Relational E-matching
Simpler, Faster, and Optimal

Yihong Zhang, Remy Wang, Max Willsey, Zachary Tatlock
University of Washington
Relational e-matching

e-graphs \subseteq \text{relational databases}

e-matching \subseteq \text{conjunctive queries}

e.g., Q(a, c) :- R(a, b), S(b, c)

relational queries only involving joins
Overview

- Background
- Relational e-matching
- Evaluations and discussions
Overview

- Background
- Relational e-matching
- Evaluations and discussions
E-graphs are everywhere!

Spores [VLDB ’20]
Szalinski [PLDI ’20]
Herbie [PLDI ’15]
TenSat [MLSys ’21]
Diospyoros [ASPLOS ’21]
Glenside [MAPS ’21]
egg [POPL ’20]
Z3
Ruler [OOPSLA ’21]
Metatheory.jl
CVC4
...

CVC4
E-graphs are everywhere!

**Program optimization**
- Known as “equality saturation”.
- Keeping many equivalent programs in a single e-graph.
- Non-destructive rewriting until fixpoint or timeout.

**SMT solver**
- Solving theory of equality with uninterpreted functions.
- Combining theories (Nelson-Oppen procedure)
E-graphs

- An e-graph represents a set of terms and a congruence relation $\equiv$ efficiently.

$$\text{E-class } c_f \text{ represents } f(1, g(1)), f(1, g(2)), f(2, g(1)), \ldots$$

- All equivalent with each other.
- Exponentially many terms!
E-graphs

- An e-graph represents a set of terms and a congruence relation \( \cong \) efficiently.
- E-class \( c_f \) represents \( f(1, g(1)), \)
  \( f(1, g(2)), \) \( f(2, g(1)), \ldots \)
  - All equivalent with each other.
  - Exponentially many terms!
E-graphs

- An e-graph represents a set of terms and a congruence relation \( \approx \) efficiently.
- E-class \( c_f \) represents \( f(1, g(1)), f(1, g(2)), f(2, g(1)), \ldots \) all equivalent with each other.
- Exponentially many terms!
E-graphs

- An e-graph represents a set of terms and a congruence relation efficiently.
- E-class $c_f$ represents $f(1, g(1)), f(1, g(2)), f(2, g(1)), \ldots$ all equivalent with each other.
- Exponentially many terms!
E-graphs

● An e-graph represents a set of terms and a congruence relation $\equiv$ efficiently.
● E-class $c_f$ represents $f(1, g(1))$, ...

$\text{c}_f$ $\rightarrow$ $f$ $\rightarrow$ $\text{c}_g$ $\rightarrow$ $g$ $\rightarrow$ $1$ $\rightarrow$ $2$ $\ldots$ $N$ $\rightarrow$ $f$ $\rightarrow$ $g$ $\rightarrow$ $\text{e}-\text{nodes}$ $\rightarrow$ $\text{e}-\text{classes}$
E-graphs

- An e-graph represents a set of terms and a congruence relation efficiently.
- E-class $c_f$ represents $f(1, g(1)), f(1, g(2)), \ldots$, all equivalent with each other.
- Exponentially many terms!
E-graphs

- An e-graph represents a set of terms and a congruence relation efficiently.
- E-class $c_f$ represents $f(1, g(1)), f(1, g(2)), f(2, g(1)),$ and so on.

Exponentially many terms!
E-graphs

- An e-graph represents a set of terms and a congruence relation $\cong$ efficiently.
- E-class $c_f$ represents $f(1, g(1)), f(1, g(2)), f(2, g(1)), \ldots$
E-graphs

- An e-graph represents a set of terms and a congruence relation \( \equiv \) efficiently.
- E-class \( c_f \) represents \( f(1, g(1)), f(1, g(2)), f(2, g(1)), \ldots \)
  - All equivalent to each other.
E-graphs

- An e-graph represents a set of terms and a congruence relation $\equiv$ efficiently.
- E-class $c_f$ represents $f(1, g(1)), f(1, g(2)), f(2, g(1)), \ldots$
  - All equivalent to each other.
- Exponentially many terms!
E-matching

- E-matching: pattern matching over an e-graph.
- More formally: e-matching finds substitutions from variables to e-classes such that the substituted terms are represented by the e-graph.

\[ f(\alpha, g(\alpha)) \] will match \[ f(1, g(1)) \]
\[ f(2, g(2)) \]
\[ \ldots \]
\[ f(N, g(N)) \]
, witnessed by \[ \{ \alpha \mapsto 1 \} \]
\[ \{ \alpha \mapsto 2 \} \]
\[ \ldots \]
\[ \{ \alpha \mapsto N \} \]

\[ f(1, \alpha) \] will match \[ f(1, g(1)) \]
\[ f(1, g(2)) \]
\[ \ldots \]
\[ f(1, g(N)) \]
, witnessed by \[ \{ \alpha \mapsto c_g \} \].
E-matching

