Towards a theory of search queries on dataspaces

Dirk Van Gucht
Indiana University

Joint work with George Fletcher,
Jan Van den Bussche, Stijn Vansummeren

Indiana University

November 18, 2009
1. Theory of database queries
2. Relational algebra
3. Semijoin algebra
4. Search queries
5. Dataspaces
6. Structured querying versus searching
7. Research problems
Classically, any computational problem is a function (mapping) from inputs to outputs.

E.g., route planning:
- Input: a map (graph), source, target
- Output: shortest route in graph from source to target
Database queries

- A query is a function from databases to databases

- E.g., Employee query
  - Input: history of employee hirings
  - Output: list of all employees who have been hired at least twice

- Also route planning!
Relational Algebra

▶ Language in which queries over relational databases can be expressed

▶ Every expression denotes a query
  ▶ compare arithmetic: \( \text{avg}(x, y) \) expresses the function \( \frac{x+y}{2} \)

▶ Expression is a combination of operators
  ▶ union, intersection, difference
  ▶ cartesian product (join)
  ▶ selection
  ▶ projection
  ▶ renaming
Employee query

relation \textbf{History}(\text{emp\_id, hire\_date})

\[\Pi_{H_1.\text{emp\_id}} \sigma_{H_1.\text{emp\_id} = H_2.\text{emp\_id}} \text{ AND } H_1.\text{hire\_date} \neq H_2.\text{hire\_date} \]

\[\rho_{H_1}(\text{History}) \times \rho_{H_2}(\text{History})\]

equivalently:

\[\Pi_{H_1.\text{emp\_id}} (\rho_{H_1}(\text{History}) \Join \rho_{H_2}(\text{History}))\]

\[H_1.\text{emp\_id} = H_2.\text{emp\_id}\]

\[H_1.\text{hire\_date} \neq H_2.\text{hire\_date}\]
Expressibility

- Not all queries are expressible in relational algebra

- E.g., route planning

- Not surprising
  - Ackermann function is not expressible as a primitive recursive formula
The first-order queries

- Relational algebra forms an important **core query language**
  - SQL select-statements = relational algebra + aggregates
  - XPATH 2.0 = relational algebra
  - SPARQL = relational algebra

- Queries expressible in relational algebra are called **first-order queries** (relational calculus)
Semijoin

- Recall Employee query:

\[ \Pi_{H_1.emp\_id}(\rho_{H_1}(\text{History})) \bowtie \rho_{H_2}(\text{History}) \]
\[
\begin{align*}
H_1.emp\_id &= H_2.emp\_id \\
H_1.hire\_date &\neq H_2.hire\_date
\end{align*}
\]

- We don’t need attributes of \( H_2 \) after join

- Semijoin:

\[ \Pi_{H_1.emp\_id}(\rho_{H_1}(\text{History})) \times \rho_{H_2}(\text{History}) \]
\[
\begin{align*}
H_1.emp\_id &= H_2.emp\_id \\
H_1.hire\_date &\neq H_2.hire\_date
\end{align*}
\]
The semijoin algebra (SA)

- Same as relational algebra, except: $\times$ and $\bowtie$ are replaced by $\bowcap$

- SA queries . . .
  - always return subset of the relations (modulo $\Pi$)
  - can be efficiently processed
    - sorting
    - one-pass query processing
    - linear

- SA with only equalities in join conditions
  $=\text{the linear fragment of relational algebra.}$
Searching versus (Structured) Querying

- Users of information systems do not use (full) SQL
  - Library catalog
  - Text search
  - Google, Yahoo, Bing etc
  - Amazon EC2 (searching data in the cloud)
  - mapreduce (key-value pairs)

- They can search:
  - title = OED AND author = Tompa
  - Brussel AND NOT Bruxelles
University of Waterloo
University of Waterloo is located in Waterloo, Ontario, Canada. These pages are part of UW's home on the world wide web.

