Sparkwave: Continuous Schema-Enhanced Pattern Matching over RDF Data Streams

Srdjan Komazec, Davide Cerri, Dieter Fensel
Motivation
Data streams on the Web

• **Data streams** are becoming common on the Web
  – Stock exchange tickers
    • All mayor stock exchanges are offering some real-time data services
  – Transportation services
    • IVB Innsbrucker Verkehrsbetriebe (http://www.ivb.at/earth/)
  – Social networking activity notifications
    • All major players are providing notifications and REST APIs
  – Weather information
    • http://wiki.knoesis.org/index.php/LinkedSensorData

• **Collecting, integrating and sharing** data streams
  – Cosm platform
    • http://cosm.com
Motivation

A Social Media Example

- Real-time monitoring of social media
  - Notify when at least three of my friends are posting from a certain place
  - Notify when someone talks about the same place on both Facebook and Twitter
Introduction
Processing data streams on the Web

• Stream processing on the Web needs to cope with **heterogeneity**

• Semantic Web technologies
  – Facilitating data integration by using machine processable descriptions to reconcile heterogeneities (e.g., Semantic Sensor Web)

• Impact of SW technologies over stream processing and vice versa
  – Entailments and data/event stream processing
  – Data volatility (temporality) and Semantic Web technologies

• Stream reasoning
  – Performing materialization of implicit knowledge while processing data streams
Sparkwave

What is Sparkwave?

- Sparkwave is a system for matching patterns over RDF data streams

\[
\begin{align*}
\cdot (s_i, p_i, o_i, \tau_i) \\
\cdot (s_{i+1}, p_{i+1}, o_{i+1}, \tau_{i+1}) \\
\end{align*}
\]

An RDF data stream

\[
\begin{align*}
\cdot <x a b > \land <x c ?y > \land <?y m n >
\end{align*}
\]

A pattern

RDF data stream

\[
\begin{align*}
... (<t11, a, b>, 10) \\
... (<t21, m, n>, 20) \\
... (<t12, a, b>, 40) \\
... (<t11, c, t21>, 60)...
\end{align*}
\]

MATCH!

A match over the Figure 2 pattern
Sparkwave

What Sparkwave can and can not do?

- Sparkwave exploits **schema-entailed knowledge**
  - Entailed triples are considered during matching

- Sparkwave supports data **stream processing features**
  - Time-based sliding windows and garbage collection (temporal operators and consumption strategies to be added soon)

- Sparkwave operates over a **fixed schema**
  - Schema definitions are rarely transported over a stream (e.g., sensor readings)

- Sparkwave **does not support big background knowledge**
  - The system exploits a memory-intensive approach

- Sparkwave **does not provide generic reasoning capabilities**
  - The system is built for RDF Schema (and few OWL constructs)
• Uses **Rete algorithm** as the foundation
  – Extends Rete with additional features to meet the requirements of data stream processing
  – Introduces a **networking structure** necessary to compute entailments (ε – network)
• Pattern matching algorithm for implementing production rule systems in many object/many pattern situations.
  – The basis for many popular expert system shells, including CLIPS, Jess, Drools, BizTalk, Rules Engine and Soar.
Sparkwave
Rete algorithm adaptation (RDF triples)

\(<?x a b> \land {?x c ?y} \land {?y m n}>\)
Sparkwave
Rete algorithm adaptation (time window)

- Executing time-window boundary checks in join nodes
  - Token has always an updated value of a time interval occupied by a partial match
Sparkwave

ε-network

- Rete architecture is extended with an additional network of entailment nodes called ε-network

- ε-network is a graph consisting of
  - Class and Property nodes – generate new triples
  - S and O arcs – transport subject or object value between source & destination node
  - SO and OS arcs – transport subject+object (and inverse) values between source & destination node

- ε-network is built on top of schema and optimized in accordance to RDF pattern descriptions
  - The network topology is governed by the property hierarchies (with domain and range definitions) and class hierarchies

- Outputs of ε-network are connected to inputs of α-network
## Sparkwave
### Schema entailments

