How and Why is An Answer (Still) Correct?
Maintaining Provenance in Dynamic Knowledge Graphs

Garima Gaur¶, q, Arnab Bhattacharya¶, Srikanta Bedathur†
garimag@cse.iitk.ac.in, arnabb@cse.iitk.ac.in, srikanta@cse.iitd.ac.in

¶ Indian Institute of Technology, Kanpur, India
† Indian Institute of Technology, Delhi, India
q Thanks SIGIR for covering conference registration cost
Motivation

• Knowledge Graph (KG): collection of facts

• Fact extractors extracting information from various sources

Motivation

- Knowledge Graph (KG): collection of facts
- Fact extractors extracting information from various sources
- Dynamic KGs
  - NELL is continuously at work since 2010
  - 1.9 Wikipedia edits/second

Dynamic data $\implies$ Evolving answer

- List democrats who are running for US president 2020

<table>
<thead>
<tr>
<th>On 30th Jan 2019</th>
<th>On 28th Feb 2019</th>
<th>On 30th March 2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew Yang</td>
<td>Andrew Yang</td>
<td>Bernie Sanders</td>
</tr>
<tr>
<td>Tulsi Gabbard</td>
<td></td>
<td>Joe Biden</td>
</tr>
<tr>
<td>John Delaney</td>
<td>Elizabeth Warren</td>
<td></td>
</tr>
<tr>
<td>Julián Castr</td>
<td>Amy Klobuchar</td>
<td></td>
</tr>
<tr>
<td>Kamala Harris</td>
<td>Bernie Sanders</td>
<td></td>
</tr>
</tbody>
</table>

2“Provenance Semirings”, PODS, 2007
Dynamic data $\implies$ Evolving answer

- List democrats who are running for US president 2020

<table>
<thead>
<tr>
<th>On 30th Jan 2019</th>
<th>On 28th Feb 2019</th>
<th>On 30th March 2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew Yang</td>
<td>Andrew Yang</td>
<td>Bernie Sanders</td>
</tr>
<tr>
<td>Tulsi Gabbard</td>
<td>Elizabeth Warren</td>
<td>Joe Biden</td>
</tr>
<tr>
<td>John Delaney</td>
<td>Amy Klobuchar</td>
<td></td>
</tr>
<tr>
<td>Julián Castr</td>
<td>Bernie Sanders</td>
<td></td>
</tr>
<tr>
<td>Kamala Harris</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Important to propagate changes in facts down to the precomputed (“standing”) queries

---

2“Provenance Semirings”, PODS, 2007
Dynamic data $\Rightarrow$ Evolving answer

- List democrats who are running for US president 2020

<table>
<thead>
<tr>
<th>On 30th Jan 2019</th>
<th>On 28th Feb 2019</th>
<th>On 30th March 2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew Yang</td>
<td>Andrew Yang</td>
<td>Bernie Sanders</td>
</tr>
<tr>
<td>Tulsi Gabbard</td>
<td></td>
<td>Joe Biden</td>
</tr>
<tr>
<td>John Delaney</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Julián Castr</td>
<td>Elizabeth Warren</td>
<td></td>
</tr>
<tr>
<td>Kamala Harris</td>
<td>Amy Klobuchar</td>
<td></td>
</tr>
</tbody>
</table>

- Important to propagate changes in *facts* down to the precomputed (“standing”) queries
- Need a mechanism to keep track of extraction process and the source of information
- *How* provenance\(^2\) captures how a query answer is generated
- Encode provenance as a polynomial – monomial corresponds to derivation

\(^2\)“Provenance Semirings”, PODS, 2007
Figure 1: KG encoding information about phd students, their advisors and collaborators
Find pairs of advisors and collaborators of their students such that the collaborator has a PhD and works in an institute
Find pairs of advisors and collaborators of their students such that the collaborator has a PhD and works in an institute.

Two derivations of answer \( \langle \text{Stonebraker}, \text{Ramakrishnan} \rangle \):  
- Red-colored subgraph: \( \{e_2, e_3, e_6, e_8, e_{17}\} \)  
- Blue-colored subgraph: \( \{e_2, e_3, e_5, e_{14}, e_{17}\} \)  
- Resultant polynomial is \( e_2.e_3.e_6.e_8.e_{17} + e_2.e_3.e_5.e_{14}.e_{17} \)
Problem Statement

Query result maintenance under edge update

Given a knowledge graph \( G(V, E) \) and a set of standing queries \( Q = \{Q_1, Q_2, \ldots, Q_n\} \), maintain result along with their provenance of a subset \( Q' \subseteq Q \) such that \( Q_i, \forall Q_i \in Q' \), gets affected on the deletion or insertion of an edge \( e_d, e_d \in E \).

Query re-computation is impractical due to KG size!

