Optimizing Join Enumeration in Transformation-based Query Optimizers

ANIL SHANBHAG, S. SUDARSHAN

IIT BOMBAY

anils@mit.edu, sudarsha@cse.iitb.ac.in

Query Optimization: Quick Background

System R algorithm

- Dynamic programming algorithm to find best join order
- Time complexity: $O(3^n)$ for bushy join orders
- Plan space considered includes cross products

For some common join topologies #cross-product free intermediate join results is polynomial

- E.g. chain, cycle, ..

Can we reduce optimization time by avoiding cross products?

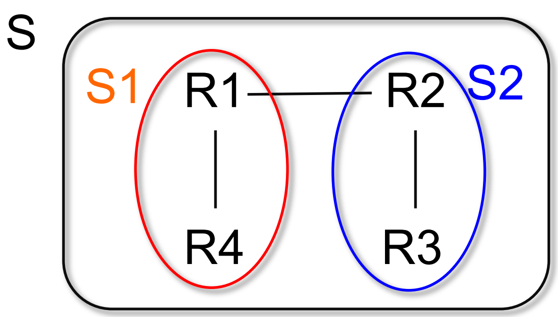
- Algorithms for generation of cross-product free join space
  - Bottom up: DPccp (Moerkotte and Newmann [VLDB06])
  - Top-down: TDMinCutBranch (Fender et al. [ICDE11]), TDMinCutConservative (Fender et al. [ICDE12])
- Time complexity is polynomial if #cross-product free intermediate join results is polynomial in size

Cross-Product-Free Join Order Enumeration using Graph Partitioning

Key idea for avoiding cross products while finding best join tree:

For set S of relations, find all ways to partition S into S1 and S2 s.t.

- the join graph of S1 is connected, and so is the join graph of S2
- there is an edge (join predicate) between S1 and S2



Simple recursive algorithm to find best plan in cross-product free join space using partitioning as above

Efficient algorithms for finding all ways to partition S into S1 and S2 as above

- MinCutLazy (Dehaan and Tompa [SIGMOD07])
- Fender et. al proposed MinCutBranch [ICDE11] and MinCutConservative [ICDE12]
  - MinCutConservative is the most efficient currently.

Volcano/Cascades Framework for Query Optimization

Based on equivalence rules: e.g. $A \bowtie B \leftrightarrow B \bowtie A$

Key benefit: easy to add rules to deal with new operators

- e.g. outerjoin group-by/aggregate, limit, ...
- Memoization technique which generalizes System R style dynamic programming applicable even with equivalence rules

Used in SQL Server, Tandem, and Greenplum, and several other databases, increasing adoption

Transformation rule sets for join order optimization:

RS-B1: Commutativity + Left Associativity: Takes O(4^n) time

RS-B2: Pellenkoft et. al [VLDB97] suggest new ruleset: O(3^n) time

Both the rulesets generate join orders with cross-products.

