Relational E-matching

Simpler, Faster, and Optimal

Yihong Zhang, Remy Wang, Max Willsey, Zachary Tatlock
University of Washington

Relational e-matching

e-graphs $\subseteq$ relational databases

e-matching $\subseteq$ conjunctive queries

relational queries only involving joins
e.g., Q(a, c) :- R(a, b), S(b, c)

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

...

E-graphs are everywhere!

Program optimization

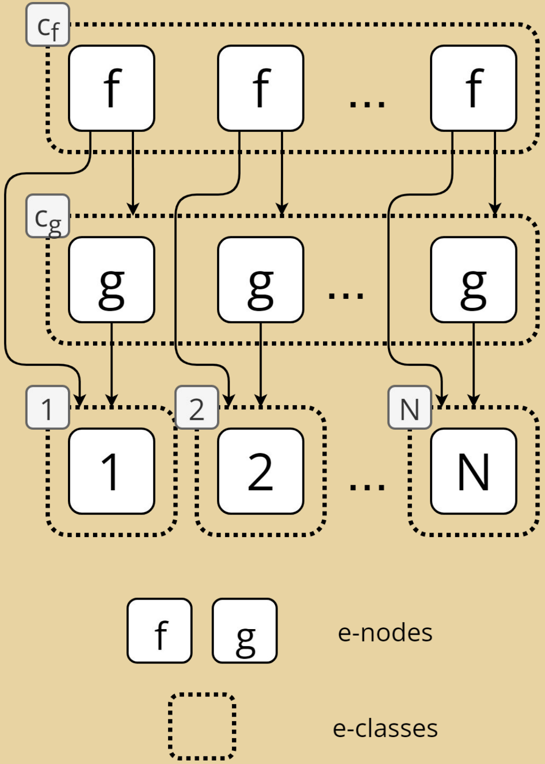
- Known as "equality saturation".
- Keeping many equivalent programs in a single e-graph.
- Non-destructive rewriting until fixpoint or timeout.

SMT solver

- Solving theory of equality with uninterpreted functions.
- Combining theories (Nelson-Oppen procedure)

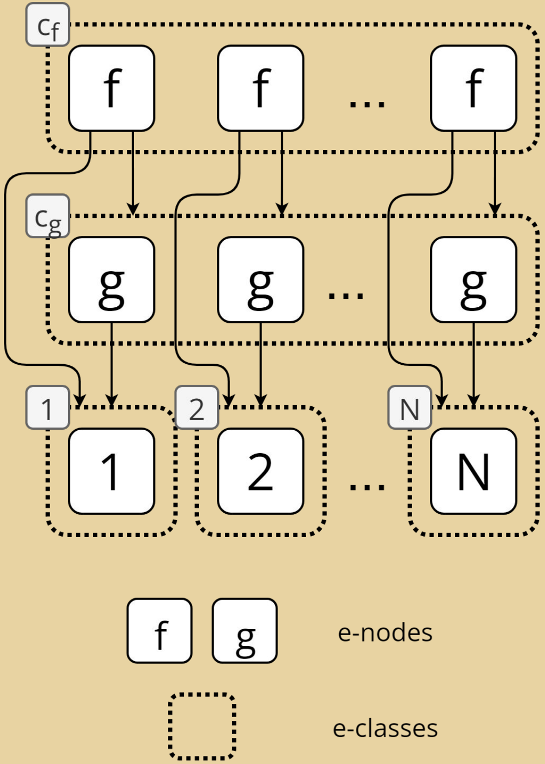
E-graphs

- An e-graph represents a set of terms and a congruence relation ≅ efficiently.



