Interactive keyword-based access to large-scale structured datasets

2nd Keystone Summer School
20 July 2016

Dr. Elena Demidova
University of Southampton
Overview

• Keyword-based access to structured data
  • Usability and expressiveness

• Preparation of data for keyword-based access
  • Indexing structured data

• Interactive query construction
  • Building structured queries with user input
Keyword-based access to structured data

Usability and expressiveness
Keyword-based access to structured data

Using structure we could refine the results: London... What do you mean?
A book title? An author?
Access to structured data: Search vs. query

Example: DBLP as a relational database containing paper-author relations

Keyword query: \( K = \{\text{Michelle, XML}\} \)

Structured query: \( Q = \sigma_{\text{michelle} \in \text{name}(\text{Author})} \bowtie \text{Write} \bowtie \sigma_{\text{xml} \in \text{title}(\text{Paper})} \)

(Example from [Yu et. al 2009])
Database queries: Expressiveness vs. usability

Usability

Easy to use

Complicated

Expressiveness

Less expressive

More expressive

Goal:
Expressive AND Easy to use

- **Keyword search**
  possibly imprecise results
  BANKS, DBXPloerer, Discover (’02)

- **Structured queries**
  language, schema
  (SQL, SPARQL, XQuery)
  QBE (’75), NLQ (’99)

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)
Preparation of data for keyword-based access

Indexing structured data at the example of relational databases
Keyword query semantics

A \( \land \)-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 database?
Full-text search on a specific database attribute

Full-text search on specific attribute is supported by major databases, e.g. using **contains** predicate:

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

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

String comparison operators (e.g. **like**):

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

Problem: need to search in each attribute separately
DB indexing for keyword search

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>-&gt; Author.a₃</td>
</tr>
<tr>
<td></td>
<td>Paper.p₁</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>XML</td>
<td>-&gt; Paper.p₂</td>
</tr>
<tr>
<td></td>
<td>Paper.p₃</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Attribute level:

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

Differences?
SQL full-text search vs. indexing

Built-in full-text search capabilities are database dependent

Contains predicate can use indexes but is neither flexible, nor generally available

String comparison operators can require sequential scan (e.g. like operator if the prefix is undefined)

Each textual attribute needs to be queried separately

In the global full-text index, the list of attributes is immediately available

Index construction cost

Storage cost (depends on the index granularity)
Query construction as a way to improve query expressiveness

Building structured queries from keywords
From keywords to structured queries: An example

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

1. Identify tuples \(\sigma\) / attributes containing keywords
   - \(\sigma_{\text{michelle} \in \text{name}(\text{Author})}: \text{michelle}\)
   - \(\sigma_{\text{xml} \in \text{title}(\text{Paper})}: \text{xml}\)
   - \(\sigma_{\text{michelle} \in \text{title}(\text{Paper})}: \text{michelle}\)

2. Identify join paths to connect all keywords in the query
   \[ Q = \text{michelle} \in \text{name}(\text{Author}) \bowtie \text{Write} \bowtie \sigma_{\text{xml} \in \text{title}(\text{Paper})} \]

   Other paths?
From keywords to structured queries: An example

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

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

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

A simplified representation of the schema graph:
Definitions and notations: The database graph

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

$V_t$ – 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 $t_i \rightarrow t_j$ or $t_j \rightarrow t_i$.

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

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

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

$\text{dis}(a1, p4) \ ?$
Interconnecting keywords: MTJNT

An answer to a \(\ell\)-keyword query is a Minimal Total Joining Network of Tuples (\(\text{MTJNT}\)).

