Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard

Marcelo Arenas    Sebastián Conca    Jorge Pérez

PUC-Chile, Universidad de Chile
SPARQL 1.0 provides limited navigational capabilities
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE
{
  ?Y :name "Maria" .
}
```
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE
{
  ?Y :name "Maria" .
}
```
SPARQL 1.0 provides limited navigational capabilities

SELECT ?X
WHERE
{
  ?X (:friendOf)* ?Y .
  ?Y :name "Maria".
}
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE {
  ?X (:friendOf)* ?Y .
  ?Y :name "Maria" .
} ← SPARQL 1.1 property path
```
SPARQL 1.0 provides limited navigational capabilities

SELECT ?X
WHERE
{
    ?X (:friendOf)* ?Y .
    ?Y :name "Maria" .
} ← SPARQL 1.1 property path
SPARQL 1.1 implementations had a poor performance

Data:
- *cliques* (complete graphs) of different size
- from 2 nodes (87 bytes) to 13 nodes (970 bytes)

RDF clique with 4 nodes (127 bytes)
SPARQL 1.1 implementations had a poor performance

SELECT * WHERE { :a0 (:p)* :a1 }
Poor performance with real Web data of small size

Data:
- Social Network data given by foaf:knows links
- Crawled from Axel Polleres’ foaf document (3 steps)
- Different documents, deleting some nodes
Poor performance with real Web data of small size

SELECT * WHERE { axel:me (foaf:knows)* ?x }
Poor performance with real Web data of small size

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>5.13</td>
<td>75.70</td>
<td>313.37</td>
<td>–</td>
</tr>
<tr>
<td>10.9KB</td>
<td>8.20</td>
<td>325.83</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>11.4KB</td>
<td>65.87</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>13.2KB</td>
<td>292.43</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

(time in seconds, timeout = 1hr)
Poor performance with real Web data of small size

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>5.13</td>
<td>75.70</td>
<td>313.37</td>
<td>–</td>
</tr>
<tr>
<td>10.9KB</td>
<td>8.20</td>
<td>325.83</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>11.4KB</td>
<td>65.87</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>13.2KB</td>
<td>292.43</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

(time in seconds, timeout = 1hr)

Is this a problem of these particular implementations?
We show that this is a problem of the specification

Any implementation that follows SPARQL 1.1 standard is doomed to show the same behavior
We show that this is a problem of the specification

Any implementation that follows SPARQL 1.1 standard is doomed to show the same behavior

Technical contributions:

- Experimental study of property paths
- Complete study of the complexity of path evaluation
- Identification of the main sources of complexity (counting!)
- Proposal for a semantics with efficient evaluation
We show that this is a problem of the specification

Any implementation that follows SPARQL 1.1 standard is doomed to show the same behavior

Technical contributions:

- Experimental study of property paths
- Complete study of the complexity of path evaluation
- Identification of the main sources of complexity (counting!)
- Proposal for a semantics with efficient evaluation

Impact on W3C standard:

- Normative semantics of SPARQL 1.1 property paths will be changed to overcome the issues raised in our paper
Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard

Marcelo Arenas  Sebastián Conca  Jorge Pérez

PUC-Chile, Universidad de Chile
Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard would have Prevented?

Marcelo Arenas    Sebastián Conca    Jorge Pérez

PUC-Chile, Universidad de Chile
Outline

Motivation

Experimental results

Complexity results

Existential semantics: a proposal
Property paths match regular expressions, but also count!

Property paths: regular expressions (/, |, *)

```
SELECT ?X
WHERE { :a (:p)* ?X }
```
Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, *)

```
SELECT ?X
WHERE { :a (:p)* ?X }
```

- :a
- :b
- :c
- :d
- :d
Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, *)

```
SELECT ?X
WHERE { :a (:p)* ?X }
```

- :a
- :b
- :c
- :d
- :d
Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, *)

```
SELECT ?X
WHERE { :a (:p)* ?X }
```

![](image)
Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, *)

SELECT ?X
WHERE { :a (:p)* ?X }

?X
: a
: b
: c
: d
: c
: d
Property paths match regular expressions, but also *count*!

Property paths: regular expressions (/, |, *)

