ROX: Run-time Optimization of XQueries

Riham Abdel Kader\textsuperscript{1}  Peter Boncz\textsuperscript{2}  Stefan Manegold\textsuperscript{2}  Maurice van Keulen\textsuperscript{1}

\textsuperscript{1}University of Twente

\textsuperscript{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

```
root
  |
  open_auction
    |
    current
      |
      bidder
        |
        text()
```
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

```
root
  open_auction  card = 12K
    current
    bidder      card = 31K
      text() < 145
```
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

```
root
    /
  open_auction  card = 12K
     /
current     bidder  card = 31K  81K
         /
  text()  < 145  > 145
```
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: 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
1. Motivation

2. Run-time Optimizer

3. Experiments

4. Conclusion and Future Work

Outline

ROX: Run-time Optimization of XQueries
Proposed Approach

User Query → Compile → Join Graph → ROX → Output

Input → Results

Interleave Optimization and Execution Steps
Proposed Approach

User Query $\rightarrow$ Compile $\rightarrow$ Join Graph $\rightarrow$ ROX

Input $\rightarrow$ Output $\rightarrow$ Results

Interleave Optimization and Execution Steps
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

User Query → Compile → Join Graph → ROX → Output

Input

Interleave Optimization and Execution Steps

Choose Join with Smallest Output → Execute Join → Update Graph

Results
Sampling: Estimate Cardinality of a Join

Motivation

Run-time Optimizer

Experiments

Conclusion and Future Work

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

\[ \text{card}(T_1 \bowtie T_2) = \text{card}(\text{SampleSet}(T_1, \tau) \bowtie T_2) \times \text{card}(T_1) \div \tau \]
Sampling: Estimate Cardinality of a Join

\[ \text{card}(T_1 \bowtie T_2) = \text{card}(\text{SampleSet}(T_1, \tau) \bowtie T_2) \times \text{card}(T_1) \div \tau \]

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:
- $\text{card}(a // b) = 200$
- $\text{card}(b // c) = 100$

but what is:
- $\text{card}(a // b // c) = ?$
Chain Sampling: Estimate Cardinality of a Chain of Joins

Query: \( a//b//c \)

With join sampling, we know that:

\[
\text{card}(a//b) = 200 \\
\text{card}(b//c) = 100 \\
\]

but what is:

\[
\text{card}(a//b//c) = ?
\]
Proposed Approach

User Query → Compile → Join Graph → ROX

Input → Output

Interleave Optimization and Execution Steps

Choose Join with Smallest Output → Execute Join → Update Graph

Results

ROX: Run-time Optimization of XQueries
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

http://monetdb.cwi.nl/
Run-time Optimizer

doc(SIGMOD.xml)  doc(VLDB.xml)  doc(Bioinformatics.xml)

//

author

//

author

//

author

//

text() = text()

CHILD

TEXT INDEX

CHILD

PARENT

TEXT INDEX

ROX: Run-time Optimization of XQueries
Run-time Optimizer

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

ROX: Run-time Optimization of XQueries
Run-time Optimizer

Iterative Phase: Update weights of edges
Run-time Optimizer

Iterative Phase: Update weights of edges
Iterative Phase: Pick edge with smallest weight and execute
Iterative Phase: Chain sample the branches and execute the most reducing chain
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
Iterative Phase: Chain sample the branches and execute the most reducing chain
Run-time Optimizer

doc(SIGMOD.xml)  doc(VLDB.xml)  doc(Bioinformatics.xml)

//
author
1 /

//
author
5 /

//
author
3 /

text() = 4

2

text() = 4

ROX: Run-time Optimization of XQueries
**Motivation**

**Run-time Optimizer**

**Experiments**

**Conclusion and Future Work**

---

**Chain Sampling: Avoiding Local Optimum**

Query: \( x[./a_1//a_2]//b_1//b_2 \)

Without Chain Sampling

![Diagram of query execution without chain sampling]

\[
\text{cost} = 50 + 100 + 20 + 110 = 280
\]

With Chain Sampling

![Diagram of query execution with chain sampling]

\[
\text{cost} = 10 + 20 + 110 + 10 = 150
\]

---

**ROX: Run-time Optimization of XQueries**
Chain Sampling: Avoiding Local Optimum

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

Without Chain Sampling

With Chain Sampling

**Without Chain Sampling**

**With Chain Sampling**
**Chain Sampling: Avoiding Local Optimum**

Query: \( x[.//a_1//a_2]///b_1///b_2 \)

Without Chain Sampling

\[
\begin{array}{c}
50 \\
50 \\
110
\end{array}
\quad 50 \\
\quad 100 \\
\quad 100 \\
\quad 200 \\
\quad 110 \\
\quad 220
\]

\[
\text{cost} = 50 + 100 + 20 + 110 = 280
\]

