Keyword query interpretation over structured data

Advanced Methods of IR
Elena Demidova
SS 2015

Materials used in the slides:
Database queries: expressiveness vs. usability

Usability

Expressiveness

Less expressive

More expressive

Complicated

Easy to use

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

Goal:
Expressive AND Easy to use

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

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

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

1. Identify tuples / 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

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

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

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

1. Knowledge of the \textbf{schema graph} (tables, attributes, join paths)
2. Knowledge of \textbf{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

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

Primary keys: Author.\text{TID}, Write.\text{TID}, Paper.\text{TID}, Cite.\text{TID}

Text attributes: Author.\text{Name}, Paper.\text{Name}
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 $G_s$.

$V_t$ – 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 $t_i \rightarrow t_j$ 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. $t_i \rightarrow t_1 \text{, } \ldots \text{, } t_n \rightarrow t_j$.

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 \( t_i, t_j \).

\[ \text{dis}(a1, p4)? \]
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 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**:

`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`.

```
SELECT * FROM Author WHERE contains(Author.Name, „Michelle“);
```

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

```
SELECT * FROM Author WHERE Author.Name LIKE '％michelle％';
```

**Differences?**
DB indexing

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.a₃  Paper.p₁  ...</td>
</tr>
<tr>
<td>XML</td>
<td>Paper.p₂  Paper.p₃  ...</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>
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 not 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)
Interconnecting keywords: MTJNT

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

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 fk-reference in the schema i.e. either $R_i \rightarrow R_j$ or $R_j \rightarrow R_i$ (ignoring direction).

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

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

$T_{max}$ – a size control parameter to define the maximum number of tuples in MTJNT.
Keyword query answers: MTJNT examples

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

MTJNTs = \{?\}

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

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

\[ T_{max} = 5 \]

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

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 (\(a_3\), "Michelle")
contains (\(p_1\), "Michelle")
contains (\(p_2\), "XML")
contains (\(p_3\), "XML")

**MTJNTs:**

![Graphical representation of MTJNT examples]
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

**Idea:** Generate structured queries first
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

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

CNs:
CN examples

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

**CNs:**

**MTJNTs:**

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

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

**CNs:**

**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:**

**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“)
CN generation algorithms

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

Complexity?
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?
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**
CN generation: Pruning rules

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 \((Q, \text{Tmax}, G_S)\)

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

**Input:** an \(l\)-keyword query \(Q = \{k_1, k_2, \ldots, k_l\}\), the size control parameter \(\text{Tmax}\), 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
7: \(C \leftarrow C \cup\{T\}\); continue
8: if the size of \(T < \text{Tmax}\) then
9: for all \(R_i \in T\) do
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: Work in groups

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

Algorithm 1 Discover-CNGen (Q, Tmax, GS)

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

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

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

Work in Groups (10 minutes):

Write down the essential steps of the algorithm until the first valid (i.e. total and minimal) CN is generated
CN generation: An example

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

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

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
7. \(C \leftarrow C \cup \{T\}; \text{continue}\)
8. if the size of \(T < T_{\text{max}}\) then
9. for all \(R_i \in T\) do
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 \(C\)

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

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

... dequeue: \(T_3\), check if \(T_3\) is minimal and total, add \(T_3\) 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:
References and further reading

References:


Further reading:
