Natural Language Question Answering Over Knowledge Graph—A Data-Driven Approach

Lei Zou

Peking University
Institute of Computer Science and Technology

Joint work with Jeffery Xu Yu(CUHK), Haixun Wang (Google), Tamer Ozsu (UW), Lei Chen (HKUST), Ruizhe Huang, Shuo Han, Youhuan Li, Dongyan Zhao (PKU)

RDF and Semantic Web

- RDF is a language for the conceptual modeling of information about web resources
- A building block of semantic web
  - Facilitates exchange of information
  - Search engines can retrieve more relevant information
  - Facilitates data integration (mashes)
- Machine understandable
  - Understand the information on the web and the interrelationships among them

Outline

RDF Introduction

gAnswer: Natural Language Question Answering over RDF

- A Data Driven Approach [Zou et al., SIGMOD 2014; Zheng et al., SIGMOD 2015]

Outline

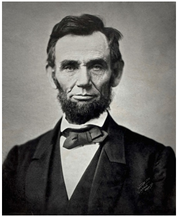
RDF Introduction

gAnswer: Natural Language Question Answering over RDF

A Data Driven Approach [Zou et al., SIGMOD 2014; Zheng et al., SIGMOD 2015]

RDF Introduction

http://en.wikipedia.org/wiki/Abraham_Lincoln

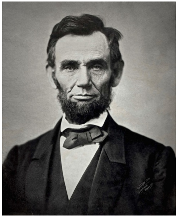


- Everything is an uniquely named resource

RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names

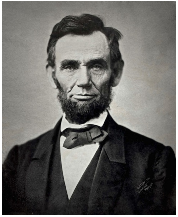
xmlns:y=http://en.wikipedia.org/wiki
y:Abraham_Lincoln



RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names
- Properties of resources can be defined

xmlns:y=http://en.wikipedia.org/wiki
y:Abraham_Lincoln



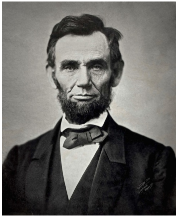
Abraham_Lincoln:hasName "Abraham Lincoln"
Abraham_Lincoln:BornOnDate: "1809-02-12"
Abraham_Lincoln:DiedOnDate: "1865-04-15"

RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names
- Properties of resources can be defined
- Relationships with other resources can be defined

xmlns:y=http://en.wikipedia.org/wiki

y:Abraham_Lincoln



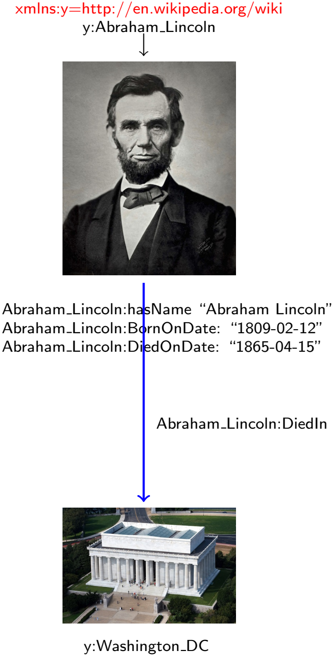
Abraham_Lincoln:hasName "Abraham Lincoln"
Abraham_Lincoln:BornOnDate: "1809-02-12"
Abraham_Lincoln:DiedOnDate: "1865-04-15"

Abraham_Lincoln:DiedIn

y:Washington_DC

RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names
- Properties of resources can be defined
- Relationships with other resources can be defined
- Resources can be contributed by different people/groups and can be located anywhere in the web
  - Integrated web "database"

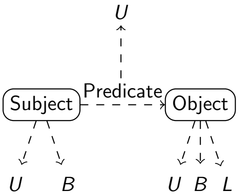


RDF Data Model

- Triple: Subject, Predicate (Property), Object (s, p, o)

Subject:	the entity that is described (URI or blank node)
Predicate:	a feature of the entity (URI)
Object:	value of the feature (URI, blank node or literal)

- $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$
- Set of RDF triples is called an RDF graph



U: set of URIs
B: set of blank nodes
L: set of literals

Subject	**Predicate**	**Object**
Abraham_Lincoln	hasName	"Abraham Lincoln"
Abraham_Lincoln	BornOnDate	"1809-02-12"
Abraham_Lincoln	DiedOnDate	"1865-04-15"

RDF Example Instance

Prefix: y=http://en.wikipedia.org/wiki

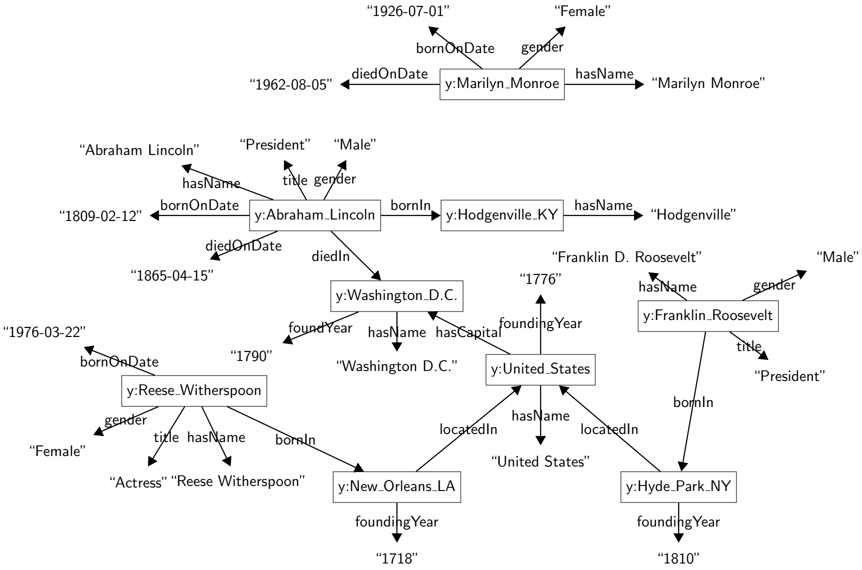
Subject	Predicate	Object
y: Abraham_Lincoln	hasName	“Abraham Lincoln”
y: Abraham_Lincoln	BornOnDate	“1809-02-12”’
y: Abraham_Lincoln	DiedOnDate	“1865-04-15”
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y: Abraham_Lincoln	DiedIn	y: Washington_DC
y:Abraham_Lincoln	title	“President”
y:Abraham_Lincoln	gender	“Male”
y: Washington_DC	hasName	“Washington D.C.”
y:Washington_DC	foundingYear	“1790”
y:Hodgenville_KY	hasName	“Hodgenville”
y:United_States	hasName	“United States”
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	“1776”
y:Reese_Witherspoon	bornOnDate	“1976-03-22”
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	“Reese Witherspoon”
y:Reese_Witherspoon	gender	“Female”
y:Reese_Witherspoon	title	“Actress”
y:New_Orleans_LA	foundingYear	“1718”
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	“Franklin D. Roosevelt”
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	“President”
y:Franklin_Roosevelt	gender	“Male”
y:Hyde_Park_NY	foundingYear	“1810”
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	“Female”
y:Marilyn_Monroe	hasName	“Marilyn Monroe”
y:Marilyn_Monroe	bornOnDate	“1926-07-01”
y:Marilyn_Monroe	diedOnDate	“1962-08-05”

URI

Literal

URI

RDF Graph


"1926-07-01"
"Female"
bornOnDate
gender
"1962-08-05"
diedOnDate
y:Marilyn_Monroe
hasName
"Marilyn Monroe"
"Abraham Lincoln"
"President"
"Male"
hasName
title
gender
"1809-02-12"
bornOnDate
y:Abraham_Lincoln
bornIn
y:Hodgenville_KY
hasName
"Hodgenville"
diedOnDate
diedIn
"1865-04-15"
"Franklin D. Roosevelt"
"Male"
"1776"
hasName
gender
y:Washington_D.C.
y:Franklin_Roosevelt
foundYear
hasName
hasCapital
foundingYear
"1976-03-22"
title
"1790"
bornOnDate
"Washington D.C."
y:United_States
"President"
y:Reese_Witherspoon
bornIn
gender
hasName
title
hasName
locatedIn
locatedIn
bornIn
"Female"
"United States"
"Actress"
"Reese Witherspoon"
y:New_Orleans_LA
y:Hyde_Park_NY
foundingYear
foundingYear
"1718"
"1810"

RDF Query Model

- Query Model - **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- Given U (set of URIs), L (set of literals), and V (set of variables), a SPARQL expression is defined recursively:
  - an atomic triple pattern, which is an element of

$$(U \cup V) \times (U \cup V) \times (U \cup V \cup L)$$

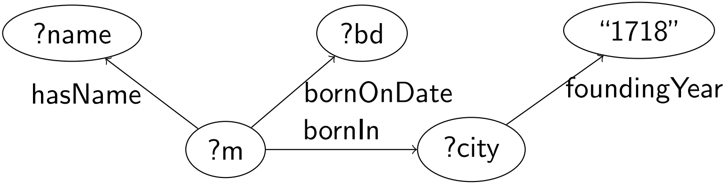
    - ?x hasName "Abraham Lincoln"
  - P FILTER R, where P is a graph pattern expression and R is a built-in SPARQL condition (i.e., analogous to a SQL predicate)
    - ?x price ?p FILTER(?p $<$ 30)
  - $P1$ AND/OPT/UNION $P2$, where $P1$ and $P2$ are graph pattern expressions
- Example:

```
SELECT ?name
WHERE {
  ?m <bornIn> ?city . ?m <hasName> ?name .
  ?m<bornOnDate> ?bd . ?city <foundingYear> ''1718''.
  FILTER(regex(str(?bd),''1976''))
}
```

SPARQL Queries

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city . ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

FILTER(*regex*(*str*(?bd), “1976”))