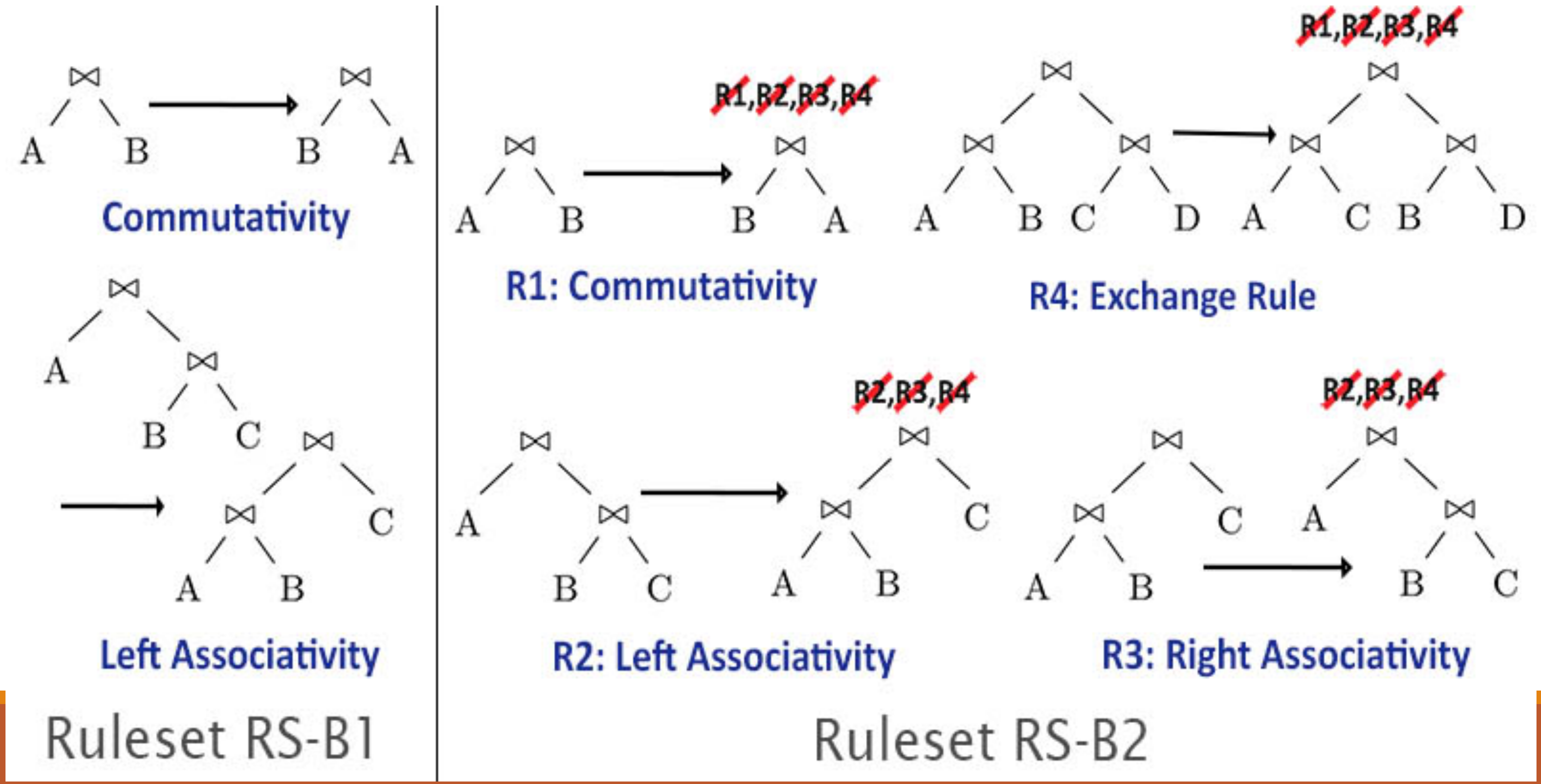
- **Key contribution of paper: Efficient rulesets that avoid cross-products**

Rulesets with Cross-Product Suppression (CPS)

RS-B1-CPS/RS-B2-CPS: modification of RS-B1/RS-B2 to suppress cross-products, i.e. block transformation if the result has cross-product

RS-B1-CPS and RS-B2-CPS have been used in some implementations

- but not obvious if they are complete, i.e. generate the entire search space



RS-B1-CPS
Proof of Completeness

Theorem: RS-B1-CPS is complete i.e. any cross-product free tree Q1 can be converted to any other cross-product free tree Q2 using RS-B1-CPS

