Equality Saturation for Tensor Graph Optimization

Authors: Yichen Yang, Phitchaya Mangpo Phothilimthana, Yisu Remy Wang, Max Willsey, Sudip Roy, Jacques Pienaar

Presenter: Ken Jin

Motivation

AI/Tensor compilers often have multiple destructive optimization passes

Problems:

1. Optimization opportunities are pass-dependent
2. Multiple passes can be slow and do repeated work

Motivating Example

`(a * 2) / 2` should rewrite to a

Rewrite rules:

1. Strength reduction: `a * n` →`a << log2(n)`, where `n` is a power of 2
2. Algebraic simplification: `na / n` → `a`

`(a * 2) / 2`

Motivating Example

`(a * 2) / 2` should rewrite to a

Rewrite rules:

1. Strength reduction: `a * n` →`a << log2(n)`, where `n` is a power of 2
2. Algebraic simplification: `na / n` → `a`

`(a * 2) / 2`

Motivating Example

`(a * 2) / 2` should rewrite to a

Rewrite rules:

1. Strength reduction: `a * n` →`a << log2(n)`, where `n` is a power of 2
2. Algebraic simplification: `na / n` → `a`

`(a * 2) / 2`

→ `(a << 1) / 2`

Motivating Example

`(a * 2) / 2` should rewrite to a

Rewrite rules:

1. Strength reduction: `a * n` →`a << log2(n)`, where `n` is a power of 2
2. Algebraic simplification: `na / n` → `a`

`(a * 2) / 2`

→ `(a << 1) / 2`

→ **`blocked!`**

Motivating Example

`(a * 2) / 2` should rewrite to a

Rewrite rules:

1. Strength reduction: `a * n` →`a << log2(n)`, where `n` is a power of 2
2. Algebraic simplification: `na / n` → `a`

`(a * 2) / 2`

→ `(a << 1) / 2`

→ **`blocked!`**

Solution

Key insight: In traditional compilers, it is possible to keep rewrite candidates around without destructively changing things, using *equality saturation*. Apply this traditional technique to tensor graph optimizers (this part is a novel application).

Intuitively: keep the candidates around and search for an “optimal” program.

Solution

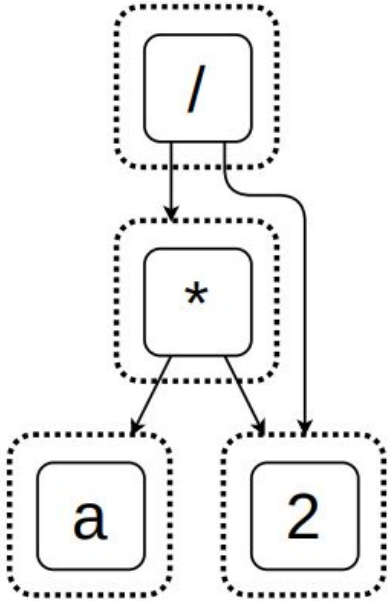
e-graphs data structures:

- Nodes in the tensor graph = Equivalence nodes (e-nodes)
- Equivalent set of nodes = Equivalence classes (e-classes)

Equality saturation and its phases:

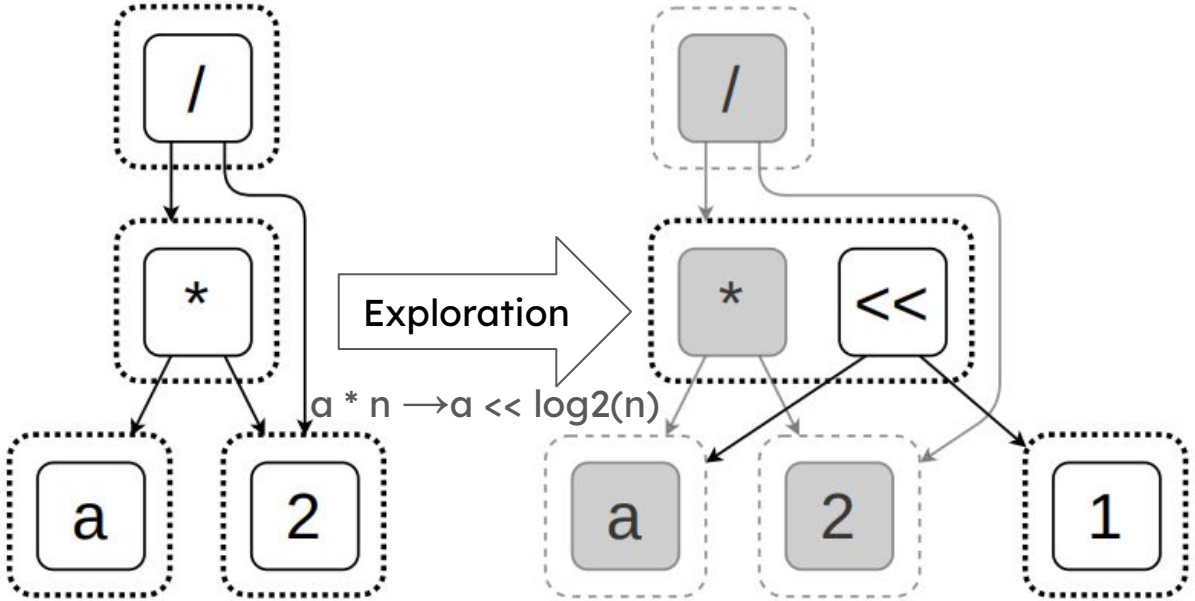
1. Exploration (building the e-graph)
2. Extraction (choosing the optimal final program via a cost model)

e-graph exploration



Dotted boxes: e-classes
Edges: Connects operators to operands

e-graph exploration



Dotted boxes: e-classes
Edges: Connects operators to operands

e-graph exploration

Problem: search space can grow exponentially with multi-pattern rewrites (Rewrite rules that have more than output)