Naïve Triple Store Design

```
SELECT ?name
WHERE {
  ?m <bornIn> ?city . ?m <hasName> ?name .
  ?m<bornOnDate> ?bd . ?city <foundingYear> ''1718''.
  FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	“Abraham Lincoln”
y:Abraham_Lincoln	bornOnDate	“1809-02-12”
y:Abraham_Lincoln	diedOnDate	“1865-04-15”
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	“President”
y:Abraham_Lincoln	gender	“Male”
y:Washington_DC	hasName	“Washington D.C.”
y:Washington_DC	foundingYear	“1790”
y:Hodgenville_KY	hasName	“Hodgenville”
y:United_States	hasName	“United States”
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	“1776”
y:Reese_Witherspoon	bornOnDate	“1976-03-22”
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	“Reese Witherspoon”
y:Reese_Witherspoon	gender	“Female”
y:Reese_Witherspoon	title	“Actress”
y:New_Orleans_LA	foundingYear	“1718”
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	“Franklin D. Roosevelt”
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	“President”
y:Franklin_Roosevelt	gender	“Male”
y:Hyde_Park_NY	foundingYear	“1810”
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	“Female”
y:Marilyn_Monroe	hasName	“Marilyn Monroe”
y:Marilyn_Monroe	bornOnDate	“1926-07-01”
y:Marilyn_Monroe	diedOnDate	“1962-08-05”

Naïve Triple Store Design

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city . ?m <hasName> ?name.
   ?m<bornOnDate> ?bd . ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	“Abraham Lincoln”
y:Abraham_Lincoln	bornOnDate	“1809-02-12”
y:Abraham_Lincoln	diedOnDate	“1865-04-15”
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	“President”
y:Abraham_Lincoln	gender	“Male”
y:Washington_DC	hasName	“Washington D.C.”
y:Washington_DC	foundingYear	“1790”
y:Hodgenville_KY	hasName	“Hodgenville”
y:United_States	hasName	“United States”
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	“1776”
y:Reese_Witherspoon	bornOnDate	“1976-03-22”
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	“Reese Witherspoon”
y:Reese_Witherspoon	gender	“Female”
y:Reese_Witherspoon	title	“Actress”
y:New_Orleans_LA	foundingYear	“1718”
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	“Franklin D. Roosevelt”
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	“President”
y:Franklin_Roosevelt	gender	“Male”
y:Hyde_Park_NY	foundingYear	“1810”
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	“Female”
y:Marilyn_Monroe	hasName	“Marilyn Monroe”
y:Marilyn_Monroe	bornOnDate	“1926-07-01”
y:Marilyn_Monroe	diedOnDate	“1962-08-05”

```
SELECT T2.object
FROM   T as T1, T as T2, T as T3,
       T as T4
WHERE T1.property="bornIn"
AND T2.property="hasName"
AND T3.property="bornOnDate"
AND T1.subject=T2.subject
AND T2.subject=T3.subject
AND T4.propety="foundingYear"
AND T1.object=T4.subject
AND T4.object="1718"
AND T3.object LIKE '%1976%'
```

Naïve Triple Store Design

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city . ?m <hasName> ?name .
   ?m<bornOnDate> ?bd . ?city <foundingYear>
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	“Abraham Lincoln”
y:Abraham_Lincoln	bornOnDate	“1809-02-12”
y:Abraham_Lincoln	diedOnDate	“1865-04-15”
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	“President”
y:Abraham_Lincoln	gender	“Male”
y:Washington_DC	hasName	“Washington D.C.”
y:Washington_DC	foundingYear	“1790”
y:Hodgenville_KY	hasName	“Hodgenville”
y:United_States	hasName	“United States”
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	“1776”
y:Reese_Witherspoon	bornOnDate	“1976-03-22”
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	“Reese Witherspoon”
y:Reese_Witherspoon	gender	“Female”
y:Reese_Witherspoon	title	“Actress”
y:New_Orleans_LA	foundingYear	“1718”
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	“Franklin D. Roosevelt”
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	“President”
y:Franklin_Roosevelt	gender	“Male”
y:Hyde_Park_NY	foundingYear	“1810”
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	“Female”
y:Marilyn_Monroe	hasName	“Marilyn Monroe”
y:Marilyn_Monroe	bornOnDate	“1926-07-01”
y:Marilyn_Monroe	diedOnDate	“1962-08-05”

Too many self-joins!

```
SELECT T2.object
FROM   T as T1, T as T2, T as T3,
       T as T4
WHERE T1.property="bornIn"
AND T2.property="hasName"
AND T3.property="bornOnDate"
AND T1.subject=T2.subject
AND T2.subject=T3.subject
AND T4.propety="foundingYear"
AND T1.object=T4.subject
AND T4.object="1718"
AND T3.object LIKE '%1976%'
```

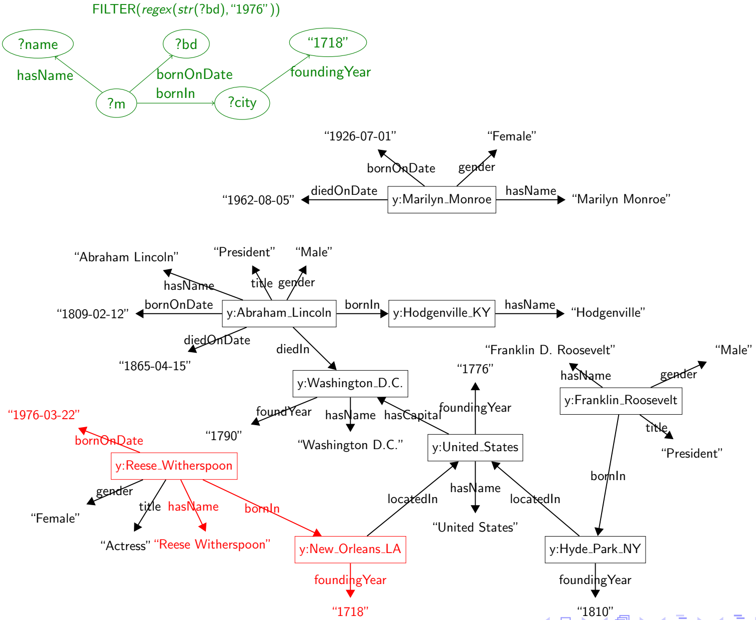
Existing Solutions

1. Property table
   - Each class of objects go to a different table $\Rightarrow$ similar to normalized relations
   - Eliminates some of the joins
2. Vertically partitioned tables
   - For each property, build a two-column table, containing both subject and object, ordered by subjects
   - Can use merge join (faster)
   - Good for subject-subject joins but does not help with subject-object joins
3. Exhaustive indexing
   - Create indexes for each permutation of the three columns
   - Query components become range queries over individual relations with merge-join to combine
   - Excessive space usage

gStore [Zou et al., VLDB 11], [Zou et al., VLDB J 14] – General Idea

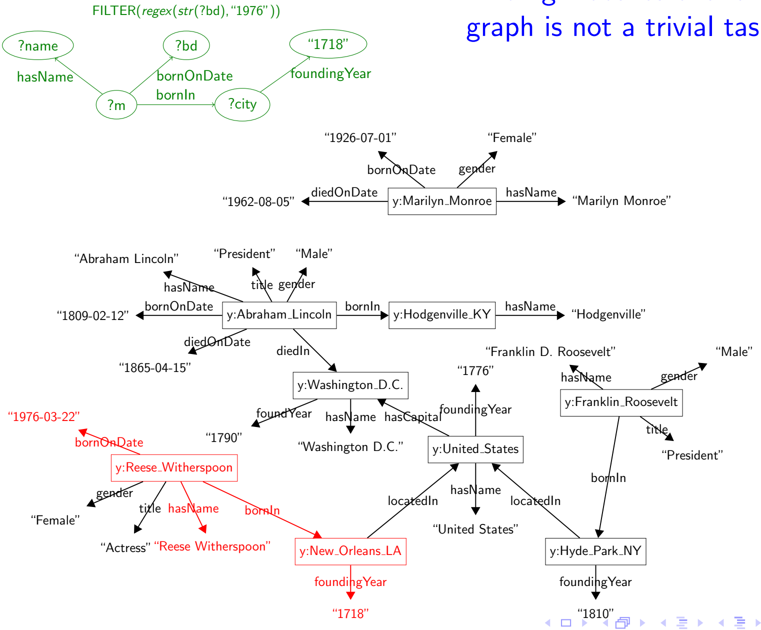
- We work directly on the RDF graph and the SPARQL query graph
  - Answering SPARQL query $\equiv$ subgraph matching
  - Subgraph matching is computationally expensive
- Use a signature-based encoding of each entity and class vertex to speed up matching
- Filter-and-evaluate
  - Use a false positive algorithm to prune nodes and obtain a set of candidates; then do more detailed evaluation on those
- We develop an index (VS*-tree) over the data signature graph (has light maintenance load) for efficient pruning

0. Start with RDF Graph G



0. Start with RDF Graph G

Finding matches over a large graph is not a trivial task!



Outline

RDF Introduction

gAnswer: Natural Language Question Answering over RDF

A Data Driven Approach [Zou et al., SIGMOD 2014; Zheng et al., SIGMOD 2015]

gAnswer: Natural Language Question Answering Over Knowledge Graph–A Graph Data Driven Approach

- An Easy-to-Use Interface to Access Knowledge Graph
  - It is interesting to both academia and industry.
  - Interdisciplinary research between database and NLP (natural language processing) communities.

gAnswer: Natural Language Question Answering Over Knowledge Graph–A Graph Data Driven Approach

- An Easy-to-Use Interface to Access Knowledge Graph
  - It is interesting to both academia and industry.
  - Interdisciplinary research between database and NLP (natural language processing) communities.

Running Example

Question: Who was married to an actor that play in Philadelphia ?

Subject	**Property**	**Object**
Antonio_Banderas	type	actor
Antonio_Banderas	spouse	Melanie_Griffith
Antonio_Banderas	starring	Philadelphia_(film)
Philadelphia_(film)	type	film
Jonathan_Demme	director	film
Philadelphia	type	city
Aaron_McKie	bornIn	Philadelphia
James_Anderson	playForTeam	Philadelphia_76ers
Constantin_Stanislavski	create	An_Actor_Prepares
Philadelphia_76ers	type	Basketball_team
An_Actor_Prepares	type	Book

Running Example

Question: Who was married to an actor that play in Philadelphia ?

Subject	Property	Object
Antonio_Banderas	type	actor
Antonio_Banderas	spouse	Melanie_Griffith
Antonio_Banderas	starring	Philadelphia_(film)
Philadelphia_(film)	type	film
Jonathan_Demme	director	film
Philadelphia	type	city
Aaron_McKie	bornIn	Philadelphia
James_Anderson	playForTeam	Philadelphia_7
Constantin_Stanislavski	create	An_Actor_Prepa
Philadelphia_76ers	type	Basketball_team
An_Actor_Prepares	type	Book

Search Needs a Shake-Up

COMMENT

BIOETHICS Growth in genome screening could cause dangerous meddling p.27 | EVOLUTION How genes and culture have shaped our ability to cooperate p.29 | CHEMISTRY Debating how life got going on the early Earth p.30 | EXHIBITION Wildlife paintings from Yukon to Yellowstone p.32

Keyword

Search

Search needs a shake-up

On the twentieth anniversary of the World Wide Web's public release, **Oren Etzioni** calls on researchers to think outside the keyword box and improve Internet trawling.

Two decades after Internet pioneer Tim Berners Lee introduced his World Wide Web project to the world using the alt.hypertext newsgroup, web search is on the cusp of a profound change — from simple document retrieval to question answering. Instead of poring over long lists of documents that contain requested keywords, users need direct answers to their questions. With sufficient scientific and financial investment, we could soon view today's keyword searching with the same nostalgia and amusement reserved for bygone technologies such as electric typewriters and vinyl records.

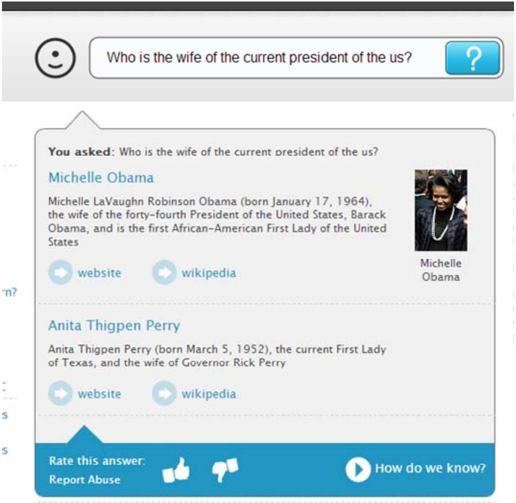
But this transformation could be unreasonably delayed. As a community, computer scientists have underinvested in tools that can synthesize sophisticated answers to questions, and have instead focused on incremental progress in lowest-common-denominator search. The classic keyword search box exerts a powerful gravitational pull. Academics and industry researchers need to achieve the intellectual 'escape velocity' necessary to revolutionize search. They must invest much more in bold strategies that can achieve natural-language searching and answering, rather than providing the electronic equivalent of the index at the back of a reference book.

Today, that 'book' is distributed over billions of web pages of uneven quality, and much effort has been directed at ranking the most useful results. Such engines readily index billions of documents, but overwhelm their users with millions of results in response to simple queries. This quandary only worsens as the number of web pages

" Academics and industry researchers must invest much more in bold strategies that can achieve natural-language searching and answering."
—Oren Etzioni, NATURE, Vol 476, p25-26, 2011.

Some Interesting Products

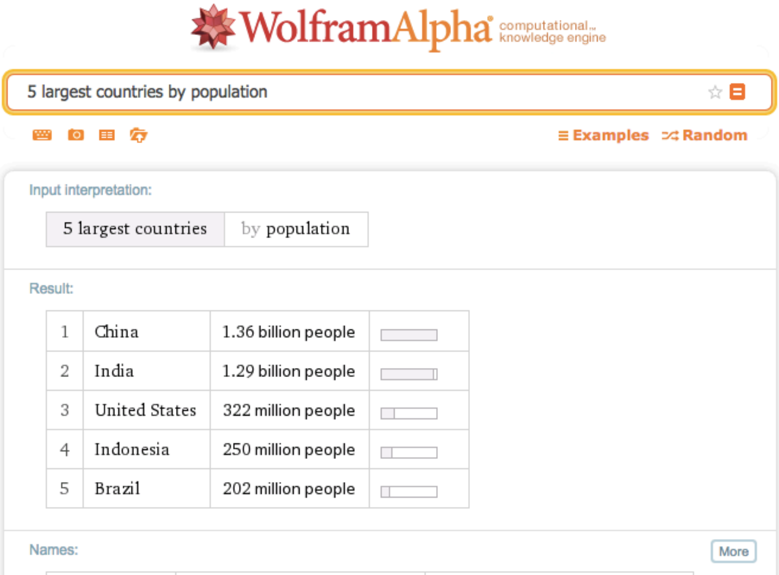
EVI— acquired by Amazon on October 2012.



William Tunstall-Pedoe: True Knowledge: Open-Domain Question Answering Using Structured Knowledge and Inference. AI Magazine 31(3): 80-92 (2010)

Some Interesting Products

WolframAlpha



1	China	1.36 billion people	
2	India	1.29 billion people	
3	United States	322 million people	
4	Indonesia	250 million people	
5	Brazil	202 million people	

Existing Solutions

Who was married to an actor that play in Philadelphia ?

Translate NL Question to structured queries