Intuition for the proof

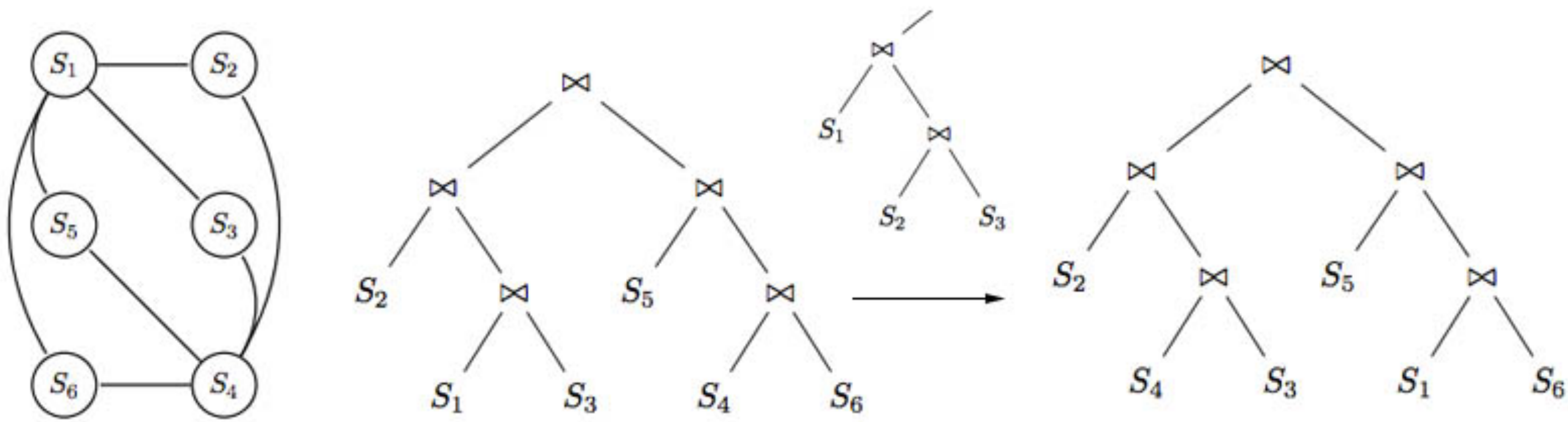
- Step 1: Given any arbitrary cross-product-free tree Q1 we can convert it into a canonical cross-product free left-deep tree
  $Qc = (..((R1 \bowtie R2) \bowtie R3)..) \bowtie Rk)$ with relations in sorted order
  using RS-B1-CPS
- Step 2: Above steps can be reversed using RS-B1-CPS for any cross-product free tree
- Can go from any Q1 to any Q2 as above via Qc

RS-B2-CPS is Incomplete

Some cross-product free trees may not be reachable from other cross-product free trees using RS-B2-CPS.

Proof of incompleteness of RS-B2-CPS using counter-example below

- Q and Q2 are both cross-product free join trees
- Starting with Q, we can go to Q2 only via application of exchange rule at root join op
- This will always result in an intermediate tree with cross-product !



(a) Join Graph J (b) Query Tree Q (c) Query Tree Q_2

Problem and Potential Fix

Problem: RS-B1-CPS and RS-B2 are complete, however

- RS-B1-CPS generates exponential number of duplicates (Pellenkoft et al.)
- RS-B2 explores significantly larger search space (no CPS)

