From relation algebra to semi-join algebra: an approach for graph query optimization

Jelle Hellings¹
Catherine L. Pilachowski²  Dirk Van Gucht²
Marc Gyssens¹  Yuqing Wu³

¹ Hasselt University
² Indiana University
³ Pomona College
Graph queries: data model
Graph queries: basic path queries

(WorksWith \cup \text{FriendOf}) \circ [\text{ParentOf}]^+ \circ \text{FriendOf}
Graph queries: basic path queries

\[
(WorksWith \cup \text{FriendOf}) \circ [\text{ParentOf}]^+ \circ \text{FriendOf}
\]
Graph queries: node-tests and branching

\[ \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf} \]
Graph queries: node-tests and branching

\[ \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf} \]
Graph querying: relation algebra

<table>
<thead>
<tr>
<th>id</th>
<th>$\cup$</th>
<th>$\circ$</th>
<th>$+$</th>
<th>$\cap$</th>
<th>$\pi$</th>
<th>$\bar{\pi}$</th>
<th>$\cap$</th>
<th>$\neg$</th>
<th>di</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPQs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2RPQs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested RPQs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigational XPath, Graph XPath</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FO[3] + transitive closure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Relation algebra and query evaluation

Cheap \((\cup, \cap, \pi, \cap, \neg)\).
Cost linearly upper bounded by operands

In between \((id, \pi)\).
Cost linearly upper bounded by \#nodes

Expensive \((\circ, +, \text{di})\).
Worst-case quadratically lower bounded by \#nodes
Naive query evaluation: an inefficient example

Return pairs of (great-grandparent, friend)

\[ \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf} \]

1. Compute (grandparent, grandchild):
   \[ X = \text{ParentOf} \circ \text{ParentOf} \]

2. Compute (great-grandparent, great-grandchild):
   \[ Y = \text{ParentOf} \circ X \]

3. Throw away the great-grandchildren:
   \[ Z = \pi_1[Y] \]

4. Compute (great-grandparent, friend):
   \[ \text{Result} = Z \circ \text{FriendOf} \]
Optimize query evaluation: add specialized operators?

Return pairs of (great-grandparent, friend)

\[ \pi_1[\text{ParentOf} \circ \text{ParentOf} \circ \text{ParentOf}] \circ \text{FriendOf} \]

1. Compute (grandparent, ???):
   \[ X = \text{ParentOf} \Join \text{ParentOf} \]

2. Compute (great-grandparent, ???):
   \[ Y = \text{ParentOf} \Join (X) \]

3. Throw away ???:
   \[ Z = \pi_1[Y] \]

4. Compute (great-grandparent, friend):
   \[ \text{Result} = Z \Join \text{FriendOf} \]

\[ \pi_1[\text{ParentOf} \Join (\text{ParentOf} \Join \text{ParentOf})] \Join \text{FriendOf} \]
Simple idea: automatic query rewriting

- Rewrite composition into semi-joins
- Rewrite transitive closure into fixpoints

\textit{In such a way that the rewritten query is equivalent}
When are expressions equivalent?

Definition
Queries \( q_1 \) and \( q_2 \) are

path-equivalent if, for every graph \( G \), \( [q_1]_G = [q_2]_G \)
(denoted by \( q_1 \equiv_{\text{path}} q_2 \))

left-projection-equivalent if, for every graph \( G \), \( [q_1]_G|_1 = [q_2]_G|_1 \)
(denoted by \( q_1 \equiv_{\pi_1} q_2 \))

right-projection-equivalent if, for every graph \( G \), \( [q_1]_G|_2 = [q_2]_G|_2 \)
(denoted by \( q_1 \equiv_{\pi_2} q_2 \))

Example

- \( R \cap S \equiv_{\text{path}} R - (R - S) \)
- \( R \circ S \equiv_{\pi_1} R \times S \)
- \( \pi_1[R \circ S] \equiv_{\text{path}} \pi_1[R \times S] \)
The main result

Collapse also holds for fragments (that include $\pi$)

Example: Nested RPQs are projection-equivalent to expressions using only id, $\cup$, $\times$, $\times$, fp, $\bowtie$, and $\pi$
Intersection $\cap$ and difference $-$

Issues when combining composition with $\cap$ or $-$

$$(\text{FriendOf} \circ \text{FriendOf}) \cap \text{FriendOf}$$

- **Restricting**: use $\cap$ and $-$ only on composition-free expressions
  - Exact syntactic fragment of $\text{FO}[3] + TC$ that is projection-equivalent to $\text{FO}[2] + \text{fixpoint}$.