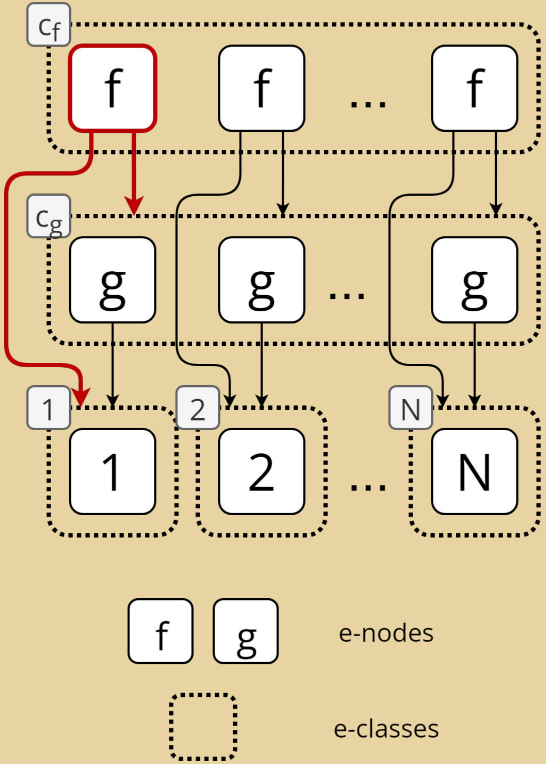
E-graphs

- An e-graph represents a set of terms and a congruence relation ≅ efficiently.
- E-class c_f represents *f(1, g(1))*,



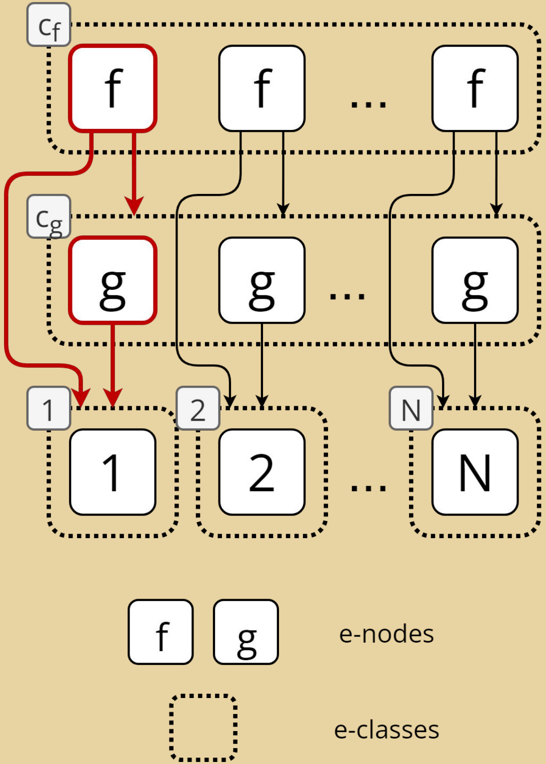
E-graphs

- An e-graph represents a set of terms and a congruence relation ≅ efficiently.
- E-class c_f represents *f(1, g(1))*,



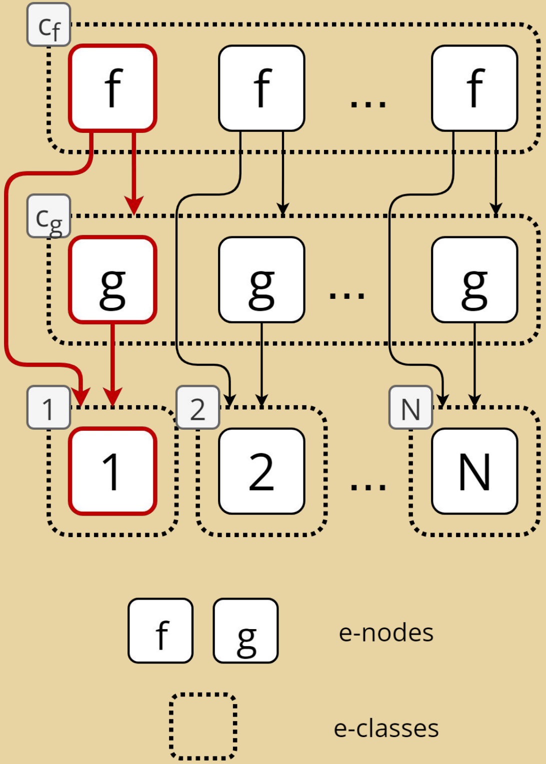
E-graphs

- An e-graph represents a set of terms and a congruence relation ≅ efficiently.
- E-class c_f represents *f(1, g(1))*,