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

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

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

\(T_{\text{max}}\) – a size control parameter to define the maximum number of tuples in MTJNT.
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
Aggregation / summarization is needed

Generation of structured queries:
Schema graph is much smaller
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

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

CNs:
CN examples

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

CNs:

\[ C_1, C_2, C_3, C_4 \]

MTJNTs:

\[ T_1, T_2, T_3, T_4, T_5, T_6, T_7 \]

Which MTJNTs are generated by which CNs?
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: $K = \{\text{Michelle, XML}\}$
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

BFS/DFS

Data structures?
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

\[ V_1 \to \text{pop} \to V_2 \to \text{push} \to V_1 \]

\[ \text{push} \to V_1 \to \text{pop} \to V_2 \to \text{push} \to V_4 \]
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 \(l\)-keyword query \(Q = \{k_1, k_2, \ldots, k_l\}\), 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: \(Q.\text{enqueue}(R_i[K'])\)
4: while \(Q \neq \emptyset\) do
5: \(T \leftarrow Q.\text{dequeue}()\)
6: if \(T\) is minimal and total and \(T\) does not satisfy Rule 1 then Rule 1: duplicate elim.
7: \(C \leftarrow C \cup \{T\}; \text{continue}\)
8: if the size of \(T < T_{\text{max}}\) then Rule 2: minimality
9: for all \(R_i \in T\) do Rule 3: avoid cycles
10: for all \((R_i, R_j) \in E(G_S)\) 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: \(Q.\text{enqueue}(T')\)
14: return \(C\);
CN generation: An example

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

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

\textbf{Input:} an \(l\)-keyword query \(Q = \{k_1, k_2, \ldots, k_l\}\), the size control parameter \(T_{\text{max}}\), the schema graph \(G_s\).

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

\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*}

enqueue: P{Michelle}, P{XML}, A{Michelle}
dequeue: T_1 \leftarrow A{Michelle}
expand: T_2 \leftarrow A{Michelle} \bowtie W{}
enqueue: T_2

... 

dequeue: T_2 \leftarrow A{Michelle} \bowtie W{}
expand: T_3 \leftarrow A{Michelle} \bowtie W{} \bowtie P{XML}
enqueue: T_3

... 

dequeue: T_3, \text{ check if } T_3 \text{ is minimal and total, add } T_3 \text{ to the result}
CN generation: Complexity and optimizations

Complexity factors:
• Size of the schema graph $G_s$ – the number of nodes and edges
• Maximum 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:
Interactive query construction

Building structured queries with user input
Query construction for large scale databases

- Freebase:
  - 22 millions entities, more than 350 millions facts
  - more than 7,500 relational tables
  - about 100 domains
  - Wikipedia, MusicBrainz, ...
  - part of the LOD cloud

- Goal:
  - Enable efficient and scalable query construction solutions for large scale data
A film adaptation starring Tom Hanks was attempted [...] after the actor's performances in The Terminal (2004).

An article in Entertainment Weekly did a comparison to the Tom Hanks film The Terminal.

Tom Hanks' character Viktor Navorski is stuck at New York's JFK airport in the United terminal in The Terminal.

Feng Zhenghu has been likened to the Tom Hanks character in The Terminal.
Structured MQL query for „Tom Hanks Terminal“

```json
{
  "!pd:/film/actor/film": [{
    "name": "Tom Hanks",
    "type": "/film/actor"
  }],
  "film": [{
    "name": "The Terminal",
    "type": "/film/film"
  }],
  "character": {
    "name": null,
    "type": "/film/performance"
  }
}
```

http://www.freebase.com/query

Requires prior knowledge of:
- ✓ Schema: above 1000 entity types (relational tables)
- ✓ Specialized query language: MQL
Query interpretation techniques

• Automatic keyword query interpretation:
  • Automatically translate keyword query in the (most likely) structured query (-ies)
  • No one size fits all – no perfect ranking for every query and every user
  • If ranking fails, navigation cost can be unacceptable
    • too many interpretations / search results

• Interactive query refinement
  • Goal: Enable users to incrementally refine a keyword query into the intended interpretation on the target database in a minimal number of interactions
Query interpretation

A query interpretation consists of:

