Keyword query interpretation over structured data

Advanced Methods of Information Retrieval
Elena Demidova
SS 2018
Recap
Query a knowledge graph: SPARQL query language

Which maize dishes are popular in the United States?

SELECT ?dish ?name WHERE {
  ?dish dbpedia:Maize.
  ?dish dbpedia:United_States.
  ?name.
}
Query a knowledge graph: SPARQL query language

Maize dishes popular in the United States (an excerpt):

<table>
<thead>
<tr>
<th>dish</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/Corn_soup">http://dbpedia.org/resource/Corn_soup</a></td>
<td>&quot;Corn soup&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Corn_soup">http://dbpedia.org/resource/Corn_soup</a></td>
<td>&quot;コーンスープ&quot;@ja</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Corn_cookie">http://dbpedia.org/resource/Corn_cookie</a></td>
<td>&quot;Corn cookie&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Pashofa">http://dbpedia.org/resource/Pashofa</a></td>
<td>&quot;Pashofa&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Taco_soup">http://dbpedia.org/resource/Taco_soup</a></td>
<td>&quot;Taco soup&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Brunswick_stew">http://dbpedia.org/resource/Brunswick_stew</a></td>
<td>&quot;Brunswick stew&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Brunswick_stew">http://dbpedia.org/resource/Brunswick_stew</a></td>
<td>&quot;Brunswick Stew&quot;@de</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Brunswick_stew">http://dbpedia.org/resource/Brunswick_stew</a></td>
<td>&quot;Estofado Brunswick&quot;@es</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Cocoa_Puffs">http://dbpedia.org/resource/Cocoa_Puffs</a></td>
<td>&quot;Cocoa Puffs&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Pudding_corn">http://dbpedia.org/resource/Pudding_corn</a></td>
<td>&quot;Pudding corn&quot;@es</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Pudding_corn">http://dbpedia.org/resource/Pudding_corn</a></td>
<td>&quot;Pudding corn&quot;@en</td>
</tr>
</tbody>
</table>
Query a knowledge graph: issues

- knowledge of the schema / unknown graph patterns
  - E.g. 62,000 different predicates in current DBpedia
- knowledge of the query language (SPARQL)
- scale / complexity of the schema and data
- incomplete / missing schema information
- noisy data / errors
Search in SPARQL literals

SPARQL FILTER functions like regex can test RDF literals.

```
SELECT ?subject ?name
WHERE {
FILTER regex(?name, "^maize", "i") }
```
Query result (an excerpt)

<table>
<thead>
<tr>
<th>subject</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/C">http://dbpedia.org/resource/C</a> Category:Maize_diseases</td>
<td>&quot;Maize diseases&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Maize_diseases">http://dbpedia.org/resource/Maize_diseases</a></td>
<td>&quot;Maize diseases&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/C">http://dbpedia.org/resource/C</a> Category:Maize</td>
<td>&quot;Maize&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Maize">http://dbpedia.org/resource/Maize</a></td>
<td>&quot;Maize&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Maize_dishes">http://dbpedia.org/resource/Maize_dishes</a></td>
<td>&quot;Maize dishes&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/C">http://dbpedia.org/resource/C</a> Category:Maize_products</td>
<td>&quot;Maize products&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Maize_varieties">http://dbpedia.org/resource/Maize_varieties</a></td>
<td>&quot;Maize varieties&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/C">http://dbpedia.org/resource/C</a> Category:Maize_production</td>
<td>&quot;Maize production&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Maize_drinks">http://dbpedia.org/resource/Maize_drinks</a></td>
<td>&quot;Maize drinks&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Maize_Kansas">http://dbpedia.org/resource/Maize_Kansas</a></td>
<td>&quot;Maize, Kansas&quot;@en</td>
</tr>
</tbody>
</table>

Returns entities of diverse entity types
Challenges in search over structured data

- Large / missing / unknown schema
  - But precise graph patterns in SPARQL / SQL

- Too many interpretations for pure literal search
  - E.g. “^maize“ in DBpedia: plants, locations, schools, etc.

