ROX: Run-time Optimization of XQueries

Riham Abdel Kader[1] Peter Boncz[2] Stefan Manegold[2]
Maurice van Keulen[1]

[1]University of Twente

[2]Centrum voor Wiskunde en Informatica (CWI)

SIGMOD 2009

Are Traditional Optimizers Enough?

Complexity in Optimizing XQueries

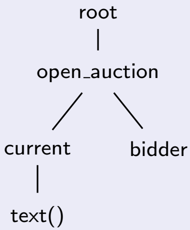
- Large number of joins. In XMark benchmark, number of joins in a query ranges between 5 and 32.
- Existence of correlations

Are Traditional Optimizers Enough?

Complexity in Optimizing XQueries

- Large number of joins. In XMark benchmark, number of joins in a query ranges between 5 and 32.
- Existence of correlations

Correlation in an XMark Document

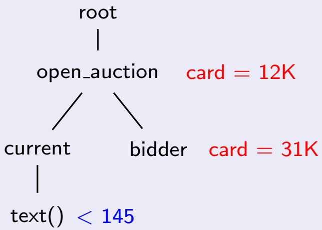


Are Traditional Optimizers Enough?

Complexity in Optimizing XQueries

- Large number of joins. In XMark benchmark, number of joins in a query ranges between 5 and 32.
- Existence of correlations

Correlation in an XMark Document

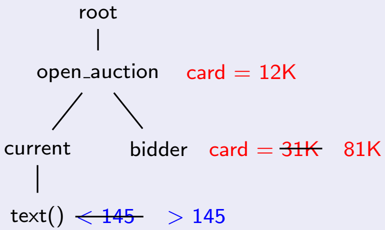


Are Traditional Optimizers Enough?

Complexity in Optimizing XQueries

- Large number of joins. In XMark benchmark, number of joins in a query ranges between 5 and 32.
- Existence of correlations

Correlation in an XMark Document



Traditional Optimizers are not Suitable

Shortcomings of Traditional Optimizers

- Accuracy of traditional estimation techniques degrades when optimizing a large number of joins.
- Traditional optimizers fail to detect correlations between attributes.
- Accurate cardinality and cost estimations in XML is still a challenge.
- Parametric queries are common in XQuery.

Solution

Solution: perform proactive optimization at run-time.

At run-time:

- accurate information about document statistics can be obtained,
- cardinality of intermediate results, selectivity and cost of operators can be accurately estimated,
- correlations can be detected.
- statistics and cost models are not needed.

Challenge: keep resource usage under control and run-time overhead as small as possible.

Adaptive Query Optimization Techniques

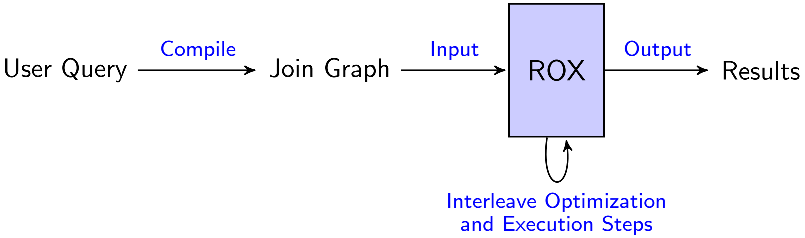
Plan-first execute-next approach, *e.g.*:

1. generate plans optimal for partitions of the data domain (*e.g.* choose plans)
2. reoptimize plans when observed costs differ from estimated ones (*e.g.* Babu et al.)
3. execute plan and feedback observations to optimizer (*e.g.* LEO-IBM)

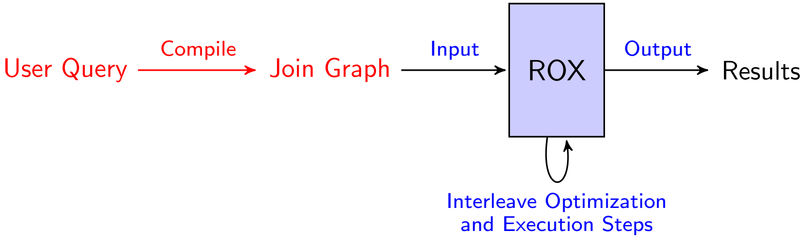
Routing approach: *e.g.* Eddies

Outline

Proposed Approach



Proposed Approach

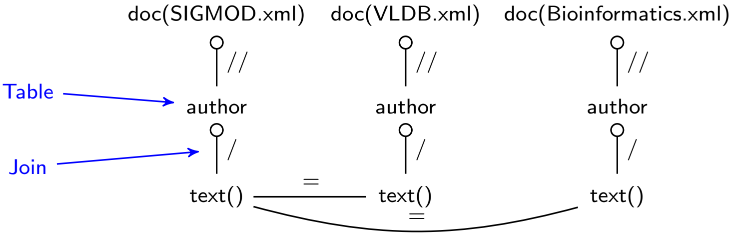


User Query to Join Graph

```
for $a1 in doc("SIGMOD.xml")//author,
    $a2 in doc("VLDB.xml")//author,
    $a3 in doc("Bioinformatics.xml")//author
where $a1/text() = $a2/text() and
      $a1/text() = $a3/text()
return $a1
```