Search | Results for stargate univers water - TorrentReactor Best ...
Search | Results for stargate univers water torrent download and streaming and free - TorrentReactor.

Indoor Waterpark | Indoor Waterpark | Splash Universe
Indoor waterpark fun for the whole family and always a sunny 84 degrees. Escape with your family to Splash Universe with tons of water park fun!

Video results for univers water
Stargate Universe: Water
43 min - Oct 31, 2009
www.hulu.com

Untitled
21 min - Nov 8, 2009
water.cas.psu.edu
QueryWithAttributes API returns a set of item names and corresponding attributes that match the query expression and the attribute names requested. This can be easily re-written using Select API. For example, a QueryWithAttributes API call with parameters:

```
me.2-SizesDomainName-MyStore&QueryExpression={'Size' : 'Medium' or 'Size' : 'Small'}
```

can be re-written using Select API with parameter:

```
SelectExpression= SELECT Color, Size FROM MyStore WHERE Size = 'Medium' or Size = 'Small'
```

QueryExpression can be easily converted to SelectExpression. In most cases the operators will be the same. The following table lists various operators supported by Query and Select expression. Select API provides more powerful operators such as count, every, in, etc.

<table>
<thead>
<tr>
<th>Query /QueryWithAttributes</th>
<th>Select</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparison operators</strong></td>
<td></td>
</tr>
<tr>
<td>=, !=, &lt;, &gt;, &lt;=, &gt;=, starts-with, does-not-start-with</td>
<td>=, !=, &lt;, &gt;, &lt;=, &gt;=, like, not like, between, in, is null, is not null, every</td>
</tr>
<tr>
<td><strong>Set operators</strong></td>
<td></td>
</tr>
<tr>
<td>intersection, union, not</td>
<td>intersect, (or can be used for union see example below), not</td>
</tr>
<tr>
<td><strong>Logical operators</strong></td>
<td></td>
</tr>
<tr>
<td>and, or, not</td>
<td>and, or, not</td>
</tr>
<tr>
<td><strong>Sort operators</strong></td>
<td></td>
</tr>
<tr>
<td>sort asc, sort desc</td>
<td>Order by asc, order by desc</td>
</tr>
<tr>
<td><strong>Limiting result sets</strong></td>
<td></td>
</tr>
<tr>
<td>MaxNumberOItems parameter</td>
<td>limit</td>
</tr>
<tr>
<td><strong>Counting</strong></td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>count</td>
</tr>
</tbody>
</table>
An abstract **dataspace** is a set of **objects**

Each **object** is a set of **items**

E.g., set of webpages
  - each webpage = set of strings

E.g., classical relation is a set of tuples
  - each tuple = set of attribute-value pairs
Attribute-value pairs

- Tuple

<table>
<thead>
<tr>
<th>epm_imp</th>
<th>hire_date</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>20091021</td>
<td>programmer</td>
</tr>
</tbody>
</table>

- Set of attribute-value pairs

<table>
<thead>
<tr>
<th>att</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp_id</td>
<td>1234</td>
</tr>
<tr>
<td>hire_date</td>
<td>20091021</td>
</tr>
<tr>
<td>job</td>
<td>programmer</td>
</tr>
</tbody>
</table>
Attribute-value dataspaces