• A set of **keyword interpretations** $I$ that map a keyword to a value of an attribute (also interpretations as an attribute or table name are possible)

$$\sigma_{\text{hanks} \in \text{name}(\text{Actor})}: \text{hanks}$$

$$\sigma_{\text{cruise} \in \text{name}(\text{Actor})}: \text{cruise}$$

$$\sigma_{2001 \in \text{year}(\text{Movie})}: 2001$$

• A **query template** $T$

$$T = \sigma_{? \in \text{name}(\text{Actor})} \Join \text{Acts} \Join \sigma_{? \in \text{year}(\text{Movie})} \Join \text{Acts} \Join \sigma_{? \in \text{name}(\text{Actor})}$$
Query interpretation

\[ K = "\text{hanks cruise 2001}" \]

\[ \sigma_{\text{hanks} \in \text{name(Actor): hanks}} \]
\[ \sigma_{\text{2001} \in \text{year(Movie): 2001}} \]
\[ \sigma_{\text{cruise} \in \text{name(Actor): cruise}} \]

\[ T = \sigma?_{\text{name(Actor)}} \bowtie \text{Acts} \bowtie \sigma?_{\text{year(Movie)}} \bowtie \text{Acts} \bowtie \sigma?_{\text{name(Actor)}} \]

\[ Q = \sigma_{\text{hanks} \in \text{name(Actor)}} \bowtie \text{Acts} \bowtie \sigma_{\text{2001} \in \text{year(Movie)}} \bowtie \text{Acts} \bowtie \sigma_{\text{cruise} \in \text{name(Actor)}} \]

partial interpretation of \( K \), sub-query of \( Q \)

complete interpretation of \( K \) (structured query)

\( \triangleright \) interpretation space of \( K \)
Query hierarchy

$\mathcal{K} = \text{“Tom Hanks 2001”}$
Query construction options (QCO)

Idea: use partial interpretations (sub-queries) as user interaction items (QCO)

Problem: large number of queries – and sub-queries (QCOs)

\[
\sigma_{\text{name}(\text{Actor}): \text{hanks}} 
\bowtie \sigma_{\text{year}(\text{Movie}): 2001} 
\sigma_{\text{name}(\text{Actor}): \text{cruise}} 
\bowtie \sigma_{\text{year}(\text{Movie}): 2001} 
\]

\[
Q' = \sigma_{\text{name}(\text{Actor})} \bowtie \text{Acts} \bowtie \sigma_{\text{year}(\text{Movie}): 2001} 
\]

How to select a QCO to present to the user?
Query construction plan (QCP) as a binary tree

Idea: use sub-query relations to organize the options in a (binary) tree structure

The root node is the entire interpretation space

Remove queries conflicting with QCO₁

Yes → QCO₁: \( \sigma_{\text{hanks} \in \text{name} \ (\text{Actor})} \) hanks

Yes → QCO₂: \( \sigma_{2001 \in \text{year} \ (\text{Movie})} \) 2001

No → QCO...

Remove queries that subsume QCO₁

A leaf node is a single complete query interpretation

Problem: How to find an optimal QCP?
Defining a cost function for QCP

Idea: define a cost function
Take query probability into account
Construction of the most likely queries should not incur much cost

$$\text{Cost}(QCP) = \sum_{\text{leaf} \in QCP} \text{depth(leaf)} \times P(\text{leaf})$$

Given a keyword query $K$, how to compute the probability of leaf nodes (i.e. complete query interpretations of $K$)?

$K$ (a keyword query) = \{hanks, 2001, cruise\}
$Q$ (a leaf node of QCP) =

$$\sigma_{\text{hanks} \in \text{name \ (Actor)}} \Join \text{Acts} \Join \sigma_{2001 \in \text{year \ (Movie)}} \Join \text{Acts} \Join \sigma_{\text{cruise} \in \text{name \ (Actor)}}$$

$P(\text{leaf}) = P(Q|K)$: the conditional probability that, given $K$, $Q$ is the user intended complete interpretation of $K$. 
Query interpretation: assumptions

Assumption 1 (Keyword Independence): Assume that the interpretation of each keyword in a keyword query is independent from the other keywords.