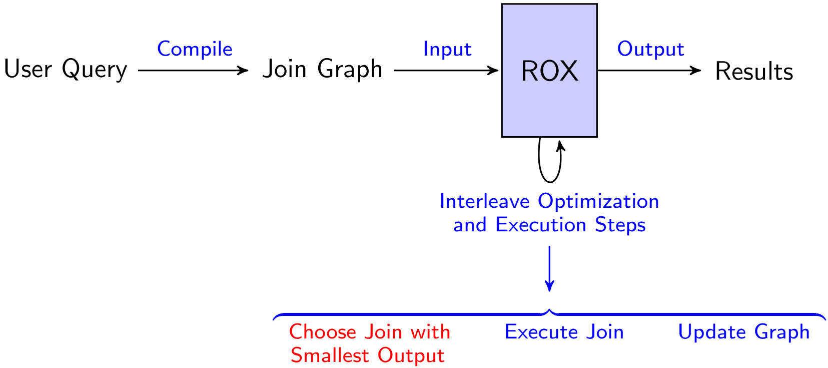
User Query to Join Graph

```
for $a1 in doc("SIGMOD.xml")//author,
    $a2 in doc("VLDB.xml")//author,
    $a3 in doc("Bioinformatics.xml")//author
where $a1/text() = $a2/text() and
      $a1/text() = $a3/text()
return $a1
```

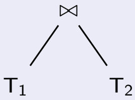


http://www-db.informatik.uni-tuebingen.de/research/pathfinder

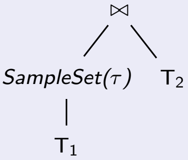
Proposed Approach



Sampling: Estimate Cardinality of a Join



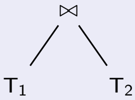
Full Join Execution



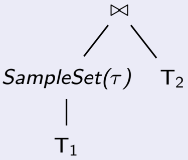
Sampled Join Execution

$$card(T_1 \bowtie T_2) = card(SampleSet(T_1, \tau) \bowtie T_2) \times card(T_1) \div \tau$$

Sampling: Estimate Cardinality of a Join



Full Join Execution



Sampled Join Execution

$$card(T_1 \bowtie T_2) = card(SampleSet(T_1, \tau) \bowtie T_2) \times card(T_1) \div \tau$$

Sampling is applied on operators whose cost is linear to their input and requires no fixed prior investments like sorting.

Chain Sampling: Estimate Cardinality of a Chain of Joins

Query: $a//b//c$

With join sampling, we know that:

card(a//b) = 200

card(b//c) = 100

but what is:

card(a//b//c) = ?

Chain Sampling: Estimate Cardinality of a Chain of Joins

Query: $a//b//c$

With join sampling, we know that:

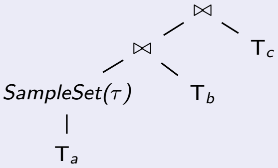
card(a//b) = 200

card(b//c) = 100

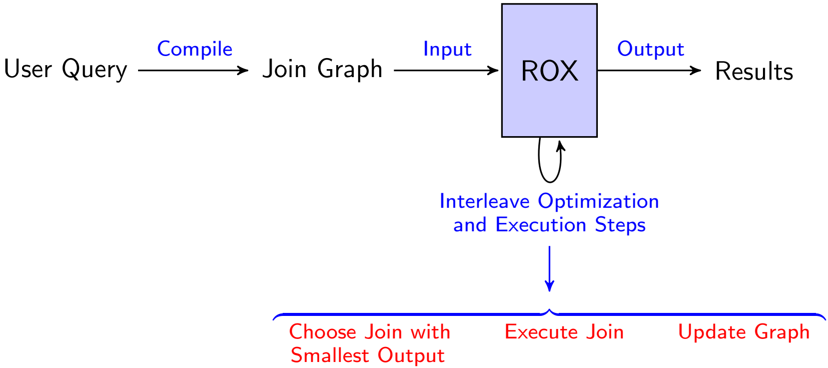
but what is:

card(a//b//c) = ?

Chain Sampling



Proposed Approach



Playing Field

The ROX algorithm is implemented on top of MonetDB/XQuery.

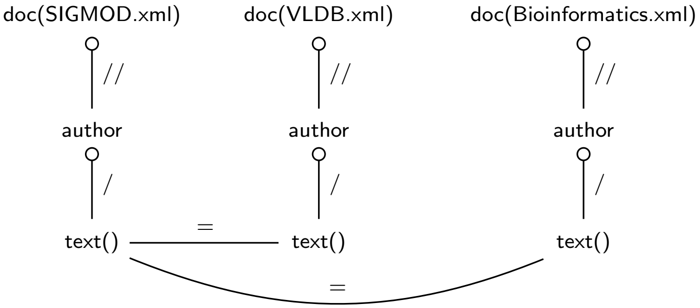
MonetDB provides:

- Efficient intermediate result materialization
- Efficient and cheap sampling techniques
- Fast access to XML nodes in the document using element, text and attribute indices