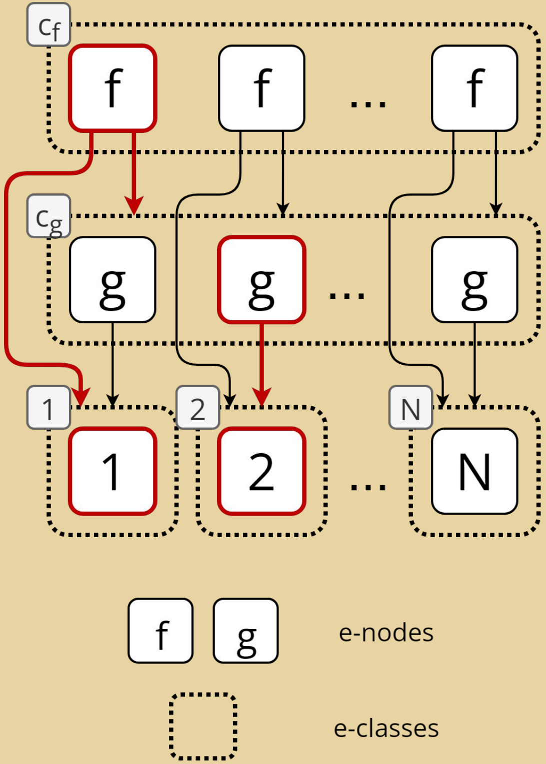
E-graphs

- An e-graph represents a set of terms and a congruence relation ≅ efficiently.
- E-class c_f represents *f(1, g(1))*,



