Graph Analytics for Subject-Matter Experts: Or, Connecting Graph Analytics to the Data that Needs It

Steve Reinhardt
spr@YarcData.com
The uRiKA developers are extending SPARQL’s excellent graph-matching capabilities with global graph-analytic operations, via an interface that researchers can use to plug in their own algorithms.
Agenda

- How I viewed graph analytics while working on KDT
- How I view graph analytics while working on uRiKA
- Extending SPARQL for global operations
Knowledge Discovery Workflow

1. Cull relevant data
2. Build input graph
3. Analyze input graph

- Graph Analysis is the core of the workflow
- Input methods and formats are ad hoc
- Modest heterogeneity of data
- Complex algorithms are high value
Current Mental Model for Graph Analytics

**Workflow**

1. Extract and convert data to RDF
2. Ingest into database

- Extracting, representing, and converting data as a graph is hard
- The data (both vertices and edges) in the graph will be highly heterogeneous
- Complex algorithms are high value
- Reasoning about data is an important capability
- The RDF/SPARQL community is investing heavily to make those technologies do steps 1, 2, and part of 3 very well
- How can we build on that investment to make graph analytics more widely usable?
uRiKA Architecture

RDF / SPARQL

RDF/SPARQL Interpreter

SPARQL query engine ("endpoint")

x86 Service Nodes

Threadstorm processors
Emerging Standards: RDF and SPARQL

Resource Description Framework (RDF)

- Designed to enable a) semantic web searching and b) integration of disparate data sources
- W3C standard formats
- Every datum represented as subject/predicate/object(/graph)
  - Ideally with each of those expressed with a URI
- Standard ontologies in some domains
  - *e.g.*, SnoMED/CT for clinical medical terms
- Example:

```r
<ncbitax:NCBITaxon_8401> rdf:type owl:Class .
<http://cs.org/pDNS#2> cs:hasQueryDomain <http://cs.org/domain#ns1.secure.net> .
nf:NetflowRecord_2 nf:hasDstAddr <http://netflow.org/nf#IPAddress_255.255.23.18> .
```
Emerging Standards: RDF and SPARQL

- **SPARQL Protocol and RDF Query Language (SPARQL)**
  - Enables matching of graph patterns
  - Reminiscent of SQL
  - **Minimal ability to do**

---

# Lehigh University Benchmark (LUBM) Query 9

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>

WHERE
{ ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?Z rdf:type ub:Course .
}
```

---

- **PREFIX == shorthand for a URI**
- **variables to be returned from the query**
- “find each student who took a course from her advisor”
Choose only vertices representing people, and edges between those vertices representing phone calls and text messages during the last hour

Calculate BC for the vertices in the graph created by those vertices and edges

```python
# in KDT
def vfilter(self, vTypes):
    return self.type in vTypes

def efilter(self, eTypes, sTime, eTime):
    return (self.type in eTypes) and ((self.sTime > sTime) and (self.eTime < eTime))

wantedVTypes = (People)
wantedETypes = (PhoneCall, TextMessage)
start = dt.now() - dt.timedelta(hours=1)
end = dt.now()
bc = G.centrality('BC', filter=((vfilter, wantedVTypes), (efilter, wantedETypes, start, end)))```
Example: Betweenness Centrality on Semantic Graph (2)

# in SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX comm: <http://www.comm.org#> # a fictional communication ontology
SELECT ?p1, ?p2
WHERE
  { ?call rdf:type comm:PhoneCall . } UNION { ?call rdf:type comm:TextMessage .}
  ?call comm:hasTime ?time
  FILTER ( ?time > xsd:dateTime("2012-07-11T09:30:00Z") )
}
#
# Here, need to do betweenness centrality, but no good interface
#
Example 2: More-complicated pattern

