**Background**

An information worker
- Has no exact knowledge of the database and its underlying schema.
- Seeks queries to cover information needs.
- Spends a lot of time to go over the database schema and manually discover the queries in need.

**Observation**

The information worker knows a few example tuples that should be present in the output of the queries.

**System Task**

Given a spreadsheet filled with example tuples return the top-k most relevant minimal PJ-queries.

**Spreadsheet-Style Search**

**Input:** Spreadsheet

**Output:** Top-k Minimal PJ Queries

**Scoring Model // What makes a good score?**

- **Row Containment**
  - Are the example tuples contained in the output of the PJ-query and the desired PJ-query?
  - Penalize relationship-alignment errors.

- **Column Containment**
  - Are the example tuples contained in the output of the PJ-query and the desired PJ-query?
  - Penalize alignment errors.

**Balancing Factor a**

And all example tuples, ranking them by the column score.

**Evaluation w/ Early Termination**

- **Theoretical result:** The score of a PJ-query given a spreadsheet is bounded by the column score.
- **Column score:** Efficiently computed using the indices built offline.
- **Idea:** Use column score as the upper bound score.

**Strategy**

- Evaluate PJ-queries in the decreasing order of upper bound score.
- Terminate when current upper bound score less than the k-th ranked evaluated PJ-query.
- Guarantee minimal evaluation set of PJ queries.

**Sharing Sub-PJ Queries**

- Reuse previously computed intermediate results for Sub-PJ queries to efficiently evaluate multiple PJ queries.
- **But:** Cache is limited.
- **We want to evaluate the minimum amount of PJ queries (to early terminate).**

**Caching-Evaluation Scheduling Problem**

- The two extremes of PJ queries evaluation.
  - **One at a time:** Minimum evaluation set.
  - **All at once:** Maximum share of Sub-PJ queries.

**NP-hard even when the minimum evaluation set is given by an oracle.**

**Experiments // Evaluation**

- **Database:** CSUPP, 4759.7, 12374, 105, 821, 63

- **Evaluation strategies:**
  - Naive: Evaluation of all minimal PJ-queries.
  - Baseline: Evaluation w/ early termination strategy.
  - FastTopK: Evaluation w/ sharing sub-PJ-queries and early termination.

**SPREADSHEET GENERATION**

50 spreadsheets generated from meaningful PJ-queries (added 2 alignment errors) bucketed under low, medium, high cost classes.

**Contributions**

- **Introducing top-k spreadsheet style search and a novel scoring model for error resilient but efficient PJ query discovery.**
- **Efficient score the PJ queries using a novel Enumeration-Evaluation framework.**
- **Focusing on evaluation using the framework:**
  - We introduced a strong baseline that evaluates the minimum number of minimal PJ-queries to terminate early.
  - Reused the baseline by sharing sub-PJ-queries and limiting evaluation.
  - Introduced optimizations to further increase speedup.

- **Examinations of our model and approaches include:**
  - Incremental computation of the output as soon as the user types in a cell.
  - OR semantica to allow partial mapping of columns.

---

**S4: Top-k Spreadsheet Style Search for Query Discovery**

Fotis Psalidhas

Columbia University, New York, NY

fotis@cs.columbia.edu

Bolin Ding

Kaushik Chakrabarti

Microsoft Research, Redmond, WA

Surajit Chaudhuri

bolind, kaushik, surajitc@microsoft.com