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

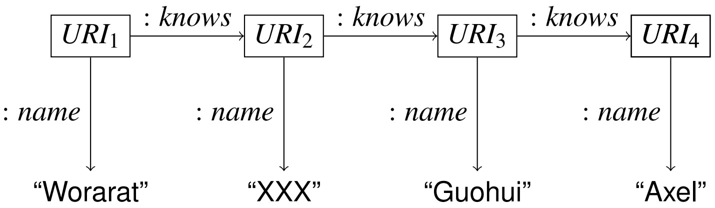
Evaluation Complexity

Counting Complexity

Conclusion

Property Path: in SPARQL 1.0 Query

SPARQL 1.0 provides limited navigational capabilities

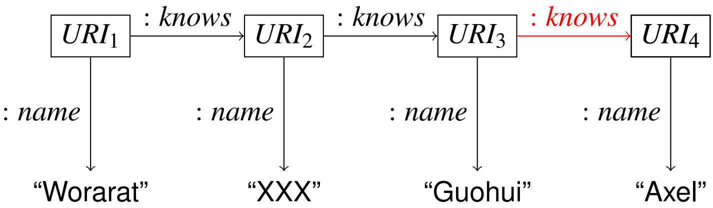


Example Query

```
SELECT ?x
WHERE
{
  ?x :knows ?y .
  ?y :name "Axel" .
}
```

Property Path: in SPARQL 1.0 Query

SPARQL 1.0 provides limited navigational capabilities

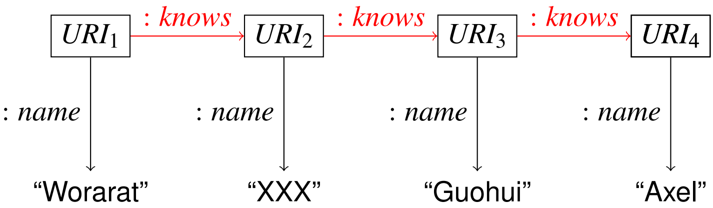


Example Query

```
SELECT ?x
WHERE
{
   ?x :knows ?y .
   ?y :name "Axel" .
}
```

Property Path: in SPARQL 1.0 Query

SPARQL 1.0 provides limited navigational capabilities



Example Query

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

Outline

Property Path Syntax in SPARQL 1.1

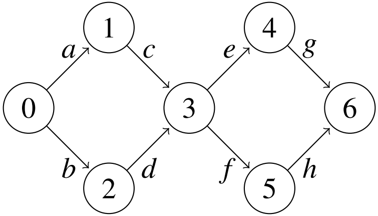
Syntax Form	Matches
iri	An IRI. A path of length one.
$\hat{}elt$	Inverse path (object to subject).
$!iri$ or $!(iri_1\|...\|iri_n)$	An IRI. Negated property set. An IRI which is not one of iri_i. $!iri$ is short for $!(iri)$.
$!\hat{}iri$ or $!(iri_1\|...\|iri_j\|\hat{}iri_{j+1}\|\ldots\|\hat{}iri_n\|)$	An IRI. Negated property set. An IRI which is not one of iri_i. $!iri$ is short for $!(iri)$.
(elt)	A group path elt, brackets control precedence.
$(elt1)$ / $(elt2)$	A sequence path of elt1 followed by elt2.
$(elt1)$ | $(elt2)$	A alternative path of elt1 or elt2 (all possibilities are tried).
elt^*	A path of zero or more occurrences of elt.
elt^+	A path of one or more occurrences of elt.
$elt?$	A path of zero or one occurrences of elt.
$elt\{n, m\}$	A path of between n and m occurrences of elt.
$elt\{n\}$	A path of exactly n occurrences of elt.
$elt\{n, \}$	A path of n or more occurrences of elt.
$elt\{, n\}$	A path of between 0 and n occurrences of elt.

elt: is a path element, which may itself be composed of path syntax constructs.

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.



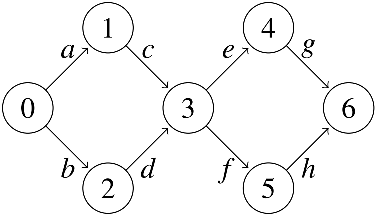
Main interesting problems:

- 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.



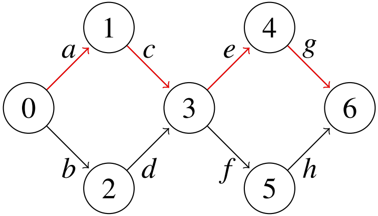
Main interesting problems:

- 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

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

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



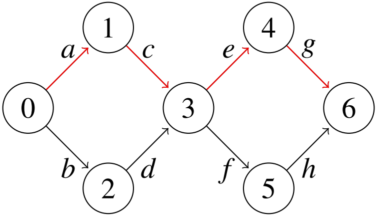
Main interesting problems:

- 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.