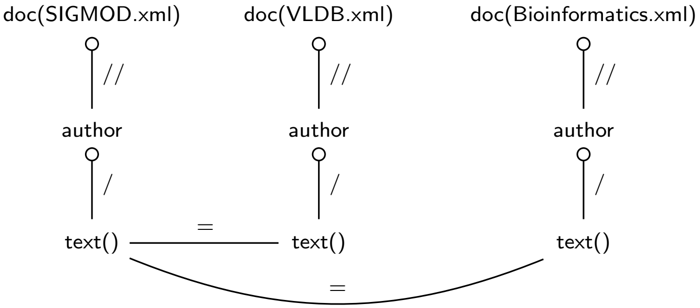
MonetDB

http://monetdb.cwi.nl/

Run-time Optimizer

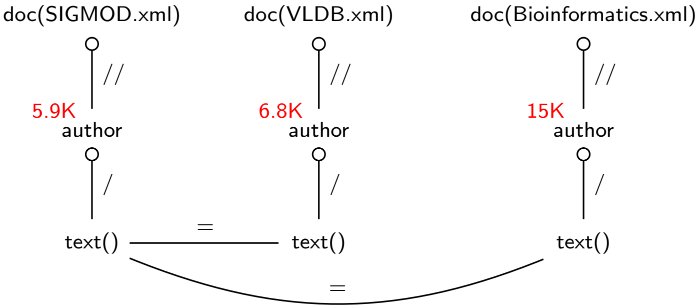


Run-time Optimizer



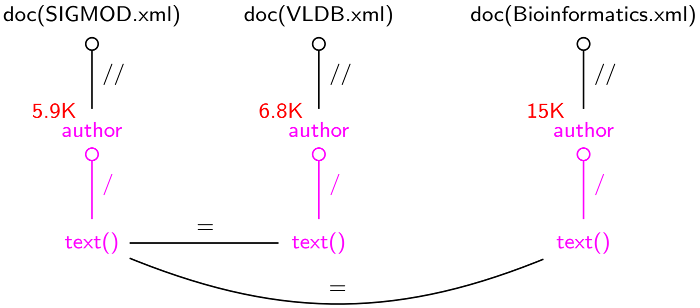
Initialization Phase: Estimate cardinality of nodes, and XPath steps and joins (assign weights to edges)

Run-time Optimizer



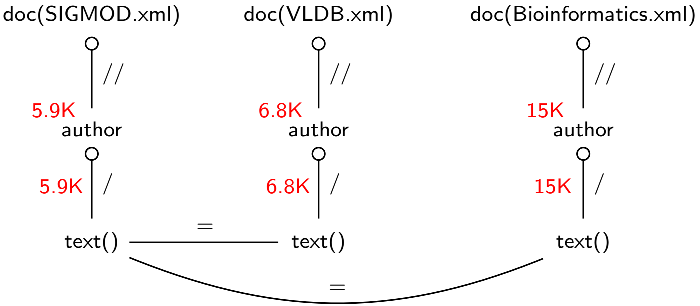
Initialization Phase: Estimate cardinality of nodes, and XPath steps and joins (assign weights to edges)

Run-time Optimizer



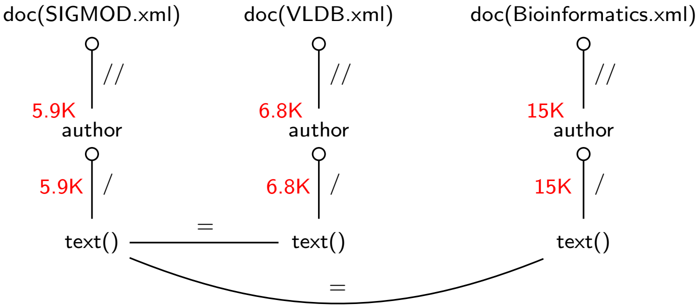
Initialization Phase: Estimate cardinality of nodes, and XPath steps and joins (assign weights to edges)

Run-time Optimizer



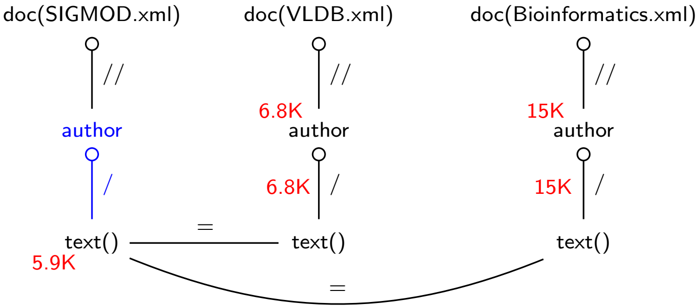
Initialization Phase: Estimate cardinality of nodes, and XPath steps and joins (assign weights to edges)

Run-time Optimizer



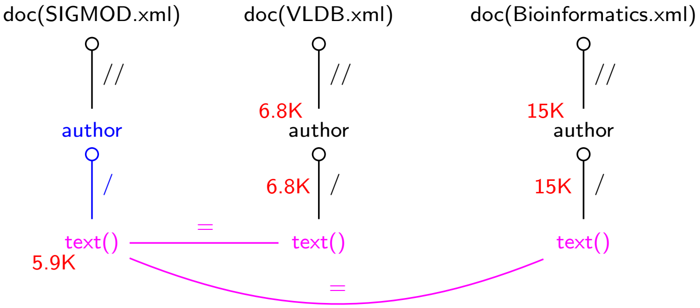
Iterative Phase: Pick edge with smallest weight (join with smallest output) and execute

Run-time Optimizer



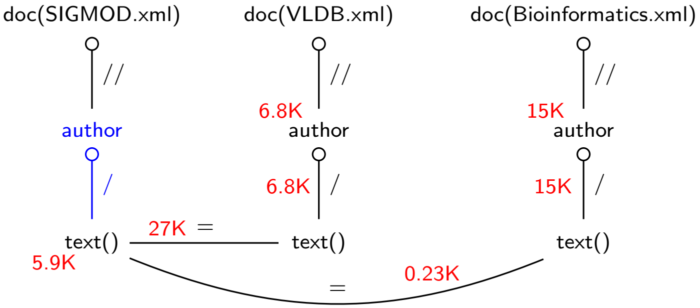
Iterative Phase: Pick edge with smallest weight (join with smallest output) and execute

