UNIVERSITY OF CAMBRIDGE

Equality Saturation For Tensor Graph Superoptimization

Authors: Yang, Y., Phothilimthana, P. M., Wang, Y. R., Willsey, M., Roy, S., and Pienaar, J.

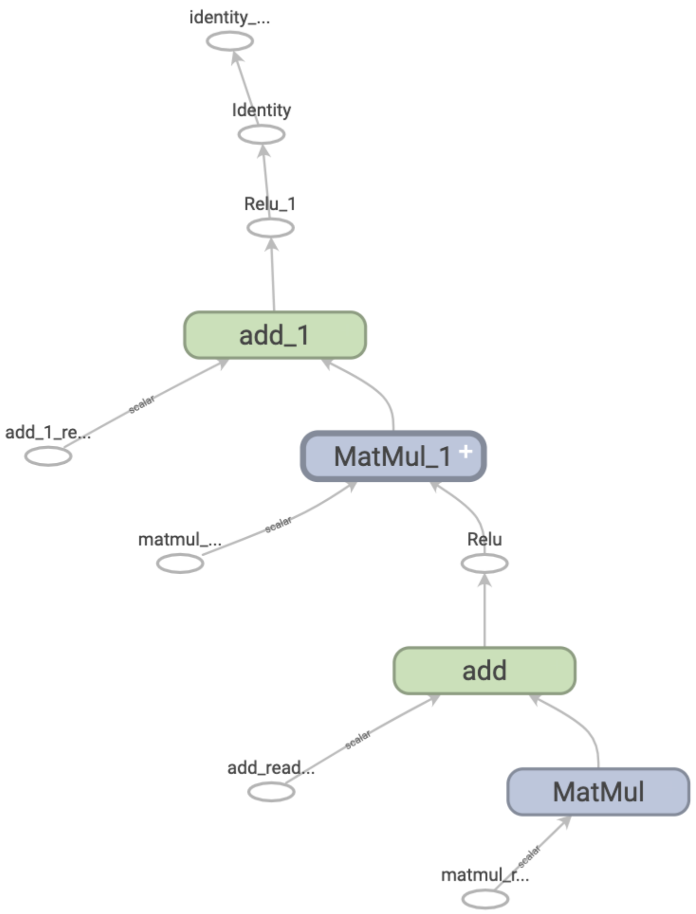
Year of Publication: 2021

Presenter: Hetong Shen

CRSid: HS899

TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.



TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

- Enumerate through potential substitutions of graphs and find the optimal one

TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

- Term Rewriting $(a \cdot 2)/2 \rightarrow a$

Useful

$(x \cdot y)/z = x \cdot (y/z)$

$x/x=1$

$(a \cdot 2)/2 \rightarrow a \cdot (2/2) \rightarrow a$ ☑

Not useful

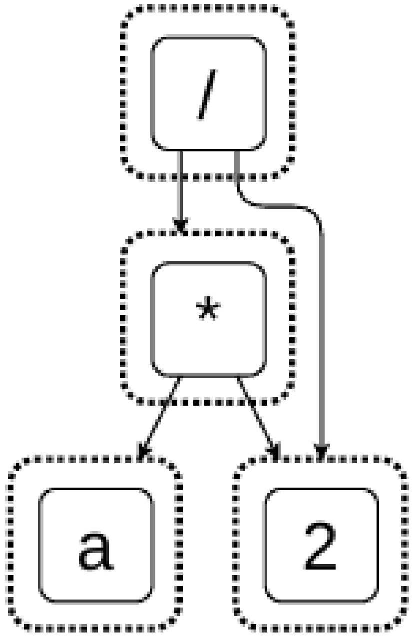
$x \cdot 2 = x \ll 1$

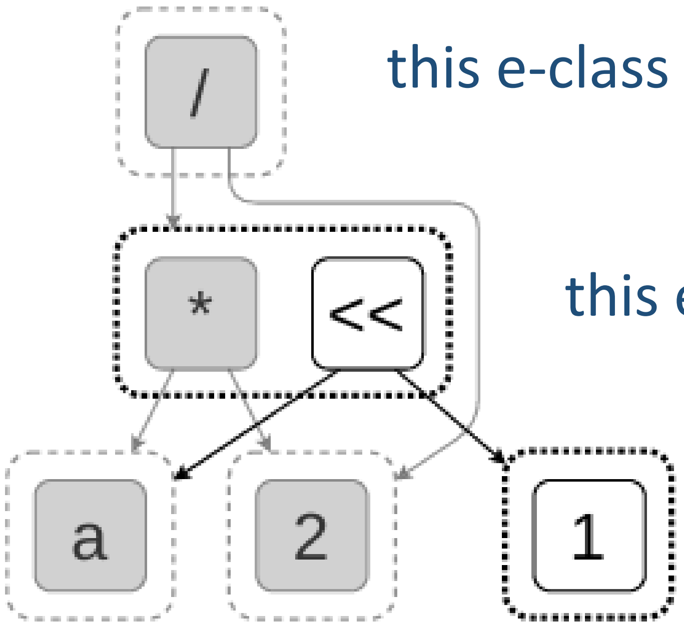
$x \cdot y = y \cdot x$

$(a \cdot 2)/2 \rightarrow (a \ll 1)/2$ ☒

TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

- E-graphs: (a·2)/2
- Term rewriting: (a·2)/2 → (a<<1)/2



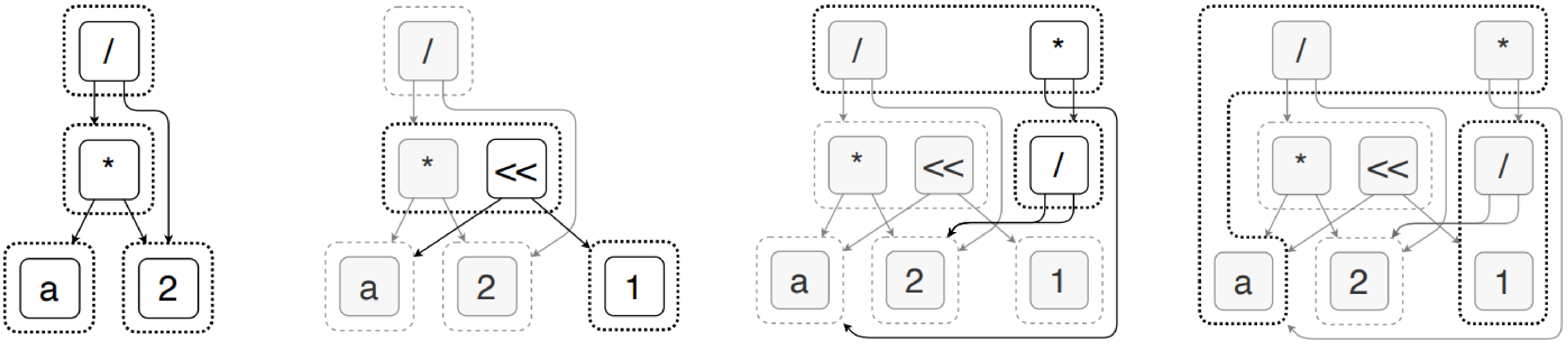


this e-class represents (a * 2) / 2 and (a << 1) / 2

this e-class represents (a * 2) and (a << 1)

TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

- Grow a E-graph



(a) Initial e-graph contains $(a \times 2)/2$.

(b) After applying rewrite $x \times 2 \rightarrow x \ll 1$.

(c) After applying rewrite $(x \times y)/z \rightarrow x \times (y/z)$.

(d) After applying rewrites $x/x \rightarrow 1$ and $1 \times x \rightarrow x$.

TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

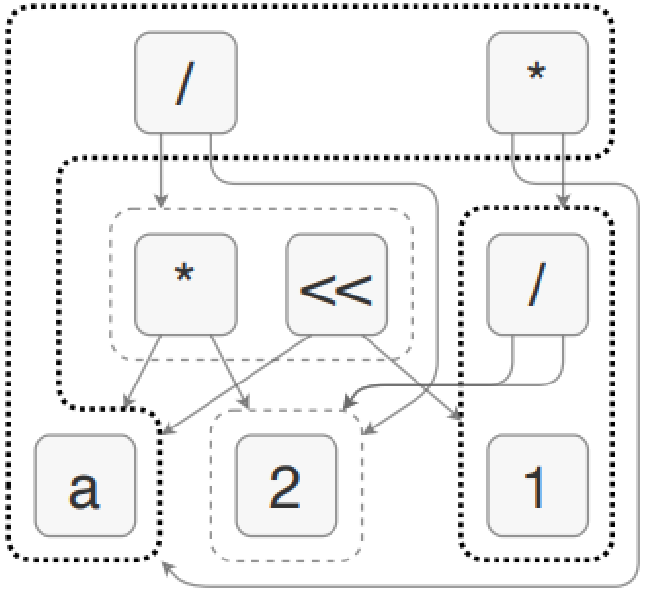
- Equality Saturation

$x \cdot 2 \rightarrow x \ll 1$

$(x \cdot y)/z \rightarrow x \cdot (y/z)$

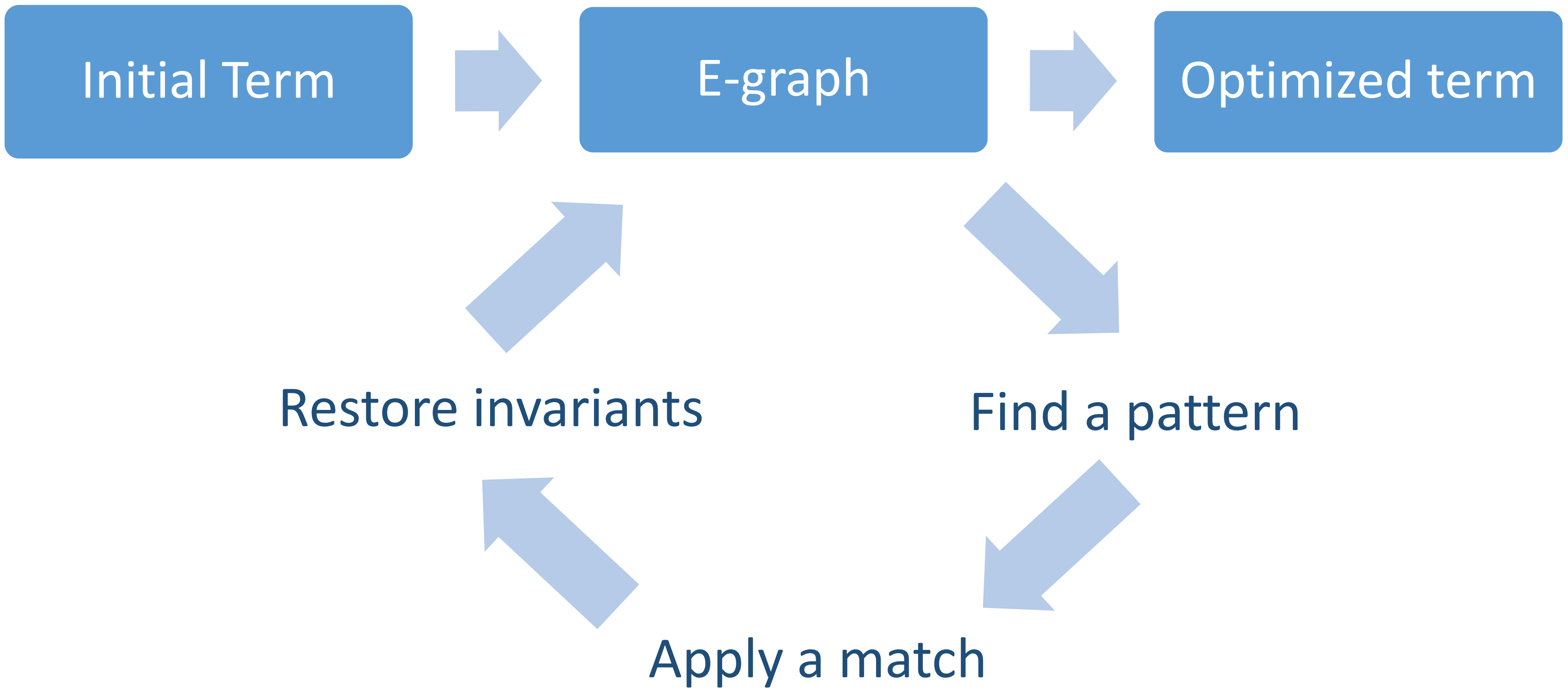
$x/x \rightarrow 1$

$x \cdot 1 \rightarrow x$



TENSAT, a tensor graph superoptimization framework that employs equality saturation on E-Graphs.

- Equality Saturation

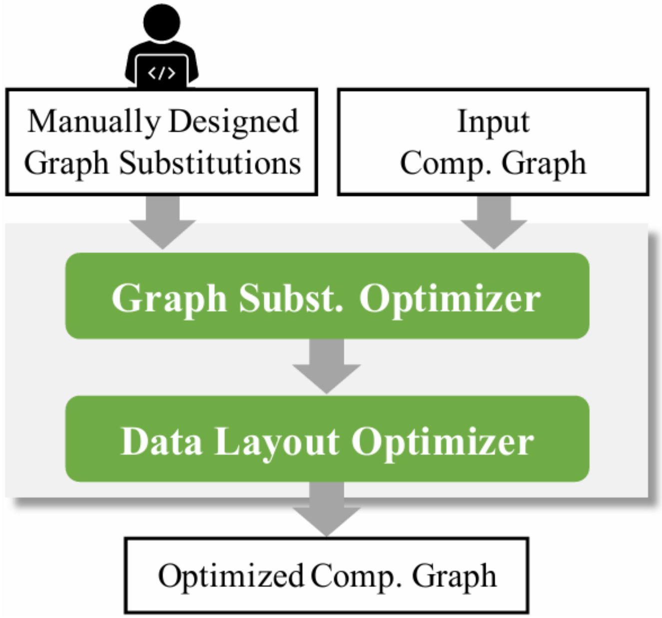


Challenges

- When doing graph rewriting to determine the order of applying the rewrite rules :
  - manually curated set of rewrite rules.
  - heuristic.

Sequential Substitution

- However, sequential substitution often leads to sub-optimal:
  - The non-comprehensive set of rewrite rules.
  - The sub-optimal graph substitution heuristic.
  - Rule choice problem



Existing Works

- Graph Rewrite Optimizations
  - TASO
  - NeuRewriter
- Superoptimization
  - Short sequences of low-level instructions
  - Denali
- Equality Saturation Applications
  - Optimize in other fields: ML, CAD simplification, Numerical Accuracy.