- Objects are arbitrary sets of AV-pairs

<table>
<thead>
<tr>
<th>name</th>
<th>Anne</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper</td>
<td>p1</td>
</tr>
<tr>
<td>location</td>
<td>Brussels</td>
</tr>
<tr>
<td>phone</td>
<td>022222785</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper</td>
<td>p1</td>
</tr>
<tr>
<td>paper</td>
<td>p2</td>
</tr>
<tr>
<td>location</td>
<td>Namur</td>
</tr>
<tr>
<td>likes</td>
<td>voetbal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>naam</th>
<th>Ellen</th>
</tr>
</thead>
<tbody>
<tr>
<td>artikel</td>
<td>p2</td>
</tr>
<tr>
<td>artikel</td>
<td>p3</td>
</tr>
<tr>
<td>plaats</td>
<td>Brussel</td>
</tr>
<tr>
<td>plaats</td>
<td>Namen</td>
</tr>
<tr>
<td>houdtvan</td>
<td>rugby</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>drink_type</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Orval</td>
</tr>
<tr>
<td>kind</td>
<td>Trappist</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>paper_id</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>SQL</td>
</tr>
<tr>
<td>proceedings</td>
<td>VLDB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>paper_id</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>OED</td>
</tr>
<tr>
<td>author</td>
<td>Tompa</td>
</tr>
<tr>
<td>journal</td>
<td>unknown</td>
</tr>
</tbody>
</table>
“Database of everything”

- Alon Halevy (University of Washington, Google)

- Very similar to Semantic Web
  - RDF
  - Linked Data
A-V dataspace as RDF store

- RDF store: set of triples
  - (subject, predicate, object)

- View A-V dataspace $D$ as set of triples:
  - $\{(oid, att, val) : oid \in D \& (att, val) \in oid\}$

<table>
<thead>
<tr>
<th>oid</th>
<th>att</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Anne</td>
</tr>
<tr>
<td>1</td>
<td>paper</td>
<td>p1</td>
</tr>
<tr>
<td>1</td>
<td>location</td>
<td>Brussels</td>
</tr>
<tr>
<td>1</td>
<td>phone</td>
<td>022222785</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>paper_id</td>
<td>p3</td>
</tr>
<tr>
<td>6</td>
<td>title</td>
<td>OED</td>
</tr>
<tr>
<td>6</td>
<td>author</td>
<td>Tompa</td>
</tr>
<tr>
<td>6</td>
<td>journal</td>
<td>unknown</td>
</tr>
</tbody>
</table>
RDF triple store as A-V dataspace

- Use 3 special attributes
  - subject
  - predicate
  - object

- RDF triple store is just a relation over the scheme
  \{subj, pred, obj\}

- Already know that a relation is a dataspace!

- No RDFS
Searching Dataspaces

- Abstract Dataspace
  - set of objects
  - object: set of items

- Abstract keywords
  - predicate on items

- E.g., when items are strings:
  - string contains “Water”
  - string contains “Univers”
  - strings following “Waterloo” in OED
  - synonyms of “data”
Every keyword $k$ is an expression

Meaning:
- Retrieve all objects containing some item satisfying $k$

If $e_1$ and $e_2$ are expressions then so are:
- $e_1 \text{ AND } e_2$
- $e_1 \text{ OR } e_2$
- $e_1 \text{ AND NOT } e_2$

Meaning: union, intersection, set difference

Waterloo AND NOT(Toronto OR Vancouver)
Dataspace search queries

- Database query:
  - mapping from databases to databases

- Dataspace query:
  - mapping \( q \) from dataspaces to dataspaces

- Dataspace search query:
  - such that \( q(D) \subset D \) for each \( D \)

- Bit like semijoin queries ...
Your search - frank - did not match any documents.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
What dataspace search queries are expressible in BSL?

- BSL queries are **safe**
  - Only return objects containing some item satisfying (matching) some keyword that we used

- BSL queries are **additive**
  
  \[ q(D) = \text{union of all } q(o) \text{ for all } o \in D \]
BSL queries are finitely distinguishing

- Only distinguish objects using some finite set $K$ of keywords

- $o_1$ and $o_2$ are $K$-equivalent if for each $k \in K$, $o_1$ matches $k \iff o_2$ matches $k$

- When $o_1$ and $o_2$ from $D$ are $K$-equivalent then $o_1 \in q(D) \iff o_2 \in q(D)$
A dataspace query $q$ is expressible in BSL if (and only if) $q$ is additive, and for some finite set $K$ of keywords,