Search „Maize“ in DBpedia (an excerpt)

```
http://dbpedia.org/resource/Beet,_Maize_&_Corn
http://dbpedia.org/resource/La_Villéenne-Bellenoye-et-la-Maize
http://dbpedia.org/resource/Maizels,_Westerberg_&_Co
http://dbpedia.org/resource/Colligny-Maizery
http://dbpedia.org/resource/2014_Canberra_United_W-League_season_Melissa_Maizels_1
http://dbpedia.org/resource/2014_Melbourne_Victory_W-League_season_Melissa_Maizels_1
http://dbpedia.org/resource/Maize,_Kansas
http://dbpedia.org/resource/Maizet
http://dbpedia.org/resource/Kellee_Maize
http://dbpedia.org/resource/Men_of_Maize
http://dbpedia.org/resource/Fort_de_Maizeret
http://dbpedia.org/resource/Maize_waevil
http://dbpedia.org/resource/Michael_Maize
http://dbpedia.org/resource/Integration_(Kellee_Maize_album)
http://dbpedia.org/resource/International_Maize_and_Wheat_Improvement_Center
http://dbpedia.org/resource/Maize_South_High_School
http://dbpedia.org/resource/Presle_des_Maizeaux
http://dbpedia.org/resource/Maize_chlorotic_dwarf_virus
```

https://dbpedia.org/sparql
Search in knowledge graphs / structured data

- **Using structured query language**
  - Using full-text search of a structured query language
    - SPARQL / SQL
- **Indexing (e.g. RDF literals / string values) using an external IR engine**
  - Indexing textual content using a dedicated full-text indexing engine, e.g. Elastic search, Lucene
- **Handling search queries that address several nodes in a graph**
  - Specialized approaches / later in this lecture
Aims of the session “Keyword query interpretation over structured data”

- **Lecture:**
  - Analyse aspects of:
    - usability and expressiveness in queries and search over structured data
  - Understand the concepts and algorithms to:
    - transform a keyword query into a structured query over a relational database

- **Hands-on:**
  - Get practical experience with:
    - Query and search relational data
    - Algorithms to conduct keyword search on relational data
Querying structured data: expressiveness vs. usability

Usability

Easy to use

Keyword search
possibly imprecise results
BANKS, DBXPlorer,
Discover (’02)

Complicated

Less expressive

Structured queries
language, schema
(SQL, SPARQL, XQuery)
OBE (’75), NLQ (’99)

More expressive

Expressiveness

Goal:
Expressive AND
Easy to use

adapted from: [Tata et. al 2008]
Database queries: expressiveness vs. usability

- **Database queries:**
  - knowledge of database schema
  - knowledge of query language syntax

- **Keyword search:**
  - Easy-to-use but imprecise
  - Ambiguous: unclear information need

- **Keyword query interpretation:**
  - Automatically translate keyword query in a (most likely) structured query (-ies)
From keywords to structured queries: An example
From keywords to structured queries: An example

\[ K = \{ \text{Michelle, XML} \} \]
From keywords to structured queries: An example

\[ K = \{\text{Michelle}, \text{XML}\} \]
From keywords to structured queries: An example

\[ K = \{ \text{Michelle, XML} \} \]

1. Identify tuples / attributes containing keywords

\[ \sigma_{\text{michelle} \in \text{name (Author)}}: \text{michelle} \]
\[ \sigma_{\text{xml} \in \text{title (Paper)}}: \text{xml} \]
\[ \sigma_{\text{michelle} \in \text{title (Paper)}}: \text{michelle} \]

2. Identify join paths to connect all keywords in the query

\[ Q = \sigma_{\text{michelle} \in \text{name (Author)}} \Join \text{Write} \Join \sigma_{\text{xml} \in \text{title (Paper)}} \]

Other paths?
From keywords to structured queries: An example

\[ K = \{ \text{Michelle, XML} \} \]
\[ Q = \sigma_{\text{michelle} \in \text{name} \text{(Author)}} \bowtie \text{Write} \bowtie \sigma_{\text{xml} \in \text{title} \text{(Paper)}} \]

The translation \( K \rightarrow Q \) requires:

1. Knowledge of the **schema graph**
   (tables, attributes, join paths)
2. Knowledge of **keyword occurrences**
3. Efficient algorithms
Definitions and notations: The schema graph

**Schema graph**: a directed graph $G_s (V, E)$

- $V$ – the set of relation schemas $\{R_1, R_2, \ldots, R_n\}$. An instance of a relation schema is a set of tuples (i.e. a database table).