TENSAT:

Re-implementation of the TASO compiler using equality saturation

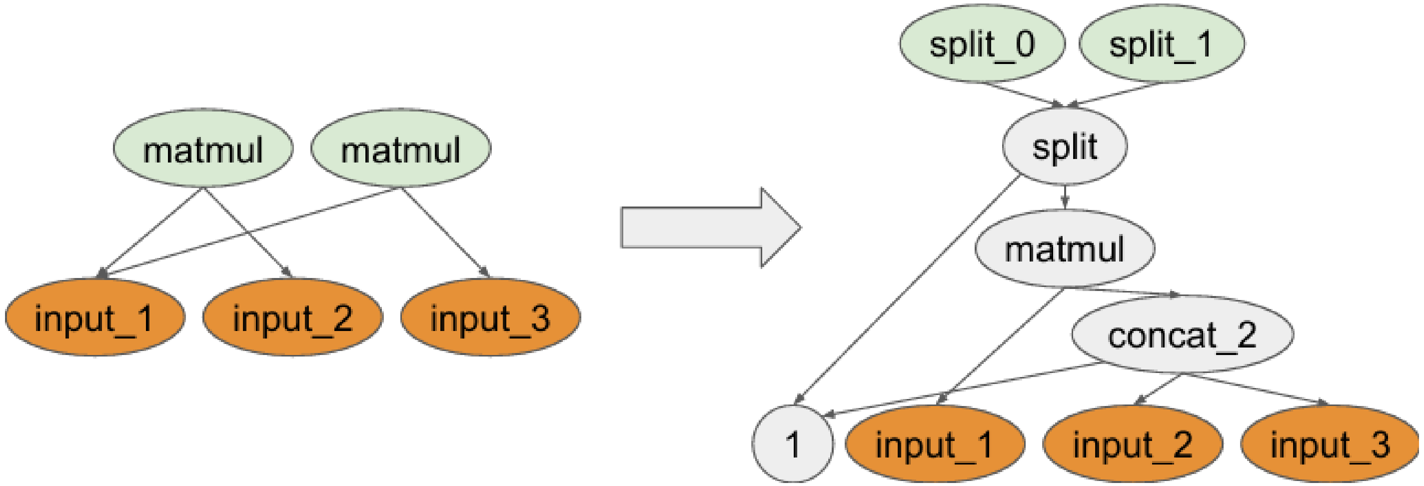
TENSAT's Representations

- Representing Tensor Computation Graphs

Operator	Description	Inputs	Type signature
ewadd	Element-wise addition	$\text{input}_1, \text{input}_2$	(T, T) $\rightarrow$ T
ewmul	Element-wise multiplication	$\text{input}_1, \text{input}_2$	(T, T) $\rightarrow$ T
matmul	Matrix multiplication	activation, $\text{input}_1, \text{input}_2$	(N, T, T) $\rightarrow$ T
conv [a]	Grouped convolution	$\text{stride}_h, \text{stride}_w$, pad., act., input, weight	(N, N, N, N, T, T) $\rightarrow$ T
relu	Relu activation	input	T $\rightarrow$ T
tanh	Tanh activation	input	T $\rightarrow$ T
sigmoid	Sigmoid activation	input	T $\rightarrow$ T
poolmax	Max pooling	input, $\text{kernel}_{\{h,w\}}$, $\text{stride}_{\{h,w\}}$, pad., act.	(T, N, N, N, N, N, N) $\rightarrow$ T
poolavg	Average pooling	input, $\text{kernel}_{\{h,w\}}$, $\text{stride}_{\{h,w\}}$, pad., act.	(T, N, N, N, N, N, N) $\rightarrow$ T
transpose [b]	Transpose	input, permutation	(T, S) $\rightarrow$ T
enlarge [c]	Pad a convolution kernel with zeros	input, ref-input	(T, T) $\rightarrow$ T
concat_n [d]	Concatenate	axis, $\text{input}_1, \ldots, \text{input}_n$	(N, T, ..., T) $\rightarrow$ T
split [e]	Split a tensor into two	axis, input	(N, T) $\rightarrow$ TT
split_0	Get the first output from split	input	TT $\rightarrow$ T
split_1	Get the second output from split	input	TT $\rightarrow$ T
merge [f]	Update weight to merge grouped conv	weight, count	(T, N) $\rightarrow$ T
reshape [g]	Reshape tensor	input, shape	(T, S) $\rightarrow$ T
input	Input tensor	identifier [h]	S $\rightarrow$ T
weight	Weight tensor	identifier [h]	S $\rightarrow$ T
noop [i]	Combine the outputs of the graph	$\text{input}_1, \text{input}_2$	(T, T) $\rightarrow$ T

TENSAT's Representations

- Representing Rewrite Rules
  - Single pattern rewrite rules
  - Multiple pattern rewrite rules



Source: (matmul ?input$_1$?input$_2$), (matmul ?input$_1$?input$_3$)
Target: (split$_0$ (split 1 (matmul ?input$_1$ (concat$_2$ 1 ?input$_2$?input$_3$)))),

(split$_1$ (split 1 (matmul ?input$_1$ (concat$_2$ 1 ?input$_2$?input$_3$))))

TENSAT

- Rule choice problem
    - Solution: first generates all rewritten terms, leaving the choice of which term to select to the extraction procedure
- Exploration Phase
- Extraction Phase

Exploration Phase

- Search for matches of all rewrite rules in the current e-graph, and add the target patterns and equivalence relations to the e-graph
    - Single pattern rewrite rules and Multiple pattern rewrite rules

Algorithm 1 Applying multi-pattern rewrite rules

Input: starting e-graph $\mathcal{G}$, set of multi-pattern rewrite rules $\mathcal{R}_m$.
Output: updated e-graph $\mathcal{G}$.