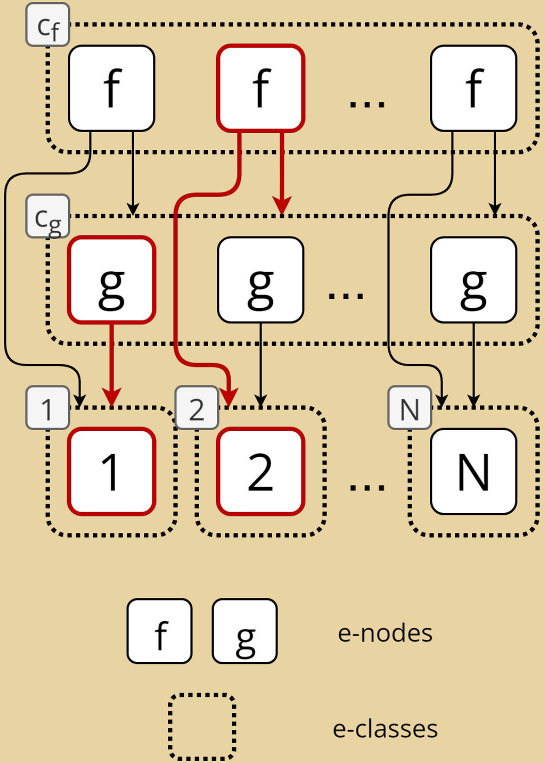
E-graphs

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



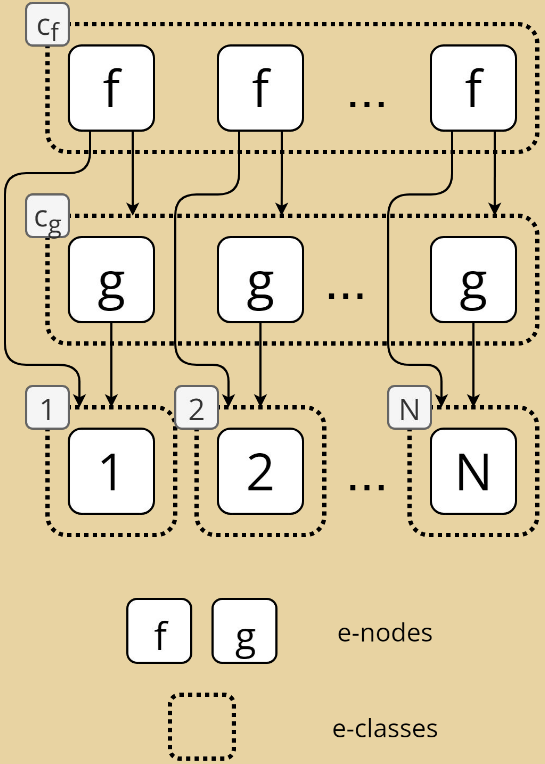
E-graphs

- 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))*,



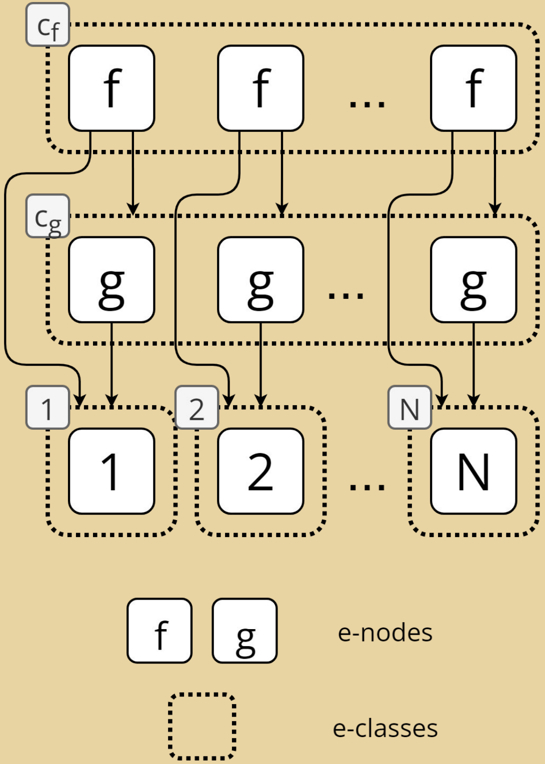
E-graphs

- 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))*, …



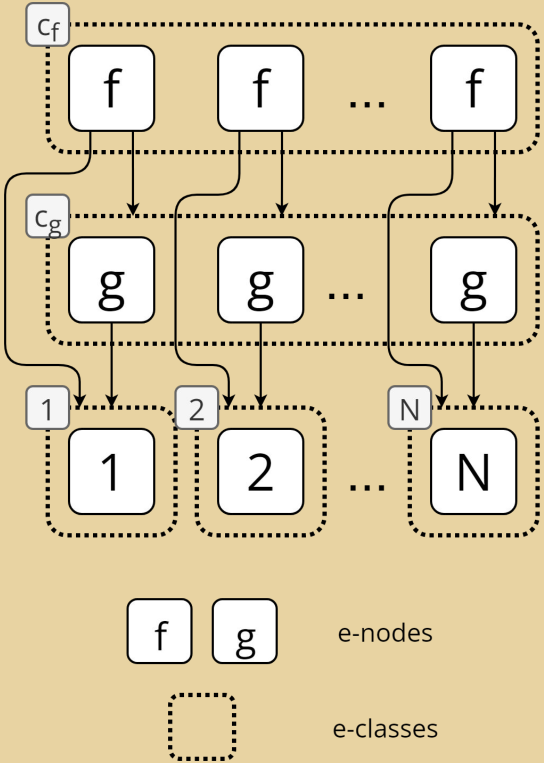
E-graphs

- 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))*, …
    - All equivalent to each other.



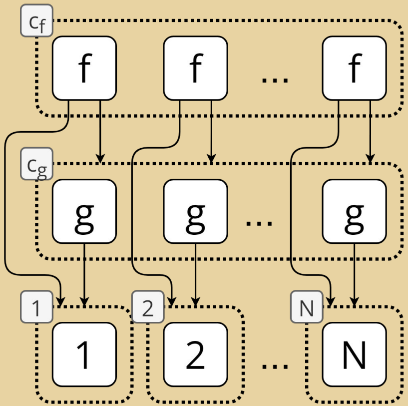
E-graphs

- 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))*, …
    - 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.



$f(a, g(a))$ will match $\begin{array}{c} f(1, g(1)) \\ f(2, g(2)) \\ \dots \\ f(N, g(N)) \end{array}$, witnessed by $\begin{array}{c} \{a \mapsto 1\} \\ \{a \mapsto 2\} \\ \dots \\ \{a \mapsto N\} \end{array}$.

$f(1, a)$ will match $\begin{array}{c} f(1, g(1)) \\ f(1, g(2)) \\ \dots \\ f(1, g(N)) \end{array}$, witnessed by $\{a \mapsto c_g\}$.

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 th substituted terms are represented by the
- NP complete w.r tern size.
- Responsible for 60–90% of the run time in equality saturation.

Bottleneck!