- Framework **HUKA** which incrementally maintains the query result and its provenance under edge insertion or deletion.
HUKA – maintaining How provenance under Updates to Knowledge grAph
Handling Edge Insertion: Primary Idea

Shifting focus from exact matches of a query pattern to its partial matches
Shifting focus from exact matches of a query pattern to its partial matches

**Potential Match (PM)**

Any subgraph $S$ of the knowledge graph $G$ which can become an exact match of a query $Q$ after a *single* edge insertion is called a potential match.
Shifting focus from exact matches of a query pattern to its partial matches

**Potential Match (PM)**

Any subgraph $S$ of the knowledge graph $G$ which can become an exact match of a query $Q$ after a single edge insertion is called a potential match.
Handling Edge Insertion: Primary Idea

Shifting focus from exact matches of a query pattern to its partial matches

Potential Match (PM)
Any subgraph $S$ of the knowledge graph $G$ which can become an exact match of a query $Q$ after a *single* edge insertion is called a potential match.
HUKA Framework

- Incremental insertion handling approach – *one* edge at a time
- Addressing three sub-problems:
  1. Pre-compute potential matches (PM) of each query
  2. After insertion, efficiently identify *transformed* PM
  3. Maintain PM to ensure correctness while handling subsequent updates
HUKA Framework

- Incremental insertion handling approach – *one* edge at a time
- Addressing three sub-problems:
  1. Pre-compute potential matches (PM) of each query
  2. After insertion, efficiently identify transformed PM
  3. Maintain PM to ensure correctness while handling subsequent updates
- HUKA operates in two phases
  - Query Registration
  - Update Processing
Query Registration

0. Submit standing query

1. Subquery Construction

2. Annotate KG

3. Execution Plan

Diagram: Graph representation of query registration with nodes and edges labeled with predicates.

Nodes:
- N1, N2, N3, N4, N5, N6, N7, N8, N9, N10

Predicates:
- P1, P2, P3

Connections:
- N1 → N2
- N2 → N3
- N3 → N4
- N4 → N1
- N5 → N6
- N6 → N7
- N7 → N8
- N8 → N9
- N9 → N10

Query: (?x, ?y, ?z)

P1 → (?x, ?y)

P2 → (?y, ?z)

P3 → (?x, ?a)

Execution Plan:
- {P1, P3, P2}
- (P1, P3, P2)
- (P1, P2)
- (P1, P3)
- P2
- P1
- P3
Task 1: PM computation

$G_1$: 1 : 1 PM

$G_2$: 1 : M PM

• Insert $\langle \text{Sarawagi}, \text{worksIn}, \text{IITB} \rangle$: $G_1$ and $G_2$ becomes exact match
Task 1: PM computation

\( G_1: 1 : 1 \text{ PM} \)

\begin{array}{c}
\text{Carey} \\
\text{hadAdvisor} \\
\text{Stonebraker} \\
\text{worksIn} \\
\text{MIT} \\
\text{coAuthor} \\
\text{Sarawagi} \\
\text{hasDegree} \\
\text{PhD}
\end{array}

\( G_2: 1 : M \text{ PM} \)

\begin{array}{c}
\text{Sarawagi} \\
\text{hadAdvisor} \\
\text{PhD} \\
\text{coAuthor} \\
\text{Godbole} \\
\text{hasDegree} \\
\text{PhD}
\end{array}

- Insert \( \langle \text{Sarawagi}, \text{worksIn}, \text{IITB} \rangle \): \( G_1 \) and \( G_2 \) becomes exact match
- Unmatched triple patterns:
  - \( G_1: \langle ?collab, \text{worksIn}, ?org1 \rangle \)
  - \( G_2: \langle ?collab, \text{worksIn}, ?org1 \rangle \) and \( \langle ?prof, \text{worksIn}, ?org2 \rangle \)
- Types of potential matches:
  - \( 1 : 1 \text{ PM} \): New edge matches to single triple pattern
  - \( 1 : M \text{ PM} \): New edge satisfies multiple triple constraints
Task 1: PM Computation

- 1:1 PM (pre-computed): Satisfies subqueries with one less triple pattern

Diagram:

- Node `?org1`: PhD
- Edge `workIn`
- Node `?collab`
- Edge `coAuthor`
- Node `?stud`
- Edge `hadAdvisor`
- Node `?prof`
- Edge `workIn`
- Node `?org2`
- Edge `hasDegree`

Diagram:

- Node `Carey`
- Edge `coAuthor`
- Node `Sarawagi`
- Edge `hasDegree`
- Node `Stonebraker`
- Edge `workIn`
- Node `PhD`
- Node `MIT`
Task 1: PM Computation

• 1 : 1 PM (pre-computed): Satisfies subqueries with one less triple pattern

1 : 1 PM (pre-computed):

• ?org1
• worksIn
• ?collab
• hasDegree
• PhD
• coAuthor
• ?stud
• hadAdvisor
• ?prof
• coAuthor
• hasDegree
• worksIn
• ?org2

• 1 : M PM (lazily computed): On appropriate (expected) edge insertion,
  • If new edge satisfies all the unmatched triple patterns
  • PM directly becomes an exact match
  • An exact match also a partial match – satisfies all subqueries
Task 2: KG annotation

- Efficiently check if the new edge has converted a PM to an exact match
- **Connection points**: PM node expecting an edge
Task 2: KG annotation

- Efficiently check if the new edge has converted a PM to an exact match
- **Connection points**: PM node expecting an edge

