Relational E-Matching
Simpler, Faster, and Optimal

Yihong Zhang, University of Washington
PLDI 2021 Student Research Competition
E-Graphs are everywhere!

Spores [VLDB '20]  
Herbie [PLDI '15]  
Szalinski [PLDI '20]  
TenSat [MLSys '21]  
Diospyoros [ASPLOS '21]  
Glenside [MAPS '21]  
egg [POPL '20]  
Z3  
CVC4  
Metathtetory.jl  
...

E-Graphs

g(f(a, a))

e-class 1 represents

e-nodes

e-classes
E-Graphs

g(f(a, a))

e-class 1 represents

e-nodes

e-classes
E-Graphs

e-class 1 represents

\[ g(f(a, a)) \quad f(a, g(a)) \]
\[ g(f(a, b)) \quad f(a, g(b)) \]
\[ g(f(b, a)) \quad f(b, g(a)) \]
\[ g(f(b, b)) \quad f(b, g(b)) \]
\[ f(a, g(c)) \]
\[ f(b, g(c)) \]

exponentially many terms!
E-Matching

> E-matching is the query that finds patterns in an e-graph.

> Responsible for 60-90% of the run time.

\[ f(a, g(a)), f(a, g(b)) \]
\[ f(b, g(a)), f(b, g(b)). \]

All witnessed by subst. \( \{\alpha \mapsto 4\} \).
Existing E-Matching Algorithms

\( f(\alpha, g(\alpha)) \)

for all e-class \( c \) in e-graph \( E \):
  for all \( f \) node \( n_1 \) in \( c \):
    subst = \{ \alpha \mapsto n_1.child_1, \text{root} \mapsto c \}
  for all \( g \) node \( n_2 \) in \( n_1.child_2 \):
    if subst[\alpha] = \( n_2\).child_1:
      yield subst

Quadratic runtime!
Yet at most \( O(N) \) terms.
Relational E-Matching

E-Matching \subseteq Conjunctive Queries

E-Graphs \subseteq Relational Databases

Simpler

Relational queries only involving joins e.g.,

\[ Q(a, c) \leftarrow R(a, b), S(b, c) \]
## Comparison to Existing E-Matching

<table>
<thead>
<tr>
<th>Existing E-Matching</th>
<th>Relational E-Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ Top-down backtracking search only.</td>
<td>✓ Top-down, bottom-up, middle-out, etc. depending on the query optimizer.</td>
</tr>
<tr>
<td>✗ Exploits structural constraints only.</td>
<td>✓ Exploits both structural constraints and equality constraints.</td>
</tr>
<tr>
<td>✗ No theoretical guarantee.</td>
<td>✓ Achieves optimality by adapting results from database research.</td>
</tr>
</tbody>
</table>
E-Graphs → Relational Databases

Relations representing function symbols

\[ R_f \]
- `eclass-id`
- `child_1`
- `child_2`

\[ R_g \]
- `eclass-id`
- `child_1`

\[ R_a, R_b, R_c \]
- `eclass-id`
E-Graphs → Relational Databases

Relations representing function symbols

<table>
<thead>
<tr>
<th>eclass-id</th>
<th>child₁</th>
<th>child₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
E-Matching \(\rightarrow\) Conjunctive Queries

\[ f(\alpha, g(\alpha)) \]

For all e-class \(c\) in e-graph \(E\):
- For all \(f\) node \(n_1\) in \(c\):
  - subst = \{\(\alpha \mapsto n_1.\text{child}_1, \text{root} \mapsto c\}\)
  - For all \(g\) node \(n_2\) in \(n_1.\text{child}_2\):
    - If subst[\(\alpha\)] = \(n_2.\text{child}_1\):
      - Yield subst

Linear

\[ Q(\text{root}, \alpha) \leftarrow R_f(\text{root}, \alpha, x), R_g(x, \alpha) \]

For all \(r_f\) in \(R_f\):
- If \( (r_f.\text{child}_2, r_f.\text{child}_1) \) in \(R_g\):
  - Yield \{\(\text{root} \mapsto R_f.\text{eclass-id}, \alpha \mapsto R_f.\text{child}_1\}\}

Quadratic
Traditional two-way join algorithms like hash join and merge-sort join are provably suboptimal.

We propose to use the generic join algorithm to solve the generated conjunctive query.

Fix a pattern \( p \), let \( M(p,E) \) be the set of substitutions yielded by e-matching on an e-graph \( E \) with size \( n \), relational e-matching runs in time

\[
\tilde{O} \left( \max_E |M(p,E)| \right).
\]
(Preliminary) Evaluations

- Asymptotic speed-ups on patterns with equality constraints (the cyclic and non-linear acyclic patterns).
- Similar performance on patterns with no equality constraints (the linear pattern).

Speed-ups over existing e-matching algorithms (de Moura and Bjørner)
Thank you!
Why Faster?

\( f(\alpha, g(\alpha)) \) enumerates all terms of the form \( f(\alpha, g(\beta)) \), and check if \( \alpha = \beta \) only before yielding.

\[ Q(\text{root}, \alpha) \leftarrow R_f(\text{root}, \alpha, x), R_g(x, \alpha) \]
builds indices on both \( \alpha \) and \( x \); only enumerates terms where constraints on \( \alpha \) and \( x \) are both satisfied.

Equality constraints are exploited to prune the search space!
Traditional two-way join algorithms like hash join and merge-sort join are not optimal.

For example, for conjunctive query whose corresponding hypergraph is cyclic, two-way join algorithms will enumerate asymptotically more atoms than needed.

\[- (\alpha + \beta) \times (\alpha + \gamma).\]
Generic Join Algorithm

> Multi-way join algorithm that avoids enumerating unnecessarily large intermediate relations.
> Worst-case optimal & efficient for both cyclic and acyclic queries.