Orion: Enabling Suggestions in a Visual Query Builder for Ultra-Heterogeneous Graphs

Nandish Jayaram
Member of Technical Staff 3
Pivotal

(work done as a student at the University of Texas at Arlington)

Rohit Bhoopalam
Chengkai Li
Vassilis Athitsos

Eighth TUC Meeting, June 22-23, 2016
Ultra-heterogeneous Entity Graphs

Large, complex and schema-less graphs capturing millions of entities and billions of relationships between entities.

Linked Open Data: 52 billion RDF triples
Freebase: 1.8 billion triples
DBpedia: 470 million triples
Yago: 120 million triples
Structured Queries are Difficult to Write

**SQL QUERY:**

SELECT Founder.subj, Founder.obj
FROM Founder,
    Nationality,
    HeadquarteredIn
WHERE
    Founder.subj = Nationality.subj AND
    Founder.obj = HeadquarteredIn.subj

**SPARQL QUERY:**

SELECT ?company ?founder WHERE {
    ?company dbprop:headquartered_in db:Silicon_Valley .
}
Simpler Query Paradigms

Keyword Search

- [Kargar+11], BLINKS [He+07]
  - Challenging to articulate exact query intent by keywords

Approximate Query Answering

- NESS [Khan+11]: uses neighborhood-based indexes to quickly find approximate matches to a query graph;
- TALE [Tian+08]: approximate large graph matching
  - Users still have to formulate the initial query graph
Visual Query Builders

Relational Databases: CLIDE [Petropoulos+06]

Graph Databases: VOGUE [Bhowmick+13], PRAGUE [Jin+12], Gblender [Jin+10], GRAPHITE [Chau+08]

Single Large Graphs: QUBLE [Hung+13]

- Require a good knowledge of the underlying schema
- No automatic suggestions regarding what to include in the query graph
Orion

- Interactive GUI for building query components
- Iteratively suggests edges based on their relevance to the user’s query intent, according to the partial query graph so far
Orion GUI

Dynamic list of all possible user actions at any given moment

Control panel for various settings and tips
Active Mode

Grey edges and nodes automatically suggested in **active mode:**

- Accepted by user (blue): **positive edges**
- Ignored by user: **negative edges**
Passive Mode

A new node added in **passive mode**

A new edge added in **passive mode**
Concepts

Edges in partial query graph (positive edges)
- starring, director, music

Edges rejected by users (negative edges)
- education, nationality

Candidate edges
- producer, writer, editor

Query Session:
<starring, director, music, education, nationality>
Concepts

Query Log (W)

<table>
<thead>
<tr>
<th>Id</th>
<th>Query Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>w1</td>
<td>education, founder, nationality</td>
</tr>
<tr>
<td>w2</td>
<td>starring, music, director</td>
</tr>
<tr>
<td>w3</td>
<td>nationality, education, music, starring</td>
</tr>
<tr>
<td>w4</td>
<td>artist, title, writer, director</td>
</tr>
<tr>
<td>w5</td>
<td>director, founder, producer</td>
</tr>
<tr>
<td>w6</td>
<td>writer, editor, genre</td>
</tr>
<tr>
<td>w7</td>
<td>award, movie, director, genre</td>
</tr>
<tr>
<td>w8</td>
<td>education, founder, nationality</td>
</tr>
</tbody>
</table>

Problem

Given a query log, a query session, and a set of candidate edges, rank the candidate edges by their relevance to the user’s query intent.
Random Decision Path (RDP)

\[<\text{starring, director, music, education, nationality}>\]

- Choose edges from the query session randomly, to form RDPs

- Each decision path selects a subset of the query log, with no more than ‘\(\tau\)’ rows
- Grow a path incrementally until its support in the query log drops below ‘\(\tau\)’
Random Decision Path: Scoring

- For each RDP, use its corresponding query log subset to compute the support of each candidate edge.
- Final score of each candidate is its average score across all RDPs.
- If $R$ is the set of all RDPs:

$$score(e) = \frac{1}{|R|} * \sum_{Q_i \in R} sup(e, Q_i, W)$$

$$sup(e, Q_i, W) = \frac{|\{w | w \in W, Q_i \cup \{e\} \subseteq w\}|}{|\{w | w \in W, Q_i \subseteq w\}|}$$
Query Log

Nonexistent (almost)

Simulate and bootstrap

- Find positive edges
  - Wikipedia and data graph
  - Data graph only
- SPARQL query log [Morsey+11]
- Inject negative edges
**Query Log Simulation: Wikipedia + Data Graph**

*Use Sentences in Wikipedia Articles to Identify Positive Edges*

### Early life [edit]

Yang was born in Taipei, Taiwan on November 6, 1968, and moved to San Jose, California at the age of ten with his mother and younger brother.[4] He claimed that despite his mother being an English teacher, he only knew one English word (shoe) on his arrival. Becoming fluent in the language in three years, he was then placed into an Advanced Placement English class.[5]

Yang graduated from Sierramont Middle School and Piedmont Hills High School in San Jose and went on to earn a Bachelor of Science and a Master of Science in electrical engineering from Stanford University, where he was a member of Phi Kappa Psi fraternity.[6][7]

### Nodes Mapped:

- **Jerry Yang**, Electrical Engineering
- Stanford University, Phi Kappa Psi

- <degree, almaMater, frat_member>
Query Log Simulation: Data Graph Only

Represent Each Node as an Itemset of Positive Edges

Generate Frequent Itemsets of Varying Sizes

- Each frequent itemset of edges forms positive edges
Query Log Simulation: Injecting Negative Edges