- **Data models**: usage of $\cap$ and $-$ is sometimes redundant
  - Sibling-ordered trees: $\text{FO}^{\text{tree}} \preceq_{\pi} \text{FO}[2] + \text{fixpoints}$.
  - Downward queries on trees [DBPL 2015]
  - ...

- **Partial rewriting**: keep compositions when necessary
The rewrite functions - partial rewriting

\[ \tau(e) \equiv_{\text{path}} e \quad \tau_{\pi_1}(e) \equiv_{\pi_1} e \quad \tau_{\pi_2}(e) \equiv_{\pi_2} e \]

\[ \tau_{\circ_1}(e; \varepsilon) \equiv_{\pi_1} e \times \varepsilon \quad \tau_{\circ_2}(e; \varepsilon) \equiv_{\pi_2} \varepsilon \times e \]

Example

\[ \pi_1[(\text{WorksOn} \circ \text{WorksOn}^\sim) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]
The rewrite functions - partial rewriting

\[ \tau(e) \equiv \text{path } e \]
\[ \tau_{\pi_1}(e) \equiv \pi_1 e \]
\[ \tau_{\pi_2}(e) \equiv \pi_2 e \]
\[ \tau_{\circ_1}(e; \varepsilon) \equiv \pi_1 e \times \varepsilon \]
\[ \tau_{\circ_2}(e; \varepsilon) \equiv \pi_2 \varepsilon \times e \]

Example

\[ \pi_1[((\text{WorksOn } \circ \text{WorksOn}^\perp) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]

\[ \tau(e) \]
The rewrite functions - partial rewriting

\[ \tau(e) \equiv \text{path } e \quad \tau_{\pi_1}(e) \equiv \pi_1 e \quad \tau_{\pi_2}(e) \equiv \pi_2 e \]

\[ \tau_{\circ_1}(e; \varepsilon) \equiv \pi_1 e \times \varepsilon \quad \tau_{\circ_2}(e; \varepsilon) \equiv \pi_2 \varepsilon \times e \]

Example

\[ \pi_1[((WorksOn \circ WorksOn^\circ) \cap FriendOf) \circ EditorOf] \circ StudentOf \]

\[ \tau(e) = \tau_{\pi_2}(\pi_1[((W \circ W^\circ) \cap F) \circ E]) \times \tau(S) \]
The rewrite functions - partial rewriting

\[ \tau(e) \equiv \text{path } e \quad \tau_{\pi_1}(e) \equiv \pi_1 e \quad \tau_{\pi_2}(e) \equiv \pi_2 e \]

\[ \tau_{\circ_1}(e; \varepsilon) \equiv \pi_1 e \times \varepsilon \quad \tau_{\circ_2}(e; \varepsilon) \equiv \pi_2 \varepsilon \times e \]

Example

\[ \pi_1[(((\text{WorksOn} \circ \text{WorksOn}^\land) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]

\[ \tau(e) = \tau_{\pi_2}(\pi_1[(((W \circ W^\land) \cap F) \circ E)] \times \tau(S) \]
\[ = \pi_1[\tau_{\pi_1}(((W \circ W^\land) \cap F) \circ E)] \times S \]
The rewrite functions - partial rewriting

\[ \tau(e) \equiv \text{path } e \quad \tau_{\pi_1}(e) \equiv \pi_1 e \quad \tau_{\pi_2}(e) \equiv \pi_2 e \]

\[ \tau_{\circ_1}(e; \varepsilon) \equiv \pi_1 e \rtimes \varepsilon \quad \tau_{\circ_2}(e; \varepsilon) \equiv \pi_2 \varepsilon \rtimes e \]

Example

\[ \pi_1[((\text{WorksOn} \circ \text{WorksOn}^\triangle) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]

\[ \tau(e) = \tau_{\pi_2}(\pi_1[((W \circ W^\triangle) \cap F) \circ E]) \rtimes \tau(S) \]

\[ = \pi_1[\tau_{\pi_1}(((W \circ W^\triangle) \cap F) \circ E)] \rtimes S \]

\[ = \pi_1[\tau_{\circ_1}(((W \circ W^\triangle) \cap F; \tau_{\pi_1}(E)))] \rtimes S \]
The rewrite functions - partial rewriting

\[ \tau(e) \equiv_{\text{path}} e \quad \tau_{\pi_1}(e) \equiv_{\pi_1} e \quad \tau_{\pi_2}(e) \equiv_{\pi_2} e \]

\[ \tau_{\circ_1}(e; \varepsilon) \equiv_{\pi_1} e \times \varepsilon \quad \tau_{\circ_2}(e; \varepsilon) \equiv_{\pi_2} \varepsilon \times e \]