```
SELECT ?y
WHERE {
?x starring Philadelphia_(film).
?x type actor.
?x spouse ?y . }
```

Query Processing

Melanie Griffith

Existing Solutions

Ambiguity

Who was married to an actor that play in Philadelphia ?

```
SELECT ?y
WHERE {
?x starring Philadelphia_(film).
?x type actor.
?x spouse ?y . }
```

Translate NL Question to structured queries

Philadelpha

Philadelpha_(film)

Philadelpha_76ers

Query Processing

Melanie Griffith

Existing Solutions

Ambiguity

Who was married to an actor that play in Philadelphia?

playForTeam
↓
starring
↓
director

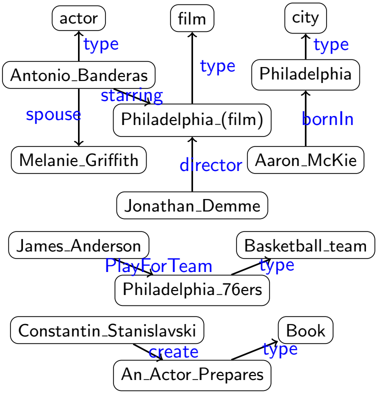
Translate NL Question to structured queries

```
SELECT ?y
WHERE {
?x starring Philadelphia_(film).
?x type actor.
?x spouse ?y . }
```

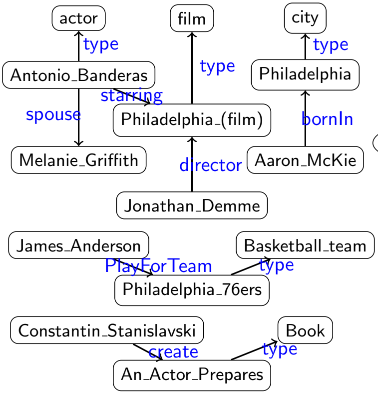
Query Processing

Melanie Griffith

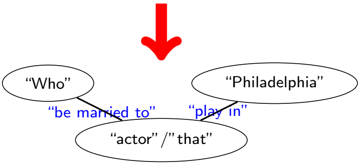
Our Method: Data Driven [Zou et al., SIGMOD 14]



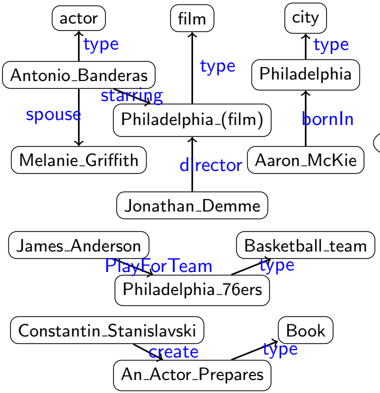
Our Method: Data Driven [Zou et al., SIGMOD 14]



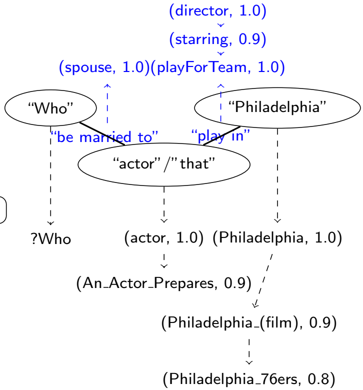
Who was married to an actor that play in Philadelphia ?



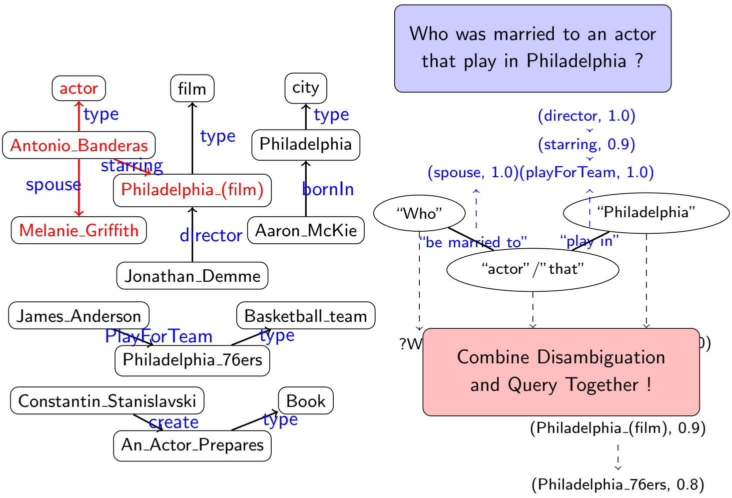
Our Method: Data Driven [Zou et al., SIGMOD 14]



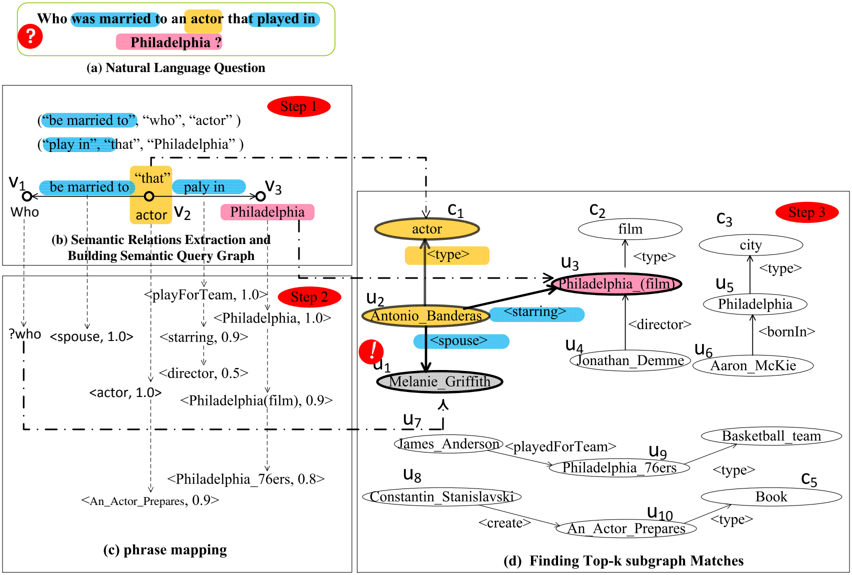
Who was married to an actor that play in Philadelphia ?



Our Method: Data Driven [Zou et al., SIGMOD 14]



Our Method: Framework



Our Method: Offline

- **TASK**: Building a paraphrase dictionary to map relation phrases to predicates in RDF graph.
- **METHOD**: Data Driven Approach

Relation Phrases	Predicates or Predicate Paths	Confidence Probability
"be married to"	<spouse>	1.0
"play in"	<starring>	0.9
"play in"	<director>	0.5
"uncle of"	<hasChild> <hasChild> <hasChild>	0.8
… …	… …	… …

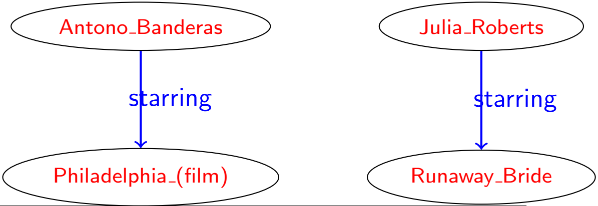
Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩), (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),......
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),......

Our Method: Offline

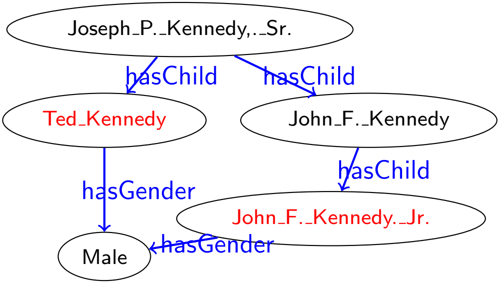
- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths



Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),......
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),......

Our Method: Offline

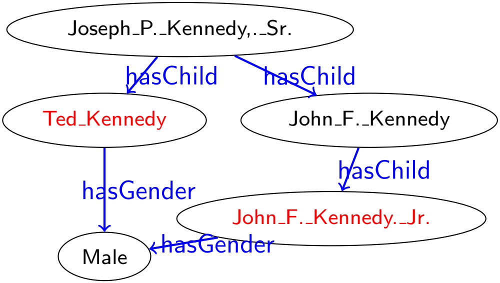
- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

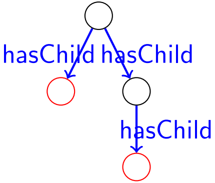


Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),......
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),......

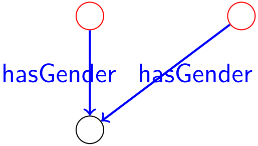
Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths



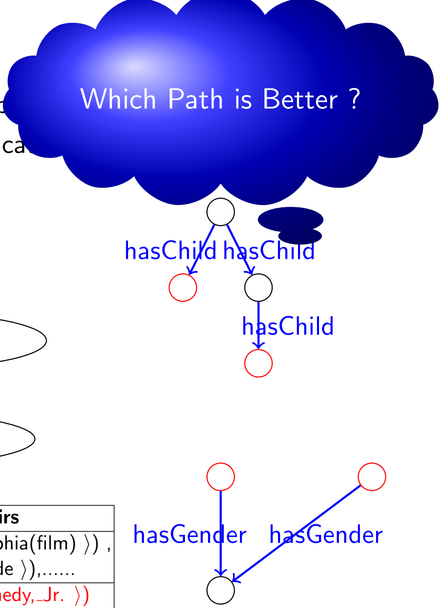


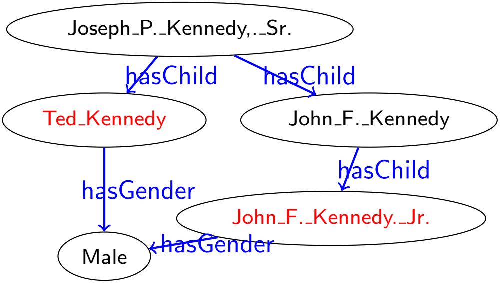
Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),......
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),......



Our Method: Offline

- **INPUT**: Relation Phrases and Supp
- **OUTPUT**: Possible Mapping Predica



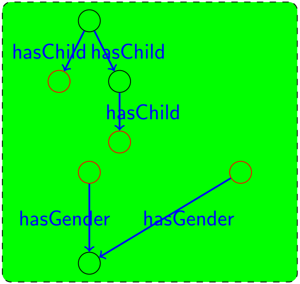


Relation Phrase	**Supporting Entity Pairs**
“play in”	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),......
“uncle of”	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),......

Our Method: Offline

relation phrase: rel_i

"uncle of"

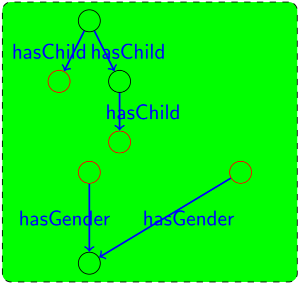


virtual document

Our Method: Offline

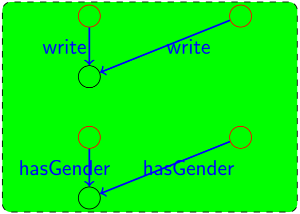
relation phrase: rel_i

"uncle of"



virtual document

"co-author with"

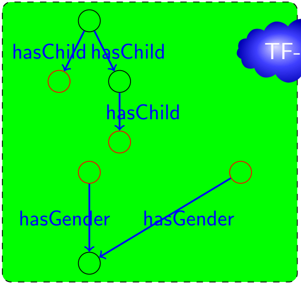


virtual document

Our Method: Offline

relation phrase: rel_i

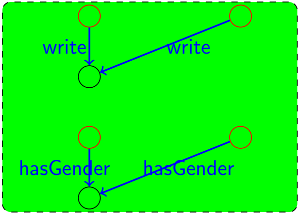
"uncle of"



TF-IDF

virtual document

"co-author with"



virtual document

Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