Run-time Optimizer



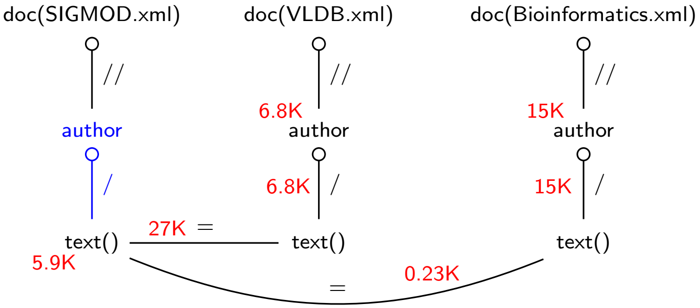
Iterative Phase: Update weights of edges

Run-time Optimizer



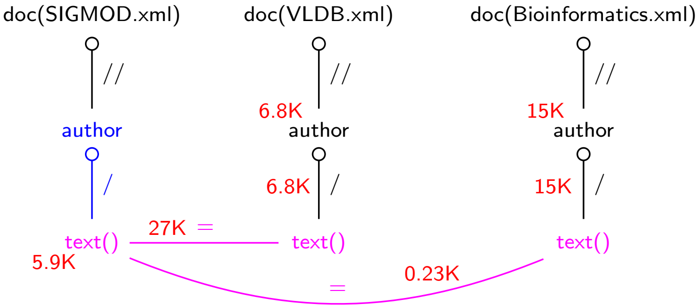
Iterative Phase: Update weights of edges

Run-time Optimizer



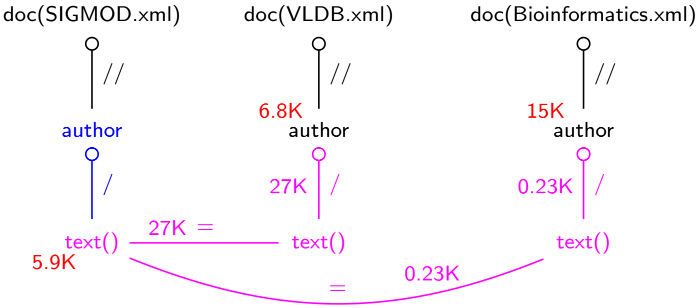
Iterative Phase: Pick edge with smallest weight and execute

Run-time Optimizer



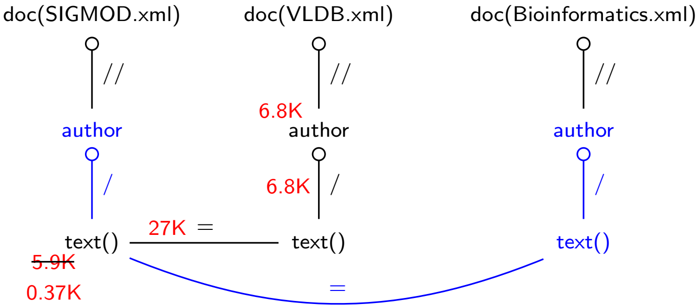
Iterative Phase: Chain sample the branches and execute the most reducing chain

Run-time Optimizer



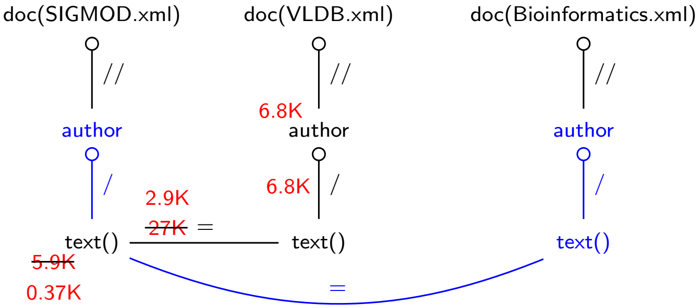
Iterative Phase: Chain sample the branches and execute the most reducing chain

Run-time Optimizer



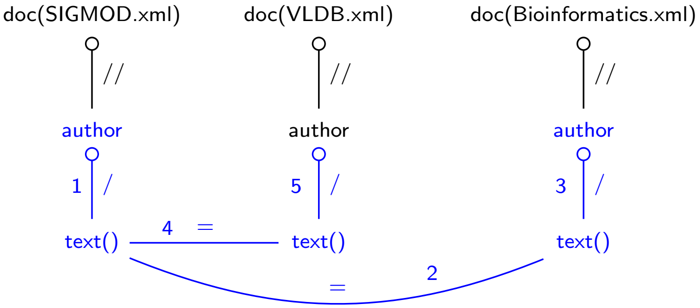
Iterative Phase: Chain sample the branches and execute the most reducing chain

