
  - Simple path requirement
  - Duplicates in path counting
- Possible solutions:
  - Avoid simple path requirement (like XPATH)
  - Existential Semantics
  - Only count paths for some specific number of occurrence (e.g. shortest paths)
Conclusion

- The property path in SPARQL 1.1 query can make it intractable
- Two requirements makes property queries difficult
  - Simple path requirement
  - Duplicates in path counting
- Possible solutions:
  - Avoid simple path requirement (like XPATH)
  - Existential Semantics
  - Only count paths for some specific number of occurrence (e.g. shortest paths)
Conclusion

- The property path in SPARQL 1.1 query can make it intractable
- Two requirements makes property queries difficult
  - Simple path requirement
  - Duplicates in path counting
- Possible solutions:
  - Avoid simple path requirement (like XPATH)
  - Existential Semantics
  - Only count paths for some specific number of occurrence (e.g. shortest paths)
Latest Stories in Editors’ Draft

- Simple path requirements in ZeroOrMorePath (*), OneOrMorePath (+) are removed
- http://lists.w3.org/Archives/Public/public-rdf-dawg-comments/2012Apr/0004.html

The changes from the current Last Call working draft are as follows:

- The semantics of *, +, and ? are changed to be non-counting (they no longer preserve duplicates)
- The /, |, and ! remain unchanged as in the current draft (they preserve duplicates)
- The curly brace forms -- {n}, {n,m}, {n}, {,m} -- have all been removed
9.4 Arbitrary Length Path Matching

Connectivity between the subject and object by a property path of arbitrary length path can be found using the "zero or more" property path operator, `*`, and the "one or more" property path operator, `+`. There is also a "zero or one" connectivity property path operator, `?`.

For example, finding all the the possible types of a resource, including supertypes of resources, can be achieved with:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x ?type
{ ?x rdf:type/rdfs:subClassOf* ?type
}
```

Similarly, finding all the people :x connects to via the foaf:knows relationship,

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://example/>
SELECT ?person
{ :x foaf:knows+ ?person
}
```

Such connectivity matching does not introduce duplicates (it does not incorporate any count of the number of ways the connection can be made) even if the repeated path itself would otherwise result in duplicates.

The graph matched may include cycles. Connectivity matching is defined so that matching cycles does not lead to undefined or infinite results.
References

Arenas, M., Conca, S., and Pérez, J. (2012). Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In WWW '12, pages 629–638, New York, NY, USA. ACM.