Existing e-matching algorithms

Existing e-matching algorithms

f(α, g(α))

⇩

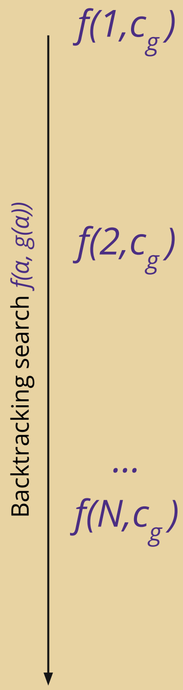
```
for e-class c in e-graph E:
```

Backtracking search *f(α, g(α))*

Existing e-matching algorithms

$f(a, g(a))$

```
for e-class c in e-graph E:
  for f-node n1 in c:
```



Existing e-matching algorithms

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

```
for e-class c in e-graph E:
  for f-node n_1 in c:
    subst = {root ↦ c, α ↦ n_1.child_1}
```

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

$f(1, c_g)$

$f(2, c_g)$

…

$f(N, c_g)$

Existing e-matching algorithms

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

```
for e-class c in e-graph E:
  for f-node n_1 in c:
    subst = {root ↦ c, α ↦ n_1.child_1}
    for g-node n_2 in n_1.child_2:
```

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

$f(1,c_g)$ $g(1)$
$g(2)$
…
$g(N)$
$f(2,c_g)$ $g(1)$
$g(2)$
…
$g(N)$
… …
$f(N,c_g)$ $g(1)$
$g(2)$
…
$g(N)$

Existing e-matching algorithms

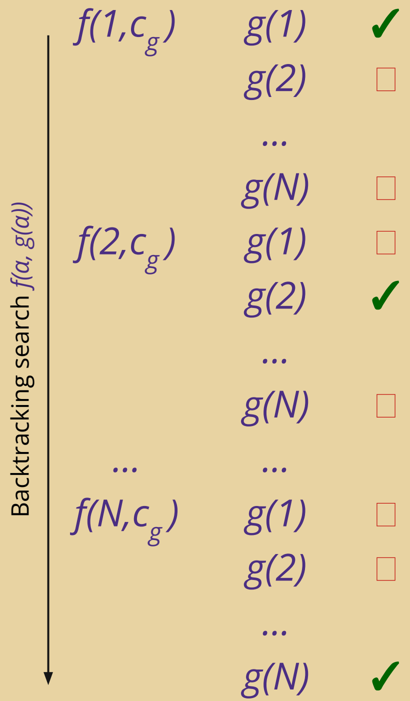
f(α, g(α))

```
for e-class c in e-graph E:
  for f-node n_1 in c:
    subst = {root ↦ c, α ↦ n_1.child_1}
    for g-node n_2 in n_1.child_2:
      if subst[α] = n_2.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:
    subst = {root ↦ c, α ↦ n_1.child_1}
    for g-node n_2 in n_1.child_2:
      if subst[α] = n_2.child_1:
        yield subst
```



Existing e-matching algorithms

f(α, g(α))

```
for e-class c in e-graph E:
  for f-node n1 in c:
    subst = {root ↦ c, α ↦ n1.child1}
    for g-node n2 in n1.child2:
      if subst[α] = n2.child1:
        yield subst
```

$O(N^2)$!

Yet at most O(N) matches

Backtracking search *f(α, g(α))*

$f(1,c_g)$	$g(1)$	✓
	$g(2)$	□
	…	
	$g(N)$	□
$f(2,c_g)$	$g(1)$	□
	$g(2)$	✓
	…	
	$g(N)$	□
…	…	
$f(N,c_g)$	$g(1)$	□
	$g(2)$	□
	…	
	$g(N)$	✓

