EVALUATION OF QUERY GENERATORS FOR ENTITY SEARCH ENGINES

Stefan Endrullis, Andreas Thor, Erhard Rahm

University of Leipzig

http://dbs.uni-leipzig.de

Aug. 24, 2009
**Entity Search Engine (ESE)**

Examples: Google Scholar, Amazon Advanced Search for DVDs

- domain specific search engine
- web interface for querying certain type of entities

**Author** Return articles written by

e.g., "PJ Hayes" or McCarthy

**Publication** Return articles published in

e.g., *J Biol Chem* or *Nature*

- set of predicates $p_1, \ldots, p_m$
- corresponding search values $v_1, \ldots, v_m$
- query interpretation: $q \approx \bigwedge_{1 \leq i \leq m, v_i \neq \epsilon} p_i(v_i)$
- result: top-k entities
**Motivation**

Common data integration task: find specific set of entities at ESE

Input entities $S$

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>...</th>
<th>$a_u$</th>
</tr>
</thead>
</table>

Output entities $T$

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$b_1$</th>
<th>...</th>
<th>$b_y$</th>
</tr>
</thead>
</table>

Queries $q_1, \ldots, q_n$

Relevant for
- offline data integration (e.g. data warehousing)
- online data integration (e.g. mashups)

Examples:
- list of products $\rightarrow$ find prices
- list of research papers of venue $\rightarrow$ enrich with citation counts
Challenges

Concept of Query Generators

Evaluation of Query Generators

Initial Evaluation for Bibliographic Domain

Summary and Outlook
Challenges

Problem:
- find given set of entities at an entity search engine

Challenges:
- how to find effective and efficient search queries?
- myriad of different queries (predicate combinations)
- data quality problems (misspelling, missing values, synonyms, ...)
- limited number of requests per time interval
- limited number of entities per result page
**Concept of Query Generators**

**Query Generator:**
- function that maps a set of entities (multi-valued relation) to a set of search queries

![Diagram of Query Generator](image)

4 general properties based on generic data structures:
- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination
**Concept of Query Generators**

Query Generator:
- function that maps a set of entities (multi-valued relation) to a set of search queries

4 general properties based on generic data structures:
- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination

Authors: Stefan Endrullis, Andreas Thor, Erhard Rahm

Title: Evaluation of Query Generators for Entity Search Engines
Query Generator:
- function that maps a set of entities (multi-valued relation) to a set of search queries

4 general properties based on generic data structures:
- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination

Authors
Title
Year
Authors $	ext{first}$ 
intitle
intitle
intitle

Stefan Endrullis, Andreas Thor, Erhard Rahm
Evaluation of Query Generators for Entity Search Engines
Concept of Query Generators

Query Generator:
- function that maps a set of entities (multi-valued relation) to a set of search queries

4 general properties based on generic data structures:
- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination

\[ \begin{array}{cccc}
  a_1 & a_2 & \ldots & a_u \\
  s_1 & & & \\
  s_2 & & & \\
\end{array} \xrightarrow{\text{Query Generator}} \begin{array}{c}
  q_1 \\
  \vdots \\
  q_n \\
\end{array} \]

Input entities $S$

_partition $S$ into $S_1, \ldots, S_n$_
**Concept of Query Generators**

Query Generator:
- function that maps a set of entities (multi-valued relation) to a set of search queries

4 general properties based on generic data structures:
- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination

\[
\begin{array}{cccc}
  a_1 & a_2 & \ldots & a_u \\
  s_1 & & & \\
  s_2 & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
  a_1 & a_2 & \ldots & a_u \\
  s_1 & & & \rightarrow S_1 \rightarrow q_1 \\
  s_2 & & & \rightarrow S_2 \rightarrow q_2 \\
  s_3 & & & \rightarrow S_3 \rightarrow q_3 \\
\end{array}
\]
Concept of Query Generators

Query Generator:
- function that maps a set of entities (multi-valued relation) to a set of search queries

4 general properties based on generic data structures:
- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination
CONCEPT OF QUERY GENERATORS

Query Generator:

- function that maps a set of entities (multi-valued relation) to a set of search queries

4 general properties based on generic data structures:

- attribute-predicate mapping
- search value generation
- partitioning: naive, frequent value
- aggregation: query combination
Naive Query Generator

Properties:

- one query per input entity
- OR aggregation to reduce number of queries
  - not applicable to any ESE

Example:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Smith, Jones}</td>
<td>The question to 42</td>
</tr>
<tr>
<td>{Williams, Smith}</td>
<td>Don’t Panic</td>
</tr>
</tbody>
</table>

