DBXplorer

- “programming knuth”
  - 1st result:
- “soumen sunita”
  - No result!

Info
- MSR, ICDE 2002
- One of the first to adopt the relation-based approach
- Discussed many issues: (1) index creation & compression; (2) “join tree” generation; and (3) efficient SQL generation without full-text index

Designed to use SQLs to find results, all algorithms utilizes this fact
Data and Query Models

- Data Model
  - Relational model
  - Granularity = tuple

- Query Model
  - Query answers are instances of minimal and total join trees

- Ranking
  - Only by the size of the result
  - Exact top-k ranking (but barely useful)
    - Many ties intra- & inter- join tree instances
Query Processing

Basic idea:

1. If we known which table+col+(row)
each keyword appears
2. Then we can generate SQLs that
   \textit{might} generate results
3. Execute the SQLs in the increasing
   order of their sizes until $k$ results
   are obtained

Use col-based or cell-based
index

Breadth-first enumeration of all
qualified subtrees from the schema
graph

Results automatically ranked
“soumen sunita”

1st result

Info

- UCSD, VLDB 2002
- Relation-based
- Emphasizes on the candidate network generation algorithm and intra-SQL sharing optimization
Data and Query Models

- **Data Model**
  - Relation-based, same as DBXplorer

- **Query Model**
  - Query answers are instances of *promising*, *minimal* and *total* candidate networks

- **Ranking**
  - Only by the size of the result
  - Exact top-k ranking (but barely useful)
Query Processing

1. Generate the *tuple set graph* from the schema graph and query keywords

2. Breadth-first enumeration of all Candidate Networks (CNs)

3. Rewrite the list of CNs into an execution schedule

4. Execute it
DISCOVER2

- “soumen sunita”
  - with better ranking
- Info
  - UCSD, VLDB 2003
  - Relation-based
  - Emphasizes on IR-style ranking function and optimized execution algorithm for top-$k$ queries
Data and Query Models

- Data Models Remains the Same
- Query Model:
  - Allow both AND and OR semantics (for multiple-keyword query)
- Ranking
  - 1st to introduce IR-style ranking heuristics
  - \[ \text{Score}(n, Q) = \sum_{w \in Q \cap n} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s \cdot dl / avdl} \cdot \ln \frac{N + 1}{df} \]
  - Property: tuple monotonicity

Results 1 - 10 of about 809,000 for sunita. (0.35 seconds)

Results 1 - 10 of about 207,000 for soumen. (0.20 seconds)

You can get it for free (in most RDBMSs)!
Query Processing

1. Generate the tuple set graph
2. Generate a list of CNs
3. Execution algorithms optimized for top-k queries
   - Naïve → Sparse → Single pipeline → Global pipeline
   - Hybrid

Do not do unnecessary work! Push top-k constraints inside!

Naïve
- Retrieve top-k results from each CN
  - ORDER BY + LIMIT
- Merge them to obtain top-k query result

tuple monotonicity ensures the correctness
Sparse

**Idea**
- Start from the smallest CNs (*heuristic*)
- Prune the rest of the CNs using the current top-$k$ score & MPSs of the remaining CNs.

**Question:** Can we avoid executing some CNs?

**Answer:** $CN_2$ does not need to be executed, as its result, if any, won’t make it into top-2 results.
### Pipelined

**ResultHeap:**

<table>
<thead>
<tr>
<th>1st</th>
<th>$h_1-a_8$</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd</td>
<td>$h_3-a_8$</td>
<td>1.9</td>
</tr>
</tbody>
</table>

**Idea**
- (Intra CN) calculate the MPFSs to select the next tuple to probe (the seen parts of) other relations
- Early stop based on MPFS scores

**Answer:** execution on $CN_1$ can stop now

---

**$CN_1$**

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>1.8</th>
<th>$a_8$</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_3$</td>
<td>1.6</td>
<td>$a_5$</td>
<td>2.1</td>
</tr>
<tr>
<td>$h_7$</td>
<td>1.2</td>
<td>$a_9$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

- $MPFS(h_1-a_5) = 1.95$
- $MPFS(h_3-a_8) = 1.90$
- $MPFS(h_3-a_8) = 1.90$
- $MPFS(h_1-a_9) = 1.40$
- $MPFS(h_7-a_8) = 1.70$
- $MPFS(h_1-a_8) = 1.40$

---

**Question:** Can we avoid executing a CN to its full?
Global Pipelined

Can we generalize the idea to multiple CNs?

<table>
<thead>
<tr>
<th>CN₁</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>h₁</td>
<td>1.8</td>
<td>a₈</td>
</tr>
<tr>
<td>h₃</td>
<td>1.6</td>
<td>a₅</td>
</tr>
<tr>
<td>h₇</td>
<td>1.2</td>
<td>a₉</td>
</tr>
</tbody>
</table>

**Question:** Can we generalize the idea to multiple CNs?

```
get_MPFS()
next()
```
Global Pipelined

- Idea
  - (Inter-CN) Activate the CN that has the max MPFS
  - Early stop based on MPFS scores and the score of the current $k$th result

**Question:** Can we generalize the idea to multiple CNs?

**Answer:** algorithm can terminate when the max of MPFSs is smaller than the score of the last top-$k$ result

**Question:** Can we generalize the idea to multiple CNs?
SPARK

- “soumen sunita”
  - Improved ranking
    - problem fixes
    - integration of AND and OR semantics
  - Efficient query processing algorithm
    - Why challenging?

- Info
  - UNSW, SIGMOD 2007*
  - Relation-based
  - Starts with more effective ranking methods and then focuses on efficient algorithms for the non-monotonic score aggregate function
Details

- See the other lecture notes
Summary

- **Trend**
  - DB-style query (proximity search) → DB+IR-style query for DB (DISCOVER2)
  - Optimization for “effectiveness”
  - Optimization for “efficiency”
  - No convincing study on effectiveness yet, 😞

- **Graph-based vs. relation-based approach**
  - Different design choices
    - Working with a variety of data, or
    - Fully exploit/integrate with the RDBMS
  - Analogy to database query processing & optimization
    - dynamic vs static query optimization
  - Could be complementary

---

E.g., Indexing terms:
- Implement/use (external) inverted index
- create a symbol table and index it using full-text index
Outline

- Introduction
- IR Preliminaries
- Systems
- Open Issues
Benchmarking

- Datasets and Queries
  - Vary the distributions of structure and keywords
  - Medium to large, real datasets
  - Large number of real queries

- Ranking
  - Not only numbers, but user experience/satisfaction is important too!
Advanced Features

- Combine keyword search with other constraints
  - Language issues
  - Algebra?
  - (Rank-aware) Query optimization

- Finetune the result
  - “Find similar” ... ➔ goal-oriented
  - Personalized ranking?

- Visualization and Better User Interaction
  - Easy to understand the result
  - Drill-down?
Q & A

thanks!