Existing e-matching algorithms

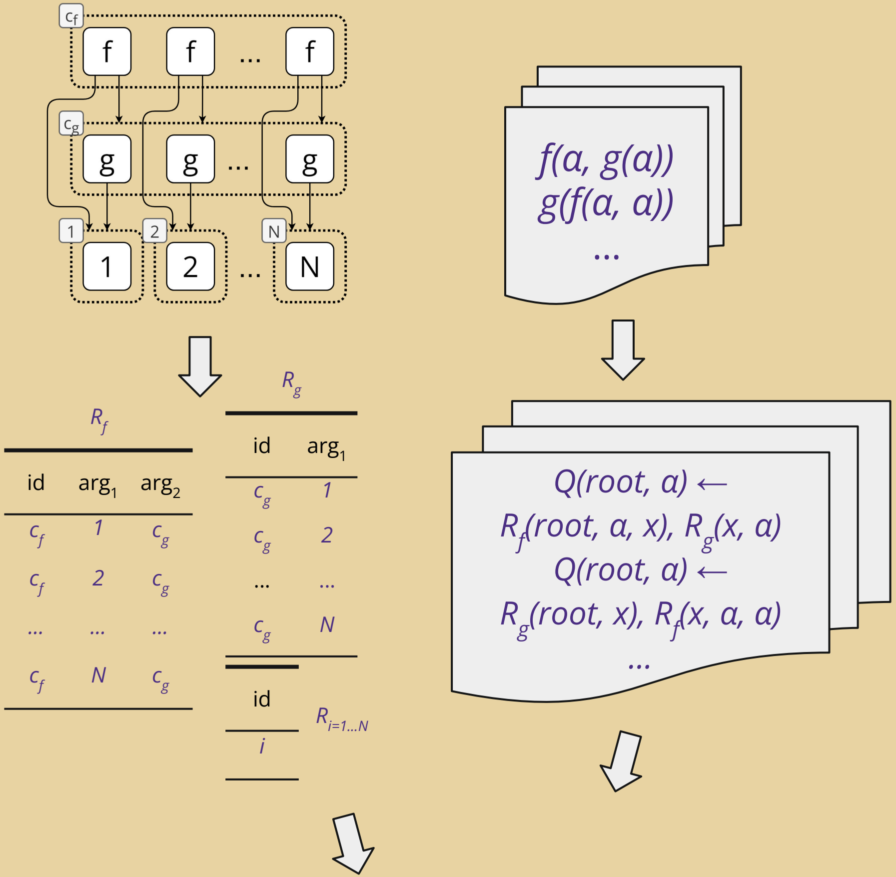
- Relies on naive backtracking.
- Uses several *ad hoc* optimizations for specific patterns.
- No data complexity bound.

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!



Observations

E-matching

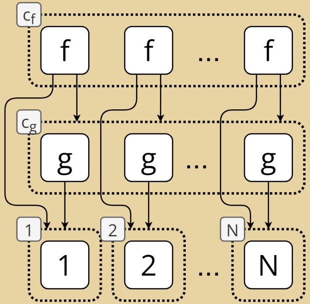
Finding substitutions such that the substituted terms are present in the e-graph.

🤝

Conjunctive queries

Finding substitutions such that the substituted atoms are present in the relational database.

Relational e-matching workflow



R_f

id	arg_1	arg_2
c_f	1	c_g
c_f	2	c_g
…	…	…
c_f	N	c_g

R_g

id	arg_1
c_g	1
c_g	2
…	…
c_g	N

$R_{i=1...N}$

id
i

well suited for equality saturation

$f(a, g(a))$
$g(f(a, a))$
…

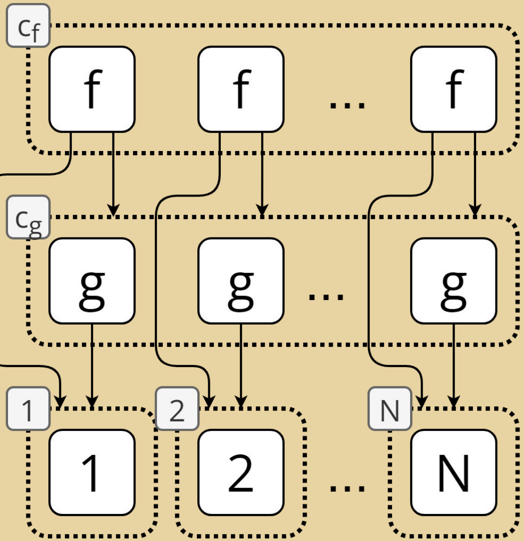
$Q(root, a) \leftarrow R_f(root, a, x), R_g(x, a)$
$Q(root, a) \leftarrow R_g(root, x), R_f(x, a, a)$
…

Relational e-matching

well suited for
equality saturation

- 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!