canonicalized S-expr e_c = Set({})
for rule $r \in \mathcal{R}_m$ **do**
    for $i = 0, \ldots, |r| - 1$ **do** ▷ $|r|$: #S-exprs in source pattern
        (e, rename_map) = CANONICAL(r.source[i])
        e_c.insert(e)
        r.map[i] = rename_map
    end for
end for
for iter = 0, ..., MAX_ITER **do**
    M = SEARCH($\mathcal{G}, e_c$) ▷ all matches for all patterns
    for rule $r \in \mathcal{R}_m$ **do**
        for $i = 0, \ldots, |r| - 1$ **do**
            canonical matches mc_i = M[r.source[i]]
            matches m_i = DECANONICAL(mc_i, r.map[i])
        end for
        for $(\sigma_0, \ldots, \sigma_{|r|-1}) \in \mathrm{m}_0 \times \cdots \times \mathrm{m}_{|r|-1}$ **do**
            if COMPATIBLE($(\sigma_0, \ldots, \sigma_{|r|-1})$) **then**
                APPLY($\mathcal{G}, r, \sigma_0, \ldots, \sigma_{|r|-1}$)
            end if
        end for
    end for
end for
return $\mathcal{G}$

Extraction Phase – 1st Approach Greedy

- Cost Model
- Greedy Extraction:
  - For each e-class, computes the total cost of the subtrees rooted on each of the e-nodes, and picks the e-node with the smallest subtree cost
  - Not guaranteed to extract the graph with the minimum cost

Extraction Phase – 2nd Approach ILP

- ILP Extraction:
  - Objective function and constraints

$$\text{Minimize: } f(x) = \sum_i c_i x_i$$

Subject to:

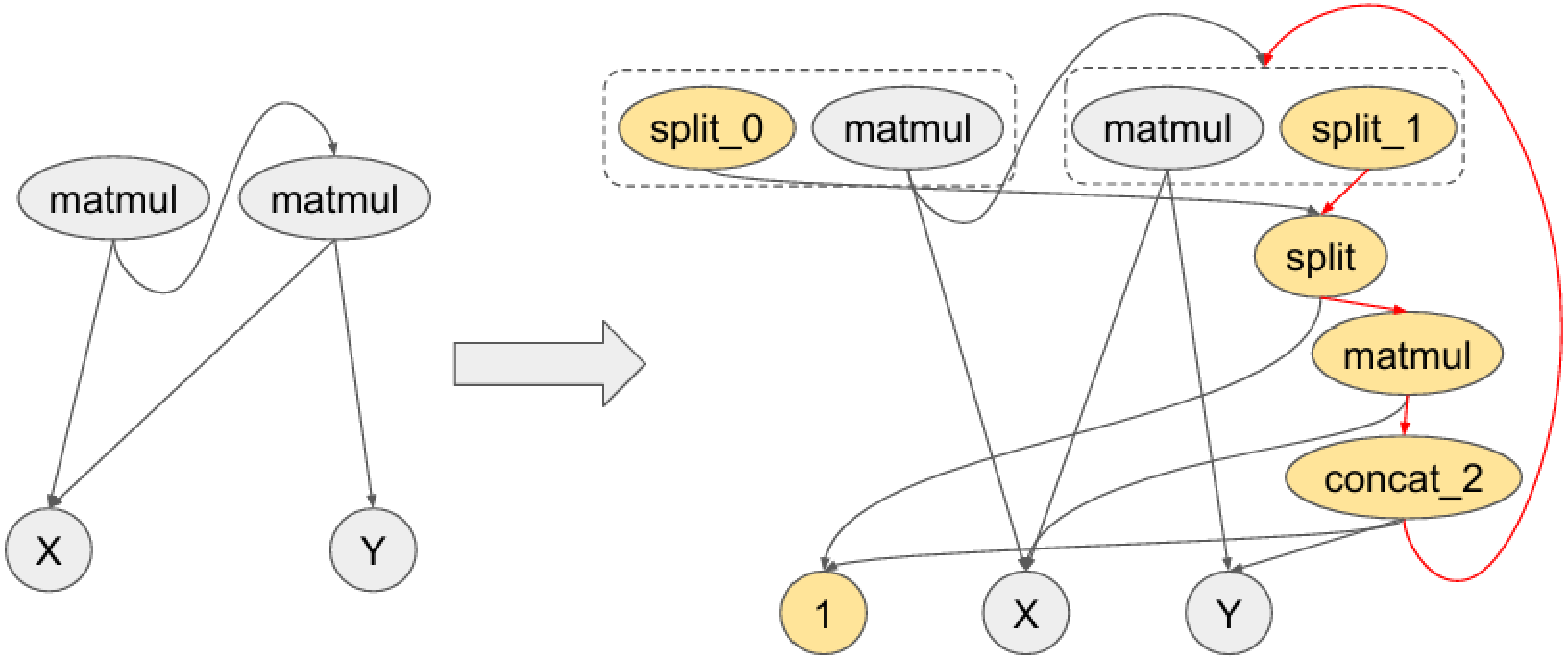
$$\boxed{\begin{aligned} & x_i \in \{0,1\}, && (1)\\ & \sum_{i \in e_0} x_i = 1, && (2)\\ & \forall i, \forall m \in h_i, x_i \leq \sum_{j \in e_m} x_j, && (3) \end{aligned}}$$

