A Framework for Testing Query Transformation Rules

HICHAM G. ELMONGUI, VIVEK NARASAYYA, RAVI RAMAMURTHY
SIGMOD’09

Presenter: ZHANG, Yushan @Prism
Outline

➢ Background of this research
➢ The Challenges
➢ Framework Design
➢ Experiment
➢ Reflections
Query Optimization: Overview

Two Requirements:

- Produce faster plans
- Guarantee correct results
Transformation Rule: example

Let $E$ denote a relational algebra expression, $\theta$ is a predicate

E.g.

$$\sigma_{\theta_1 \land \theta_2}(E) \equiv \sigma_{\theta_1}(\sigma_{\theta_2}E), \text{ a cascade of } \sigma$$

$$E_1 \bowtie_{\theta} E_2 \equiv E_2 \bowtie_{\theta} E_1, \text{ theta-join is commutative}$$
Testing Aspects

- **Coverage**
  - a given transformation rule (*rule*) should be executed with different queries

- **Correctness**
  - a given transformation rule should NOT alter the results of a query
  - a pair of transformation rules should NOT alter the results of a query

- Performance... (not considered in this paper)
The Challenges

➢ Efficiently generate queries that exercise a particular rule (Generation Problem)
➢ Randomly generation [1][2] takes many trials before find a query - slow
➢ Randomly generated queries are hard to interpret - complex

➢ Efficiently execute test suites for correctness testing (Compression Problem)
➢ Intuitive approach: turn off the rules one by one, and compare the results
➢ Problem: un-optimized queries could be extremely slow

Generation with Rule Patterns

➢ Captures the *sufficient condition* for a rule to execute is hard
  ➢ Pull up GBAgg requires certain functional dependency

➢ Use **Rule Patterns** (*necessary condition*)
  ➢ A query exercising the given rule must have the corresponding pattern
  ➢ Initialize the “wholes” in the pattern
  ➢ Support rule composition (extends to test multiple rules)

**Take-away message:**
➢ reduce the trials with known necessary conditions
Test Suite Compression (TSC)

➢ Very **high cost** for executing the test suite
   ➢ A query could exercise multiple rules
   ➢ A rule is exercised by multiple rules

Intuition (**bipartite graph problem**):

A set of transformation rules: \( R = \{ r_1, r_2, \ldots r_n \} \)

A test suite for a rule has \( k \) queries

Overall test suites for \( R \): \( TS = \bigcup_i TS_i \)

(with cost) Nodes are rules and queries

(with cost) Edge from \( r_i \) to \( q_j \) denotes \( r_i \) is exercised when \( q_j \) is optimized.

**Question:** what is the optimal solution for this example?

\[
\begin{align*}
TS_1 &= \{ q_1 \} \\
TS_2 &= \{ q_2 \} \\
Cost(TS_1) &= 100 + 180 \\
Cost(TS_2) &= 100 + 120 \\
Cost(TS) &= Cost(TS_1) + Cost(TS_2)
\end{align*}
\]

**Goal:** Minimize cost of TS

**Constraint:** Each rule is exercised \( k \) times
Test Suite Compression (TSC)

- Very **high cost** for executing the test suite
- A query could exercise multiple rules
- A rule is exercised by multiple rules

Intuition (**bipartite graph problem**):
A set of transformation rules: \( R = \{r_1, r_2, \ldots, r_n\} \)
A test suite for a rule has \( k \) queries
Overall test suites for \( R \): \( TS = \bigcup_i TS_i \)
(with cost) Nodes are rules and queries
(with cost) Edge from \( r_i \) to \( q_j \) denotes \( r_i \) is exercised when \( q_j \) is optimized.

Optimal solution:
\[
\text{Cost}(TS) = 100 + 120 + 120
\]
Test Suite Compression (TSC)

➢ Reduction from the Set Cover Problem to TSC (NP-Hard)
  ➢ Only approximation algorithm is possible

➢ Two algorithms
  ➢ Applying the Set Cover Heuristic (ignores edge cost)
    ➢ Constrained Set Multicover algorithms (SMC)
  ➢ TopKIndependent algorithm (considers edge cost)
    ➢ Consider edge costs for each rule separately, sort them and use greedy picks
    ➢ Further optimization to boost the picking process
Framework Overview

- **Background**
- **The Challenges**
- **Framework Design**

Diagram:
- **Query Generation**
  - Generate logical query tree
  - Generate SQL

- **Correctness Validation**
  - Test suite generation
  - Test suite compression
  - Test suite execution

- **Query Optimizer**
  - Rules Exercised
  - Turn On/Off Rule

- **DBMS**
  - Execute
Experiment Setup

The approach is prototyped in Microsoft SQL Server

Databases: TPC-H

Used 30 rules of the optimizer

- *The efficiency of query generation*: random vs. pattern
- *The effectiveness of two algorithms*: SetMultiCover vs. TopKIndependent
- *The importance of exploiting monotonicity* (further optimization on TopKIndependent)
Random vs. Pattern based Generation

- **Number of Trials (Singleton rules)**
  - Random and Pattern comparisons

- **Number of Trials (Rule Pairs)**
  - Random and Pattern comparisons

- **Query Generation Time (Rule Pairs)**
  - Random and Pattern comparisons
Compression Quality

**Background**

**The Challenges**

**Framework Design**

**Experiments**

**Y-axis:**

Log scale

Optimizer estimated cost
Summary

This paper considers the problem of testing transformation rules of a query optimizer.

Main Contributions:

➢ Efficiently generate queries to cover the given rules
  ➢ Use query patterns to reduce the random generations trials

➢ Efficiently execute test suites for correctness testing
  ➢ Use approximation to pick a subset of the queries
Reflections

- This paper has a very clean and clear writing style, which worth a detailed reading
- There is a well-established logic between the defined problem and the solution
- A typical example is used throughout the paper
- The approach illustration is simple to understand without the theory