Definition

Given a predicate path L , the *tf-value* of L in $PS(rel_i)$ is defined as follows:

$$tf(L, PS(rel_i)) = |\{Path(v_i^j, v_i'^j) | L \in Path(v_i^j, v_i'^j)\}|$$

The *idf-value* of L over the whole relation phrase dictionary $T = \{rel_1, ..., rel_n\}$ is defined as follows:

$$idf(L, T) = \log \frac{|T|}{|\{rel_i \in T | L \in PS(rel_i)\}| + 1}$$

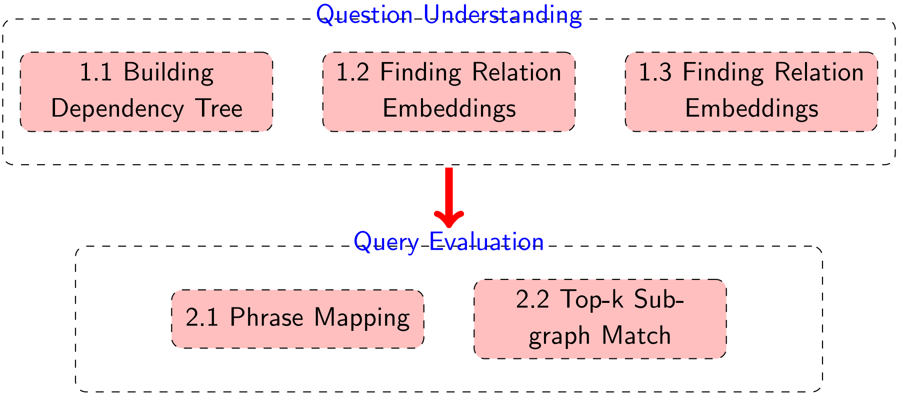
The *tf-idf value* of L is defined as follows:

$$\mathrm{tf-idf}(L, PS(rel_i), T) = tf(L, PS(rel_i)) \times idf(L, T)$$

We define the confidence probability of mapping relation phrase *rel* to predicate or predicate path L as follows.

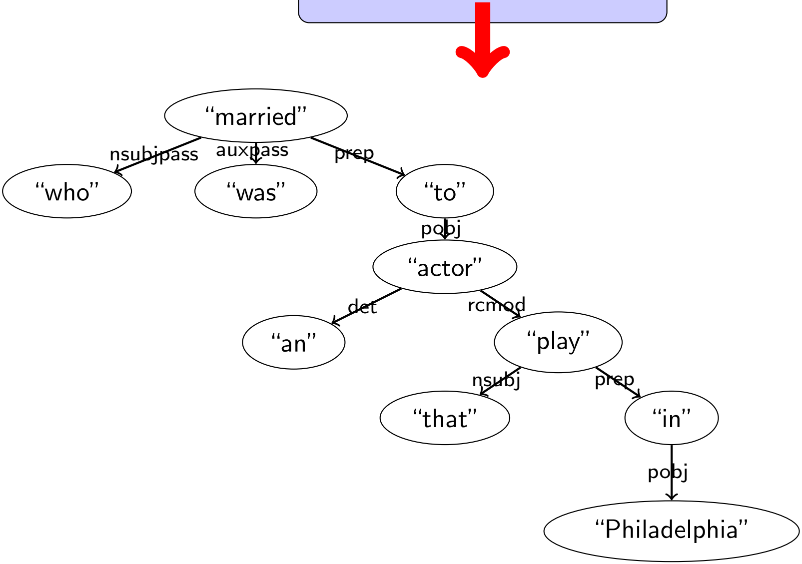
$$\delta(rel, L) = \mathrm{tf-idf}(L, PS(rel_i), T) \tag{1}$$

Online: Framework



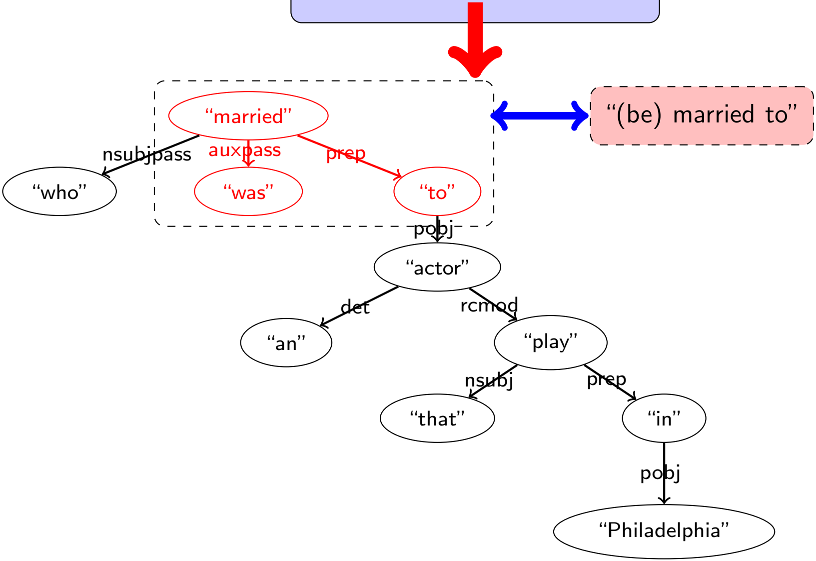
Online: Step 1.1

Who was married to an actor that play in Philadelphia ?



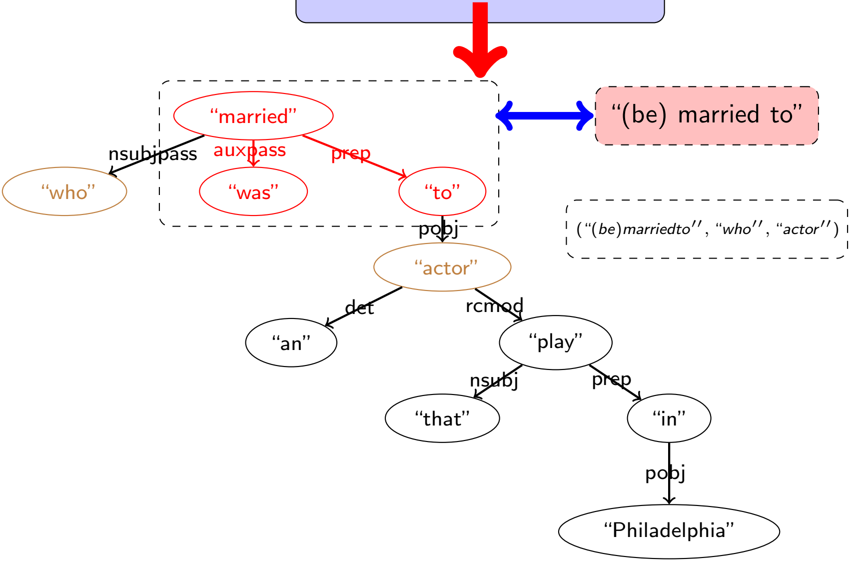
Online: Step 1.2

Who was married to an actor that play in Philadelphia ?



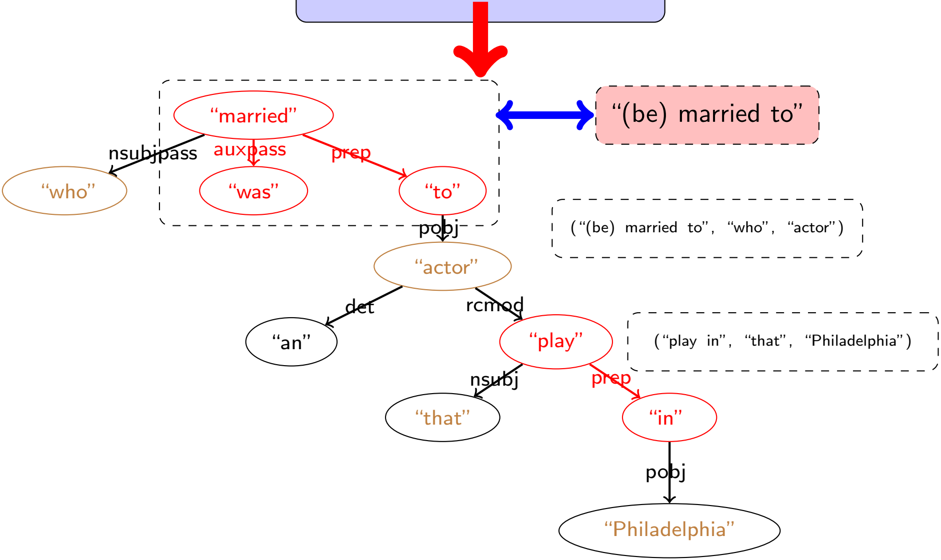
Online: Step 1.3

Who was married to an actor that play in Philadelphia ?



Online: Step 1.3

Who was married to an actor that play in Philadelphia ?



Online: Step 1.4

Who was married to an actor that play in Philadelphia ?

(“(be) married to”, “who”, “actor”)

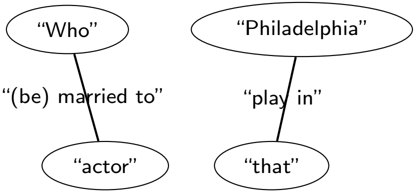
(“play in”, “that”, “Philadelphia”)

Online: Step 1.4

Who was married to an actor that play in Philadelphia ?

(“(be) married to”, “who”, “actor”)

(“play in”, “that”, “Philadelphia”)

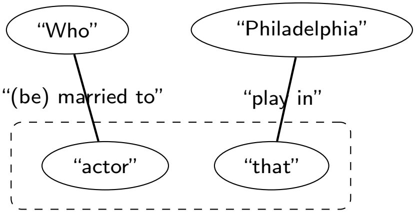


Online: Step 1.4

Who was married to an actor that play in Philadelphia ?

("(be) married to", "who", "actor")

("play in", "that", "Philadelphia")



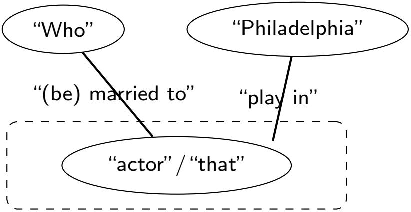
coreference resolution

Online: Step 1.4

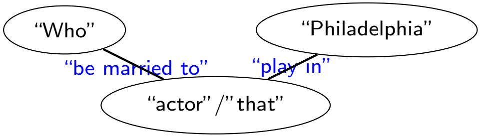
Who was married to an actor that play in Philadelphia ?

(“(be) married to”, “who”, “actor”)

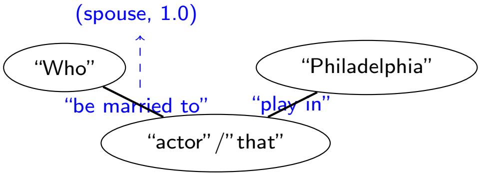
(“play in”, “that”, “Philadelphia”)



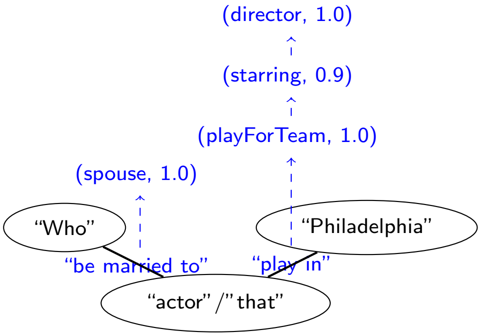
Online: Step 2.1 Mapping Edge



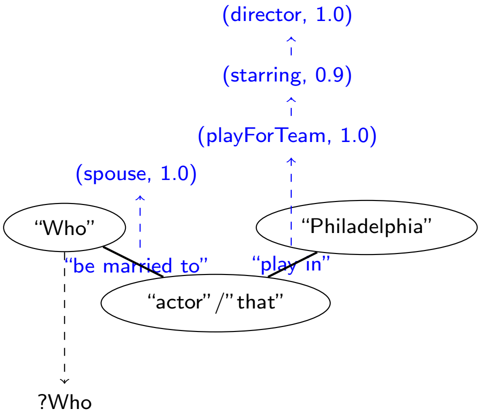
Online: Step 2.1 Mapping Edge



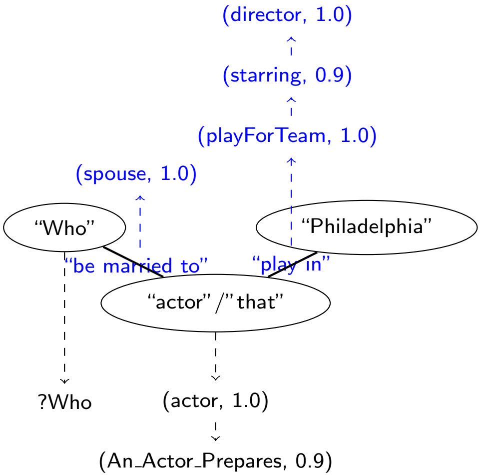
Online: Step 2.1 Mapping Edge



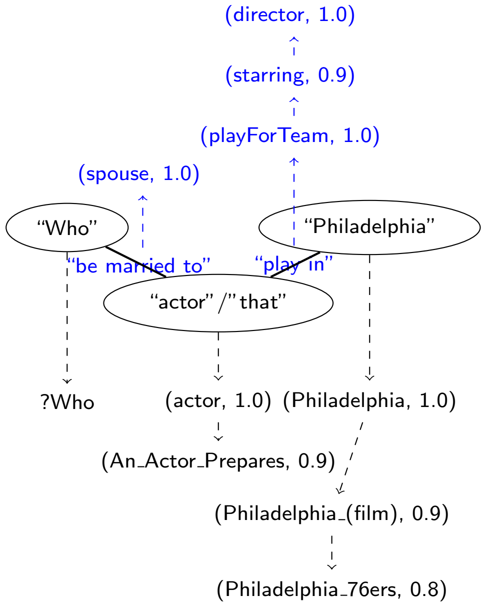
Online: Step 2.2 Mapping Vertices



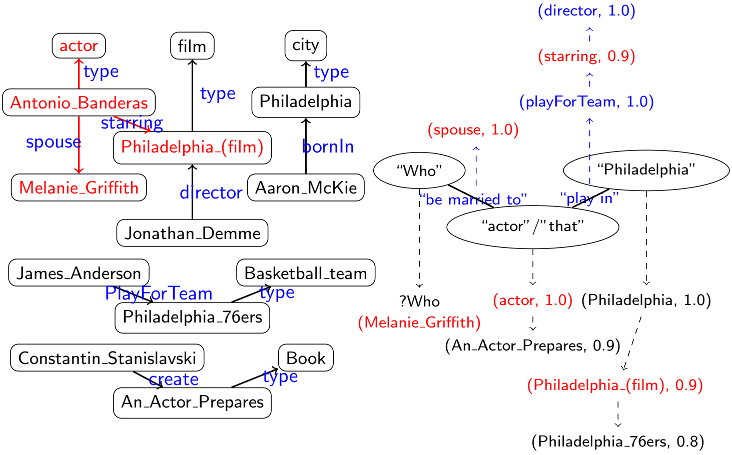
Online: Step 2.2 Mapping Vertices



Online: Step 2.2 Mapping Vertices

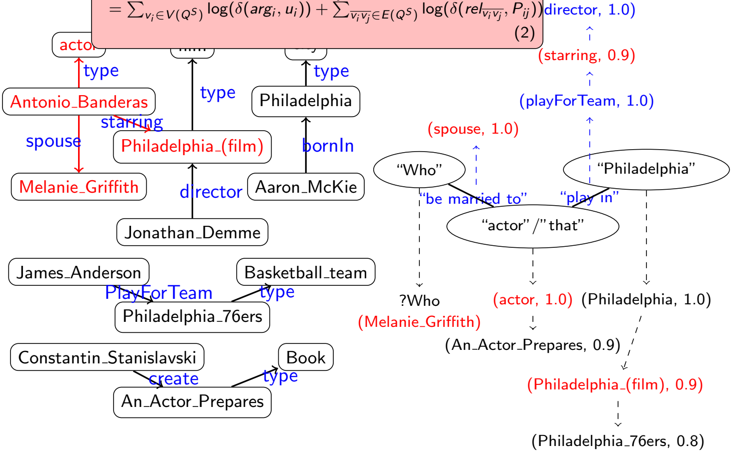


Online: Step 2.3 Finding Top-k Matches



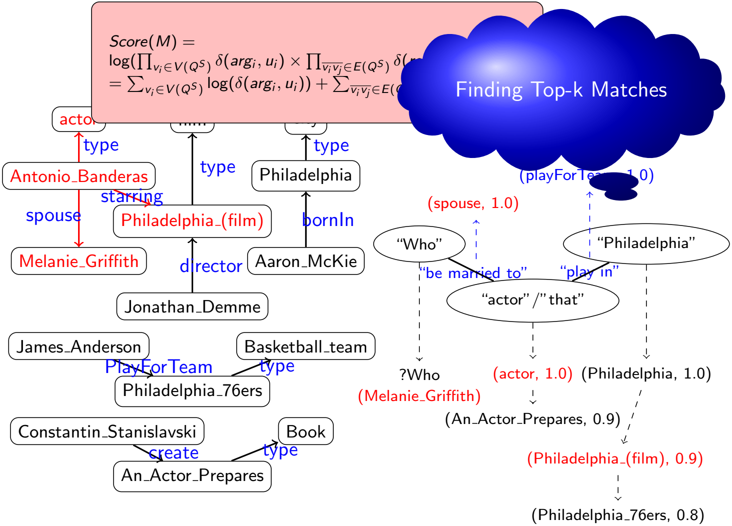
Online: Step 2.3 Finding Top-k Matches

$$Score(M) = \log\left(\prod_{v_i \in V(Q^S)} \delta(arg_i, u_i) \times \prod_{\overline{v_i v_j} \in E(Q^S)} \delta(rel_{\overline{v_i v_j}}, P_{ij})\right) = \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + \sum_{\overline{v_i v_j} \in E(Q^S)} \log(\delta(rel_{\overline{v_i v_j}}, P_{ij})) \quad (2)$$



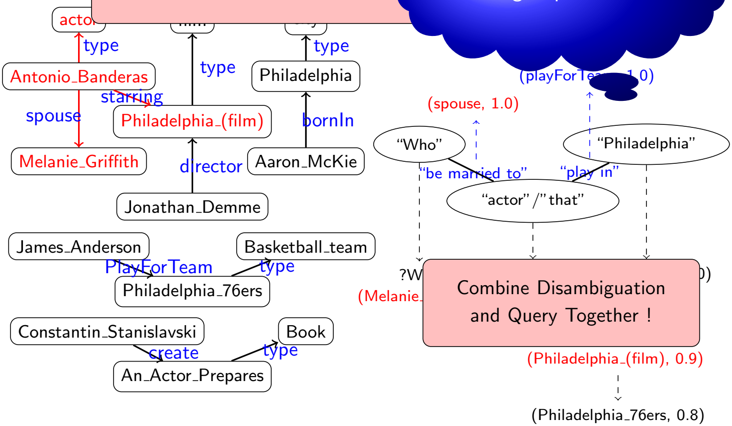
Online: Step 2.3 Finding Top-k Matches

$Score(M) =$
$\log(\prod_{v_i \in V(Q^S)} \delta(arg_i, u_i) \times \prod_{\overline{v_i v_j} \in E(Q^S)} \delta(r$
$= \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + \sum_{\overline{v_i v_j} \in E(Q$



Online: Step 2.3 Finding Top-k Matches

$Score(M) =$
$\log(\prod_{v_i \in V(Q^S)} \delta(arg_i, u_i) \times \prod_{\overline{v_i v_j} \in E(Q^S)} \delta(r$
$= \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + \sum_{\overline{v_i v_j} \in E(Q$



Online: Step 2.3 Finding Top-k Matches

Theorem

Finding Top-k subgraph matches of Q^S over RDF graph G is an NP-hard problem.

Online: Step 2.3 Finding Top-k Matches

Theorem

