Equality Saturation For Tensor Graph Superoptimisation [5]

Authors: Yichen Yang, Phitchaya Mangpo Phothilimthana, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar.
Gooogle and MIT CSAIL

Presented by Umer Hasan (suh25)

Michelmas 2025

UNIVERSITY OF CAMBRIDGE

Background on tensor graph rewriting and equality saturation

- Tensor graph g and semantics-preserving rewrites R to find lower cost g'.
- Equality saturation [4] is a compiler optimisation technique.

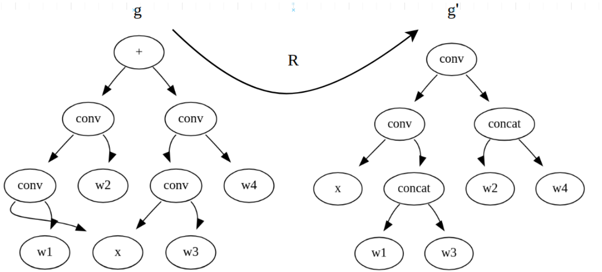


Figure 1: A rewrite useful for NasNet-A [6].
Left: original. Right: optimised.
The dimensions are (`out_channels`, `in_channels`, `height`, `width`) and the operations are `concat(w1, w3, axis=0)`, `concat(w2, w4, axis=1)`

UNIVERSITY OF CAMBRIDGE

Motivation

What is it?

TENSAT is a tensor graph superoptimisation framework.

Motivation

What is it?

TENSAT is a tensor graph superoptimisation framework.

Why do we need it?

Motivation

What is it?

TENSAT is a tensor graph superoptimisation framework.

Why do we need it?

- Runtime speedup of computation.

UNIVERSITY OF CAMBRIDGE

Motivation

What is it?

TENSAT is a tensor graph superoptimisation framework.

Why do we need it?

- Runtime speedup of computation.
- Optimisation time speedup.

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.

UNIVERSITY OF CAMBRIDGE

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.
- Naively generating all sequences requires exponential time and space.

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.
- Naively generating all sequences requires exponential time and space.

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.
- Naively generating all sequences requires exponential time and space.

Approach: Equality Saturation

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.
- Naively generating all sequences requires exponential time and space.

Approach: Equality Saturation

- Uses an E-graph to store all equivalent versions simultaneously.

UNIVERSITY OF CAMBRIDGE

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.
- Naively generating all sequences requires exponential time and space.

Approach: Equality Saturation

- Uses an E-graph to store all equivalent versions simultaneously.
- Rewrites are additive: $g_i \to g_i'$ means g_i and g_i' exist in the same equivalence class.

Features: Equality Saturation (ES)

Problem: The Phase Ordering Challenge

- Traditional graph rewriting is sequential and destructive.
- Naively generating all sequences requires exponential time and space.

Approach: Equality Saturation

- Uses an E-graph to store all equivalent versions simultaneously.
- Rewrites are additive: $g_i \rightarrow g_i'$ means g_i and g_i' exist in the same equivalence class.
- Rules apply until saturation (monotonic growth), there is no backtracking.

UNIVERSITY OF CAMBRIDGE

An e-graph has e-classes and e-nodes.

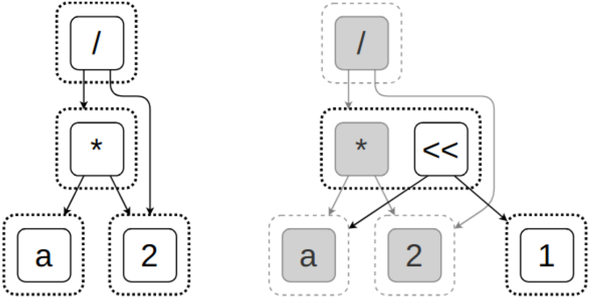


Figure 1. Left: An e-graph representing the term $(a \times 2)/2$. Dotted boxes show e-classes, and arrows connect e-nodes to their e-class children. *Right:* The e-graph after applying the rewrite $x \times 2 \to x \ll 1$. Only a few e-nodes were added (highlighted in white), and the result represents both the initial and rewritten terms.

Features: Extraction

Problem: Greedy Extraction Inefficiency

For each e-class, calculate subtree costs for each e-node, select cheapest e-node. Not globally optimal. No subgraph sharing. Computing subtree costs is an arduous process.

Features: Extraction

Problem: Greedy Extraction Inefficiency

For each e-class, calculate subtree costs for each e-node, select cheapest e-node. Not globally optimal. No subgraph sharing. Computing subtree costs is an arduous process.

Approach: Global Integer Linear Program (ILP) Extraction

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

Subject to:

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

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

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

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

$$\forall m, 0 \leq t_m \leq 1, \tag{5}$$

with i nodes, m e-classes, cost c_i, indicator var x_i, children h_i and e-class g_i.

Features: Efficient Cycle Filtering

Problem: Cycles / Deadlock

E-Graphs represent equality ($A = B$) (bidirectional, not DAG). "Check and reject" is inefficient.