- $q$ is $K$-safe and
- $q$ is $K$-distinguishing
Application to relational selection queries

- Recall: relation = set of tuples = set of objects

- Object = set of attribute-value pairs

- Keywords: $A = c$
  - $A$: attribute from the given relation scheme
  - $c$: arbitrary constant

- Also wildcard keyword: *

- Example BSL query:
  * AND NOT(job=programmer OR emp_id = 1234)

- Same as relational algebra using only $\cup$, $\neg$, $\sigma_{A=c}$
Negated keywords (if you don’t have them)
  ▶ retrieve all objects containing an item not matching “Waterloo”
  ▶ not finitely distinguishing over positive keywords

Normally will use boolean-closed repertoire of keywords
Neither expressible in BSL

- Retrieve all objects sharing an item with an object matching “Waterloo”

- Retrieve all co-authors of “Frank Tompa”

- Not additive

- We cannot do joins or even semijoins

- Want to do such “associate search”
Similarity relations (simrels)

- How to link (associate) two objects?
  - hard wire links between objects in the dataspace
  - not necessary
  - not flexible

- Better: use simrels between items
  - a simrel is a binary predicate on items
Examples of simrels

▶ Equality

▶ Translation on city names:
  ▶ Namur \textit{trans} Namen
  ▶ Bruxelles \textit{trans} Brussel
  ▶ Anvers \textit{trans} Antwerpen

▶ Equal-value on A-V pairs:
  ▶ (likes, Tompa) \textit{eqval} (name, Tompa)

▶ Equal-attribute on A-V pairs:
  ▶ (name, Tompa) \textit{eqval} (name, Gonnet)
If \( k \) and \( k' \) are keywords, and \( \approx \) is a simrel, then \( k \approx k' \) is a simlink

Meaning: binary predicate on items
  - will be used to link (associate) objects

\[ i_1 [k \approx k'] i_2 \] if
  - \( i_1 \) matches \( k \)
  - \( i_2 \) matches \( k' \)
  - \( i_1 \approx i_2 \)

Example on string items, with sub-string and wildcard keywords and translation simrel:

“Citadelle de Namur” [Namur trans *] “Citadel van Namen”
Linking objects using simlinks

- For objects $o_1$ and $o_2$, $o_1[k \approx k']o_2$ if
  - $o_1$ contains some item $i_1$
  - $o_2$ contains some item $i_2$
  - $i_1[k \approx k']i_2$ if

- New associative search operator on dataspace:
  \[
  \text{LINK} \ [k \approx k'](S)
  \]

- retrieve all objects in the dataspace that are linked by $[k \approx k']$ to some objects in $S$
  \[
  \text{LINK}[\text{Namur trans *]} \ (\text{Citadel})
  \]
BSL extended with link operator

Parametrized by choice of:
- keywords (already as BSL)
- simrels (for link operator)

What is the expressiveness of ASL?

Link operators is like semijoin . . .

\[ e_1 \text{ AND } \text{LINK}[\theta](e_2) \]
\[ e_1 \bowtie_{\theta} e_2 \]
ASL on A-V dataspaces

Keywords:
- literals & wildcards
  (name: Frank) (name: *) (*: Frank)
- negation on values
  (likes: \neg(\text{Heineken,Budweiser}))
- negation on attributes
  (\neg(paper\_id, title):Kriek)
- negation on both values and attributes
  (\neg(paper\_id, title): \neg(\text{Heineken,Budweiser}))

Simrels:
- eq, eq\_val, eq\_att
Example query

- Retrieve all people located in Waterloo who have published a paper in CACM:

  \[(\text{location: Waterloo}) \text{ AND } \text{LINK}[(\text{paper: *}) \text{ eq_val (paper_id: *)]}(\text{journal: CACM})\]

- Which queries can we express?
A-V dataspace as relation

- We saw this already: set of (oid, att, val)