- E-matching: pattern matching over an e-graph.
- More formally: e-matching finds substitutions from variables to e-classes such that the substituted terms are represented by the e-graph.
E-matching

- E-matching: pattern matching over an e-graph.
- More formally: e-matching finds substitutions from variables to e-classes such that the substituted terms are represented by the e-graph.
- NP complete w.r.t. the pattern size.
E-matching

- E-matching: pattern matching over an e-graph.
- More formally: e-matching finds substitutions from variables to e-classes such that the substituted terms are represented by the e-graph.
- NP complete w.r.t. the pattern size.
- Responsible for 60–90% of the run time in equality saturation.

Bottleneck!
Existing e-matching algorithms
Existing e-matching algorithms

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

for e-class \texttt{c in e-graph E:}
Existing e-matching algorithms

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

for e-class \( c \) in e-graph \( E \):
for \( f \)-node \( n_1 \) in \( c \):

Backtracking search \( f(\alpha, g(\alpha)) \)

\( f(1, c_g) \)
\( f(2, c_g) \)
\( \ldots \)
\( f(N, c_g) \)
Existing e-matching algorithms

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

for e-class \( c \) in e-graph \( E \):
for \( f \)-node \( n_1 \) in \( c \):
\[
\text{subst} = \{ \text{root} \mapsto c, \ \alpha \mapsto n_1.\text{child}_1 \} 
\]
Existing e-matching algorithms

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

for e-class \( c \) in e-graph \( E \):

for \( f \)-node \( n_1 \) in \( c \):

\[ \text{subst} = \{ \text{root} \mapsto c, \ \alpha \mapsto n_1.\text{child}_1 \} \]

for \( g \)-node \( n_2 \) in \( n_1.\text{child}_2 \):

Backtracking search \( f(\alpha, g(\alpha)) \)

\[ f(1, c_g) \quad g(1) \quad g(2) \quad \ldots \quad g(N) \]

\[ f(2, c_g) \quad g(1) \quad g(2) \quad \ldots \quad g(N) \]

\[ f(N, c_g) \quad g(1) \quad g(2) \quad \ldots \quad g(N) \]
Existing e-matching algorithms

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

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

Backtracking search \( f(\alpha, g(\alpha)) \)

\[
\begin{align*}
f(1,c_g) & \quad g(1) \quad \checkmark \\
& \quad g(2) \\
& \quad \ldots \\
& \quad g(N) \\
\hline
f(2,c_g) & \quad g(1) \quad \square \\
& \quad g(2) \quad \checkmark \\
& \quad \ldots \\
& \quad g(N) \\
\hline
\vdots & \quad \vdots \\
\hline
f(N,c_g) & \quad g(1) \quad \square \\
& \quad g(2) \\
& \quad \ldots \\
& \quad g(N) \quad \checkmark
\end{align*}
\]
Existing e-matching algorithms

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

\[
\text{for e-class } c \text{ in e-graph } E: \\
\text{for } f\text{-node } n_1 \text{ in } c: \\
\quad \text{subst} = \{\text{root} \mapsto c, \alpha \mapsto n_1.\text{child}_1\} \\
\text{for } g\text{-node } n_2 \text{ in } n_1.\text{child}_2: \\
\quad \text{if } \text{subst}[\alpha] = n_2.\text{child}_1: \\
\quad \quad \text{yield } \text{subst}
\]
Existing e-matching algorithms

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

\[
\text{for e-class } c \text{ in e-graph } E: \\
\quad \text{for } f\text{-node } n_1 \text{ in } c: \\
\quad \quad \text{subst} = \{ \text{root} \mapsto c, \ \alpha \mapsto n_1.\text{child}_1 \} \\
\quad \quad \text{for } g\text{-node } n_2 \text{ in } n_1.\text{child}_2: \\
\quad \quad \quad \text{if } \text{subst}[\alpha] = n_2.\text{child}_1: \\
\quad \quad \quad \quad \text{yield subst}
\]

O(N^2)!

Yet at most O(N) matches

Backtracking search \( f(\alpha, g(\alpha)) \)

| \( f(1,c_g) \) | \( g(1) \) | ✓ |
| \( f(2,c_g) \) | \( g(2) \) | ☐ |
| ... | ... | ... |
| \( f(N,c_g) \) | \( g(1) \) | ☐ |
| ... | ... | ... |
| ... | ... | ... |

For each node \( n_1 \) in class \( c \):

\[
\text{yield subst}
\]

Marked node \( n_2 \): ✓
Existing e-matching algorithms

- Relies on naive backtracking.
- Uses several *ad hoc* optimizations for specific patterns.
- No data complexity bound.
Overview

- Background
- **Relational e-matching**
- Evaluation and future work
Relational e-matching

- Takes an e-graph and a list of patterns.
- Transforms the e-graph to a relational database.
- Compiles all e-matching patterns to conjunctive queries.
- Run the conjunctive query on the relational database!