Run-time Optimizer

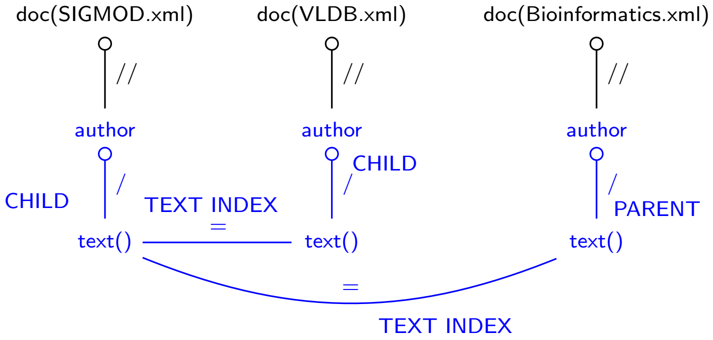


Iterative Phase: Chain sample the branches and execute the most reducing chain

Run-time Optimizer



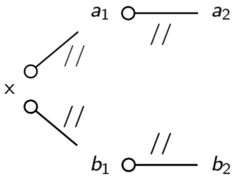
Run-time Optimizer



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

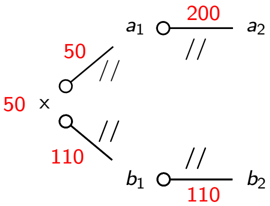
Without Chain Sampling



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

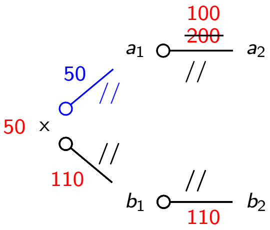
Without Chain Sampling



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

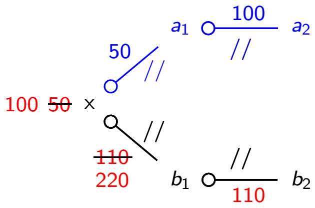
Without Chain Sampling



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

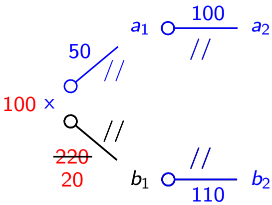
Without Chain Sampling



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

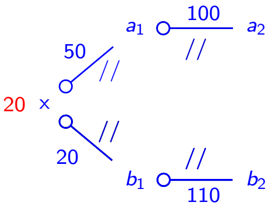
Without Chain Sampling



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

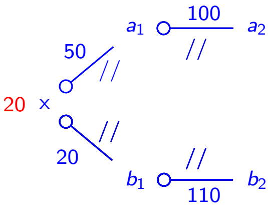
Without Chain Sampling



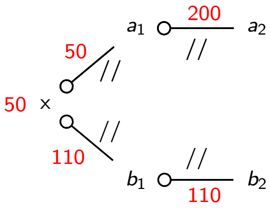
Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

Without Chain Sampling

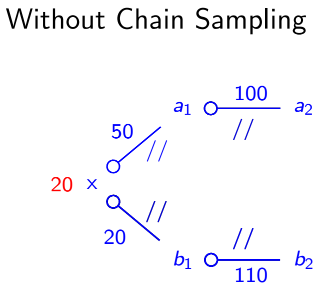


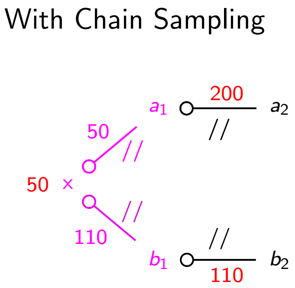
With Chain Sampling



Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

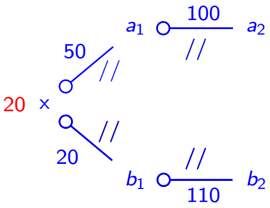




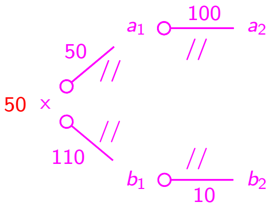
Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

Without Chain Sampling



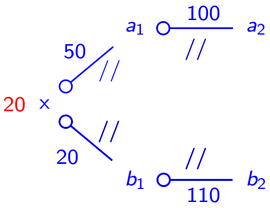
With Chain Sampling



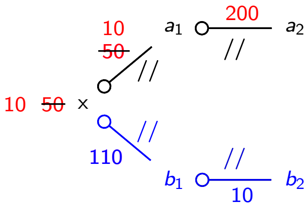
Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

Without Chain Sampling



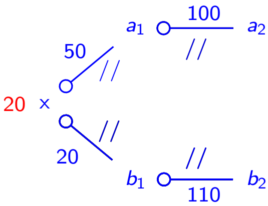
With Chain Sampling



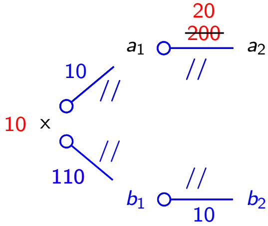
Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