<table>
<thead>
<tr>
<th>Rule</th>
<th>If</th>
<th>Then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf1</td>
<td>(x p y)</td>
<td>(p rdf:type rdf:Property)</td>
</tr>
<tr>
<td>rdfs2</td>
<td>(p rdfs:domain c) (x p y)</td>
<td>(x rdfs:type c)</td>
</tr>
<tr>
<td>rdfs3</td>
<td>(p rdfs:range c) (x p y)</td>
<td>(y rdfs:type c)</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>(x p y)</td>
<td>(x rdfs:type rdfs:Resource)</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>(x p y)</td>
<td>(y rdfs:type rdfs:Resource)</td>
</tr>
<tr>
<td>rdfs5</td>
<td>(p rdfs:subPropertyOf q) (q rdfs:subPropertyOf r)</td>
<td>(p rdfs:subPropertyOf r)</td>
</tr>
<tr>
<td>rdfs6</td>
<td>(p rdf:type rdf:Property)</td>
<td>(P rdfs:subPropertyOf p)</td>
</tr>
<tr>
<td>rdfs7</td>
<td>(p rdfs:subPropertyOf q) (x p y)</td>
<td>(x q y)</td>
</tr>
<tr>
<td>rdfs8</td>
<td>(c rdf:type rdfs:Class)</td>
<td>(c rdfs:subClassOf rdfs:Resource)</td>
</tr>
<tr>
<td>rdfs9</td>
<td>(c rdfs:subClassOf d) (x rdf:type c)</td>
<td>(x rdfs:type d)</td>
</tr>
<tr>
<td>rdfs10</td>
<td>(c rdf:type rdfs:Class)</td>
<td>(c rdfs:subClassOf c)</td>
</tr>
<tr>
<td>rdfs11</td>
<td>(c rdfs:subClassOf d) (d rdfs:subClassOf e)</td>
<td>(c rdfs:subClassOf e)</td>
</tr>
<tr>
<td>rdfs12</td>
<td>(p rdf:type rdfs:ContainerMembershipProperty)</td>
<td>(p rdfs:subPropertyOf rdfs:Member)</td>
</tr>
<tr>
<td>rdfs13</td>
<td>(x rdf:type rdfs:Datatype)</td>
<td>(x rdfs:subClassOf rdfs:Literal)</td>
</tr>
<tr>
<td></td>
<td>(p owl:inverseOf q)</td>
<td>(q owl:inverseOf p)</td>
</tr>
<tr>
<td></td>
<td>(p owl:inverseOf q) (x p y)</td>
<td>(y q x)</td>
</tr>
<tr>
<td></td>
<td>(p rdf:type owl:SymmetricProperty) (x p y)</td>
<td>(y p x)</td>
</tr>
</tbody>
</table>
Sparkwave
ε- network - Example

:UserAccount a rdfs:Class .
:Post a rdfs:Class .
:Tweet a rdfs:Class ;
   rdfs:subClassOf :Post .
:FacebookPost a rdfs:Class ;
   rdfs:subClassOf :Post .
:Place a rdfs:Class .
:hasCreator a rdf:Property ;
   rdfs:domain :Post ;
   rdfs:range :UserAccount .
:creatorOf a rdf:Property ;
   owl:inverseOf :hasCreator .
:relatedTo a rdf:Property ;
   rdfs:domain :Post ;
   rdfs:range :Place .
:talksAbout a rdf:Property ;
   rdfs:subPropertyOf :relatedTo .
:postedFrom a rdf:Property ;
   rdfs:subPropertyOf :relatedTo .
a) An example of social media analytics ontology
b) An RDF graph pattern in a form of a C-SPARQL query
Example
\(\epsilon\)-network – Pre-processing
Example
Rete network
Evaluation

Performance tests – settings

- Adaptation of Berlin SPARQL Benchmark (BSBM) dataset
  - E-commerce use-case in which products are offered by different vendors and consumers post reviews about products

- We use different product type hierarchies
  - 329 product types / 4-levels hierarchy
  - 22,527 product types / 6-levels hierarchy

- Test stream contains 200,000 offers (~ 2.2 million triples)

- Tests performed on a Linux box
  - CPU Intel Core i7 620M (4 MB cache, 2.66 GHz)
  - 4 GB RAM
Evaluation
Evaluation queries

PREFIX bsbm-vocab: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?offer ?product
FROM STREAM <http://bsbm.org/stream>
WHERE {
  ?offer bsbm-vocab:validFrom ?from .
  ?offer bsbm-vocab:validTo ?to .
  ?offer dc:date ?date .
  ?product bsbm-vocab:producer ?producer
  ?product rdf:type bsbm-inst:ProductType2
}

?vendor bsbm-voc:country http://downlode.org/rdf/iso-3166/countries#GB
}

Pattern 1

Pattern 2

Pattern 3
Evaluation

Throughput (time window [ms])

- Pattern 1, schema has 329 classes, target product type has 40 subclasses
- Sparkwave is significantly better only for smaller time windows
- Impact of reasoning over Sparkwave performance is limited (15-20%)
Evaluation
Throughput (#static triples)

• Patterns 2 and 3, schema has 329 classes, target product type has 40 subclasses, variable amount of static instances

• The loss of performance due to the presence of static knowledge is modest

• Difference between the patterns is not so significant
Evaluation

Memory consumption

- Pattern 1, schema has 329 classes, target product type has 40 subclasses
- Memory consumption is lower but rises
Further Work

- **Sparwave RDF graph pattern expression language**
  - Extending C-SPARQL and borrowing temporal operators from EP-SPARQL
  - Allocation of sub-patterns to different streams
  - Allocation of sub-patterns to different time-based windows in cascading fashion

- **Sparkwave engine enhancements**
  - Speed, speed, speed! (parallel processing?)
  - Support for logical, data and temporal operators
  - Support for full set of event consumption strategies
    - Currently we support UNRESTRICTED strategy
Conclusions

• Sparkwave is a system for graph pattern detection on RDF data streams, featuring schema entailments and time-windows

• Based on Rete algorithm
  – Extended with $\varepsilon$ – network to enable entailments
  – Extended to support time-window

• Evaluated over BSBM
  – Advantage in throughput for smaller time window sizes
  – Lower memory footprint

https://github.com/skomazec/Sparkweave
Questions?