With Chain Sampling

\[
\begin{array}{c}
10 \\
10 \\
10 \\
110
\end{array}
\quad 110
\quad 200
\quad 20
\quad 10
\quad 10
\quad 50
\quad 50
\quad 50
\quad 10
\quad 10
\quad 20
\quad 110
\quad 10
\quad 50
\]

\[
\text{cost} = 10 + 20 + 110 + 10 = 150
\]
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

\( \text{cost} = 10 + 20 + 110 + 10 = 150 \)
**Chain Sampling: Avoiding Local Optimum**

Query: \( x[.//a_1//a_2]//b_1//b_2 \)

Without Chain Sampling

Without Chain Sampling, the cost is calculated as:

\[ \text{cost} = 50 + 100 + 20 + 110 = 280 \]

With Chain Sampling

With Chain Sampling, the cost is calculated as:

\[ \text{cost} = 10 + 20 + 110 + 10 = 150 \]
Chain Sampling: Avoiding Local Optimum

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

Without Chain Sampling

With Chain Sampling

ROX: Run-time Optimization of XQueries
Chain Sampling: Avoiding Local Optimum

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

Without Chain Sampling

With Chain Sampling

ROX: Run-time Optimization of XQueries
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**

Without chain sampling, the cost is calculated as:

\[\text{cost} = 50 + 100 + 20 + 110 = 280\]

**With Chain Sampling**

With chain sampling, the cost is calculated as:

\[\text{cost} = 10 + 20 + 110 + 10 = 150\]
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**

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

**cost** = 10 + 20 + 110 + 10 = 150

ROX: Run-time Optimization of XQueries
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:

\[
\text{cost}(p_i) + \text{cost}(p_j|p_i) < \text{cost}(p_j)
\]

Example:

\[
\begin{align*}
\text{cost}(p_1) &= 250 \\
\text{cost}(p_2) &= 120 \\
\text{cost}(p_1|p_2) &= 50 \\
\end{align*}
\]

\[+\]

\[= 170\]
Chain Sampling: Stopping Condition

Stopping Condition

Execute path $p_i$ if:

$$\text{cost}(p_i) + \text{cost}(p_j|p_i) < \text{cost}(p_j)$$

Example:

- $\text{cost}(p_1) = 250$
- $\text{cost}(p_2) = 120$
- $\text{cost}(p_1|p_2) = 50$

$$\text{cost}(p_1|p_2) = 50 + 170 = 170$$

Scale Factor

$$\text{cost}(p_1|p_2) = \text{cost}(p_1) \times sf(p_2)$$

$$sf(p_2) = \frac{\text{card}(x|p_2)}{\text{card}(x)} = \frac{10}{50} = 0.2$$
Outline

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

**Used XQuery**

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

```xquery
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
```

ROX: Run-time Optimization of XQueries
Experiments Setup

ROX: Run-time Optimization of XQueries
Experiments Setup

ROX: Run-time Optimization of XQueries

Motivation | Run-time Optimizer | Experiments | Conclusion and Future Work

Experiments Setup

- Root: DOC1.xml
- Root: DOC2.xml
- Root: DOC3.xml
- Root: DOC4.xml

4500 conferences in DBLP

409515972723000 combinations of 4 documents
23 documents from 5 research areas (DB, DM, IR, AI, BI) are selected.

834 combinations of 4 documents yielding non empty results.
Experiments Setup

ROX: Run-time Optimization of XQueries
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

Motivation
Run-time Optimizer
Experiments
Conclusion and Future Work

impact of Join order on intermediate result sizes

classical opt

Cumulative (intermediate) join result cardinality

Documents: 1=VLDB, 2=ICDE, 3=ICIP, 4=ADBIS

join orders:
(2-1)-(3-4)
(2-1)-3-4
(2-1)-4-3
(2-4)-1-3 <= c
(2-4)-(3-1)
(2-4)-3-1
(2-4)-3-2
(3-1)-2-4
(3-1)-4-2
(3-2)-1-4
(3-2)-(4-1)
(3-2)-4-1
(3-4)-1-2
(3-4)-(2-1)
(3-4)-2-1 <= R
(4-1)-2-3
(4-1)-(3-2)
(4-1)-3-2

ROX: Run-time Optimization of XQueries
Execution Times

Document combinations: clustered by area distribution (2:2, 3:1, 4:0); ordered by ascending correlation

Normalized query evaluation time relative to optimal plan

ROX: Run-time Optimization of XQueries
Execution Times

Document combinations: clustered by area distribution (2:2, 3:1, 4:0); ordered by ascending correlation

ROX: Run-time Optimization of XQueries

- full run (incl. sampling)
- pure plan (excl. sampling)

Normalized query evaluation time relative to optimal plan

2:2 3:1 4:0

correlation/10

ROX:

full run (incl. sampling)
pure plan (excl. sampling)
Execution Times

Document combinations: clustered by area distribution (2:2, 3:1, 4:0); ordered by ascending correlation

Join order:
- smallest

ROX:
- full run (incl. sampling)
- pure plan (excl. sampling)

+ correlation/10
Execution Times

ROX: Run-time Optimization of XQueries
Outline

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

ROX: Run-time Optimization of XQueries
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.