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

\[ \text{Relational e-matching} \]

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

\[ \text{Run the conjunctive query on the relational database!} \]
Observations

E-matching
Finding substitutions such that the substituted terms are present in the e-graph.

 Conjunctive queries
Finding substitutions such that the substituted atoms are present in the relational database.
Relational e-matching workflow

well suited for equality saturation

\[
\begin{align*}
Q(\text{root}, \alpha) &\leftarrow R_f(\text{root}, \alpha, x), R_g(x, \alpha) \\
Q(\text{root}, \alpha) &\leftarrow R_g(\text{root}, x), R_f(x, \alpha, \alpha)
\end{align*}
\]
Relational e-matching

- Takes an e-graph and a list of patterns.
- Transforms the e-graph to a relational database.
- Compiles all e-matching patterns to conjunctive queries.
- Run the conjunctive query on the relational database!

well suited for equality saturation
E-graphs are relational databases
E-graphs are relational databases
E-matching is conjunctive queries

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

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

\[ \text{ind} = \{\} \]
\[ \text{for} \ (x, \alpha) \ \text{in} \ R_g : \ # \ \text{build index} \]
\[ \text{ind}.\text{insert}((x, \alpha)) \]

build hash

\[ R_g(c_g, 1) \]
\[ R_g(c_g, 2) \]
\[ ... \]
\[ R_g(c_g, N) \]
E-matching is conjunctive queries

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

\[
\text{ind} = \{\}
\]

\[
\text{for (x, } \alpha \text{) in } R_g: \quad \# \text{ build index}
\]

\[
\text{ind}.\text{insert((x, } \alpha\text{))}
\]

\[
\text{for (root, } \alpha, x\text{) in } R_f: \quad \# \text{ probe}
\]

\[
\text{if (} \alpha, x\text{) in ind:}
\]

\[
\text{yield } \{\text{root } \mapsto \text{ root}, \alpha \mapsto \alpha\}
\]

\[
R_g(c_g, 1)
\]

\[
R_g(c_g, 2)
\]

\[
\ldots
\]

\[
R_g(c_g, N)
\]

\[
R_f(c_f, 1, c_f)
\]

\[
R_f(c_f, 2, c_f)
\]

\[
\ldots
\]

\[
R_f(c_f, N, c_f)
\]

\[
\text{O}(N^2)
\]

\[
\text{O}(N)
\]

\[
38
\]
Why is relational e-matching faster?

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

Build indices on both $\alpha$ and $x$, and only enumerate terms where constraints on both $x$ and $\alpha$ are satisfied.
E-matching is conjunctive queries

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

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

\[ f(1, c_g), g(1) \quad \checkmark \]
\[ \ldots \]
\[ g(N) \quad \square \]
\[ \ldots \]
\[ f(N, c_g), g(1) \quad \square \]
\[ \ldots \]
\[ g(N) \quad \checkmark \]

Backtracking search

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

\( \text{ind} = {} \)

for \( (\text{id}, \text{arg}_1) \) in \( R_g \):
  # build index
  \( \text{ind}.\text{insert}((\text{id}, \text{arg}_1)) \)

for \( (\text{id}, \text{arg}_1, \text{arg}_2) \) in \( R_f \):
  # probe
  if \( (\text{arg}_2, \text{arg}_1) \) in \( \text{ind} \):
    yield \{root \mapsto \text{id}, \alpha \mapsto \text{arg}_2\}

\[ R_g(c_g, 1) \quad \checkmark \]
\[ R_g(c_g, 2) \quad \checkmark \]
\[ \ldots \]
\[ R_g(c_g, N) \quad \checkmark \]

build hash

\[ R_f(c_f, 1, c_f) \quad \checkmark \]
\[ R_f(c_f, 2, c_f) \quad \checkmark \]
\[ \ldots \]
\[ R_f(c_f, N, c_f) \quad \checkmark \]

probe
Comparison to traditional e-matching

Traditional e-matching

✖ Exploits structural constraints only.

Relational e-matching

✓ Exploits both structural constraints and equality constraints.
Comparison to traditional e-matching

Traditional e-matching

✖ Exploits structural constraints only.
✖ Top-down backtracking search only.

Relational e-matching

✔ Exploits both structural constraints and equality constraints.
✔ Top-down, bottom-up, middle-out, etc. depending on the query optimizer.
Comparison to traditional e-matching

Traditional e-matching

✖ Exploits structural constraints only.

✖ Top-down backtracking search only.