Choose people via a more complicated pattern, e.g. only Students from LUBM Query 9

```python
# in KDT
<<No way to do this today>>
def vfilter(self, vTypes):
    return self.type in vTypes

def efilter(self, eTypes, sTime, eTime):
    return (self.type in eTypes) and ((self.sTime > sTime) and (self.eTime < eTime))

wantedVTypes = (People)
wantedETypes = (PhoneCall, TextMessage)
start = dt.now() - dt.timedelta(hours=1)
end = dt.now()
bc = G.centrality('BC', filter=((vfilter, wantedVTypes), (efilter, wantedETypes, start, end)))
```
Example: More-complicated pattern (2)

# in SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX comm: <http://www.comm.org#>  # a fictional communication ontology
SELECT ?x, ?call
WHERE
{
  ?X rdf:type ub:Student .
  ?Y rdf:type ub:Faculty .
  ?Z rdf:type ub:Course .
  { ?call rdf:type comm:PhoneCall . } UNION { ?call rdf:type comm:TextMessage .}
  ?call comm:hasCaller ?X .
  ?call comm:hasTime ?time
  FILTER ( ?time > xsd:dateTime("2012-07-11T09:30:00Z") )
}
```

#
# Here, need to do betweenness centrality, but no good interface
#

Extending Global Operations in SPARQL

- SPARQL already has the idea of global graph operations
  
  ```sparql
  SELECT ?var1 ?var2
  WHERE
  {
  ... 
  } ORDER BY ?var ?var2
  
  or
  
  SELECT ?var1 ?var2
  WHERE
  {
  ... 
  } GROUP BY ?var ?var2
  
  ... we just need to denote the right function, inputs, and outputs
  
  - One possibility: property functions
    - Can be defined externally just as any other RDF object
    - (Obviously) needs to point to a function that can be executed within the SPARQL endpoint
    - Syntax
      ```sparql
      (?in1 ?in2 ... ) <function-name> (?out1 ?out2 ... )
      ```
Example: BC in SPARQL

# in SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX comm: <http://www.comm.org#>  # a fictional communication ontology
PREFIX ga:   <http://www.graphanalytics.org#>  # graph-analytic operation identifiers
SELECT ?X, ?call, ?bc
WHERE
{ ?X rdf:type ub:Student  .  
   ?Y rdf:type ub:Faculty .  
   ?Z rdf:type ub:Course .  
   ?X ub:takesCourse ?Z .}
{ ?call rdf:type comm:PhoneCall . } UNION { ?call rdf:type comm:TextMessage .}
{ ?call comm:hasCaller ?X . } UNION { ?call comm:hasCallee ?X . }
} ORDER BY (?call ?X) ga:BC (?bc ?X)
Key Points

- **The set of functions will be extensible**
  - (Still need to point to function(s) that can execute within the SPARQL endpoint!)

- **APIs will be provided so that researchers can implement their own functions**
  - Think CombBLAS or Mex

- **SPARQL extended with global graph ops provides a mechanism for graph-analytic researchers to test their algorithms on real data at scale**

- **If you’re interested in using this interface, please contact me**
uRiKA Architecture with Extensions

RDF / SPARQL

Extension functions

RDF/SPARQL Interpreter

SPARQL query engine ("endpoint")

x86 Service Nodes

Threadstorm processors
YarcData $100K Graph-Analytic Challenge

- A challenge to focus attention on big-data graph analytics with societal value
- Solutions via RDF and SPARQL
- Submissions open now until Sep 15, 2012
- Criteria: importance and complexity of the problem, scalability and performance of solution, innovativeness of solution
- $70K first prize, $13K, $8K, $3K, $3K, $3K
The uRIKA developers are extending SPARQL’s excellent graph-matching capabilities with global graph-analytic operations, via an interface that researchers can use to plug in their own algorithms.
Learn More

- Semantic Web for the Working Ontologist, by Dean Allemang and James Hendler, ISBN 978-0123859655

- RDF: www.w3.org/RDF/
- SPARQL: www.w3.org/TR/rdf-sparql-query/