$$\forall i, \forall m \in h_i, t_{g(i)} - t_m - \epsilon + A(1 - x_i) \geq 0, \quad (4)$$

$$\forall m, 0 \leq t_m \leq 1, \quad (5)$$

Extraction Phase – 2nd Approach ILP

- ILP Extraction:
    - Objective function and constraints
    - Cycles



Extraction Phase – 2nd Approach ILP

- ILP Extraction:
  - Objective function and constraints
  - Have cycles vs. no cycles

$$\text{Minimize: } f(x) = \sum_i c_i x_i$$

Subject to:

$$x_i \in \{0, 1\}, \tag{1}$$

$$\sum_{i \in e_0} x_i = 1, \tag{2}$$

$$\forall i, \forall m \in h_i, x_i \leq \sum_{j \in e_m} x_j, \tag{3}$$

$$\boxed{\begin{aligned} &\forall i, \forall m \in h_i, t_{g(i)} - t_m - \epsilon + A(1 - x_i) \geq 0, && (4) \\ &\forall m, 0 \leq t_m \leq 1, && (5) \end{aligned}}$$

Extraction time (s)	k_{multi}	With cycle real	With cycle int	Without cycle
BERT	1	0.96	0.98	**0.16**
	2	>3600	>3600	**510.3**
NasRNN	1	1116	1137	**0.32**
	2	>3600	>3600	**356.7**
NasNet-A	1	424	438	**1.81**
	2	>3600	>3600	**75.1**

Table 5. Effect of whether or not to include cycle constraints in ILP on extraction time (in seconds), on BERT, NasRNN, and NasNet-A. For the cycle constraints, we compare both using real variables and using integer variables for the topological order variables t_m.

Extraction Phase – Comparison

- Greedy vs. ILP Extraction:
  - Greedy extraction is slow: it makes the choices on which node to pick separately and greedily, without considering the interdependencies between the choices.
  - ILP Guaranteed to give a valid graph (no cycles) with the lowest cost

Graph Runtime (ms)	Original	Greedy	ILP
BERT	1.88	1.88	**1.73**
NasRNN	1.85	1.15	**1.10**
NasNet-A	17.8	22.5	**16.6**

Bottle Neck and Cycle Filtering

- Vanilla cycle filtering:
- Efficient cycle filtering in exploration phase:
    - Pre-filtering
    - Post processing

Algorithm 2 Exploration phase with efficient cycle filtering

Input: starting e-graph $\mathcal{G}$, set of rewrite rules $\mathcal{R}$.
Output: updated e-graph $\mathcal{G}$, filter list l

$l = \{\}$
for iter = 0, . . . , MAX_ITER **do**
  descendants map d = GETDESCENDANTS($\mathcal{G}, l$)
  matches = SEARCH($\mathcal{G}, \mathcal{R}, l$)
  for match $\in$ matches **do**
    if not WILLCREATECYCLE(match, d) **then**
      APPLY($\mathcal{G}$, match)
    end if
  end for
  while true **do**
    cycles = DFSGETCYCLES($\mathcal{G}, l$)
    if len(cycles) == 0 **then**
      break
    end if
    for cycle $\in$ cycles **do**
      RESOLVECYCLE($\mathcal{G}, l$, cycle)
    end for
  end while
end for
return $\mathcal{G}, l$

Bottle Neck and Cycle Filtering

- Vanilla cycle filtering vs. Efficient cycle filtering

Exploration time (s)	k_{multi}	Vanilla	Efficient
BERT	1	0.18	**0.17**
	2	32.9	**0.89**
NasRNN	1	1.30	**0.08**
	2	2932	**1.47**
NasNet-A	1	3.76	**1.27**
	2	>3600	**8.62**

Table 6. Comparison between vanilla cycle filtering and efficient cycle filtering, on the exploration phase time (in seconds) for BERT, NasRNN, and NasNet-A.

Evaluation – Set Up

- TENSAT Implementation:
    - Developed in Rust
    - Equality saturation library egg
- ILP solver:
    - Utilized SCIP

Evaluation – Set Up

The models evaluated:

- BERT (Devlin et al., 2019)
- ResNeXt-50 (Xie et al., 2017)
- NasNet-A (Zoph et al., 2018)
- NasRNN (Zoph & Le, 2017)
- Inception v3 (Szegedy et al., 2016)
- VGG-19 (Liu & Deng, 2015)
- SqueezeNet (Iandola et al., 2017)

- Limit the number of nodes in the e-graph Nmax = 50000
- Limit number of iterations for exploration kmax = 15

Evaluation – Speed Up

- TASO vs TENSAT
- Equality saturation covers a much larger space of equivalent graphs than sequential backtracking search.
- K: K multi
- Inception: Optimizer can achieve a better speedup given longer optimization time.