```
SELECT ?X
WHERE { :a (:p)* ?X }
```

But what if we have cycles?
SPARQL 1.1 document provides a special procedure to handle cycles (and make the count)

Evaluation of \textit{path}*  

\begin{quote}
\textit{``the algorithm extends the multiset of results by one application of \textit{path}. If a node has been visited for \textit{path}, it is not a candidate for another step. A node can be visited multiple times if different paths visit it.''}
\end{quote}

SPARQL 1.1 Last Call (Jan 2012)
SPARQL 1.1 document provides a special procedure to handle cycles (and make the count)

Evaluation of *path*

"the algorithm extends the multiset of results by one application of *path*. If a node has been visited for *path*, it is not a candidate for another step. A node can be visited multiple times if different paths visit it."

SPARQL 1.1 Last Call (Jan 2012)

- W3C document provides a procedure (*ArbitraryLengthPath*)
- We formalize this procedure in the paper
Counting the number of solutions...

Data: Clique of size $n$

\[ \{ :a0 (:p)* :a1 \} \]

every solution is a copy of the empty mapping (| | in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

$$\{ :a0 (:p)^* :a1 \}$$

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* (| | in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* (\(\|\|\) in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

\[
\{ :a_0 (:p)^* :a_1 \} \quad \{ :a_0 ((:p)^*)^* :a_1 \}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
<th>$n$</th>
<th># Sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
<td>6</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* (⊥ ⊥ in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

\[
\begin{align*}
\{ :a0 (:p)* :a1 \} & \quad \{ :a0 ((:p)*)* :a1 \} & \quad \{ :a0 (((:p)*)*)* :a1 \}
\end{align*}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
<th>$n$</th>
<th># Sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
<td>6</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* (\| \| in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

\[
\begin{align*}
\{ :a0 (:p)* :a1 \} & \quad \{ :a0 ((:p)*)* :a1 \} & \quad \{ :a0 (((:p)*)*)* :a1 \}
\end{align*}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
<th>$n$</th>
<th># Sol.</th>
<th>$n$</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
<td>4</td>
<td>305</td>
<td>4</td>
<td>–</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
<td>5</td>
<td>418,576</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>–</td>
<td>6</td>
<td>–</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* ($\bot \bot$ in ARQ)
More on counting the number of solutions...

Data: foaf links crawled from the Web

\{
\text{axel:me (foaf:knows)^* ?x}
\}
More on counting the number of solutions...

Data: foaf links crawled from the Web

\[ \{ \text{axel:me} \ (\text{foaf:knows})^* \ ?x \} \]

<table>
<thead>
<tr>
<th>File</th>
<th># URIs</th>
<th># Sol.</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>38</td>
<td>29,817</td>
<td>2MB</td>
</tr>
<tr>
<td>10.9KB</td>
<td>43</td>
<td>122,631</td>
<td>8.4MB</td>
</tr>
<tr>
<td>11.4KB</td>
<td>47</td>
<td>1,739,331</td>
<td>120MB</td>
</tr>
<tr>
<td>13.2KB</td>
<td>52</td>
<td>8,511,943</td>
<td>587MB</td>
</tr>
<tr>
<td>14.8KB</td>
<td>54</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
More on counting the number of solutions...

Data: foaf links crawled from the Web

\[
\{ \text{axel:me (foaf:knows)}^* \ ?x \}
\]

<table>
<thead>
<tr>
<th>File</th>
<th># URIs</th>
<th># Sol.</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>38</td>
<td>29,817</td>
<td>2MB</td>
</tr>
<tr>
<td>10.9KB</td>
<td>43</td>
<td>122,631</td>
<td>8.4MB</td>
</tr>
<tr>
<td>11.4KB</td>
<td>47</td>
<td>1,739,331</td>
<td>120MB</td>
</tr>
<tr>
<td>13.2KB</td>
<td>52</td>
<td>8,511,943</td>
<td>587MB</td>
</tr>
<tr>
<td>14.8KB</td>
<td>54</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

What is really happening here?
More on counting the number of solutions...

Data: foaf links crawled from the Web

\[
\{ \text{axel:me} \: (\text{foaf:knows})^{*} \: ?x \}
\]

<table>
<thead>
<tr>
<th>File</th>
<th># URIs</th>
<th># Sol.</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>38</td>
<td>29,817</td>
<td>2MB</td>
</tr>
<tr>
<td>10.9KB</td>
<td>43</td>
<td>122,631</td>
<td>8.4MB</td>
</tr>
<tr>
<td>11.4KB</td>
<td>47</td>
<td>1,739,331</td>
<td>120MB</td>
</tr>
<tr>
<td>13.2KB</td>
<td>52</td>
<td>8,511,943</td>
<td>587MB</td>
</tr>
<tr>
<td>14.8KB</td>
<td>54</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

What is really happening here?

Theory can help!
Outline

Motivation

Experimental results

Complexity results

Existential semantics: a proposal
A bit on complexity classes...

We measure the complexity by using *counting-complexity classes*

<table>
<thead>
<tr>
<th>NP</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sat</strong>: is a propositional formula satisfiable?</td>
<td><strong>CountSat</strong>: how many assignments satisfy a propositional formula?</td>
</tr>
</tbody>
</table>
A bit on complexity classes...

We measure the complexity by using *counting-complexity classes*

<table>
<thead>
<tr>
<th><strong>NP</strong></th>
