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

Garima Gaur[¶,a], 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

[a] Thanks SIGIR for covering conference registration cost

Motivation

- Knowledge Graph (KG): collection of *facts*
- Fact extractors extracting information from various sources

[1] https://en.wikipedia.org/wiki/Wikipedia:Statistics

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 [1]

[1] https://en.wikipedia.org/wiki/Wikipedia:Statistics

Dynamic data $\implies$ Evolving answer

- List democrats who are running for US president 2020

On 30th Jan 2019
Andrew Yang
Tulsi Gabbard
John Delaney
Julián Castr
Kamala Harris

On 28th Feb 2019
Andrew Yang
⋮
Elizabeth Warren
Amy Klobuchar
Bernie Sanders

On 30th March 2020
Bernie Sanders
Joe Biden

[2] “Provenance Semirings”, PODS, 2007

Dynamic data $\implies$ Evolving answer

- List democrats who are running for US president 2020

On 30th Jan 2019
Andrew Yang
Tulsi Gabbard
John Delaney
Julián Castr
Kamala Harris

On 28th Feb 2019
Andrew Yang
⋮
Elizabeth Warren
Amy Klobuchar
Bernie Sanders

On 30th March 2020
Bernie Sanders
Joe Biden

- Important to propagate changes in *facts* down to the precomputed ("standing") queries

[2] "Provenance Semirings", PODS, 2007

Dynamic data $\implies$ Evolving answer

- List democrats who are running for US president 2020

On 30th Jan 2019
Andrew Yang
Tulsi Gabbard
John Delaney
Julián Castr
Kamala Harris

On 28th Feb 2019
Andrew Yang
⋮
Elizabeth Warren
Amy Klobuchar
Bernie Sanders

On 30th March 2020
Bernie Sanders
Joe Biden

- 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

Toy KG

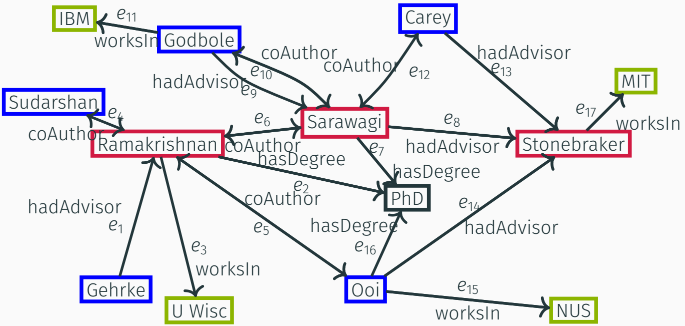


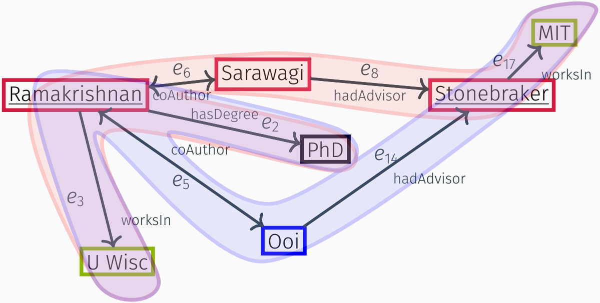
Figure 1: KG encoding information about phd students, their advisors and collaborators

Provenance Polynomial

- *Find pairs of advisors and collaborators of their students such that the collaborator has a PhD and works in an institute*

Provenance Polynomial

- *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 $\langle Stonebraker, Ramakrishnan \rangle$:
  - 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

Handling Edge Insertion: Primary Idea

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

Potential Match (PM)

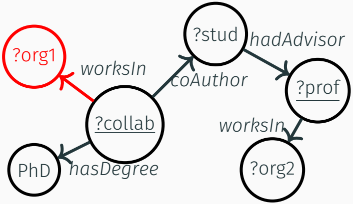
Any subgraph S of the knowledge graph G which can become an exact match of a query Q after a *single* edge insertion is called a **potential match**.

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**.

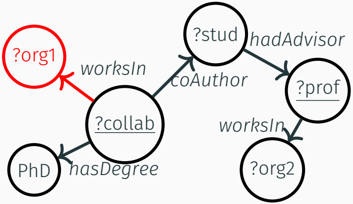


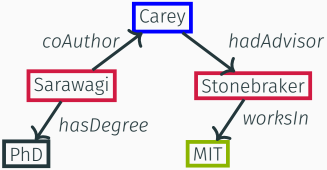
Handling Edge Insertion: Primary Idea

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**.