- $E$ - the set of edges $R_i \rightarrow R_j$ between two relation schemas. An edge is a primary key to foreign key relation.

**TID** – primary key attribute (i.e. tuple identifier).

**Text attribute** – an attribute allowing full-text search.
An example: The DBLP schema graph

\[ V = \{ \text{Author}, \text{Write}, \text{Paper}, \text{Cite} \} \]
\[ E = \{ \text{Author.TID} \rightarrow \text{Write.AID}, \text{Paper.TID} \rightarrow \text{Write.PID}, \]
\[ \text{Paper.TID} \rightarrow \text{Cite.PID1}, \text{Paper.TID} \rightarrow \text{Cite.PID2} \} \]

Primary keys: Author.TID, Write.TID, Paper.TID, Cite.TID

Text attributes: Author.Name, Paper.Title
An example: The DBLP schema graph

A simplified representation of the schema graph:

- Author
  - TID
  - Name
- Write
  - TID
  - AID
  - PID
- Paper
  - TID
  - Title
  - PID1
  - PID2
- Cite
  - TID
  - PID1
  - PID2

- Author -> Write
- Write -> Paper
- Paper -> Cite
- AID -> PID
- PID1 -> PID2
Definitions and notations: The database graph

The *database graph*: a directed graph $G_D (V_t, E_t)$ on the schema graph $G_s$.

$V_t$ – the set of tuples $\{t_1, t_2, \ldots , t_n\}$.

$E_t$ - the set of edges between tuples.

Two tuples $t_i$ and $t_j$ are *connected* if there exists a foreign key (fk) reference $t_i -> t_j$ or $t_j -> t_i$.

Two tuples $t_i, t_j$ are *reachable* if there exists a sequence of connections between them, e.g. $t_i -> t_1, \ldots, t_n -> t_j$.

The *distance* between two tuples $\text{dis}(t_i, t_j)$ is the *minimal* number of connections between $t_i, t_j$ (ignoring edge directions).
An example: The DPLP database graph

The distance between two tuples \( \text{dis}(t_i, t_j) \) is the minimal number of connections between \( t_i, t_j \).

\[ \text{dis}(a1, p4)? \]
Keyword query

A \( l \)-keyword query \( K = \{k_1, k_2, \ldots, k_l\} \) – a set of keywords of size \( l \).

\( K \) semantics (typically): search for interconnected tuples that jointly contain \( \{k_1, k_2, \ldots, k_l\} \).

How can we find the tuples containing \( \{k_1, k_2, \ldots, k_l\} \) in a relational database?
Full-text search on specific database attributes

Full-text search on specific attributes is supported by major databases, e.g. using \textit{contains predicate}:

\[\text{contains}(R.A, k_i)\] – the predicate selecting all tuples from a relation \(R\) that contain keyword \(k_i\) in the text attribute \(R.A\).

\[\text{SELECT * FROM Author WHERE contains(Author.Name, „Michelle“);}\]

String comparison operators (e.g. \textit{like}):

\[\text{SELECT * FROM Author WHERE Author.Name LIKE ´%michelle%´;}\]

\textbf{Differences?}
Indexing DB content using external inverted index

Inverted index using Lucene, Solr, Elasticsearch...