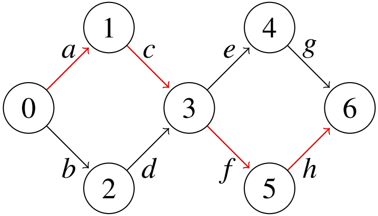
Main interesting problems:

- 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

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

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



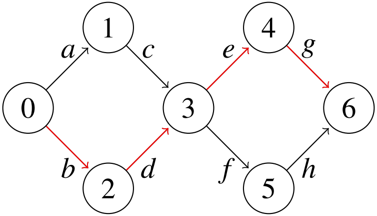
Main interesting problems:

- 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.



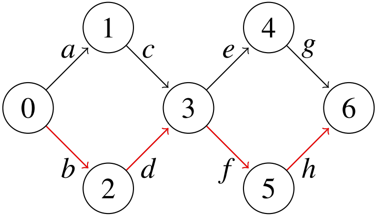
Main interesting problems:

- 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

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)

Outline

Introduction

Limitation of navigational capabilities in SPARQL 1.0

SPARQL 1.1 property path

Experiments on Evaluation and Counting

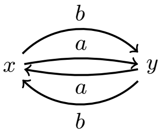
Complexity

Evaluation Complexity

Counting Complexity

Conclusion

Experiments on Evaluation



[Arenas et al., 2012]

```
ASK WHERE { x (a|b){1,k} y }
```

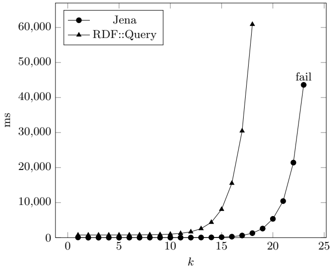


Figure: Evaluation time for Jena and RDF::Query.

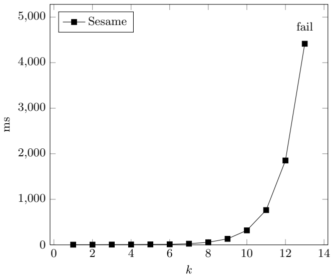


Figure: Evaluation time for Sesame.

Experiments on Counting

[Losemann and Martens, 2012]

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

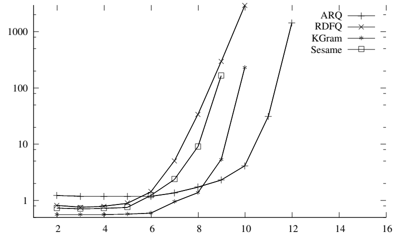
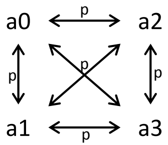


Figure: Time in seconds for processing the queries w.r.t. the clique size n (time axis in log-scale)

a)



b)

@prefix : http://example.org/ .
:a0 :p :a1, :a2, :a3 .
:a1 :p :a0, :a2, :a3 .
:a2 :p :a0, :a1, :a3 .
:a3 :p :a0, :a1, :a2 .

q

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

Outline

Outline

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

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

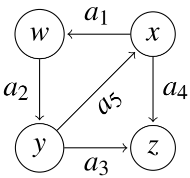
Simple path

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

Regular path (path)

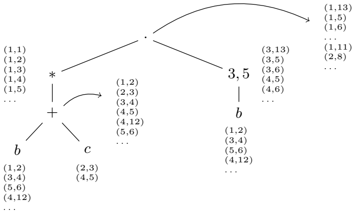
A path in a graph is a sequence of nodes such that from each of its nodes there is an edge to the next node in the sequence

Find paths from x to z.

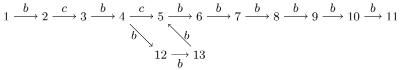


Path	a_4	a_1, a_2, a_3	a_1, a_2, a_5, a_4
Regular?	$\surd$	$\surd$	$\surd$
Simple?	$\surd$	$\surd$	$\times$ (x visited twice)

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.

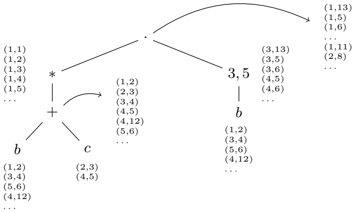
Figure: Illustration of polynomial-time dynamic programming algorithm

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

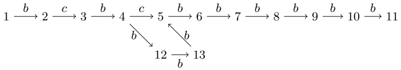
The Complexity of Evaluation

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 $(b+c)^* b^{3,5}$ and the graph in Fig. 2(b).