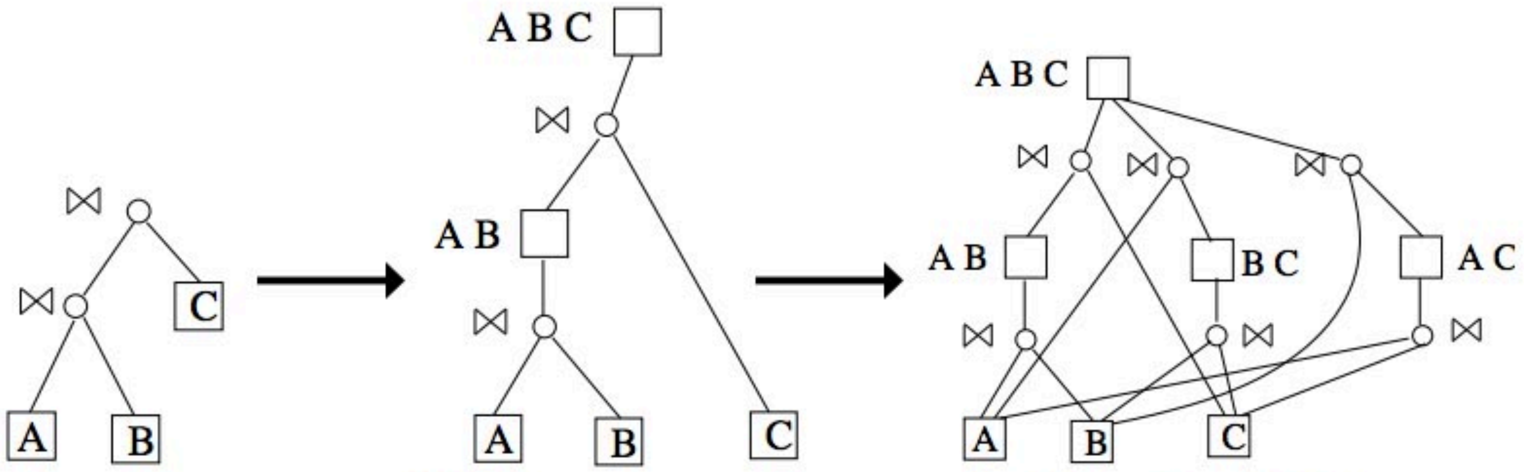
Key idea: incorporate graph-partitioning based top-down enumeration into Volcano/Cascades framework

AND-OR DAG Representation in Volcano/Cascades

Repeatedly apply a set of rules until fixedpoint

Store the alternatives efficiently using AND-OR DAG representation .

Example shows join enumeration for a simple query in transformation-based QO :



(a) Initial Query (b) DAG representation (c) Expanded DAG

Join Sets

For applying graph-partitioning based enumeration, we need to create a join graph consisting of nodes being joined

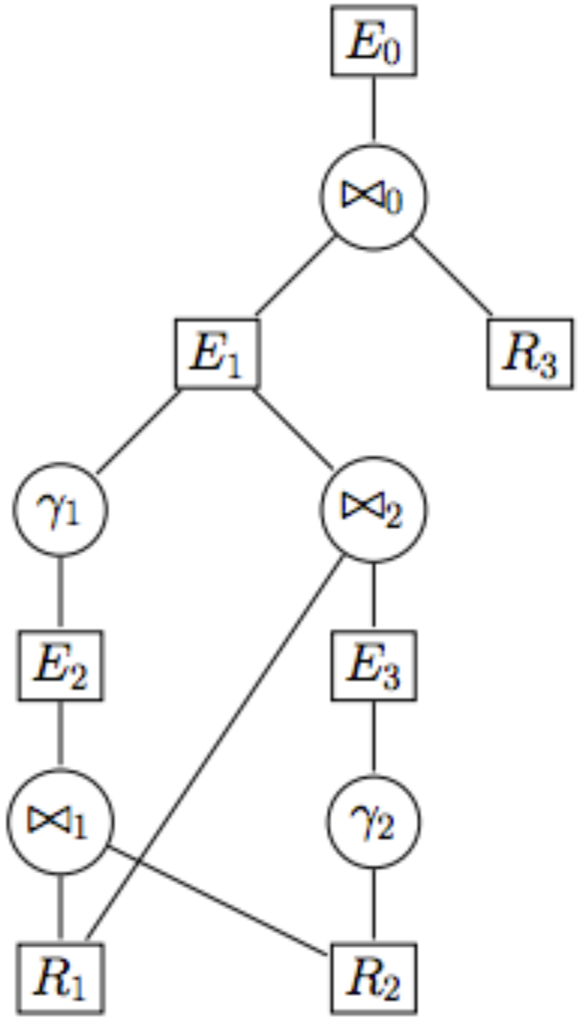
A **maximal join set** at an equivalence node E is a maximal set of equivalence nodes Ei being joined below E such that none of the Ei have any join operators below them.