Without Chain Sampling



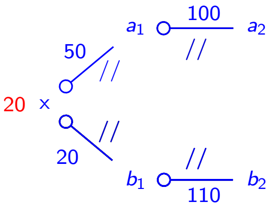
With Chain Sampling



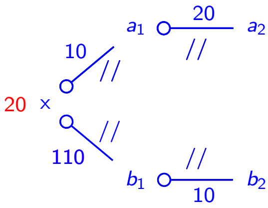
Chain Sampling: Avoiding Local Optimum

Query: $x[.//a_1//a_2]//b_1//b_2$

Without Chain Sampling



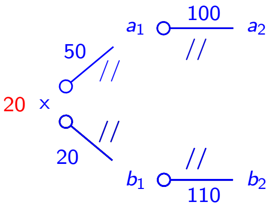
With Chain Sampling



Chain Sampling: Avoiding Local Optimum

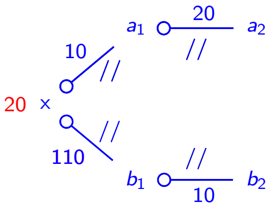
Query: $x[.//a_1//a_2]//b_1//b_2$

Without Chain Sampling



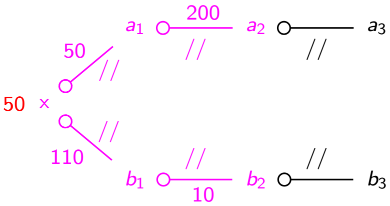
$cost = 50 + 100 + 20 + 110 = 280$

With Chain Sampling



$cost = 10 + 20 + 110 + 10 = 150$

Chain Sampling: Stopping Condition

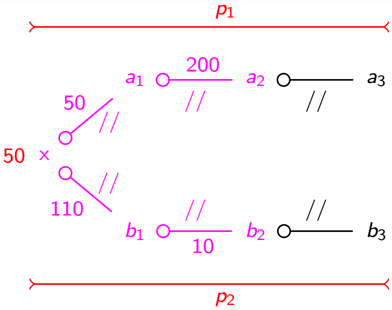


Stopping Condition

Execute path p_i if:

$cost(p_i) + cost(p_j|p_i) < cost(p_j)$

Chain Sampling: Stopping Condition



Stopping Condition

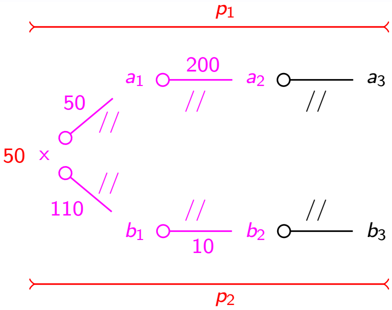
Execute path p_i if:

$cost(p_i) + cost(p_j | p_i) < cost(p_j)$

Example:

$$\left.\begin{array}{l} cost(p_1) = 250 \\ cost(p_2) = 120 \\ cost(p_1|p_2) = 50 \end{array}\quad + \right] = 170$$

Chain Sampling: Stopping Condition



Stopping Condition

Execute path p_i if:

$cost(p_i) + cost(p_j|p_i) < cost(p_j)$

Example:

$cost(p_1) = 250$
$cost(p_2) = 120$
$cost(p_1|p_2) = 50$ $+$ $= 170$

Scale Factor

$cost(p_1|p_2) = cost(p_1) * sf(p_2)$

$sf(p_2) = card(x|p_2) \div card(x) = 10 \div 50 = 0.2$

Outline

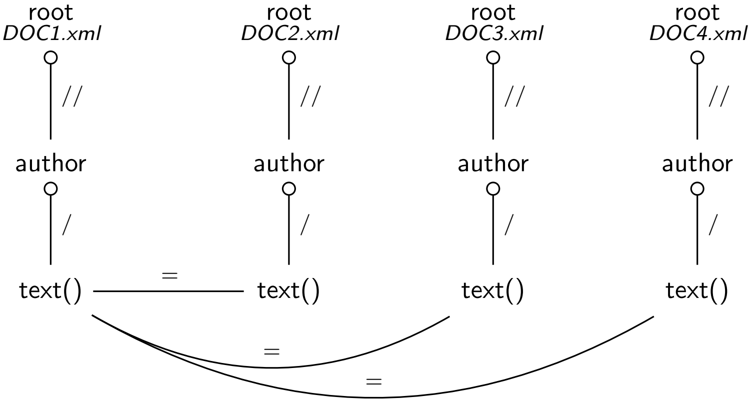
Experiments

Used XQuery

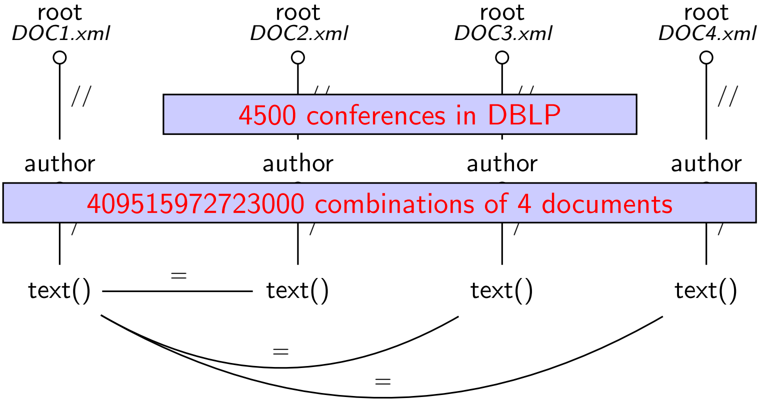
4-way join query, matching authors from 4 documents.

```
for $a1 in doc("DOC1.xml")//author,
    $a2 in doc("DOC2.xml")//author,
    $a3 in doc("DOC3.xml")//author,
    $a4 in doc("DOC4.xml")//author
where $a1/text() = $a2/text() and
      $a1/text() = $a3/text() and
      $a1/text() = $a4/text()
return $a1
```