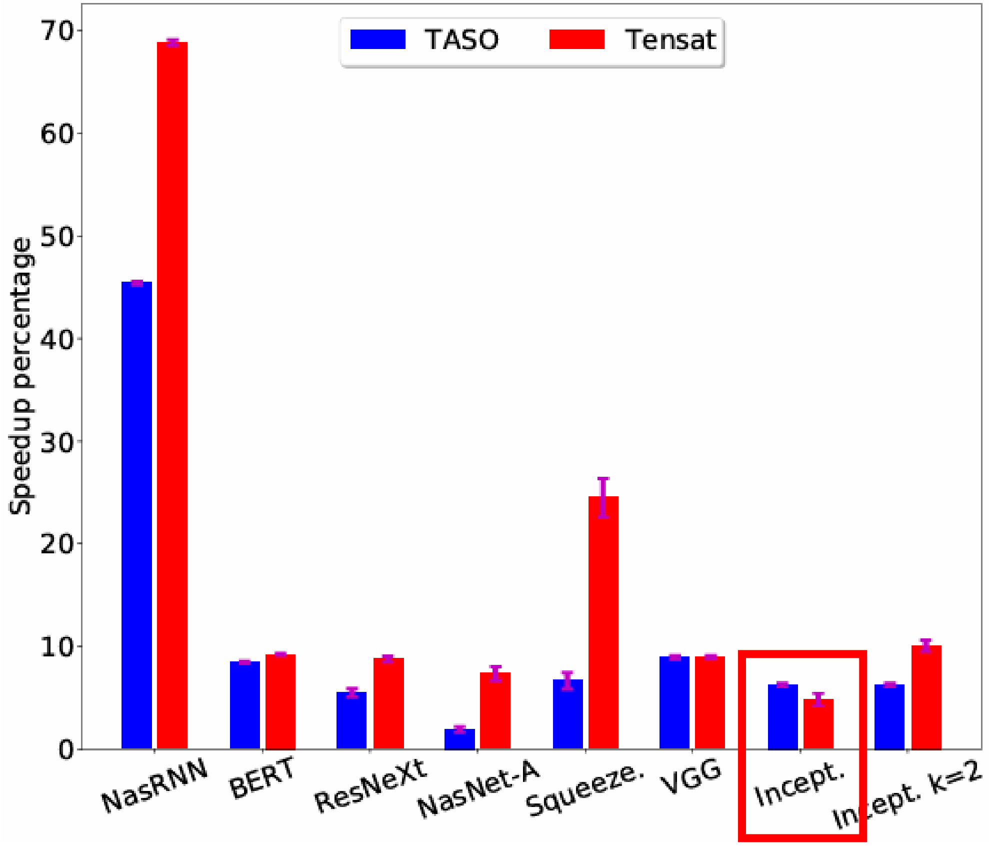


Figure 4. Speedup percentage of the optimized graph with respect to the original graph: TASO v.s. TENSAT. Each setting (optimizer $\times$ benchmark) is run for five times, and we plot the mean and standard error for the measurements.

Evaluation – Optimization Time

- TASO vs TENSAT
- TENSAT can not only cover a much larger search space, but also in less time