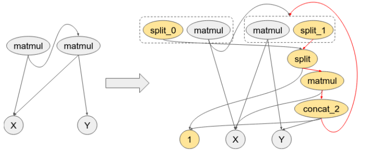
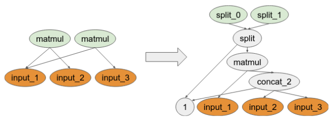


Figure 3. Example on how a valid rewrite can introduce cycles into the e-graph. RHS is the resulting e-graph after applying the rewrite rule from Figure 2 to the LHS. Dotted lines circles the e-classes. We omit the e-classes with a single node for clarity. If the node split_1 is picked in the right e-class, then the resulting graph will have a cycle (indicated by the red edges).



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$_3$))))

Features: Efficient Cycle Filtering

Problem: Cycles / Deadlock

E-Graphs represent equality ($A = B$) (bidirectional, not DAG). "Check and reject" is inefficient.

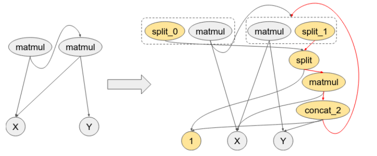
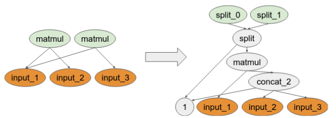


Figure 3. Example on how a valid rewrite can introduce cycles into the e-graph. RHS is the resulting e-graph after applying the rewrite rule from Figure 2 to the LHS. Dotted lines circles the e-classes. We omit the e-classes with a single node for clarity. If the node split_1 is picked in the right e-class, then the resulting graph will have a cycle (indicated by the red edges).



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$_3$))))

Approach: ILP Constraints

Topological sort with t_m guarantees acyclic execution.

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

Subject to:

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

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

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

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

$$\forall m, 0 \leq t_m \leq 1, \tag{5}$$

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**

UNIVERSITY OF CAMBRIDGE

Results

Top-line: 16% speedup over TASO [1]. **48x** less time optimising.

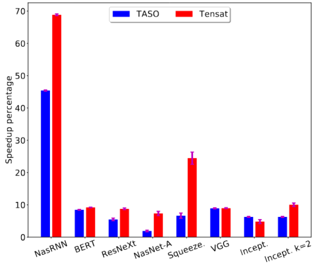


Figure 2: Speedup percentage of the optimised graph, mean and std of 5 runs.

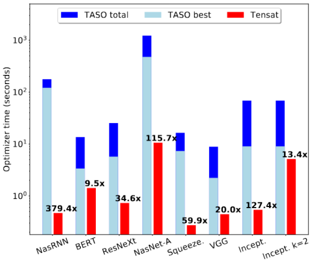


Figure 3: Optimisation time. TASO Best indicates when its found the best result during search.

My Thoughts

- Paper demonstrates high clarity.

My Thoughts

- Paper demonstrates high clarity.
- E-graph can expand indefinitely: $x- > x + 0$ (max. 50k iter).

My Thoughts

- Paper demonstrates high clarity.
- E-graph can expand indefinitely: $x- > x + 0$ (max. 50k iter).
- Blindly applying every rule consumes a lot of memory. Enter Omelette (RL) [3] or MCTS.

My Thoughts

- Paper demonstrates high clarity.
- E-graph can expand indefinitely: $x- > x + 0$ (max. 50k iter).
- Blindly applying every rule consumes a lot of memory. Enter Omelette (RL) [3] or MCTS.
- Can they start extracting and exploring at the same time?

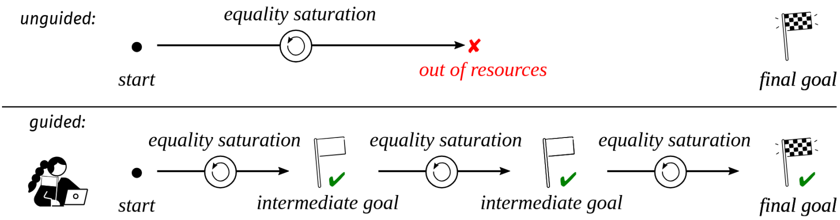


Figure 4: Guided equality saturation [2]

My Thoughts 2

- Why did the authors compare only with TASO (cost-based backtracking)? TensorFlow isn't a superoptimiser.

My Thoughts 2

- Why did the authors compare only with TASO (cost-based backtracking)? TensorFlow isn't a superoptimiser.
- I think a future work section would have been useful.

My Thoughts 2

- Why did the authors compare only with TASO (cost-based backtracking)? TensorFlow isn't a superoptimiser.
- I think a future work section would have been useful.
- No demonstration of integration into production. Maybe this is outside the scope of the research paper.

References I

Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken.
Taso: optimizing deep learning computation with automatic generation of graph substitutions.
In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 47–62, New York, NY, USA, 2019. Association for Computing Machinery.

T. Kundefinedhler, A. Goens, S. Bhat, T. Grosser, P. Trinder, and M. Steuwer.
Guided equality saturation.
Proc. ACM Program. Lang., 8(POPL), Jan. 2024.

Z. Singh.
Deep reinforcement learning for equality saturation., 2022.

References II

R. Tate, M. Stepp, Z. Tatlock, and S. Lerner.
Equality saturation: A new approach to optimization.
Logical Methods in Computer Science, Volume 7, Issue 1, Mar. 2011.

Y. Yang, P. M. Phothilimtha, Y. R. Wang, M. Willsey, S. Roy, and J. Pienaar.
Equality saturation for tensor graph superoptimization, 2021.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le.
Learning transferable architectures for scalable image recognition, 2018.

Q & A