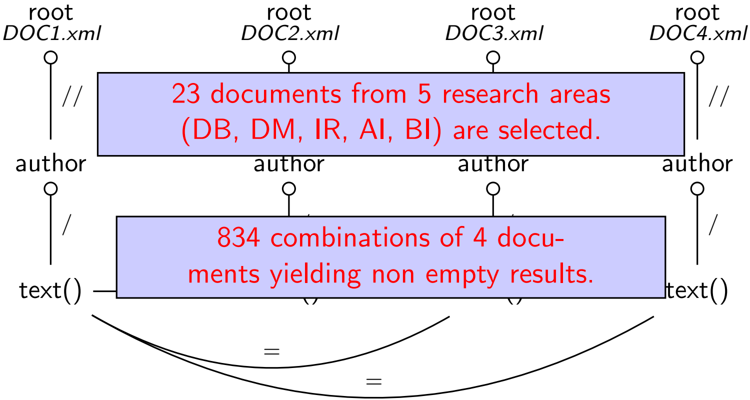
Experiments Setup



Experiments Setup



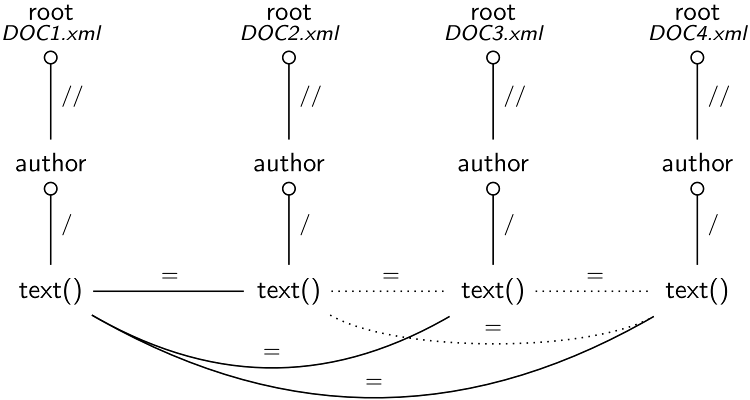
Experiments Setup



23 documents from 5 research areas (DB, DM, IR, AI, BI) are selected.

834 combinations of 4 documents yielding non empty results.

Experiments Setup



Experiments Setup: Query Plans

Plans Generated by Classical Compile Time Optimizer

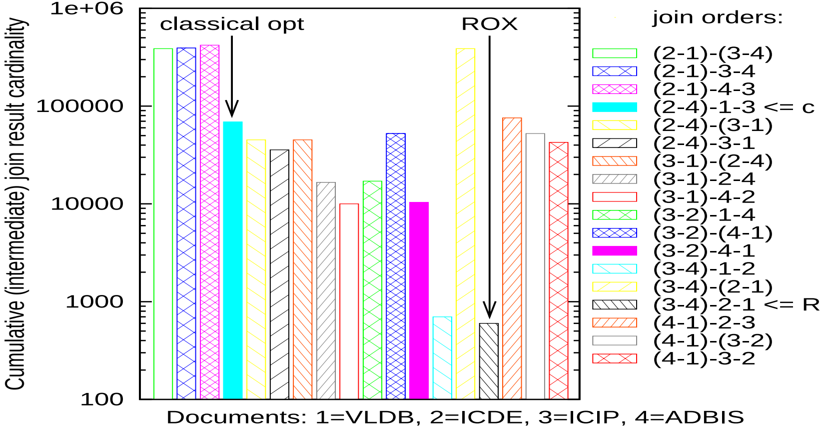
Equi-joins are ordered based on a “smallest-input-first” heuristic.

All Physical Plans in the Search Space

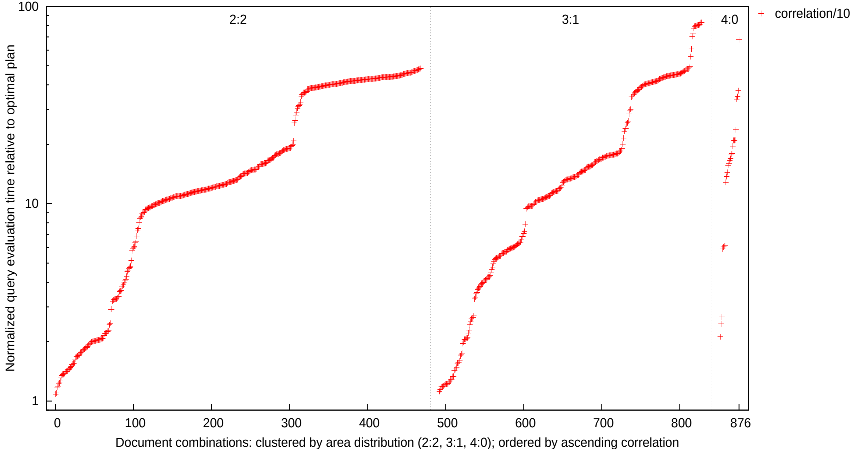
Total number of plans is 88800.

Plans are categorized based on the equi-join order: 18 categories.