Queries:

- \( q_1 = \text{intitle(question 42)} \)
- \( q_2 = \text{intitle(panic)} \)
Naive Query Generator

Properties:

- one query per input entity
- OR aggregation to reduce number of queries
  - not applicable to any ESE

Example:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Smith, Jones}</td>
<td>The question to 42</td>
</tr>
<tr>
<td>{Williams, Smith}</td>
<td>Don’t Panic</td>
</tr>
</tbody>
</table>

Queries:

- \( q_1 = \text{intitle}(\text{question} \ 42) \)
- \( q_2 = \text{intitle}(\text{panic}) \)
Naive Query Generator

Properties:

- one query per input entity
- OR aggregation to reduce number of queries
  - not applicable to any ESE

Example:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Smith, Jones}</td>
<td>The <strong>question</strong> to 42</td>
</tr>
<tr>
<td>{Williams, Smith}</td>
<td>Don’t <strong>Panic</strong></td>
</tr>
</tbody>
</table>

Queries:

- $q_1 = \text{intitle}(\text{question 42})$
- $q_2 = \text{intitle}(\text{panic})$
Frequent Value Query Generator

Properties:
- input analysis → identify frequent item sets (FIS) on attribute values
- variation of Apriori algorithm
- partitioning based on FISs with largest coverage → reduce #queries
- applicable to any ESE

Example:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Smith, Jones}</td>
<td>The question to 42</td>
</tr>
<tr>
<td>{Williams, Smith}</td>
<td>Don’t Panic</td>
</tr>
</tbody>
</table>

Queries:
- $q_1 = \text{author}(\text{Smith})$
Frequent Value Query Generator

Properties:

- input analysis → identify frequent item sets (FIS) on attribute values
- variation of Apriori algorithm
- partitioning based on FISs with largest coverage → reduce #queries
- applicable to any ESE

Example:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Smith, Jones}</td>
<td>The question to 42</td>
</tr>
<tr>
<td>{Williams, Smith}</td>
<td>Don’t Panic</td>
</tr>
</tbody>
</table>

Queries:

- $q_1 = \text{author(Smith)}$
Frequent Value Query Generator

Properties:
- input analysis → identify frequent item sets (FIS) on attribute values
- variation of Apriori algorithm
- partitioning based on FISs with largest coverage → reduce #queries
- applicable to any ESE

Example:

<table>
<thead>
<tr>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Smith, Jones}</td>
<td>The question to 42</td>
</tr>
<tr>
<td>{Williams, Smith}</td>
<td>Don’t Panic</td>
</tr>
</tbody>
</table>

Queries:
- $q_1 = \text{author}(\text{Smith})$
automatic evaluation of several query generators for different datasets

- matching between \( S \) and \( T \) → identify same real world objects
- query statistics \((S, T, M, \#\text{requests})\) stored in data warehouse to perform evaluation afterwards
Evaluation Measures

Mapping example:

\[
\begin{align*}
S & \quad T \\
\bullet & \quad \bullet \\
\end{align*}
\]

\[
\text{domain}(M) \quad \text{range}(M)
\]

Evaluation measures:

\[
\begin{align*}
p\text{recision} & := \frac{|\text{range}(M)|}{|T|} = \frac{3}{4} \\
\text{cov}erage & := \frac{|\text{domain}(M)|}{|S|} = \frac{2}{3} \\
\text{efficiency} & := \frac{|\text{domain}(M)|}{\text{requests}} = 2
\end{align*}
\]
- ESE: Google Scholar, at most 300 results ($3 \times 100$)
- 10 query generators
- 60 ($= 4 \cdot 3 \cdot 5$) datasets based on DBLP:
  - 4 categories (Author, Title, Venue, Random)
  - 3 dataset sizes (5, 30, 100)
  - 5 different datasets for each combination
naive QGs:
- better coverage than frequent value QGs (except #4 which is too precise)

frequent value QGs:
- partially high coverage variations between different categories
aggregation raises efficiency

- frequent value QGs:
  - efficiency strongly depends on dataset characteristics (category)
raise efficiency by following the next link depending on its estimated precision (estimation based on current precision)

Example:
- aim: next link precision $\geq 5\%$
- conclusion: follow next link only iff current precision $\geq 15.8\%$
Summary and Future Work

Summary:

- approach for flexible query generators
- approach for automatic evaluation of query generators
- initial evaluation for bibliographic domain

Future Work:

- adaptive search strategies to automatically select most efficient query generator (combination)
- query generators for additional domains and ESEs