Finding Top-k subgraph matches of Q^S over RDF graph G is an NP-hard problem.

Require: **Input**: A semantic query graph Q^S and a RDF G. **Output**: Top-k SPARQL Queries, i.e., the top-k matches from Q^S to G.

- **for** each candidate list L_{r_i}, $i = 1, \ldots\ldots, |E(Q^S)|$ **do**
  - Sorting all candidate relations in L_{r_i} in a non-ascending order
- **for** each candidate list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
  - Sorting all candidate entities/classes (i.e., vertices in G') in L_{arg_j} in a non-ascending order.
- Set cursor c_i to the head of L_{r_i} and cursor c_j to the head of L_{arg_j}, respectively.
- Set the upper bound $Upbound(Q)$ and the threshold $\theta = -\infty$
- **while** true **do**
  - **for** each cursor c_j in list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
    - Performance a exploration based subgraph isomorphism algorithm from cursor c_j, such as VF2, to find subgraph matches (of Q^S over G), which contains c_j.
  - Update the threshold θ to be the top-k match sore so far.
  - Move all cursors c_i and c_j by one step forward in each list.
  - Update the upper bound $Upbound(Q)$ according to Equation **??**.
  - **if** $\theta \geq Upbound(Q)$ **then**
    - Break // TA-style stopping strategy

Online: Step 2.3 Finding Top-k Matches

Theorem

Finding Top-k subgraph matches of Q^S
NP-hard problem.

TA-style Top-k Algorithm !

Require: **Input**: A semantic query graph Q^S and a RDF G. **Output**
matches from Q^S to G.

for each candidate list L_{r_i}, $i = 1, \ldots\ldots, |E(Q^S)|$ **do**
  Sorting all candidate relations in L_{r_i} in a non-ascending order
for each candidate list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
  Sorting all candidate entities/classes (i.e., vertices in G') in L_{arg_j} in a non-ascending order.
Set cursor c_i to the head of L_{r_i} and cursor c_j to the head of L_{arg_j}, respectively.
Set the upper bound $Upbound(Q)$ and the threshold $\theta = -\infty$
while true **do**
  for each cursor c_j in list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
    Performance a exploration based subgraph isomorphism algorithm from cursor c_j, such as VF2, to find subgraph matches (of Q^S over G), which contains c_j.
  Update the threshold θ to be the top-k match sore so far.
  Move all cursors c_i and c_j by one step forward in each list.
  Update the upper bound $Upbound(Q)$ according to Equation **??**.
  if $\theta \geq Upbound(Q)$ **then**
    Break // TA-style stopping strategy

Experiments: Datasets

- RDF repository: DBPedia

Table : Statistics of RDF Graph

	DBpedia
Number of Entities	5.2 million
Number of Triples	60 million
Number of Predicates	1643
Size of RDF Graphs (in GB)	6.1

- Relation Phrase Dictionary: Patty

Table : Statistics of Relation Phrase Dataset

	wordnet-wikipedia	**freebase-wikipedia**
Number of Textual Patterns	350,568	1,631,530
Number of Entity Pairs	3,862,304	15,802,947
Average Entity Pair Number For Each Pattern	11	9

Experiments: Online

Benchmark: QALD-3, 99 Natural Language Questions

Table : Evaluating QALD-3 Testing Questions (on DBpedia)

	Processed	Right	Partially	Recall	Precision	F-1
Our Method	76	32	11	0.40	0.40	0.40
squall2sparql	96	77	13	0.85	0.89	0.87
CASIA	52	29	8	0.36	0.35	0.36
Scalewelis	70	1	38	0.33	0.33	0.33
RTV	55	30	4	0.34	0.32	0.33
Intui2	99	28	4	0.32	0.32	0.32
SWIP	21	14	2	0.15	0.16	0.16
DEANNA	27	21	0	0.21	0.21	0.21

Experiments: Online

ID	Questions	Response Time (in ms)
Q2	Who was the successor of John F. Kennedy?	1699
Q3	Who is the mayor of Berlin?	677
Q14	Give me all members of Prodigy?	811
Q17	Give me all cars that are produced in Germany ?	297