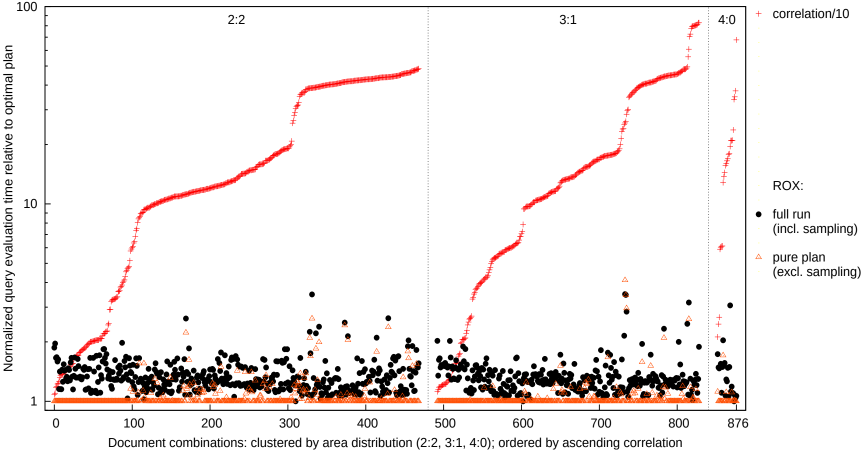
Impact of Join order on intermediate result sizes



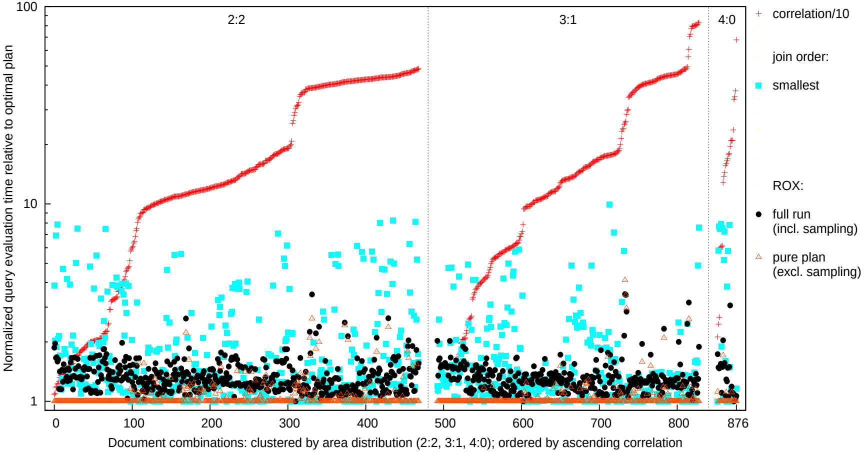
Execution Times



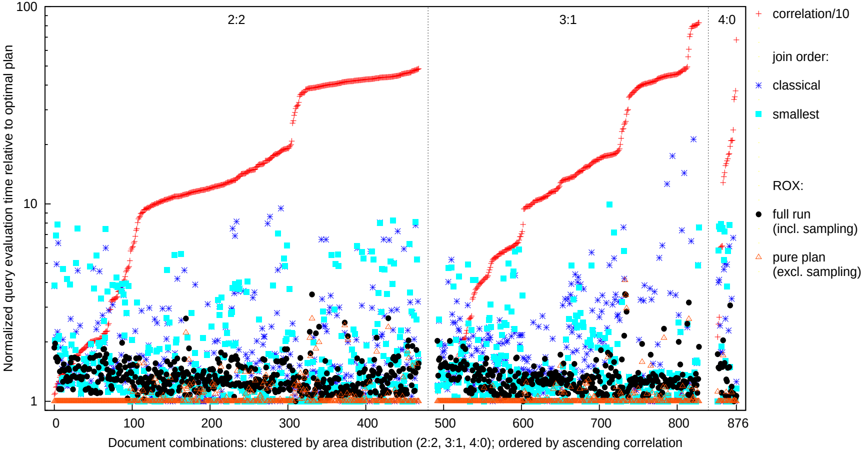
Execution Times



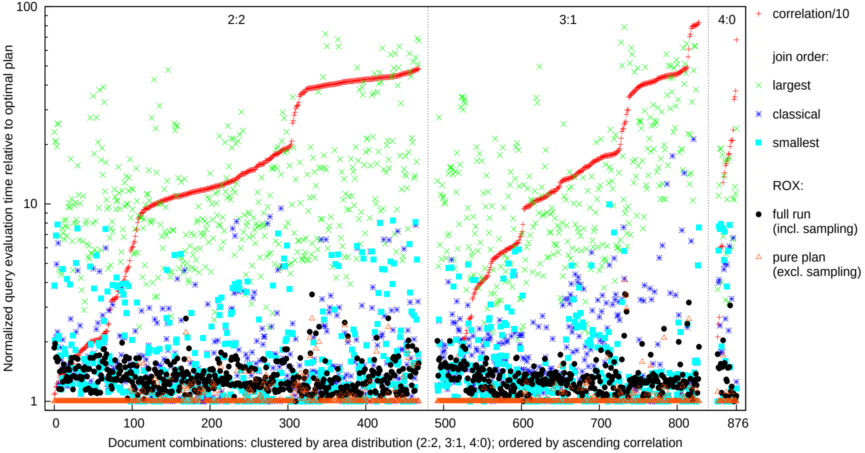
Execution Times



Execution Times



Execution Times



Outline

1. Motivation
2. Run-time Optimizer
3. Experiments
4. **Conclusion and Future Work**

Conclusion

We have developed ROX: a run-time optimizer for XQueries which interleaves optimization and execution steps.

ROX does not depend on a cost model, and can detect correlations.

Experiments have shown that ROX chooses good execution plans while keeping the sampling overhead low.

XPath steps and joins are optimized indifferently.

Future Work

Balance dynamically between the cost of sampling and the plan's execution cost.

Take the execution cost of operators into consideration while computing the weight of edges.

Generalize the approach to systems with pipelined execution.

Generalize the approach to SPARQL and SQL.