<th><strong>#P</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sat</strong>: is a propositional formula satisfiable?</td>
<td><strong>CountSat</strong>: how many assignments satisfy a propositional formula?</td>
</tr>
</tbody>
</table>

**Formally**

A function $f(\cdot)$ on strings is in **#P** if there exists a polynomial-time non-deterministic TM $M$ such that

$$f(x) = \text{number of accepting computations of } M \text{ with input } x$$
A bit on complexity classes...

We measure the complexity by using *counting-complexity classes*

<table>
<thead>
<tr>
<th>NP</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Sat</em>: is a propositional formula satisfiable?</td>
<td><em>CountSat</em>: how many assignments satisfy a propositional formula?</td>
</tr>
</tbody>
</table>

**Formally**

A function $f(\cdot)$ on strings is in $\#P$ if there exists a polynomial-time non-deterministic TM $M$ such that

$$f(x) = \text{number of accepting computations of } M \text{ with input } x$$

- *CountSat* is $\#P$-complete
Counting problem for property paths

**Input:** RDF graph $G$
Property path triple $\{ :a \ path \ :b \}$

**Output:** Count the number of solutions of $\{ :a \ path \ :b \}$ over $G$
(according to the semantics proposed by W3C)
The complexity of property paths is *intractable*

**Theorem**

\[ \text{COUNTW3C is outside } \#P \]
The complexity of property paths is \textit{intractable}

\textbf{Theorem}

$\text{CountW3C}$ \textit{is outside} $\#P$

$\text{CountW3C}$ \textit{is hard to solve even if} $P = \text{NP}$
The complexity of property paths is *intractable*

**Theorem**

\[ \text{CountW3C} \text{ is outside } \#P \]

\[ \text{CountW3C} \text{ is hard to solve even if } P = NP \]

**Proof idea**

We provide a *doubly exponential* lower bound for counting.
A *doubly exponential* lower bound for counting

Let $path_s$ be a property path of the form

$$(\cdots((:p)*)*)\cdots)*$$

with $s$ nested stars
A *doubly exponential* lower bound for counting

- Let $paths$ be a property path of the form

  $$(\cdots((:p)\ast)\ast)\cdots)\ast$$

  with $s$ nested stars

- Let $K_n$ be a clique with $n$ nodes
A doubly exponential lower bound for counting

Let $path_s$ be a property path of the form

$$\cdots((:p)*)*\cdots)*$$

with $s$ nested stars

Let $K_n$ be a clique with $n$ nodes

Let $CountClique(s, n)$ be the number of solutions of

$$\{ :a0 ~ path_s ~ :a1 \}$$

over $K_n$
A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form
  \[(\cdots((:p)*)*)\cdots)\]
  with $s$ nested stars
- Let $K_n$ be a clique with $n$ nodes
- Let $CountClique(s, n)$ be the number of solutions of
  \[\{ :a0 \ path_s \ :a1 \} \]
  over $K_n$

**Lemma**

\[
CountClique(s, n) \geq (n - 2)!^{(n-1)^{s-1}}
\]
A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form $(\cdots((:p)\ast)\ast)\cdots)\ast$ with $s$ nested stars
- Let $K_n$ be a clique with $n$ nodes
- Let $CountClique(s, n)$ be the number of solutions of \{ :a0 $path_s$ :a1 \} over $K_n$

**Lemma**

$CountClique(s, n) \geq (n - 2)!^{(n-1)^{s-1}}$

In the paper:

Recursive formula for calculating $CountClique(s, n)$
We can now explain our experimental results

*CountClique*(s, n) also allows us to *fill in the blanks*

\{
  :a0 ( (:p)* )* :a1
\}

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

*CountClique*(*s, n*) also allows us to *fill in the blanks*

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

*CountClique*\((s, n)\) also allows us to *fill in the blanks*

\[
\{ :a0 ((:p)*):a1 \}
\]

<table>
<thead>
<tr>
<th>(n)</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

*CountClique\((s, n)\)* also allows us to *fill in the blanks*

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>(n)</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

*CountClique*(s, n) also allows us to *fill in the blanks*

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305 ✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576 ✓</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

\textit{CountClique}(s, n) also allows us to \textit{fill in the blanks}

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>←28 \times 10^9</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>
We can now explain our experimental results

CountClique($s, n$) also allows us to fill in the blanks

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

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305 ✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576 ✓</td>
</tr>
<tr>
<td>6</td>
<td>– ← $28 \times 10^9$</td>
</tr>
<tr>
<td>7</td>
<td>– ← $144 \times 10^{15}$</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

CountClique\((s, n)\) also allows us to fill in the blanks

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>(n)</th>
<th>(#) Sol.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>← 28 \times 10^9</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>← 144 \times 10^{15}</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>← 79 \times 10^{24}</td>
</tr>
</tbody>
</table>
We can now explain our experimental results

*CountClique*(s, n) also allows us to *fill in the blanks*