Assumption 2 (Keyword Interpretation Independence): Assume that the probability of a keyword interpretation is independent from the part of the query interpretation the keyword is not interpreted to.
Probability of a query interpretation

\[ P(Q \mid K) = P(I, T \mid K) \]

- \( I \) is the set of keyword interpretations \( \{A_i : k_i\} \) in \( Q \)
  - \( \sigma_{2001} \in \text{year(Movie)}: 2001 \)
  - \( \sigma_{\text{cruise}} \in \text{name(Actor)}: \text{cruise} \)
  - \( \sigma_{\text{hanks}} \in \text{name(Actor)}: \text{hanks} \)

- \( T \) is the template of \( Q \)
  - \( T = \sigma_{\text{?}} \in \text{name(Actor)} \bowtie \text{Acts} \bowtie \sigma_{\text{?}} \in \text{year(Movie)} \bowtie \text{Acts} \bowtie \sigma_{\text{?}} \in \text{name(Actor)} \)

\[ P(Q \mid K) \propto \left( \prod_{k_i \in K} P(A_i : k_i \mid A_i) \right) \times P(T) \]

Estimates for \( P(T) \) and \( P(A_i : k_i \mid A_i) \)?
Probability of a keyword interpretation

- We model the formation of a query interpretation as a random process.
- For an attribute $A_i$, this process randomly picks one of its instances $a_j$ and randomly picks a keyword $k_i$ from that instance to form the expression $\sigma_{k_i} \in A_i$.
- Then, the probability of $P(\sigma_{k_i} \in A_i | \sigma ? \in A_i)$ is the probability that $\sigma_{k_i} \in A_i$ is formed through this random process.

Example:

$T = \sigma ? \in \text{name (Actor)} \bowtie \text{Acts} \bowtie \sigma ? \in \text{year (Movie)} \bowtie \text{Acts} \bowtie \sigma ? \in \text{name (Actor)}$

$Q = \sigma_{\text{hanks}} \in \text{name (Actor)} \bowtie \text{Acts} \bowtie \sigma_{2001} \in \text{year (Movie)} \bowtie \text{Acts} \bowtie \sigma_{\text{cruise}} \in \text{name (Actor)}$

$P(\sigma_{\text{hanks}} \in \text{name (Actor)} | \sigma ? \in \text{name (Actor)})$
Probability of a keyword interpretation

\[ P(\sigma_{k_i} \in A_i | \sigma \in \mathbf{A}_i) \] can be estimated using Attribute Term Frequency (ATF):

\[ \text{ATF}(k_i, A_i) = \frac{\text{TF}(k_i, A_i) + \alpha}{N_{A_i} + \alpha * B} \]

\text{ATF}(k_i, A_i) - \text{the normalized keyword frequency of } k_i \text{ in } A_i

\text{N}_{A_i} - \text{the number of keywords in } A_i

\alpha - \text{a smoothing parameter (typically } \alpha = 1: \text{ Laplace smoothing)}

B - \text{the vocabulary size}
Probability of a query template

\[ P(T) = \frac{\#occurrences(T) + \alpha}{N + \alpha^*B} \]

\#occurrences(T) - number of queries in the log using \( T \) as a template

\( N \) - total number of queries in the log

\( \alpha \) - smoothing parameter, typically set to 1

\( B \) – a constant

When the query log is absent or is not sufficient, we assume that all query templates are equally probable.
Challenges in query interpretation

- Inefficient QCOs
  - Too many keyword interpretations
  - A keyword interpretation subsumes a small proportion of the I-space, more general QCOs are needed