Q19	Give me all people that were born in Vienna and died in Berlin ?	2557
Q20	How tall is Michael Jordan ?	942
Q21	What is the capital of Canada ?	1342
Q22	Who is the governor of Wyoming ?	796
Q24	Who was the father of Queen Elizabeth II?	538
Q27	Sean Parnell is the governor of which U.S. state ?	1210
Q28	Give me all movies directed by Francis Ford Coppola.	577
Q30	What is the birth name of Angela Merkel ?	250
Q35	Who developed Minecraft ?.	2565
Q39	Give me all companies in Munich.	1312
Q41	Who founded Intel?	1105
Q42	Who is the husband of Amanda Palmer ?	1418
Q44	Which cities does the Weser flow through ?	1139
Q45	Which countries are connected by the Rhine ?	736
Q54	What are the nicknames of San Francisco ?	321
Q58	What is the time zone of Salt Lake City ?	316
Q63	Give me all Argentine films.	427
Q70	Is Michelle Obama the wife of Barack Obama ?	316
Q74	When did Michael Jackson die ?	258
Q76	List the children of Margaret Thatcher.	1139
Q77	Who was called Scarface?	719
Q81	Which books by Kerouac were published by Viking Press ?	796
Q83	How high is the Mount Everest ?	635
Q84	Who created the comic Captain America ?	589
Q86	What is the largest city in Australia ?	1419
Q89	In which city was the former Dutch queen Juliana buried ?	1700
Q98	Which country does the creator of Miffy come from ?	2121
Q100	Who produces Orangina ?	367

Experiments: Online

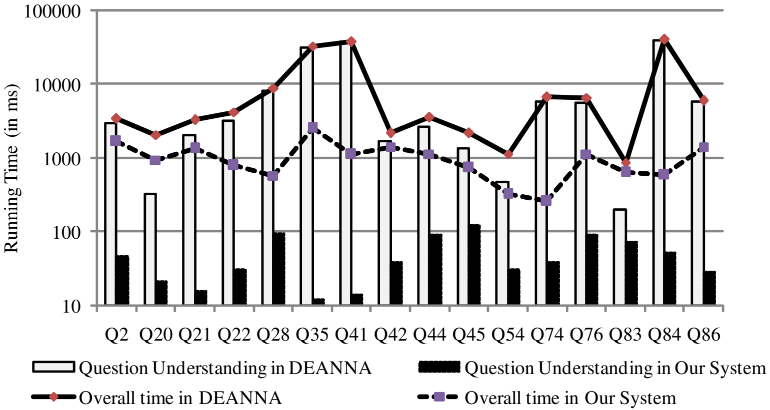


Figure : Online Running Time Comparison

Experiments: Online

QUESTION ANSWERING OVER LINKED DATA

QALD-4: Evaluation results

Workshop website: http://www.sc.cit-ec.uni-bielefeld.de/qald

Results[1] for Task 1: Multilingual question answering over DBpedia

	Total	Processed	Right	Partially	Recall	Precision	F-measure
Xser	50	40	34	6	0.71	0.72	**0.72**
gAnswer	50	25	16	4	0.37	0.37	**0.37**
CASIA	50	26	15	4	0.40	0.32	**0.36**
Intui3	50	33	10	4	0.25	0.23	**0.24**
ISOFT	50	28	10	3	0.26	0.21	**0.23**
RO_FII	50	50	6	0	0.12	0.12	**0.12**

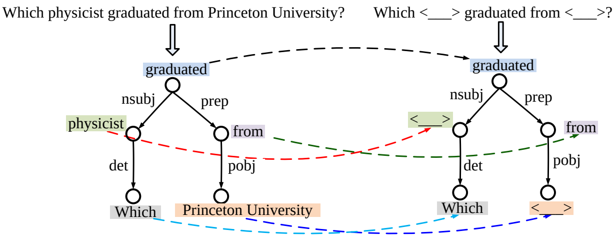
Figure : QALD-4 Results

How to Build Templates for RDF Q/A — An Uncertain Graph Similarity Join Approach [Zheng et al., SIGMOD 15]

Template

which <___> graduated from <___>?

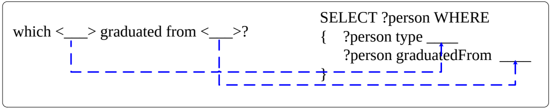
SELECT ?person WHERE
{ ?person type ____
?person graduatedFrom ____
}

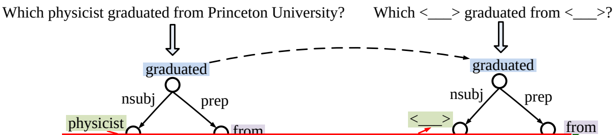


syntactic dependency tree

How to Build Templates for RDF Q/A — An Uncertain Graph Similarity Join Approach [Zheng et al., SIGMOD 15]

Template

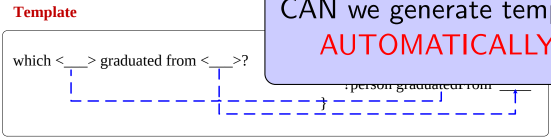




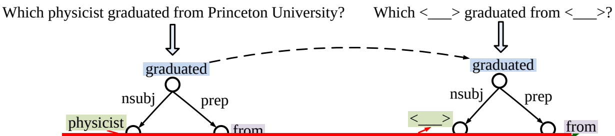
EVI (TrueKnowledge)

- Manually defined templates
- About 1200 templates in 2010
  [Tunstall-Pedoe, AI Magazine 2010]

How to Build Templates for RDF Q/A — An Uncertain Graph Similarity Join Approach [Zheng et al., SIGMOD 15]



CAN we generate templates AUTOMATICALLY ?



EVI (TrueKnowledge)

- Manually defined templates
- About 1200 templates in 2010 [Tunstall-Pedoe, AI Magazine 2010]

What do we have at hand ?

- Natural language question workload, such as Yahoo WebQuestions.
- SPARQL workload, such as DBpeida SPARQL workload.

NLQ Workload

what is the name of justin bieber brother?

Who is the daughter of Ingrid Bergman married to?

Where did saki live?

Which river does the Tower Bridge cross?

who does joakim noah play for?

In which city did John F. Kennedy die?

where are the nfl redskins from?

In which country does the Yangtze River start?

how old is sacha baron cohen?

Give me the birthdays of all actors of the television show Charmed.

who did draco malloy end up marrying?

which countries border the us?

where is rome italy located on a map?

SPARQL Query Workload