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>← 28 × 10^9</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>← 144 × 10^{15}</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td>← 79 × 10^{24}</td>
</tr>
</tbody>
</table>

79 *Yottabytes* for the answer over a file of 379 bytes
We can now explain our experimental results

*CountClique*(s, n) also allows us to *fill in the blanks*

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

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305 ✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576 ✓</td>
</tr>
<tr>
<td>6</td>
<td>– ← 28 × 10^9</td>
</tr>
<tr>
<td>7</td>
<td>– ← 144 × 10^{15}</td>
</tr>
<tr>
<td>8</td>
<td>– ← 79 × 10^{24}</td>
</tr>
</tbody>
</table>

**79 Yottabytes** for the answer over a file of 379 bytes

1 Yottabyte > the estimated capacity of all digital storage in the world
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources

Data complexity
- measure the complexity considering the query fixed
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources

*Data complexity*
- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial
Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

Data complexity

- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

**Theorem**

Data complexity of \texttt{COUNTW3C} is $\#P$-complete
Data complexity of property path is still intractable

Common assumption in Databases:

- queries are much smaller than data sources

**Data complexity**

- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

**Theorem**

Data complexity of **CountW3C** is **#P-complete**

**Corollary**

SPARQL 1.1 query evaluation is intractable in Data Complexity
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources

Data complexity
- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Theorem
Data complexity of $\text{COUNTW3C}$ is $\#P$-complete

Corollary
SPARQL 1.1 query evaluation is intractable in Data Complexity

In the paper: several other complexity results...
Outline

Motivation

Experimental results

Complexity results

Existential semantics: a proposal
An existential semantics to the rescue!

Possible solution

Do not count

Just check whether *there exists* a path satisfying the property path expression
An existential semantics to the rescue!

Possible solution

Do not count

Just check whether *there exists* a path satisfying the property path expression

Years of experiences (theory and practice) in:

- Graph Databases
- XML
- SPARQL 1.0 (PSPARQL, Gleen)

+ equivalent regular expressions giving equivalent results
Existential semantics: decision problems

**Input:** RDF graph $G$
Property path triple $\{ :a \text{ path } :b \}$

**EXISTSPath**

**Question:** Is there a path from $:a$ to $:b$ in $G$ satisfying the regular expression path?

**EXISTSW3C**

**Question:** Is the number of solutions of $\{ :a \text{ path } :b \}$ over $G$ greater than 0 (according to W3C semantics)?
Evaluating existential paths is tractable

Theorem (well-known result)

\texttt{EXISTSPath} can be solved in $O(|G| \times |\text{path}|)$
Evaluating existential paths is tractable

Theorem (well-known result)

**EXISTSPath** can be solved in $O(|G| \times |path|)$

Theorem

**EXISTSPath** and **EXISTSW3C** are *equivalent* decision problems
Evaluating existential paths is tractable

**Theorem (well-known result)**

\( \text{ExistsPath} \) can be solved in \( O(|G| \times |\text{path}|) \)

**Theorem**

\( \text{ExistsPath} \) and \( \text{ExistsW3C} \) are *equivalent* decision problems

**Corollary**

\( \text{ExistsW3C} \) can be solved in \( O(|G| \times |\text{path}|) \)
So we have possibilities for optimization
So we have possibilities for optimization

Corollary

Property path queries with SELECT DISTINCT can be efficiently evaluated
So we have possibilities for optimization

**Corollary**

Property path queries with SELECT DISTINCT can be efficiently evaluated

And we can also use DISTINCT over general queries

**Theorem**

SELECT DISTINCT SPARQL 1.1 queries are tractable in Data Complexity
SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Gleen</th>
</tr>
</thead>
</table>

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Glenn</th>
</tr>
</thead>
</table>

SPARQL 1.1 implementations do not take advantage of **SELECT DISTINCT**

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Glen</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>2.24</td>
<td>47.31</td>
<td>2.37</td>
<td>–</td>
<td>0.29</td>
<td>1.39</td>
</tr>
<tr>
<td>10.9KB</td>
<td>2.60</td>
<td>204.95</td>
<td>6.43</td>
<td>–</td>
<td>0.30</td>
<td>1.32</td>
</tr>
<tr>
<td>11.4KB</td>
<td>6.88</td>
<td>3222.47</td>
<td>80.73</td>
<td>–</td>
<td>0.30</td>
<td>1.34</td>
</tr>
<tr>
<td>13.2KB</td>
<td>24.42</td>
<td>–</td>
<td>394.61</td>
<td>–</td>
<td>0.31</td>
<td>1.38</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.33</td>
<td>1.38</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.35</td>
<td>1.42</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.44</td>
<td>1.50</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.45</td>
<td>1.52</td>
</tr>
</tbody>
</table>
SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