![Diagram showing relationships between nodes and edges]

- Annotate all the connection points – avoids materializing subquery results
- Annotation – expected edge and provenance polynomial of corresponding PM
Task 3: PM maintenance

- **Local Plan**: For each subquery
  - AND-OR tree\(^3\) – all possible execution plans
  - Best plan selection based on graph data specific cardinality estimator\(^4\)
  - Collects node signatures – *characteristic set* (CS)

\[
CS(u) = \{ P \mid \langle u, P, v \rangle \}
\]

- Cardinality estimation based on the frequency of a CS

---

\(^3\)“Materialized View Selection and Maintenance Using Multi-query Optimization”, SIGMOD, 2001

\(^4\)“Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins”, ICDE, 2011
Task 3: PM maintenance

- **Local Plan**: For *each* subquery
  - AND-OR tree\(^3\) – all possible execution plans
  - Best plan selection based on graph data specific cardinality estimator\(^4\)
  - Collects node signatures – *characteristic set (CS)*
    \[
    CS(u) = \{ P \mid \langle u, P, v \rangle \}
    \]
  - Cardinality estimation based on the frequency of a CS

- **Global Plan**: For *all* subqueries
  - Merging best local plans of all subqueries of standing queries
  - Promotes re-usability – share intermediate expression computation

\(^3\)“Materialized View Selection and Maintenance Using Multi-query Optimization”, SIGMOD, 2001
\(^4\)“Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins”, ICDE, 2011
Local Plan Construction

Figure 2: Subquery and its AND-OR tree (Boxes $\equiv$ OR; Ellipses $\equiv$ AND)

- Greedily choose the best plan – traversing bottom-up

$$\sum_{\{P_1, P_2\} \subset CS_i} Freq(CS_i) > \sum_{\{P_2, P_3\} \subset CS_i} Freq(CS_i)$$

- Global plan is a combination of best local plans
Update Processing

• **Insert**: \( \langle N6, P_1, N7 \rangle \)

1. Examine incident vertices

2. Find new PM

3.1 Annotate new CP

• HUKA also supports result maintenance under *fact* deletion

• Inverted indexes to support deletion and insertion together
Experimental Results
Setup

- Statistics of datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Vertices</th>
<th>Edges</th>
<th>Predicates</th>
<th>Queries</th>
<th>Avg. Query Size</th>
<th>Subqueries</th>
</tr>
</thead>
<tbody>
<tr>
<td>YAGO2</td>
<td>8.8M</td>
<td>23M</td>
<td>78</td>
<td>4</td>
<td>6.25</td>
<td>26</td>
</tr>
<tr>
<td>DBpedia</td>
<td>32M</td>
<td>117M</td>
<td>53K</td>
<td>215</td>
<td>3.90</td>
<td>879</td>
</tr>
</tbody>
</table>

- Query Set:
  - YAGO2: Benchmark queries used to evaluate RDF-3X\(^5\);
  - DBpedia: Real world queries over DBpedia available from the USEWOD 2014.

- Workload Configuration: Randomly generated with controlled ratio of deletion to insertion operations.

\(^5\)“The RDF-3X engine for scalable management of RDF data”, VLDB, 2010
## Efficiency Comparison

### Baselines against HUKA

<table>
<thead>
<tr>
<th>Dataset</th>
<th>HUKA&lt;sup&gt;6&lt;/sup&gt;</th>
<th>GProM&lt;sup&gt;7&lt;/sup&gt;</th>
<th>ProvSQL&lt;sup&gt;8&lt;/sup&gt;</th>
<th>Neo4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>YAGO2</td>
<td>0.119 s</td>
<td>25.121 s</td>
<td>75.657 s</td>
<td>5.709 s</td>
</tr>
<tr>
<td>DBpedia</td>
<td>1.252 s</td>
<td>5.217 s</td>
<td>6.870 s</td>
<td>99.318 s</td>
</tr>
</tbody>
</table>

### Varying workload impact

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Deletion -Heavy</th>
<th>Deletion -Moderate</th>
<th>Balanced</th>
<th>Insertion -Moderate</th>
<th>Insertion -Heavy</th>
</tr>
</thead>
<tbody>
<tr>
<td>YAGO2</td>
<td>0.062 s</td>
<td>0.091 s</td>
<td>0.126 s</td>
<td>0.146 s</td>
<td>0.169 s</td>
</tr>
<tr>
<td>DBpedia</td>
<td>0.943 s</td>
<td>1.056 s</td>
<td>1.315 s</td>
<td>1.403 s</td>
<td>1.475 s</td>
</tr>
</tbody>
</table>

<sup>6</sup> Code available at https://github.com/gaurgarima/HUKA

<sup>7</sup>“GProM-a swiss army knife for your provenance needs”, IEEE Data Engineering Bulletin, 2018

<sup>8</sup>‘ProvSQL: provenance and probability management in PostgreSQL”, VLDB, 2018
Conclusions

- First provenance-aware query result maintenance solution
- **HUKA** – an end-to-end framework to support maintenance of query result and its how provenance
- Seamlessly handles both insertion and deletion update operations
Thank you!
Questions?