There can be multiple maximal join sets at an equivalence node

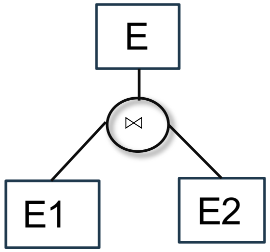
- we store all of them.

In the example to the right, at E_0

- (R_1, E_3, R_3) is a maximal join set,
- But (E_1, R_3) is not since E1 has join operator below it



Transformation Rule RS-Graph



Rule RS-Graph: matches pattern E1 ⋈ E2

On match, **For each** pair (J1, J2) where J1 ∈ join sets of E1 and J2 ∈ join sets of E2

- **If** J1 U J2 has **not** been enumerated at node E, where E is the parent equivalence node of E1 ⋈ E2
  - Call the partitioning algorithm on the join graph of J1 U J2 to generate all cross-product free partitions
  - **For each** such partition S1, (J1 U J2)\S1
    - We check if there is equivalence node representing S1 (similarly G\S1)
      - This is done efficiently by inserting a dummy n-ary join operator into the DAG and using standard Volcano/Cascades duplicate expression check .
    - If yes, we simply use the equivalence node in place of S1.
    - If not, we create a left-deep join tree of relations in S1 and insert it into the DAG. Use the equivalence node thus created for S1.

RS-Graph (Contd.)

The Volcano/Cascades framework will recursively apply RS-Graph on generated nodes to generate entire space

Join sets at a node may change as transformations are applied at child equivalence nodes

- Join sets can be maintained in a bottom-up fashion.

Theorem: RS-Graph is complete

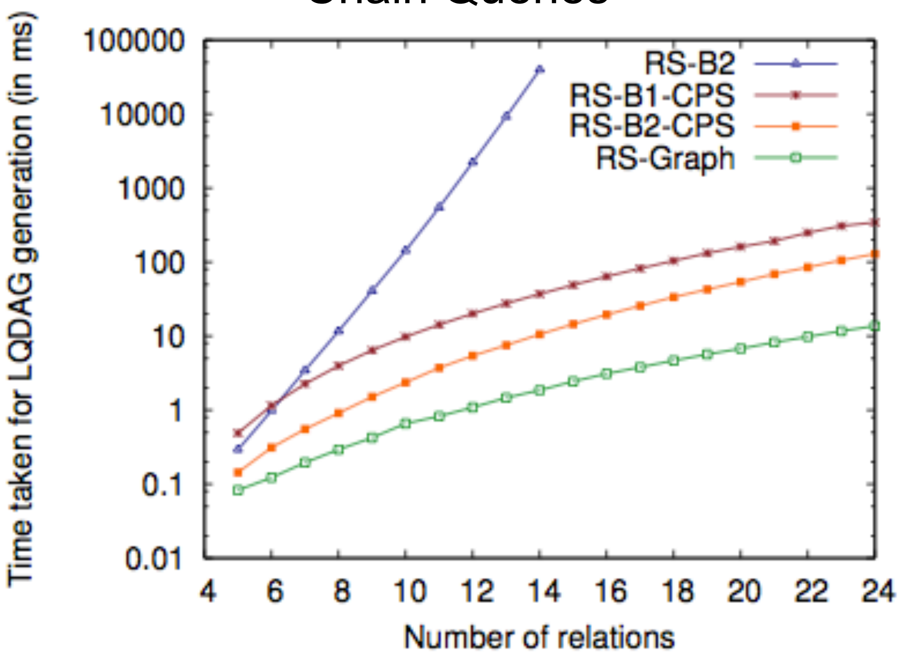
Potential risk: equivalence nodes may have many maximal join sets

Good news: For commonly encountered rulesets, each equivalence node has a single maximal join set.