E-graphs are relational databases



R_f

id	arg_1	arg_2
c_f	1	c_g
c_f	2	c_g
…	…	…
c_f	N	c_g

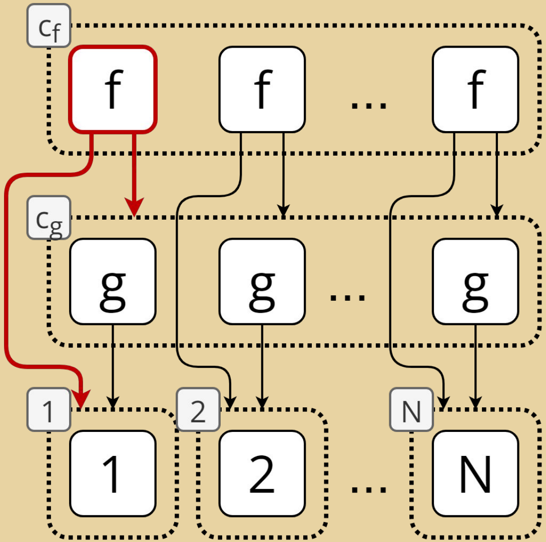
R_g

id	arg_1
c_g	1
c_g	2
…	…
c_g	N

$R_{i=1 \ldots N}$

id
i

E-graphs are relational databases



R_f

id	arg_1	arg_2
c_f	1	c_g
c_f	2	c_g
...	...	...
c_f	N	c_g

R_g

id	arg_1
c_g	1
c_g	2
...	...
c_g	N

$R_{i=1\dots N}$

id
i

E-matching is conjunctive queries

```
ind = {}
for (x, α) in R_g: # build index
  ind.insert((x, α))
```

$f(\alpha, g(\alpha))$ ➡ $Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha)$ ➡

build hash

$R_g(c_g, 1)$
$R_g(c_g, 2)$
…
$R_g(c_g, N)$

E-matching is conjunctive queries

$f(\alpha, g(\alpha))$ ➡ $Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha)$ ➡

```
ind = {}
for (x, α) in R_g: # build index
  ind.insert((x, α))
for (root, α, x) in R_f: # probe
  if (α, x) in ind:
    yield {root ↦ root, α ↦ α}
```

build hash		probe	
	$R_g(c_g, 1)$		$R_f(c_f, 1, c_f)$ ✓
	$R_g(c_g, 2)$		$R_f(c_f, 2, c_f)$ ✓
	…		…
	$R_g(c_g, N)$		$R_f(c_f, N, c_f)$ ✓

$O(N^2)$ **$O(N)$**

Why is relational e-matching faster?

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

$Q(root, \alpha) \leftarrow$
$R_f(root, \alpha, x), R_g(x, \alpha)$

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

Build indices on both α and x, and only enumerate terms where constraints on both x and α are satisfied.

structural constraints

equality constraints

E-matching is conjunctive queries

f(α, g(α)) ➡ $Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha)$

```
for e-class c in e-graph E:
  for f-node n1 in c:
    subst = {root ↦ c, α ↦ n1.child1}
    for g-node n2 in n1.child2:
      if subst[α] = n2.child1:
        yield subst
```

Backtracking search

$f(1,c_g)$	$g(1)$	✓
	…	
	$g(N)$	☐
…	…	
$f(N,c_g)$	$g(1)$	☐
	…	
	$g(N)$	✓

```
ind = {}
for (id, arg1) in Rg: # build index
  ind.insert((id, arg1))
for (id, arg1, arg2) in Rf: # probe
  if (arg2, arg1) in ind:
    yield {root ↦ id, α ↦ arg2}
```

build hash	probe
$R_g(c_g, 1)$	$R_f(c_f, 1, c_f)$ ✓
$R_g(c_g, 2)$	$R_f(c_f, 2, c_f)$ ✓
…	…
$R_g(c_g, N)$	$R_f(c_f, N, c_f)$ ✓

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

Traditional e-matching

✖ Exploits structural constraints only.

✖ Top-down backtracking search only.

✖ No theoretical guarantee.

Relational e-matching

✓ Exploits both structural constraints and equality constraints.

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

✓ Achieves optimality by adapting results from database research.

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 $Q(\overline{X}) \leftarrow R_1(\overline{X_1}), \ldots, R_m(\overline{X_m})$. Relational e-matching p on E runs in time $O\left(\sqrt{|Q(I)| \times \Pi_i |R_i|}\right) \le O\left(\sqrt{|Q(I)| \times N^m}\right)$.*

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 $O\left(\sqrt{|Q(I)| \times \Pi_i |R_i|}\right) \leq O\left(\sqrt{|Q(I)| \times N^m}\right)$.*

- 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.