Granularity:

**Tuple level:**

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michelle</td>
<td>Author.a$_3$ Paper.p$_1$ ...</td>
</tr>
<tr>
<td>XML</td>
<td>Paper.p$_2$ Paper.p$_3$ ...</td>
</tr>
</tbody>
</table>

**Attribute level:**

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michelle</td>
<td>Author.Name Paper.Title ...</td>
</tr>
<tr>
<td>XML</td>
<td>Paper.Title ...</td>
</tr>
</tbody>
</table>

Differences?
Built-in full-text search vs. external indexing

- Built-in full-text search
  - Database dependent
  - **Contains** predicate can use indexes but is neither flexible, nor not generally available
  - **String comparison operators** can require sequential scan (e.g. **like** operator if the string prefix is undefined)
  - Each textual attribute needs to be queried separately

- In a global full-text index
  - The list of attributes is immediately available
  - Index construction cost
  - Storage cost (depends on the index granularity)
Keyword query answers: MTJNTs

An answer to a \( l \)-keyword query is a Minimal Total Joining Network of Tuples (MTJNT).

\textbf{JNT} (Joining Network of Tuples) – a connected tree of tuples. Every two adjacent tuples \( t_i, t_j \) in JNT an be joined based on the fk-reference in the schema i.e. either \( R_i \rightarrow R_j \) or \( R_j \rightarrow R_i \) (ignoring edge direction).

\textbf{TJNT} (Total JNT) w.r.t. a \( l \)-keyword query \( K \) if it contains all keywords of \( K \).

\textbf{MTJNT} (Minimal TJNT) if no tuple can be removed such that JNT remains total.

\( T_{\text{max}} \) – a size control parameter to define the maximal number of tuples in a valid MTJNT.
Keyword query answers: MTJNT examples

\[ K = \{ \text{Michele, XML} \} \]

\[ T_{max} = 5 \]

MTJNTs = {?}

Work in groups: 10 minutes
Keyword query answers: MTJNT examples

\[ K = \{\text{Michele, XML}\} \]
\[ T_{\text{max}} = 5 \]

\[ \text{MTJNTs} = \{?\} \]

contains \((a_3, "\text{Michelle"})\)
contains \((p_1, "\text{Michelle"})\)
contains \((p_2, "\text{XML"})\)
contains \((p_3, "\text{XML"})\)
Keyword query answers: MTJNT examples

\[ K = \{\text{Michelle, XML}\} \]
\[ T_{\text{max}} = 5 \]
contains \((a_3, "\text{Michelle}"\))
contains \((p_1, "\text{Michelle}"\))
contains \((p_2, "\text{XML}"\))
contains \((p_3, "\text{XML}"\))

MTJNTs:
MTJNT issues

**Size and scalability:**
The data graph is potentially very large, i.e. search is very costly
The search space increases exponentially by adding new data entries

**Results semantics and presentation**
The results are heterogeneous in terms of structure, i.e.
difficult to present and understand
An overview of possible structures is needed

**Idea:** Generate structured queries first
Schema graph is much smaller than data graph
Structured queries naturally aggregate MTJNTs
Structured queries: Candidate Network (CN)

A **keyword relation**: a subset \( R_i \{K'\} \) of relation \( R_i \) that contains a subset \( K' \) of keywords from \( K \) (**and no other keywords from \( K \)**). The subset can be empty \( R_i \{ \} \).

A **Candidate Network (CN)** is a connected tree of **keyword relations**. Every two adjacent keyword relations \( R_i, R_j \) in CN are joined based on the fk-reference in the schema \( G_s \).

**CN is total** w.r.t. a \( l \)-keyword query \( K \) if its keyword relations jointly contain all keywords of \( K \).

**CN is minimal** if no keyword relation can be removed such that CN remains total.

\( T_{\text{max}} \) – a size control parameter to define the maximum number of keyword relations in CN.