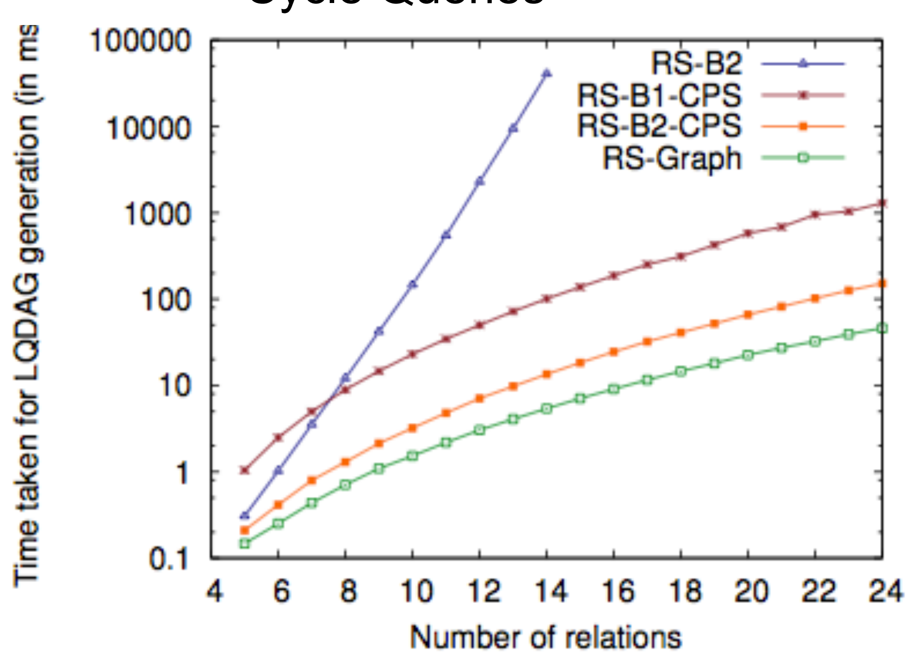
Performance

Incompleteness of RS-B2-CPS observed in cycle queries (# Eq. Nodes)
LQ DAG Expansion time (ms)

Chain Queries



Cycle Queries



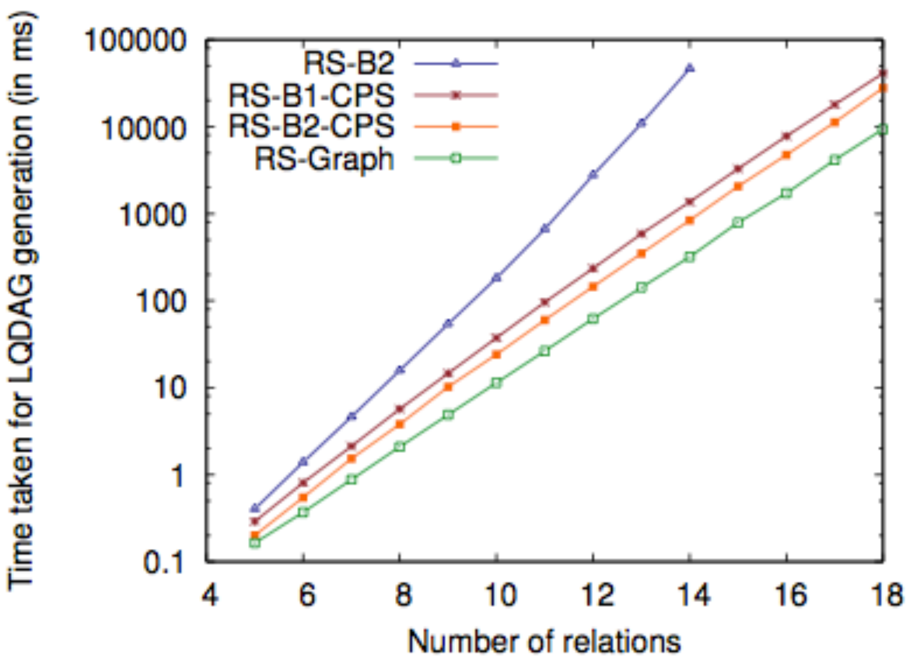
Colour code in graph: RS-B2, RS-B1-CPS, RS-B2-CPS, RS-Graph