Positive Edges List

(1) writer, starring, producer
(2) starring, editor, education
(3) editor, nationality, music

Inject Negative Edges

writer, starring, producer, editor, education (starring appears in 2)
starring, editor, education, writer, producer, nationality, music (starring appears in 1, and editor appears in 3)
editor, nationality, music, starring, education (editor appears in 2)
Experiments

System Configurations

- Double quad-core 2.0 GHz Xeon server, 24 GB RAM
- TACC: 5 Dell PowerEdge R910 server nodes, with 4 Intel Xeon E7540 2.0 GHz 6-core processors, 1 TB RAM

Datasets

- Freebase (33 M edges, 30 M nodes, 5253 edge types)
- DBpedia (12 M edges, 4 M nodes, 647 edge types)

User Studies with Freebase
Query Logs Compared

- Freebase: Wiki, Data
- DBpedia: Wiki, Data, QLog

<table>
<thead>
<tr>
<th>Query Log</th>
<th>Components Used in Query Log Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Freebase</td>
</tr>
<tr>
<td>Wiki-FB</td>
<td>Yes</td>
</tr>
<tr>
<td>Data-FB</td>
<td>Yes</td>
</tr>
<tr>
<td>Wiki-DB</td>
<td>-</td>
</tr>
<tr>
<td>Data-DB</td>
<td>-</td>
</tr>
<tr>
<td>QLog-DB</td>
<td>-</td>
</tr>
</tbody>
</table>
User Studies: Setup

15 Users for Orion, 15 Users for Naïve (A/B testing)

45 Easy, 30 Medium, and 30 Hard Query Tasks Designed

3 Easy, 2 Medium, 2 Hard Queries Assigned per Query Task

105 Query Tasks per System in Total

4 Survey Questions per Query Task

<table>
<thead>
<tr>
<th>Likert Scale Score</th>
<th>Q1: How well do you think the query graph formulated by you captures the required query intent?</th>
<th>Q2: How easy was it to use the interface for formulating this query?</th>
<th>Q3: How satisfactory was the overall experience?</th>
<th>Q4: The interface provide features necessary for easily formulating query graphs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Very Poorly</td>
<td>Very Hard</td>
<td>Unacceptable</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>2</td>
<td>Poorly</td>
<td>Hard</td>
<td>Poor</td>
<td>Disagree</td>
</tr>
<tr>
<td>3</td>
<td>Adequately</td>
<td>Neither Easy Nor Hard</td>
<td>Satisfactory</td>
<td>Uncertain</td>
</tr>
<tr>
<td>4</td>
<td>Well</td>
<td>Easy</td>
<td>Good</td>
<td>Agree</td>
</tr>
<tr>
<td>5</td>
<td>Very Well</td>
<td>Very Easy</td>
<td>Excellent</td>
<td>Strongly Agree</td>
</tr>
</tbody>
</table>

©2015-2016 The University of Texas at Arlington. All Rights Reserved.
User Studies: Conversion Rate

**Conversion Rate:**

- Percentage of query tasks completed successfully
- Successful completion measured using edge isomorphism, and not a binary notion of matching

<table>
<thead>
<tr>
<th>System</th>
<th>Queries</th>
<th>Sample Size</th>
<th>Conversion Rate ($c$)</th>
<th>z-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orion</td>
<td>All</td>
<td>105</td>
<td>$c_O = 0.74$</td>
<td>0.92</td>
<td>0.1788</td>
</tr>
<tr>
<td>Naïve</td>
<td></td>
<td></td>
<td>$c_N = 0.68$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orion</td>
<td>Medium + Hard</td>
<td>60</td>
<td>$c_O = 0.70$</td>
<td>1.36</td>
<td>0.0869</td>
</tr>
<tr>
<td>Naïve</td>
<td></td>
<td></td>
<td>$c_N = 0.58$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Orion has a higher conversion rate than Naïve for complex queries!
As the difficulty level of the query graph being constructed increases, the usability of Orion seems significantly better than Naïve’s.
Query Logs Comparison

Positive edges better captured based on the context of human usage of relationships in Wikipedia.

DBpedia is created using info-boxes in Wikipedia, and is thus very clean. Wiki-DB is highly similar to Data-DB for DBpedia.
Challenges for the LDBC Community

A benchmark query log to help improve the performance of systems such as Orion

A benchmark query set for visual query formulation, for better evaluation of systems
Orion

Prototype

http://idir.uta.edu/orion

Introduction Video

http://bit.ly/1O0GnNo
Thank You! Questions?
Time required to construct query graphs in Orion is comparable to Naïve in most cases, despite the steeper learning curve of Orion due to more features.
User Studies: Efficiency by Time

Time required to construct query graphs in Orion is comparable to Naïve in most cases, despite the steeper learning curve of Orion due to more features.
RDP better than RF and comparable to NB, despite RF and NB being light models

RDP significantly better than SVD and CAR, but worse than RF and NB
Edge Ranking Algorithms

- Simulates only Active Mode
- 43 target query graphs for Freebase
  - 6 two-edged, 10 three-edged, 9 four-edged, 17 five-edged, 1 six-edged (includes medium and hard queries from the user study)
  - 167 input instances
- 33 target query graphs for DBpedia
  - 2 three-edged, 29 four-edged, 2 five-edged
  - 130 input instances
Edge Ranking Algorithms: Efficiency by Number of Suggestions

RDP requires only 40 suggestions, 1.5-4 times fewer than other methods

RDP requires fewer suggestions compared to all other methods
Tuning RDP Parameters

RDP performs better with more random decision paths and higher query log threshold

Considering negative edges in query session is important, as it results in better performance of RDP