A CN can produce a set of possibly empty MTJNTs. One MTJNTs can be generated by exactly one CN.
CN examples

\[ K = \{\text{Michelle, XML}\}, \quad T_{\text{max}} = 5, \quad P\{\text{Michelle}\}, \quad P\{\text{XML}\}, \quad A\{\text{Michelle}\} \]

CNs:
CN examples

\[ K = \{\text{Michelle, XML}\}, \ T_{\text{max}} = 5, \ P\{\text{Michelle}\}, \ P\{\text{XML}\}, \ A\{\text{Michelle}\} \]

CNs:

MTJNTs:

Which MTJNTs are generated by which CNs?
CNs in SQL: Work in groups

\[ K = \{\text{Michelle, XML}\}, \quad T_{\text{max}} = 5, \quad P\{\text{Michelle}\}, \quad P\{\text{XML}\}, \quad A\{\text{Michelle}\} \]

**CNs:**

\[
\begin{array}{c}
C_1 \\
P\{\text{Michelle}\} P\{\text{XML}\} P\{\text{Michelle}\}
\end{array}
\begin{array}{c}
C_2 \\
P\{\text{XML}\} P\{\text{Michelle}\} A\{\text{Michelle}\} P\{\text{XML}\}
\end{array}
\]

**SQL:**

Work in groups:

Write SQL query expressions to generate \(C_1, \ldots, C_5\)

Time: 10 minutes

1 SQL expert per group?

**Tipp:** use „contains“ predicate
CNs in SQL: Work in groups

\[ K = \{\text{Michelle, XML}\}, \ T_{\text{max}} = 5, \ P (\text{“Michelle”}), \ P (\text{“XML”}), \ A (\text{“Michelle”}) \]

**CNs:**

\[
\begin{align*}
C_1 & \quad P\{\text{Michelle}\}P\{\text{XML}\} \\
C_2 & \quad P\{\text{XML}\} \\
A & \quad W\{\text{Michelle}\}P\{\text{XML}\} \\
P & \quad W\{\text{XML}\}C\{\} \\
\end{align*}
\]

**SQL: (C1)**

```
SELECT * FROM Paper AS P1, Cite AS C, Paper AS P2
WHERE contains (P1.Title, "Michelle")
AND NOT contains (P1.Title, "XML")
AND P1.TID = C.PID2
AND C.PID1 = P2.TID
AND contains (P2.Title, "XML")
AND NOT contains (P2.Title, "Michelle")
```
Given are:
1. Keyword query $K = \{k_1, k_2, \ldots, k_l\}$
2. Schema graph $G_s$
3. The nodes of $G_s$ containing each keyword $k_i$ in $K$

The Problem: Find the path(s) connecting all $\{k_1, k_2, \ldots, k_l\}$ in $G_s$ (i.e. the structured query(-ies))

Example: $K = \{\text{Michelle}, \text{XML}\}$

Complexity?
CN generation algorithms

**Complexity**: similar to the Steiner tree problem - find the shortest interconnect for a given set of objects: NP-complete.

**Approximation algorithms**: 
Iteratively explore the schema graph to construct the paths

**Algorithm ideas?**

**Data structures?**
BFS / DFS

Background knowledge:

Breadth-First-Search BFS
Depth-First-Search DFS
Search algorithms and data structures: BFS

Search on the schema graph $G_s$ (with keyword relations)
Breadth-First-Search (BFS): queue

Step i:

Step i+1:
Search algorithms and data structures: BFS

Search on the schema graph $G_s$ (with keyword relations)

Breadth-First-Search (BFS): queue

**Step j:**

**Step j+1:**
Search algorithms and data structures: DFS

Search on the schema graph $G_s$ (with keyword relations)

Depth First Search (DFS) – for top-k generation:

Stack

Differences in BFS / DFS results?
Goal:
Generate total, minimal and non-duplicating CNs

Pruning rules:
Duplicate elimination (requires graph isomorphism checking)
Pruning total but not minimal CNs
Avoiding cycles (estimated based on pk-fk references)
CN generation algorithm (BFS-based): Discover

**Algorithm 1** Discover-CNGen \((Q, T_{\text{max}}, G_S)\)

**Notation:** here \(Q\) is a keyword query!

**Input:** an \(I\)-keyword query \(Q = \{k_1, k_2, \ldots, k_I\}\), the size control parameter \(T_{\text{max}}\), the schema graph \(G_S\).