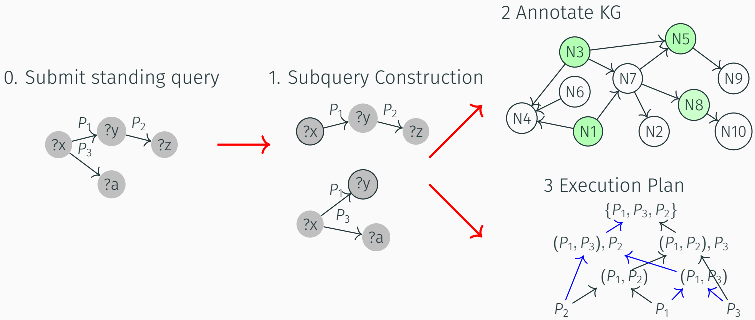
HUKA Framework

- 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

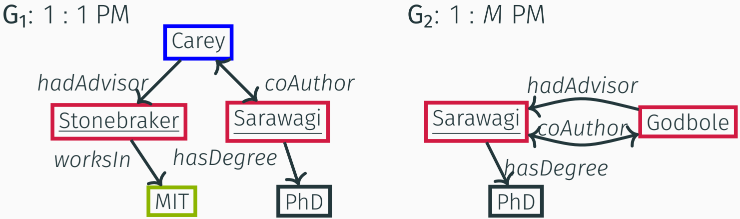
HUKA Framework

- 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

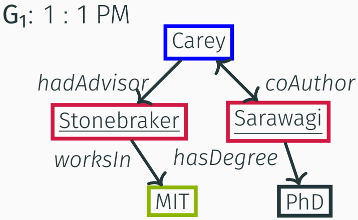


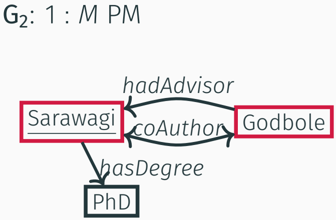
Task 1: PM computation



- Insert ⟨ *Sarawagi, worksIn, IITB* ⟩: G_1 and G_2 becomes exact match

Task 1: PM computation

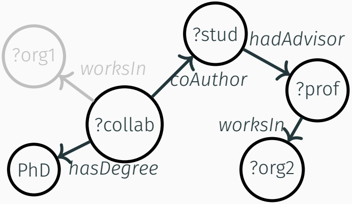


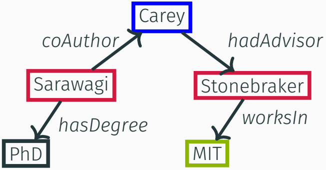


- Insert ⟨ *Sarawagi, worksIn, IITB* ⟩: G_1 and G_2 becomes exact match
- Unmatched triple patterns:
  - G_1: ⟨*?collab, worksIn, ?org1* ⟩
  - G_2: ⟨*?collab, worksIn, ?org1*⟩ and ⟨*?prof, worksIn, ?org2*⟩
- Types of potential matches:
  - **1 : 1** PM: New edge matches to *single* triple pattern
  - **1 : M** PM: New edge satisfies *multiple* triple constraints

Task1: PM Computation

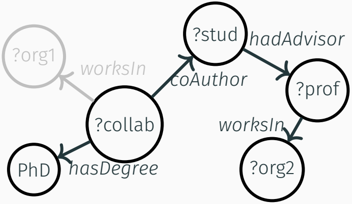
- $1:1$ PM (**pre-computed**): Satisfies subqueries with one less triple pattern

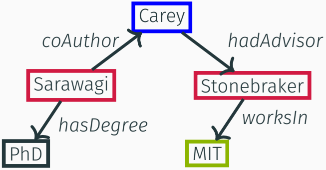




Task1: PM Computation

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

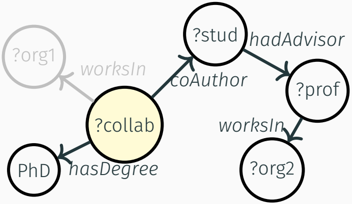


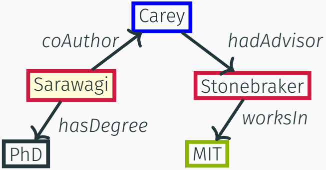


- $1:M$ PM (**lazily computed**): On appropriate (*expected*) edge insertion,
  - If new edge satisfies all the unmatched triple patterns
  - PM directly becomes an exact match
  - An exact match also a partial match – satisifes all subqueries