(b) An edge-labeled graph.

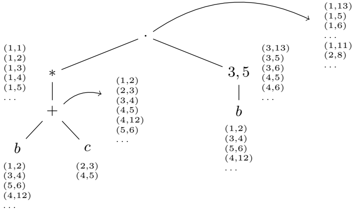
Figure: Illustration of polynomial-time dynamic programming algorithm

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

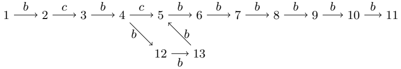
The Complexity of Evaluation

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 $(b+c)^*b^{3,5}$ 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)**.

W3C

SPARQL 1.1 Query Language

W3C Working Draft 05 January 2012

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.

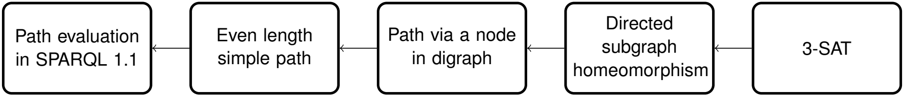
Definition: OneOrMorePath

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



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 $(aa)^+$

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

Even length simple path [Mendelzon and Wood, 1989, Lapaugh and Papadimitriou, 1984]

Let 0 and 1 be distinct symbols in Σ. 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

Even length simple path [Mendelzon and Wood, 1989, Lapaugh and Papadimitriou, 1984]

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?

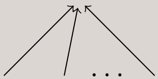
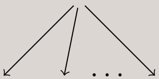
Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems

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

Directed subgraph homeomorphism [Fortune et al., 1980]

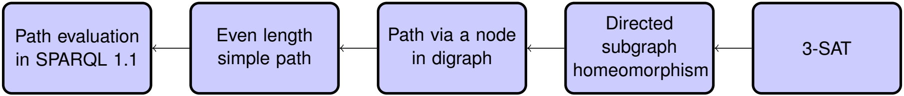
For each P not in C the *fixed subgraph homeomorphism problem* with pattern P is **NP-complete**

$C =$  or 

In the reduction of Directed subgraph homeomorphism to Path via a node problem, we use $P = \longleftarrow \; \longleftarrow$

Evaluation under Simple Path Semantics

Reduction Chain of NP-completeness Problems



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

Duplicates in SPARQL 1.1 Draft

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/> .