**Output:** the set of CNs \(C = \{C_1, C_2, \ldots\}\).

1: \(Q \leftarrow \emptyset; C \leftarrow \emptyset\)
2: for all \(R_i \in V(G_S), K' \subseteq Q\) do
3: \(Q\).enqueue\((R_i[K'])\)
4: while \(Q \neq \emptyset\) do
5: \(T \leftarrow Q\).dequeue()
6: if \(T\) is minimal and total and \(T\) does not satisfy Rule-1 then
7: \(C \leftarrow C \cup \{T\}\); continue
8: if the size of \(T < T_{\text{max}}\) then
9: for all \(R_i \in T\) do
10: \(\text{for all } (R_i, R_j) \in E(G_S) \text{ or } (R_j, R_i) \in E(G_S)\) do
11: \(T' \leftarrow T \cup (R_i, R_j)\)
12: if \(T'\) does not satisfy Rule-2 or Rule-3 then
13: \(Q\).enqueue\((T')\)
14: return \(C\);

**Rule 1:** duplicate elim.

**Rule 2:** minimality

**Rule 3:** avoid cycles
CN generation: Work in groups

Keyword relations:
A{Michelle}, P{XML}, P{Michelle}

Work in Groups (10 minutes):
Write down the essential steps of of the algorithm until the first valid (i.e. total and minimal) CN is generated

Algorithm 1 Discover-CNGen \((Q, T_{\text{max}}, G_S)\)

\[
\begin{align*}
1: \quad Q & \leftarrow \emptyset; C \leftarrow \emptyset \\
2: \quad & \text{for all } R_i \in V(G_S), K' \subseteq Q \text{ do} \\
3: \quad & Q.\text{enqueue}(R_i[K']) \\
4: \quad & \text{while } Q \neq \emptyset \text{ do} \\
5: \quad & T \leftarrow Q.\text{dequeue}() \\
6: \quad & \text{if } T \text{ is minimal and total and } T \text{ does not satisfy Rule-1 then} \\
7: \quad & C \leftarrow C \cup \{T\}; \text{ continue} \\
8: \quad & \text{if the size of } T < T_{\text{max}} \text{ then} \\
9: \quad & \text{for all } R_i \in T \text{ do} \\
10: \quad & \text{for all } (R_i, R_j) \in E(G_S) \text{ or } (R_j, R_i) \in E(G_S) \text{ do} \\
11: \quad & T' \leftarrow T \cup (R_i, R_j) \\
12: \quad & \text{if } T' \text{ does not satisfy Rule-2 or Rule-3 then} \\
13: \quad & Q.\text{enqueue}(T') \\
14: \quad & \text{return } C;
\end{align*}
\]
CN generation: An example

Keyword relations:
A\{Michelle\}, P\{XML\}, P\{Michelle\}

...
CN generation: Complexity and optimizations

Complexity factors:
• Size of the schema graph $G_s$ – the number of nodes and edges
• Maximal number of joins ($T_{max}$)
• Size of the keyword query ($l$)

The number of CNs grows exponentially with these factors.

Algorithm optimizations:
• Avoid generation of duplicate CNs by defining the expansion order
• Generate only the top-k CNs
• …
CN and MTJNT ranking factors

Ranking can be performed at **CN and MTJNT levels**

**Typical ranking factors include:**

- Size of the CN / tuple tree – preference to the short paths
- IR-Style factors
  - Frequency-based keyword weights
  - Keyword selectivity (IDF)
  - Length normalizations
- Global attribute weight in a database (PageRank / ObjectRank)

Typically, the factors are combined
Ranking query interpretations: An example

Rank the following CNs using the size factor:
Summary

• In this session we:
  ▪ Analysed the aspects of:
    ▪ usability and expressiveness in queries and search over structured data
  ▪ Considered concepts and algorithms to:
    ▪ transform a keyword query into a structured query over a relational database
  ▪ Collected practical experience with:
    ▪ Algorithms to conduct keyword search on relational data
Thank you!

Questions, Comments?

Dr. Elena Demidova

L3S Research Center
Leibniz University of Hannover

email: demidova@L3S.de
www: https://demidova.wordpress.com
References and further reading

References:


Further reading:


Materials used in the slides:

- Jeffrey Xu Yu, Lu Qin, Lijun Chang. Keyword Search in Databases.