```
1.    SELECT DISTINCT ?uri
WHERE {
      res:Albert_Einstein dbo:deathPlace ?uri .
      ?uri rdf:type dbo:City .
}
2.    SELECT DISTINCT ?uri
WHERE {
      res:Bill_Clinton dbo:child ?child .
      ?child dbo:spouse ?uri .
}
3.    PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?uri
WHERE {
      res:Brooklyn_Bridge dbo:crosses ?uri .
}
4.    SELECT DISTINCT ?date
WHERE {
      res:The_Truman_Show dbo:starring ?actor .
      ?actor dbo:birthDate ?date .
}
5.    PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?uri
WHERE {
      res:Nile dbo:sourceCountry ?uri .
}
```

What do we have at hand ?

- Natural language question workload, such as Yahoo WebQuestions.
- SPARQL workload, such as DBpeida SPARQL workload.

NLQ Workload

what is the name of justin bieber brother?

Who is the daughter of Ingrid Bergman married to?

Where did saki live?

Which river does the Tower Bridge cross?

who does joakim noah play for?

In which city did John F. Kennedy die?

where are the nfl redskins from?

In which country does the Yangtze River start?

how old is sacha baron cohen?

Give me the birthdays of all actors of the television show Charmed.

who did draco malloy end up marrying?

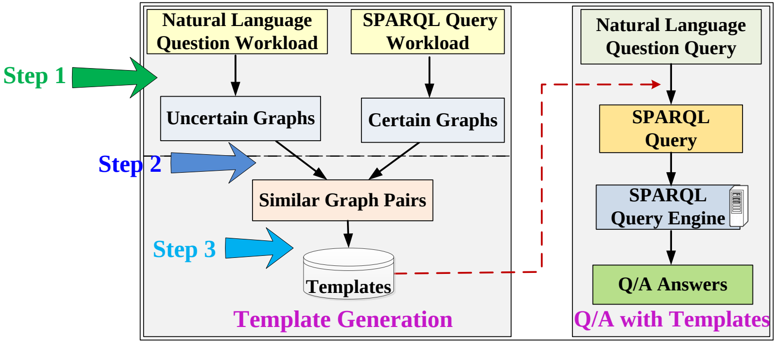
which countries border the us?

where is rome italy located on a map?

SPARQL Query Workload

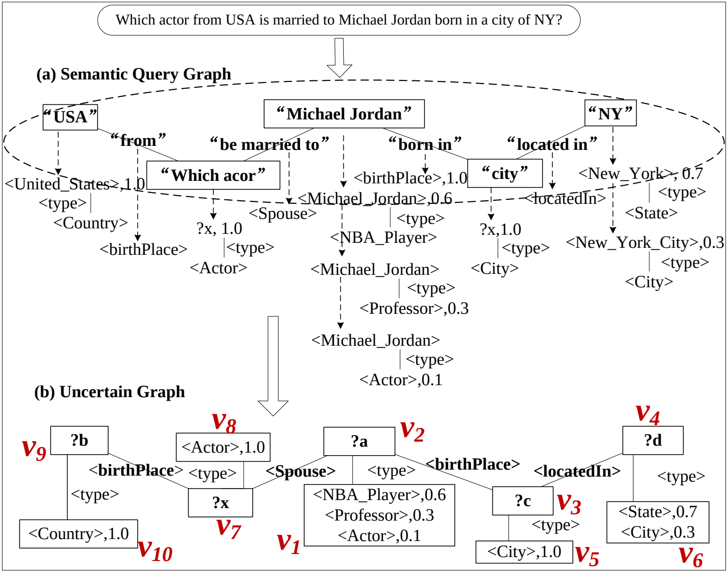
```
1.   SELECT DISTINCT ?uri
WHERE {
        res:Albert_Einstein dbo:deathPlace ?uri .
        ?uri rdf:type dbo:City .
}
2.   SELECT DISTINCT ?uri
WHERE {
        res:Bill_Clinton dbo:child ?child .
        ?child dbo:spouse ?uri .
}
3.    PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?uri
WHERE {
        res:Brooklyn_Bridge dbo:crosses ?uri .
}
4.    SELECT DISTINCT ?date
WHERE {
        res:The_Truman_Show dbo:starring ?actor .
        ?actor dbo:birthDate ?date .
}
5.    PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?uri
WHERE {
        res:Nile dbo:sourceCountry ?uri .
}
```

Framework

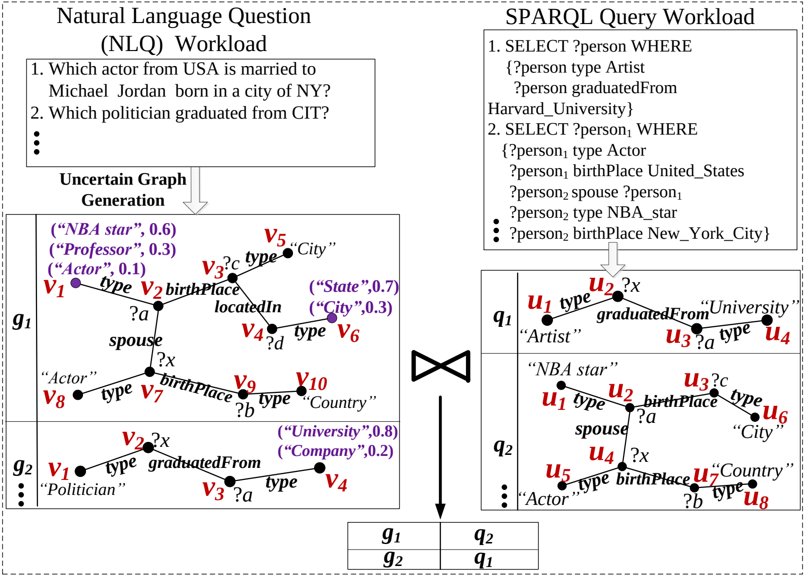


Step 1
Step 2
Step 3
Natural Language Question Workload
SPARQL Query Workload
Uncertain Graphs
Certain Graphs
Similar Graph Pairs
Templates
Template Generation
Natural Language Question Query
SPARQL Query
SPARQL Query Engine
Q/A Answers
Q/A with Templates

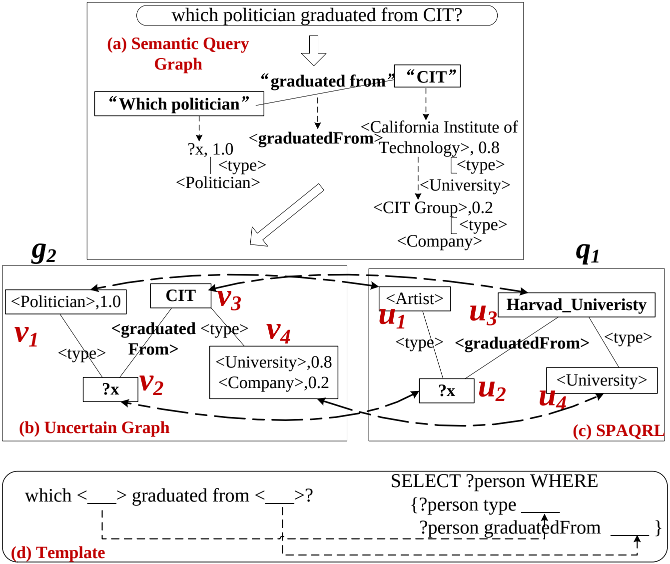
Step1: Uncertain Graph Generation



Step2: Finding Similar Graph Pairs



Step3: Template Generation



Problem Formulation

Definition (**Finding Similar Graph Pairs**)

Given a set of deterministic graphs D (derived from the SPARQL query workload), a set of uncertain graphs U (derived from natural language question workload), a graph edit distance threshold τ, and a similarity probability threshold $\alpha \in (0, 1]$, a *SimJ* query returns the graph pairs $\langle q, g \rangle$ ($q \in D$ and $g \in U$) with $SimP_{\tau}(q, g) \geq \alpha$.

Problem Formulation

Definition (**Finding Similar Graph Pairs**)

Given a set of deterministic graphs D (derived from the SPARQL query workload), a set of uncertain graphs U (derived from natural language question workload), a graph edit distance threshold τ, and a similarity probability threshold $\alpha \in (0, 1]$, a *SimJ* query returns the graph pairs $\langle q, g \rangle$ ($q \in D$ and $g \in U$) with $SimP_\tau(q, g) \geq \alpha$.

Similarity Probability

$$SimP_\tau(q, g) = \sum_{pw(g) \in PW(g)} Pr\{pw(g) | ged(q, pw(g)) \leq \tau\}$$

Existing GED Lower Bounds

- Global Filters
  - vertex/edge counting [Zeng et al., 2009]:
    $ged(q, g) \geq ||V(g)| - |V(q)|| + ||E(g)| - |E(q)||$
  - label-based lower bounds [Zhao et al., 2012]:
    $ged(q, g) \geq \Gamma(M_V(q), M_V(g)) + \Gamma(M_E(q), M_E(g))$
- n-gram based Lower Bounds
  - path-based fliter [Zhao et al., 2012]
    $$dist_P(q, g) = max(|MG(q)| - \tau \cdot D_p, |MG(g)| - \tau \cdot D_p(g))$$
  - tree-based fliter [Wang et al., 2012]
    $$dist_T(q, g) = max(|V(q)| - \tau \cdot D_t, |V(g)| - \tau \cdot D_t(g))$$
  - star-based fliter [Zeng et al., 2009]
    $$dist_S(q, g) = \frac{\mu(q, g)}{max\{4, [max\{\ \delta(q), \delta(g)\}] + 1\}}$$
- Partition-based Lower Bound
  - Pars [Zhao et al., 2013]
  - Disjoint-Partition [Zheng et al., 2015]

Challenges

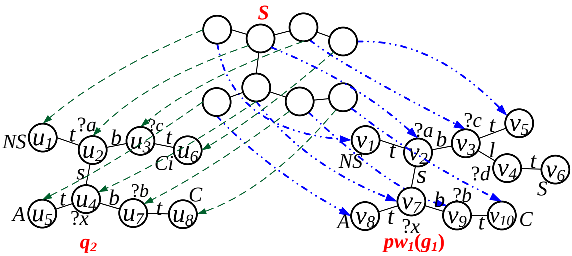
Challenges

Existing lower bounds are designed for deterministic graphs

Two Possible Variants

- ignoring all the vertex/edge labels
- enumerating all possible worlds

Common Structural Subgraph (CSS)



CSS-based Lower Bound

The relationship between GED and CSS[Brun et al., 2012]:

$$
\begin{aligned}
& ged(q, g^c) \\
= & \min_{\forall s}\{(|V(q)| + |E(q)|) - (|V(s)| + |E(s)|) && \text{(part 1)} \\
& +(|V(g^c)| + |E(g^c)|) - (|V(s)| + |E(s)|) && \text{(part 2)} \\
& +(|V_s| + |E_s|)\} && \text{(part 3)} \\
= & |V(q)| + |E(q)| + |V(g^c)| + |E(g^c)| - F(q, g^c) && (3)
\end{aligned}
$$

where V_s and E_s are the subsets of vertices and edges in CSS, whose labels need to be substituted, and we have

$$F(q, g^c) = \max_{\forall s}\{2 \cdot (|V(s)| + |E(s)|) - (|V_s| + |E_s|)\}. \quad (4)$$

$$ged(q, g) \geq |V| + |E| - \lambda_E(q, g) + \frac{dif(q, g)}{2} - \lambda_V(q, g), \quad (5)$$

CSS-based Similarity Probability Upper Bound

$$SimP_\tau(q,g) = \sum_{pw(g)\in PW(g)} Pr\{pw(g)|ged(q,pw(g)) \leq \tau\}$$

Let $C(q,g)$ be $|V| + |E| - \lambda_E(q,g) + \dfrac{dif(q,g)}{2}$(constant).

$$\begin{aligned} SimP_\tau(q,g) &\leq \sum_{pw(g)\in PW(g)} Pr\{pw(g)|\lambda_V(q,pw(g)) \geq C(q,g) - \tau\} \\ &= Pr\{\lambda_V(q,g) \geq C(q,g) - \tau\}. \end{aligned}$$

CSS-based Similarity Probability Upper Bound

m independent variables $y_1, y_2, \ldots,$ and y_m
denote $(y_1 + y_2 + \ldots + y_m)$ as a single random variable Y
by applying the Markov's inequality

$$SimP_\tau(q, g) \leq ub\_SimP_\tau(q, g) = \frac{E(Y)}{C(q, g) - \tau}$$

where $E(Y) = \sum_{i=1}^{m} E(y_i)$ and $E(y_i) = \Sigma_{l_j \in I(v_i)} Pr(l_j | l_j \cap \Sigma_V(q) \neq \emptyset)$

Cost-based Query Optimization

divide all possible worlds of an uncertain graph g into disjoint possible world groups

Algorithm 2 SimJ_OPT(q, g, τ, α)

Require: uncertain graph q, deterministic graph g, graph edit distance threshold τ, and similarity probability threshold α
Ensure: $ub\_SimP_\tau(q,g)$