<table>
<thead>
<tr>
<th>oid</th>
<th>att</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Anne</td>
</tr>
<tr>
<td>1</td>
<td>paper</td>
<td>p1</td>
</tr>
<tr>
<td>1</td>
<td>location</td>
<td>Brussels</td>
</tr>
<tr>
<td>1</td>
<td>phone</td>
<td>022222785</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>paper_id</td>
<td>p3</td>
</tr>
<tr>
<td>6</td>
<td>title</td>
<td>OED</td>
</tr>
<tr>
<td>6</td>
<td>author</td>
<td>Tompa</td>
</tr>
<tr>
<td>6</td>
<td>journal</td>
<td>unknown</td>
</tr>
</tbody>
</table>

- How does ASL compare to querying this relation using relational algebra?
ASL translated into semijoin algebra

(\text{location: Waterloo}) \text{ AND } \text{LINK}[(\text{paper: } *) \text{ eq_val} (\text{paper_id: } *)] (\text{journal: CACM})

\begin{align*}
\Pi_{\text{oid}} \sigma_{\text{att}=\text{location} \& \text{val}=\text{Waterloo}}(T) \bowtie \Pi_{\text{oid}} \sigma_{\text{att}=\text{paper}}(T) \bowtie \\
\Pi_{\text{oid}} \sigma_{\text{att}=\text{paper_id}}(T \bowtie \Pi_{\text{oid}} \sigma_{\text{att}=\text{journal} \& \text{val}=\text{CACM}}(T))
\end{align*}

▶ Only natural semijoins are used
SA queries not expressible in ASL

- Retrieve all people who have the same value for a boss and a friend attribute
- Retrieve all people who like some professor that nobody else likes
- Can prove that these are not expressible using invariance under bisimulations
Bisimilarity of Dataspace

- Dataspace $D$ and object $o$, also $D'$ and $o'$

- Natural number $n$

- We say that $(D, o) \leftrightarrow_n (D', o')$ if
  - $o$ and $o'$ match precisely the same keywords
  - moreover for $n > 0$:
    - for each simrel $\approx$ and for each object $p \in D$ such that $o \approx p$, there exists $p' \in D'$ such that $o' \approx p'$ and $(D, p) \leftrightarrow_{n-1} (D', p')$
  - vice versa (from $D'$ to $D$)
Invariance under bisimilarity

- Let $q$ be an ASL query using at most $n$ nested link operators.

- Let $(D, o) \leftrightarrow_n (D', o')$

- Then $o \in q(D)$ if and only if $o' \in q(D')$.
Retrieval queries not expressible in ASL (repeated)

- Retrieve all people who have the same value for a boss and a friend attribute
- Retrieve all people who like some beer that nobody else likes
- Can prove that these are not expressible using invariance under bisimulations
The “search” fragment of SA (semi-join algebra)

\[ E ::= T \]

\[ \quad \sigma_{\text{att}=c}(E) \]
\[ \quad \sigma_{\text{val}=c}(E) \]
\[ \quad E \cup E \]
\[ \quad E - E \]
\[ \quad \Pi_\alpha(E) \]
\[ \quad E \Join \Pi_{\text{oid}}(E) \]
\[ \quad \Pi_{\text{oid}}(E \Join \Pi_\beta(E)) \]

▶ c: constant

▶ \( \alpha \): \{oid\}, \{oid, att\}, or \{oid, val\}

▶ \( \beta \): \{att\}, \{val\}, or \{att, val\}
What have we learned?

- Searching unstructured information motivates to investigate new query languages
  - but the classical theory is still very useful:
    - relational databases
    - relational algebra
    - genericity
    - semijoin algebra
    - bisimilarity

- Querying RDF triple stores
Open research problems

- Algorithms, data structures for query processing
- Are BSL and ASL sufficient? Other primitives?
- User interface: search should be easier than full querying in SQL
- How to represent relational databases as dataspaces (or RDF) such that querying can be done by searching?