Relational e-matching

✓ Exploits both structural constraints and equality constraints.

✓ Top-down, bottom-up, middle-out, etc. depending on the query optimizer.

● E.g., e-matching $f(g(h(x)))$ on e-graphs with only one $h$-node.
Comparison to traditional e-matching

Traditional e-matching

✖ Exploits structural constraints only.
✖ Top-down backtracking search only.

Relational e-matching

✓ Exploits both structural constraints and equality constraints.
✓ Top-down, bottom-up, middle-out, etc. depending on the query optimizer.
Comparison to traditional e-matching

<table>
<thead>
<tr>
<th>Traditional e-matching</th>
<th>Relational e-matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>✖ Exploits structural constraints only.</td>
<td>✗ Exploits both structural constraints and equality constraints.</td>
</tr>
<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>✖ No theoretical guarantee.</td>
<td>✗ Achieves optimality by adapting results from database research.</td>
</tr>
</tbody>
</table>
Data complexity results

**Theorem 9.** Relational e-matching is worst-case optimal; that is, fix a pattern $p$, let $M(p, E)$ be the set of substitutions yielded by e-matching on an e-graph $E$ with $N$ e-nodes, relational e-matching runs in time $O(\max_E(|M(p, E)|))$.

**Theorem 10.** Fix an e-graph $E$ with $N$ e-nodes that compiles to a database $I$, and a fix pattern $p$ that compiles to conjunctive query $Q(X) \leftarrow R_1(X_1), \ldots, R_m(X_m)$. Relational e-matching $p$ on $E$ runs in time $O\left(\sqrt{|Q(I)| \times \prod_i |R_i|}\right) \leq O\left(\sqrt{|Q(I)| \times N^m}\right)$. 
Theorem 9. Relational e-matching is worst-case optimal; that is, fix a pattern \( p \), let \( M(p, E) \) be the set of substitutions yielded by e-matching on an e-graph \( E \) with \( N \) e-nodes, relational e-matching runs in time \( O(\max_E(|M(p,E)|)) \).

- Run time = max output size an e-graph with the same size could achieve.
- Application of database research on worst-case optimal join (WCOJ) algorithm.
Theorem 10. Fix an e-graph $E$ with $N$ e-nodes that compiles to a database $I$, and a fix pattern $p$ that compiles to conjunctive query $Q(\overline{X}) \leftarrow R_1(\overline{X}_1), \ldots, R_m(\overline{X}_m)$. Relational e-matching $p$ on $E$ runs in time $O\left(\sqrt{|Q(I)| \times \prod_i |R_i|}\right) \leq O\left(\sqrt{|Q(I)| \times N^m}\right)$.

- A more involved theorem; use the structure of the compiled conjunctive query in the proof.
- Run time = function of e-graph size and actual output size.
Overview

- Background
- Relational e-matching
- Evaluation and discussions
Evaluations

- Uses two largest test suites of egg, a state-of-the-art e-graph framework.
- Baseline: egg’s e-matching procedure.
- Uses generic join for solving conjunctive query.
- Run twice once with index building and once without.
Evaluations

6 orders of magnitude speedup

Index building takes time

Similar performance on linear patterns:

\((+ a (-1 b))\)
Multi-patterns

- Multi-patterns: list of patterns $(p_1, ..., p_n)$ to be matched simultaneously.
- Studied in literature & used by practical applications.
- Relational e-matching supports multi-patterns for free!

\[
(f(\alpha, \beta), f(\beta, \gamma))
\]

\[
Q(r1, r2, \alpha, \beta, \gamma) :- \\
R_f(r1, \alpha, \beta), R_f(r2, \beta, \gamma)
\]
Functional dependencies (FDs) in e-graph

- FD: Dependencies between attributes.
- “Every e-node uniquely identifies an e-class it belongs to” translates to FD in the relational representation.
- FD allows us to derive even tighter bound.
  - $f(g(\alpha), h(\alpha))$ translates to $Q(r, \alpha) :- R_f(r, x, y), R_g(x, \alpha), R_h(y, \alpha)$, which has a worst-case complexity of $O(N^{2/3})$;
  - FD tells us $\alpha$ uniquely determines $g(\alpha)$ and $h(\alpha)$, and therefore $f(g(\alpha), h(\alpha))$; simply enumerating $\alpha$ gives an $O(N)$ bound.
Future work

- Incremental e-matching = Incremental View Maintenance (IVM) in database.
- More join algorithms & more optimizations.
- Building on existing database management systems (DBMS).
  - We built a proof-of-concept prototype with sqlite!
  - Persistence, scalability, concurrency, ...

Thank you!
Takeaways

● Relational e-matching is simpler, faster, and optimal.
● Traditional e-matching is bad because they don’t exploit equality constraints during query planning.
● Relational database is a very powerful abstraction.