RS-Graph significantly outperforms RS-B1-CPS, RS-B2, and even RS-B2-CPS (which is incomplete). (Results on RS-B2-CPS not in paper, added subsequently)

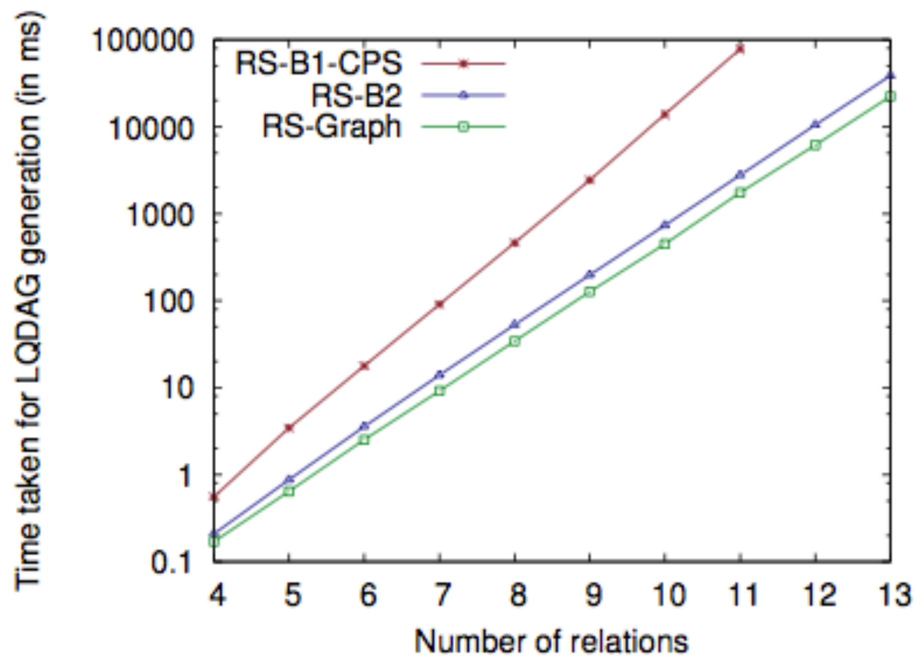
Performance

LQ DAG Expansion time (ms)

Star Queries



Clique Queries



Colour code in graph: RS-B2, RS-B1-CPS, RS-B2-CPS, RS-Graph

Further results with number of equivalence nodes, number of operation nodes, number of operation node addition attempts are in paper

Conclusion

Cross-Product Free Join Order Enumeration in Transformation-based QO is inefficient :

- RS-B1-CPS is complete but generates exponential number of duplicates
- RS-B2-CPS is incomplete
- RS-B2 explores a significantly larger space

We propose a new ruleset RS-Graph which uses join graph partitioning

- It is complete
- It does not generate duplicates
- Performs significantly better than existing rulesets

Thank You

RS-Graph is Complete

Proof consists of two parts:

- An equivalence node stores all the maximal join sets
- Having all the join sets, the RS-Graph rule generates all the join order alternatives below the equivalence node

Part 2 was shown by Pit Fender et. al, given a join set we construct the join graph. The partitioning algorithm generates all S1 $\bowtie$ S2 alternatives possible below this equivalence node.

Part 1 comes from the correctness of the join set maintenance. Interested reader may refer to the paper for this.

Potential Risk

Each equivalence node stores a set of maximal join sets. There may be multiple maximal join sets and hence we might have blow up ?

Good news: For commonly encountered rulesets, this does not happen. Each equivalence node has a single maximal join set.

Consider the example to the right:

The set of maximal join sets of E_0 consists of single entry [({R_1 E_3 R_3}, {t_2 t_0})]