Task 2: KG annotation

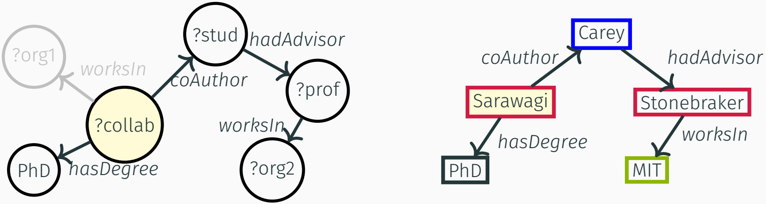
- Efficiently check if the new edge has converted a PM to an exact match
- *Connection points*: PM node expecting an edge





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



- 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

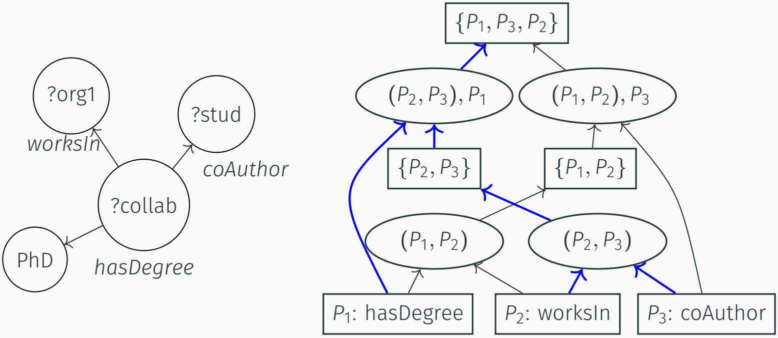


Figure 2: Subquery and its AND-OR tree (Boxes ≡ OR; Ellipses ≡ AND)

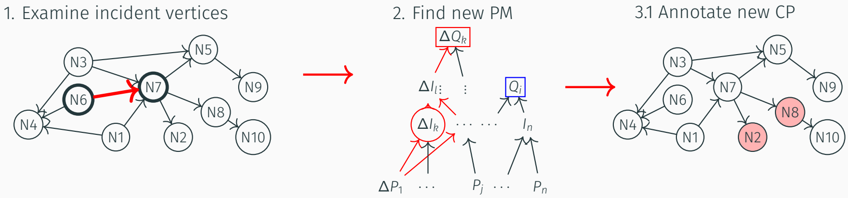
- Greedily choose the best plan – traversing bottom-up

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

- Global plan is a combination of best local plans

Update Processing

- **Insert**: $\langle N6, P_1, N7 \rangle$



- HUKA also supports result maintenance under *fact* deletion
- Inverted indexes to support deletion and insertion together

Experimental Results

Setup

- Statistics of datasets

Dataset	Vertices	Edges	Predicates	Queries	Avg. Query Size	Subqueries
YAGO2	8.8M	23M	78	4	6.25	26
DBpedia	32M	117M	53K	215	3.90	879

- **Query Set**:
  - YAGO2: Benckmark 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 forscalable management of RDF data", VLDB, 2010

Efficiency Comparison

- Baselines against HUKA

Dataset	HUKA[6]	GProM[7]	ProvSQL[8]	Neo4j
YAGO2	0.119 s	25.121 s	75.657 s	5.709 s
DBpedia	1.252 s	5.217 s	6.870 s	99.318 s

- Varying workload impact

Dataset	Deletion -Heavy	Deletion -Moderate	Balanced	Insertion -Moderate	Insertion -Heavy
YAGO2	0.062 s	0.091 s	0.126 s	0.146 s	0.169 s
DBpedia	0.943 s	1.056 s	1.315 s	1.403 s	1.475 s

[6] Code available at https://github.com/gaurgarima/HUKA

[7] "GProM-a swiss army knife for your provenance needs", IEEE Data Engineering Bulletin, 2018

[8] '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?