- Very large interpretation space
  - The number of subgraphs of the schema graph grows very sharply with the size of the schema graph. The occurrences of keywords are more numerous in a larger database. Too many query interpretations.
  - Existing query interpretation approaches rely on a completely materialized interpretation space. This is no longer feasible.
  - Need to enable incremental materialization of the interpretation space
Query construction algorithm

- Query hierarchy can become very large
- Use greedy algorithms
- Expand query hierarchy incrementally
- Use a threshold to restrict the size of the top level
- Select the QCO to be presented to the user based on Information Gain (IG)
- IG can be computed using probability of query interpretation
Query-based QCOs

- **Keyword as schema terms or attribute values**
  - `actor.name: hanks` (Hanks is in the actor’s name)

<table>
<thead>
<tr>
<th>hanks</th>
<th>actor.name</th>
<th>director.name</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminal</td>
<td>film.name</td>
<td>company.name</td>
<td>location.name</td>
</tr>
</tbody>
</table>

- **Joins using pk-fk relationships in the schema graph**
  - `actor.name: hanks – acts – film.name: terminal`
Ontology-based QCOs

- Freebase domain hierarchy
  - Arts & Entertainment, Society
- External ontologies
  - E.g. YAGO+F mapping between YAGO and Freebase
  - Person, Location, Object

![Image of person][hanks]
![Image of deceased person][hanks]
![Image of organization][hanks]
![Image of TV episode][hanks]
FreeQ query hierarchy example

Ontology-based QCOs:
Arts & Entertainment: Tom Hanks
Actor: Tom Hanks

Query-based QCOs:
Actor Name: Tom Hanks

Actor: Tom Hanks
Celebrianity: Tom Hanks
Award Nomination
Award Nominee: Tom Hanks
Nominated For: Terminal

The arrows represent sub-query relationship
A measure of QCO efficiency

Entropy of the query interpretation space:

\[ H(\zeta) = - \sum_{I \in \zeta} P(I) \times \log_2 P(I) \]

Expected information gain of a QCO as entropy reduction:

\[ IG(O) = H(\zeta) - H(\zeta|O) = H(O) \]

Entropy of O computed using P(O):

\[ H(O) = -P(O)\log_2 P(O) - P(\neg O)\log_2 P(\neg O) \]
Probability estimation for QCOs

Probability of a QCO using probabilities of the subsumed query interpretations:

\[ P(O) = \sum_{I \in \zeta(O)} P(I) \]

Estimation of QCO probability using materialized part of the query hierarchy:

\[ P(o) = \frac{\sum_{\zeta(s) \subset \zeta(o)} P(s)}{\sum_{\zeta(s) \subset \zeta(o)} P(s) + \sum_{\zeta(s) \cap \zeta(o) = \emptyset} P(s)} \]
Efficient hierarchy traversal

✓ Query initialization:
  ✓ Path indexing: for each table, index all paths leading to keywords within radius $r/2$ (bi-directional):
  ✓ Is independent of keyword query length

✓ User interaction:
  ✓ Use path index to materialize QCOs and query interpretations incrementally by BF-\(k\) and DF-\(k\)
  ✓ Start expansion with the most probable QCOs
  ✓ Thresholds, time limits

\[
T \ast \text{avg}(E_t)^{r/2}
\]
The user enters a keyword query in the search box.

FreeQ interprets the keyword query using database queries, and returns a list of possible interpretations to the user.

FreeQ suggests a set of query construction options for the user to refine her query.

The user identifies the correct database query. After double clicking this query, the user is presented with the query results.
Discussion

- Interactive query construction can enable efficient and scalable query solutions for large scale data
- It can involve ontologies to summarize and enrich database schema using abstract concepts (e.g. using YAGO ontology)
- Query interpretation space on large scale data can and should be materialized incrementally
References


Further reading


Questions, Comments?

Dr. Elena Demidova
Web and Internet Science Group
University of Southampton

e.demidova@soton.ac.uk