```
SELECT DISTINCT * WHERE { axel:me (foaf:knows)* ?x }
```

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Gleen</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>2.24</td>
<td>47.31</td>
<td>2.37</td>
<td>–</td>
<td>0.29</td>
<td>1.39</td>
</tr>
<tr>
<td>10.9KB</td>
<td>2.60</td>
<td>204.95</td>
<td>6.43</td>
<td>–</td>
<td>0.30</td>
<td>1.32</td>
</tr>
<tr>
<td>11.4KB</td>
<td>6.88</td>
<td>3222.47</td>
<td>80.73</td>
<td>–</td>
<td>0.30</td>
<td>1.34</td>
</tr>
<tr>
<td>13.2KB</td>
<td>24.42</td>
<td>–</td>
<td>394.61</td>
<td>–</td>
<td>0.31</td>
<td>1.38</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.33</td>
<td>1.38</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.35</td>
<td>1.42</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.44</td>
<td>1.50</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.45</td>
<td>1.52</td>
</tr>
</tbody>
</table>

Optimization possibilities can remain hidden in a complicated semantics
Concluding remarks and a proposal

Counting in general is not feasible, but W3C has use cases to count (when there are no cycles)
Concluding remarks and a proposal

Counting in general is not feasible, but W3C has use cases to count (when there are no cycles)

We propose to:

- Have an existential semantics as default
- Provide users with a keyword to count paths
Concluding remarks and a proposal

Counting in general is not feasible, but W3C has use cases to count (when there are no cycles)

We propose to:

- Have an existential semantics as default
- Provide users with a keyword to count paths
  even we (authors) haven’t reached a consensus on this
“How cool would it be to (reach a consensus and) have a design that meets both use cases!”

TBL yesterday

(about W3C standardization)
Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard would have Prevented?

Marcelo Arenas   Sebastián Conca   Jorge Pérez

PUC-Chile, Universidad de Chile
Outline

Motivation

Experimental results

Complexity results

Existential semantics: a proposal