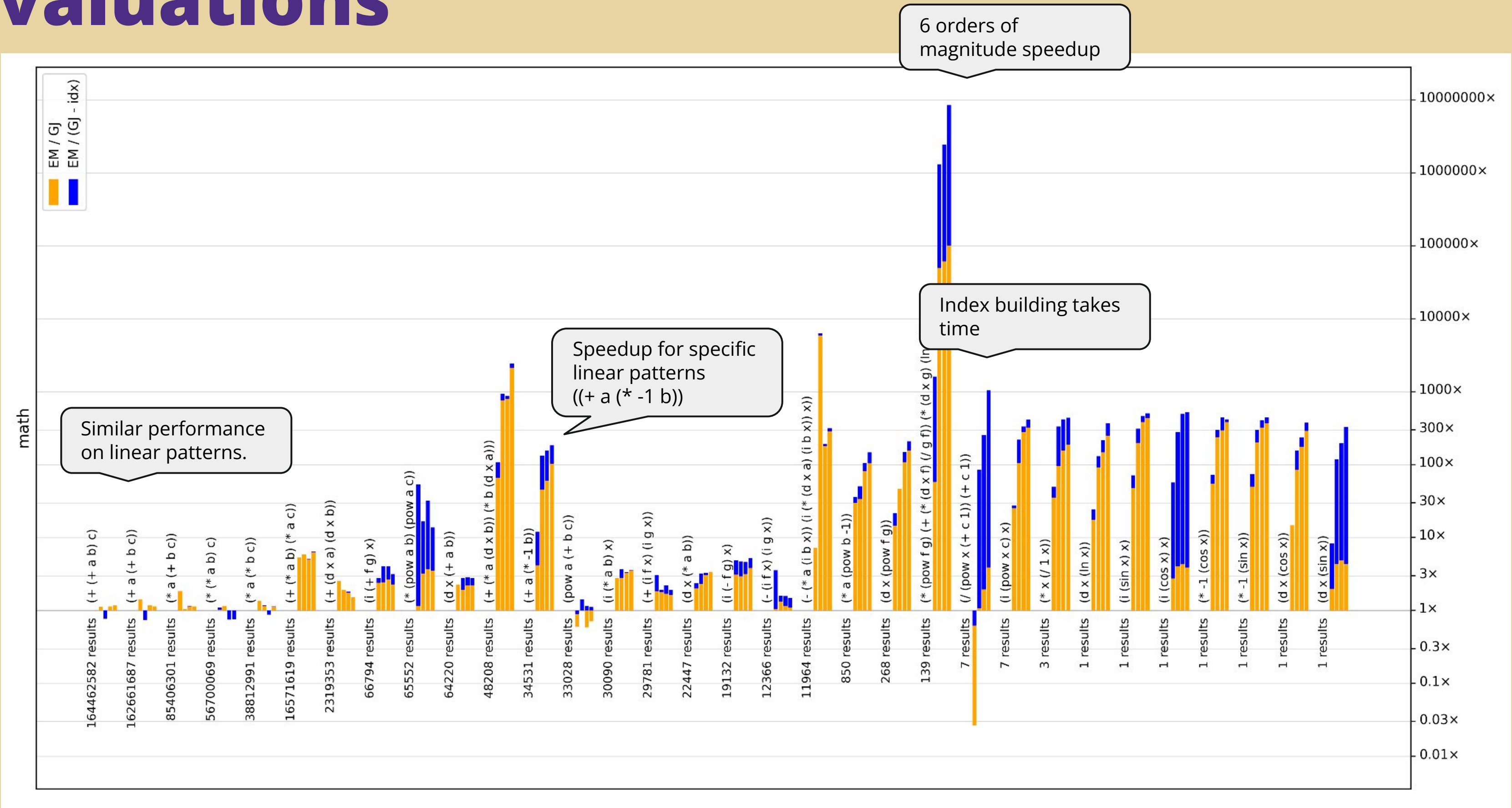
Overview

- 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



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!

$(f(\alpha, \beta), f(\beta, \gamma))$

$$\Downarrow$$

$Q(r1, r2, \alpha, \beta, \gamma)$:-
$R_f(r1, \alpha, \beta), R_f(r2, \beta, \gamma)$

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.
  - $f(g(a), h(a))$ translates to $Q(r, a) :- R_f(r, x, y), R_g(x, a), R_h(y, a)$, which has a worst-case complexity of $O(N^{2/3})$;
  - FD tells us a uniquely determines $g(a)$ and $h(a)$, and therefore $f(g(a), h(a))$; simply enumerating a 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.