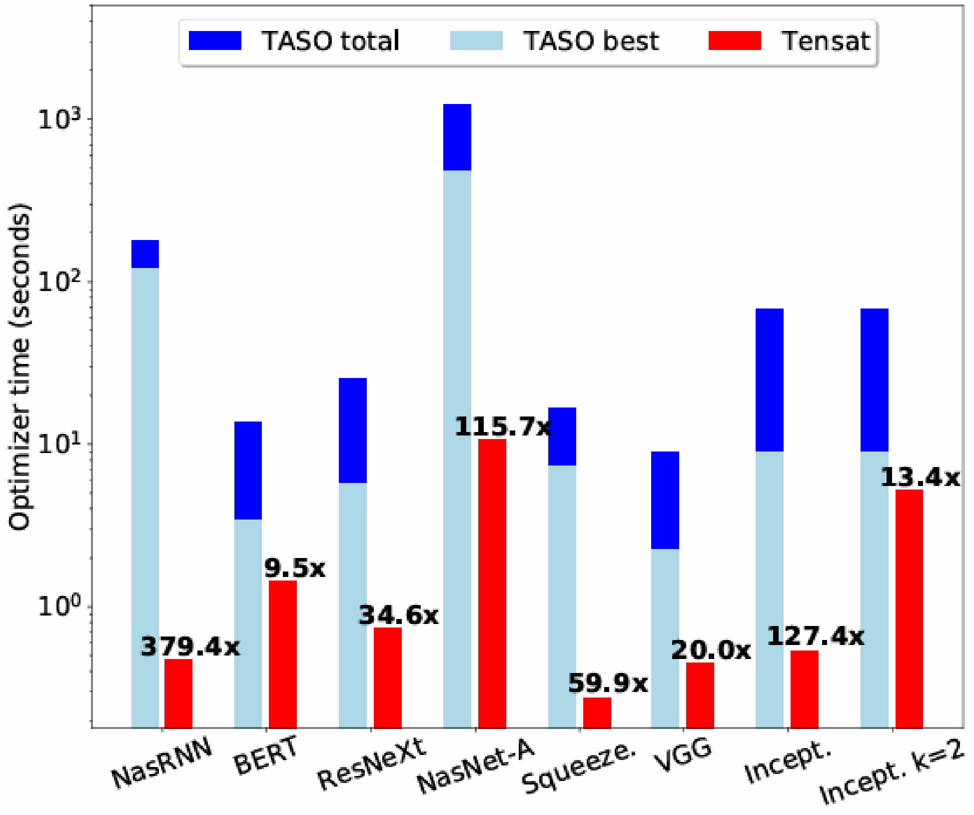
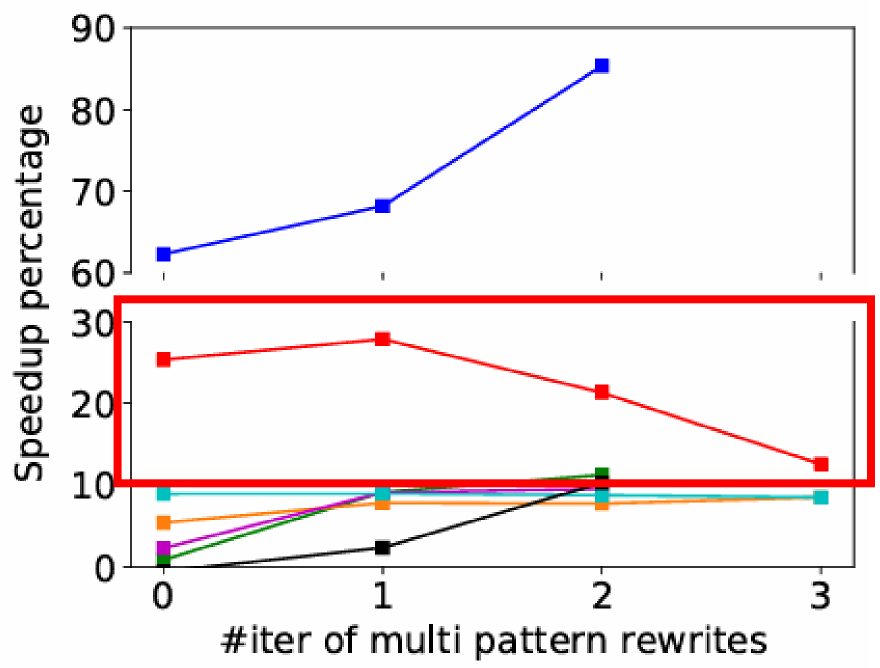
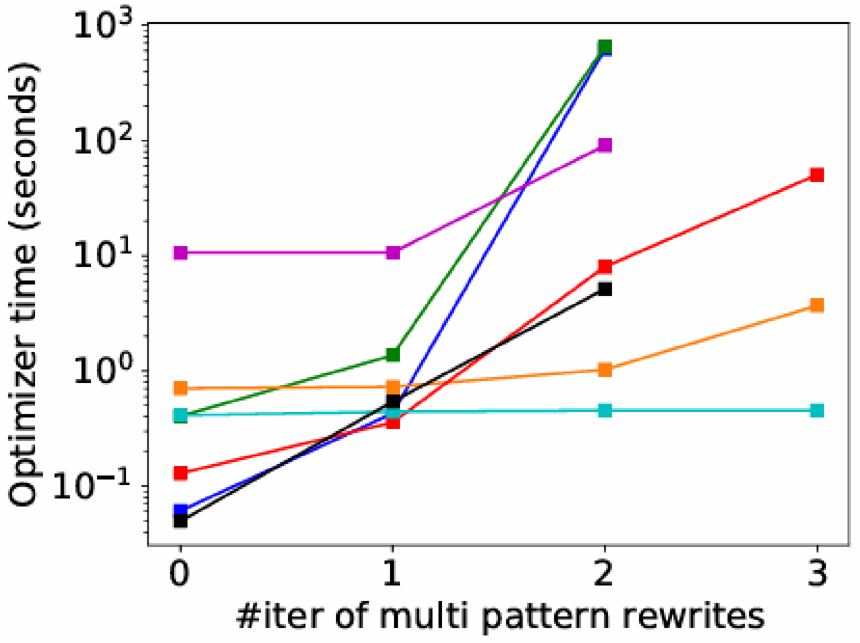


Figure 5. Optimization time (log scale): TASO v.s. TENSAT. "TASO total" is the total time of TASO search. "TASO best" indicates when TASO found its best result; achieving this time would require an oracle telling it when to stop.

Evaluation – Varying Iterations of Multi-Pattern Rewrites

- Effect of varying the number of iterations of multi-pattern rewrites $k$multi
- Squeeze-Net: discrepancy between the cost model and the real graph runtime.





Novelty

- Uses e-graph for tensor graph superoptimization
- Introduces multi pattern write rules
- Efficient cycle filtering in exploration phase

Downside

- Limitation in Scalability:
    - Multi-pattern rules for tensor graph: grow the e-graph extremely rapidly
    - Can only explore up to a certain number of iterations of multi-pattern rewrites.
    - E-graph becomes too large for the extraction phase
- Parallelism:
    - Uses cost model as TASO, which is suitable for GPU (one operator when executing graph)

Impact and Future directions

- Tackle Limitation in Scalability:
  - Selectively apply rules during exploration
  - Utilize ML techniques
- Achieve Parallelism:
  - Some hardware may execute multiple kernels in parallel
  - Needs a different cost model, such as a learned method to perform extractions
- Applications:
  - TENSAT's optimization time is small enough that can be integrated into a <u>default</u> compilation flow