Example

\[ \pi_1[((\text{WorksOn} \circ \text{WorksOn}^\cap) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]

\[ \tau(e) = \tau_{\pi_2}(\pi_1[((W \circ W^\cap) \cap F) \circ E]) \times \tau(S) \]
\[ = \pi_1[\tau_{\pi_1}(((W \circ W^\cap) \cap F) \circ E)] \times S \]
\[ = \pi_1[\tau_{\circ_1}(((W \circ W^\cap) \cap F; \tau_{\pi_1}(E)))] \times S \]
\[ = \pi_1[\tau(W \circ W^\cap) \cap \tau(F)) \times E] \times S \]
The rewrite functions - partial rewriting

\[ \tau(e) \equiv_{\text{path}} e \]
\[ \tau_{\pi_1}(e) \equiv_{\pi_1} e \]
\[ \tau_{\pi_2}(e) \equiv_{\pi_2} e \]
\[ \tau_{o_1}(e; \varepsilon) \equiv_{\pi_1} e \times \varepsilon \]
\[ \tau_{o_2}(e; \varepsilon) \equiv_{\pi_2} \varepsilon \times e \]

Example

\[ \pi_1[((\text{WorksOn} \circ \text{WorksOn}^\circ) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]

\[ \tau(e) = \tau_{\pi_2}(\pi_1[((W \circ W^\circ) \cap F) \circ E]) \times \tau(S) \]
\[ = \pi_1[\tau_{\pi_1}(((W \circ W^\circ) \cap F) \circ E)] \times S \]
\[ = \pi_1[\tau_{o_1}(((W \circ W^\circ) \cap F; \tau_{\pi_1}(E))] \times S \]
\[ = \pi_1[(((\tau(W \circ W^\circ) \cap \tau(F)) \times E] \times S \]
\[ = \pi_1[(((\tau(W) \circ \tau(W^\circ)) \cap F) \times E] \times S \].
The rewrite functions - partial rewriting

\[ \tau(e) \equiv_{\text{path}} e \quad \tau_{\pi_1}(e) \equiv_{\pi_1} e \quad \tau_{\pi_2}(e) \equiv_{\pi_2} e \]

\[ \tau_{\circ_1}(e; \varepsilon) \equiv_{\pi_1} e \times \varepsilon \quad \tau_{\circ_2}(e; \varepsilon) \equiv_{\pi_2} \varepsilon \times e \]

Example

\[ \pi_1[((\text{WorksOn} \circ \text{WorksOn}^\cap) \cap \text{FriendOf}) \circ \text{EditorOf}] \circ \text{StudentOf} \]

\[ \tau(e) = \tau_{\pi_2}(\pi_1[(((W \circ W^\cap) \cap F) \circ E)] \times \tau(S) \]
\[ = \pi_1[\tau_{\pi_1}(((W \circ W^\cap) \cap F) \circ E)] \times S \]
\[ = \pi_1[\tau_{\circ_1}(((W \circ W^\cap) \cap F; \tau_{\pi_1}(E))) \times S \]
\[ = \pi_1[[\tau(W \circ W^\cap) \cap \tau(F)) \times E] \times S \]
\[ = \pi_1[((\tau(W) \circ \tau(W^\cap)) \cap F) \times E] \times S \]
\[ = \pi_1[(((W \circ W^\cap) \cap F) \times E] \times S. \]
Query optimization

- Cost of each operator
- Input size of each operator

- Number of necessary evaluation steps
Query optimization

- Cost of each operator ✓
- Input size of each operator

- Number of necessary evaluation steps
Query optimization

- Cost of each operator ✓
- Input size of each operator

Example
Let $R = \{(1, i) \mid 0 \leq i \leq m\}$. Consider

$$R \circ R^\perp \equiv_{\pi_1} R \times R^\perp.$$  

- Number of necessary evaluation steps
Query optimization

- Cost of each operator ✓
- Input size of each operator ✓

Example

Let \( R = \{(1, i) \mid 0 \leq i \leq m\} \). Consider

\[ R \circ R^\ominus \equiv_{\pi_1} R \times R^\ominus. \]

Solution: use single-column evaluation algorithms

- Number of necessary evaluation steps
Query optimization

- Cost of each operator ✓
- Input size of each operator ✓

Example
Let \( R = \{ (1, i) \mid 0 \leq i \leq m \} \). Consider

\[
R \circ R^\perp \equiv_{\pi_1} R \times R^\perp.
\]

Solution: use single-column evaluation algorithms

- Number of necessary evaluation steps ✗
Expressions and evaluation steps

Expression size we denote the *expression size* of $e$ by $\|e\|$.