```
SimP_τ(q, g) ⟵ 0
divide g into k possible world groups PWG_1, ..., PWG_k
for i = 1 to k do
    compute lb_ged_CSS(q, PWG_i) according to Theorem 3
    if lb_ged_CSS(q, PWG_i) ≤ τ then
        compute ub_SimP_τ(q, PWG_i) according to Theorem 4
        t ⟵ ub_SimP_τ(q, PWG_i)
        ub_SimP_τ(q, g) ⟵ ub_SimP_τ(q, g) + t
return ub_SimP_τ(q, g)
```

Conclusions

- Graph Database is a Possible Way for RDF Knowledge Base Management.

Conclusions

- Graph Database is a Possible Way for RDF Knowledge Base Management.
- Subgraph Matching is a Strong Tool.

Conclusions

- Graph Database is a Possible Way for RDF Knowledge Base Management.
- Subgraph Matching is a Strong Tool.
- Using RDF repository, how to Provide Knowledge Services for Applications and Common Users?

Thank you!

gStore

gAnswer

Reference I

Brun, L., Gaüzère, B., and Fourey, S. (2012).
Relationships between graph edit distance and maximal common structural subgraph.
In *SSPR/SPR*, pages 42–50.

Wang, G., Wang, B., Yang, X., and Yu, G. (2012).
Efficiently indexing large sparse graphs for similarity search.
TKDE, 24(3):440–451.

Zeng, Z., Tung, A. K. H., Wang, J., Feng, J., and Zhou, L. (2009).
Comparing stars: On approximating graph edit distance.
PVLDB, 2(1):25–36.

Zhao, X., Xiao, C., Lin, X., Liu, Q., and Zhang, W. (2013).
A partition-based approach to structure similarity search.
PVLDB, 7(3):169–180.

Zhao, X., Xiao, C., Lin, X., and Wang, W. (2012).
Efficient graph similarity joins with edit distance constraints.
In *ICDE*.

Reference II

Zheng, W., Zou, L., Lian, X., Wang, D., and Zhao, D. (2015).
efficient graph similairty search over large graph databases.
TKDE, 24(3):440–451.