:x  :p  :y .
:y  :p  :x .
```

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

giving results of:

o
<http://example/x>
<http://example/y>
<http://example/x>

The order of results in these examples is not significant.

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: $\{\{ \underbrace{[], [], \ldots, []}_{\text{duplicates of } S_n \text{ times}} \}\}$

length	p*	path	count
1	p	01	$P_3^0 = 1$
2	pp	021, 031, 041	$P_3^1 = 3$
3	ppp	0231, 0241, 0321, 0341, 0421, 0431	$P_3^2 = 6$
4	pppp	02341, 02431, 03241, 03421, 04231, 04321	$P_3^3 = 6$
sum			$S_5 = 16$

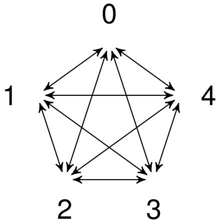
Table: Compute S_5. Recall that $P_n^k = n(n-1)\ldots(n-k+1)$

$$\boxed{S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)!}$$

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: $\{\{\ \underbrace{[], [], \ldots, []}_{\text{duplicates of } S_n \text{ times}}\ \}\}$

length	p*	path	count
1	p	01	$P_3^0 = 1$
2	pp	021, 031, 041	$P_3^1 = 3$
3	ppp	0231, 0241, 0321, 0341, 0421, 0431	$P_3^2 = 6$
4	pppp	02341, 02431, 03241, 03421, 04231, 04321	$P_3^3 = 6$
sum			$S_5 = 16$

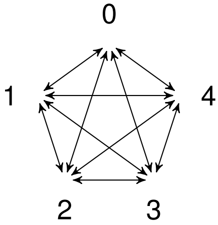
Table: Compute S_5. Recall that $P_n^k = n(n-1)\ldots(n-k+1)$

$$\boxed{S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)!}$$

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: $\{\{ \underbrace{[], [], \ldots, []}_{\text{duplicates of } S_n \text{ times}} \}\}$

length	p*	path	count
1	p	01	$P_3^0 = 1$
2	pp	021, 031, 041	$P_3^1 = 3$
3	ppp	0231, 0241, 0321, 0341, 0421, 0431	$P_3^2 = 6$
4	pppp	02341, 02431, 03241, 03421, 04231, 04321	$P_3^3 = 6$
sum			$S_5 = 16$

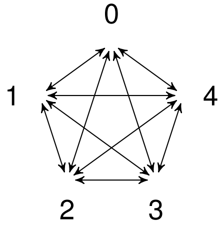
Table: Compute S_5. Recall that $P_n^k = n(n-1)\ldots(n-k+1)$

$$\boxed{S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)!}$$

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: $\{\{ \underbrace{[], [], \ldots, []}_{\text{duplicates of } S_n \text{ times}} \}\}$



length	p*	path	count
1	p	01	$P_3^0 = 1$
2	pp	021, 031, 041	$P_3^1 = 3$
3	ppp	0231, 0241, 0321, 0341, 0421, 0431	$P_3^2 = 6$
4	pppp	02341, 02431, 03241, 03421, 04231, 04321	$P_3^3 = 6$
sum			$S_5 = 16$

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

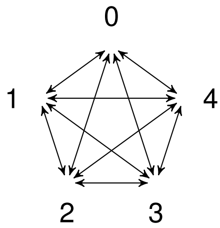
$$\boxed{S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)!}$$

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: $\{\{\ \underbrace{[], [], \ldots, []}_{\text{duplicates of } S_n \text{ times}}\ \}\}$



length	p*	path	count
1	p	01	$P_3^0 = 1$
2	pp	021, 031, 041	$P_3^1 = 3$
3	ppp	0231, 0241, 0321, 0341, 0421, 0431	$P_3^2 = 6$
4	pppp	02341, 02431, 03241, 03421, 04231, 04321	$P_3^3 = 6$
sum			$S_5 = 16$

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

$$\boxed{S_{n+1} = P_{n-1}^0 + P_{n-1}^1 + \ldots + P_{n-1}^{n-1} > P_{n-1}^{n-1} = (n-1)!}$$

Nesting of $*$

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

s	n	COUNTCLIQUE(p_s, n)	s	n	COUNTCLIQUE(p_s, n)
1	3	2	1	5	16
2	3	6	2	5	418576
3	3	42	3	5	$> 10^{23}$
4	3	1806	4	5	$> 10^{93}$
1	4	5	1	6	65
2	4	305	2	6	28278702465
3	4	56931605	3	6	$> 10^{53}$
4	4	$> 10^{23}$	4	6	$> 10^{269}$

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 #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 #DNF

An Existential Semantics to the Rescue

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

n	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen
8	1.68	32.61	1.39	9.08	0.18	1.24
9	2.00	213.99	5.34	166.82	0.20	1.23
10	3.65	2123.90	227.66	–	0.20	1.25
11	29.71	–	–	–	0.23	1.25
12	1394.06	–	–	–	0.24	1.24
13	–	–	–	–	0.27	1.24

Cliq-1D

n	ARQ	RDFQ	Psparql	Gleen
2	1.40	0.76	0.14	1.23
3	1.19	0.84	0.14	1.23
4	1.65	19.38	0.14	1.23
5	97.06	–	0.15	1.22
6	–	–	0.16	1.23
7	–	–	0.16	1.23

Cliq-2D

n	ARQ	RDFQ	Psparql	Gleen
2	1.18	0.77	0.14	1.24
3	1.41	6.78	0.14	1.23
4	–	–	0.15	1.24
5	–	–	0.15	1.24
6	–	–	0.16	1.24
7	–	–	0.16	1.24

Cliq-3D

Figure: Experiement with Existential Semantics

Outline

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
```

Latest Stories in Editors' Draft

http://www.w3.org/2009/sparql/docs/query-1.1/rq25.xml

W3C Editor's Draft

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.

Fortune, S., Hopcroft, J. E., and Wyllie, J. (1980).
The directed subgraph homeomorphism problem.
Theor. Comput. Sci., 10:111–121.

Harris, S., Seaborne, A., and Prud'hommeaux, E. (2012).
SPARQL 1.1 query language (W3C working draft 05 january 2012).
http://www.w3.org/TR/sparql11-query/.

Lapaugh, A. S. and Papadimitriou, C. H. (1984).
The even-path problem for graphs and digraphs.
Networks, 14(4):507–513.

Losemann, K. and Martens, W. (2012).
The complexity of evaluating path expressions in SPARQL.
In *PODS '12*, pages 101–112. ACM.

Mendelzon, A. O. and Wood, P. T. (1989).
Finding regular simple paths in graph databases.
In *VLDB '89*, pages 185–193. Morgan Kaufmann Publishers Inc.