Evaluation size we denote the *evaluation size* of $e$ by eval-steps($e$).

Example

$$e_1 = (((R \circ R) \circ (R \circ R)) \circ ((R \circ R) \circ (R \circ R))) \circ ((R \circ R) \circ (R \circ R))$$

$$e_2 = R \times (R \times (R \times (R \times (R \times (R \times R)))))$$

- $e_1 \equiv_{\pi_1} e_2$
- We have $\|e_1\| = 7$ and eval-steps($e_1$) = 3:
  1. $X = R \circ R$
  2. $Y = X \circ X$
  3. Result = $Y \circ Y$

- We have $\|e_2\| = 7$ and eval-steps($e_2$) = 7.
Evaluation size and unions

Example

\[ e_1 = (A \cup B) \circ (C \cup D) \circ (E \cup F) \]
\[ e_2 = A \ltimes (C \ltimes E) \cup A \ltimes (C \ltimes F) \cup \ldots \]

- \( e_1 \equiv_{\pi_1} e_2 \)
- We have \( \|e_1\| = \text{eval-steps}(e_1) = 5. \)
- We have \( \|e_2\| = \text{eval-steps}(e_2) = 23. \)
Evaluation size and unions

Example

\[ e_1 = (A \cup B) \circ (C \cup D) \circ (E \cup F) \]
\[ e_2 = A \ltimes (C \ltimes E) \cup A \ltimes (C \ltimes F) \cup \ldots \]

\[ e_1 \equiv_{\pi_1} e_2 \]

\[ \text{We have } \|e_1\| = \text{eval-steps}(e_1) = 5. \]

\[ \text{We have } \|e_2\| = \text{eval-steps}(e_2) = 23. \]

\[ e_3 = (A \ltimes X) \cup (B \ltimes X), \]
\[ X = (C \ltimes Y) \cup (D \ltimes Y), Y = (E \cup F) \]

\[ e_1 \equiv_{\pi_1} e_3' \]

\[ \text{We have } \|e_2'\| = 13 \text{ and } \text{eval-steps}(e_2') = 7. \]
Evaluation size and unions

Example

\[ e_1 = (A \cup B) \circ (C \cup D) \circ (E \cup F) \]
\[ e_2 = A \bowtie (C \bowtie E) \cup A \bowtie (C \bowtie F) \cup \ldots \]

\[ e_1 \equiv_{\pi_1} e_2 \]

\[ \text{We have } ||e_1|| = \text{eval-steps}(e_1) = 5. \]
\[ \text{We have } ||e_2|| = \text{eval-steps}(e_2) = 23. \]

\[ e_3 = (A \bowtie X) \cup (B \bowtie X), \]
\[ X = (C \bowtie Y) \cup (D \bowtie Y), \quad Y = (E \cup F) \]

\[ e_1 \equiv_{\pi_1} e_3' \]

\[ \text{We have } ||e_2'|| = 13 \text{ and eval-steps}(e_2') = 7. \]
\[ \tau_{o_i}(e; \varepsilon) \text{ does this!} \]
The main results (revised)

**Theorem**

Let $e$ be an expression. We have $\tau(e) \equiv_{\text{path}} e$, $\tau_{\pi_i}(e) \equiv_{\pi_i} e$, and

1. $\text{eval-steps}(\tau(e)) \leq u + \|e\|$;
2. $\text{eval-steps}(\tau_{\pi_i}(e)) \leq u + \|e\|$;
3. $\|\tau(e)\| = \Theta(\|e\| \cdot 2^u)$ in the worst case;
4. $\|\tau_{\pi_i}(e)\| = \Theta(\|e\| \cdot 2^u)$ in the worst case,

with $u$ the number of rewrite steps involving $\tau_{\circ_i}(e_1 \cup e_2; \varepsilon)$. 
Conclusion and future work

1. Real-life systems
2. Relational databases
3. Intersection and difference elimination
4. Extending FO[3] (e.g. counting)
The FO[2] fixpoint we use

- Notation $fp_{i,N}$ [iterative case union base case]
- $i$ specifies output column
- $N$ is a variable representing the growing output (node-test)
- Subset of traditional inflationary fixpoints

Example
The query $\pi_1[[ParentOf]^+ \circ OwnsPet]$ returns ancestors of pet-owners. We rewrite this into

$$\pi_1[fp_{1,N}[ParentOf \times N \cup ParentOf \times OwnsPet]]$$