Solution: limit number of iterations you can apply multi-pattern rewrite rules. Limit maximum nodes in the e-graph.

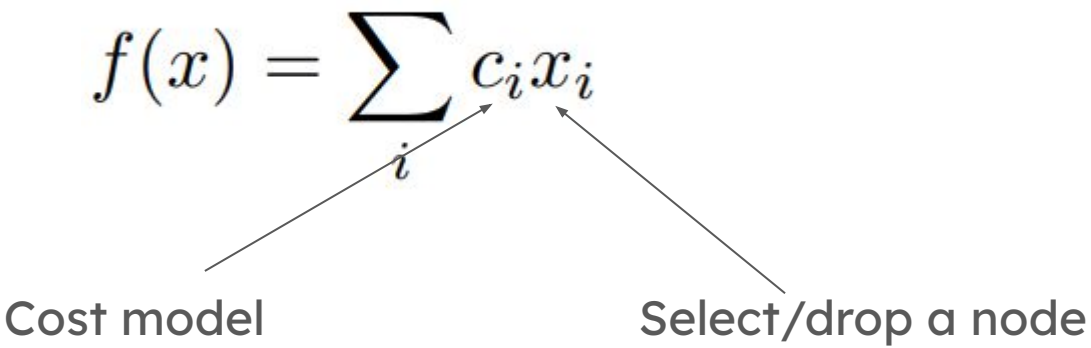
e-graph extraction

Pass the nodes and selection + cost model as a set of constraints to an Integer Linear Programming (ILP) solver. Goal is to minimize this function:

$$f(x) = \sum_i c_i x_i$$

e-graph extraction

Pass the nodes and selection + cost model as a set of constraints to an Integer Linear Programming (ILP) solver. Goal is to minimize this function:



e-graph extraction

Subject to some constraints:

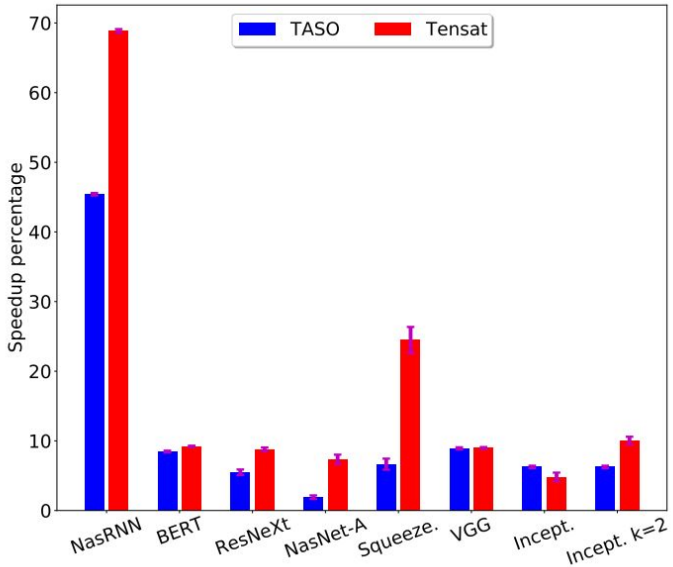
1. No cycles
2. Valid topological order
3. Etc…

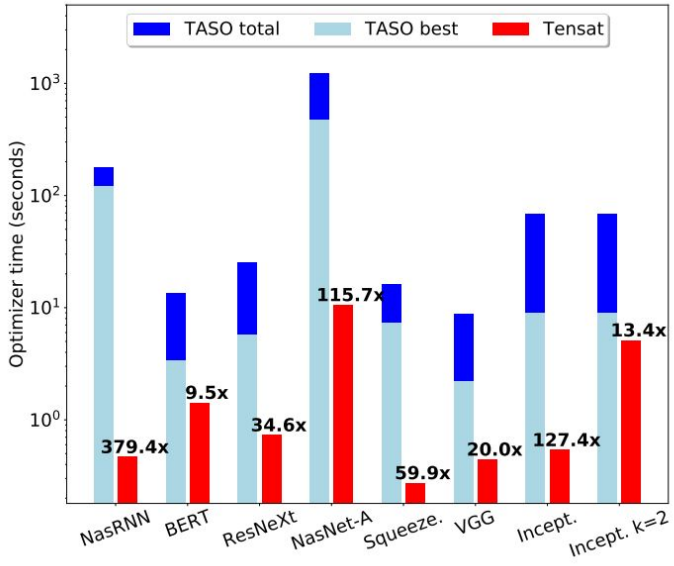
e-graph extraction

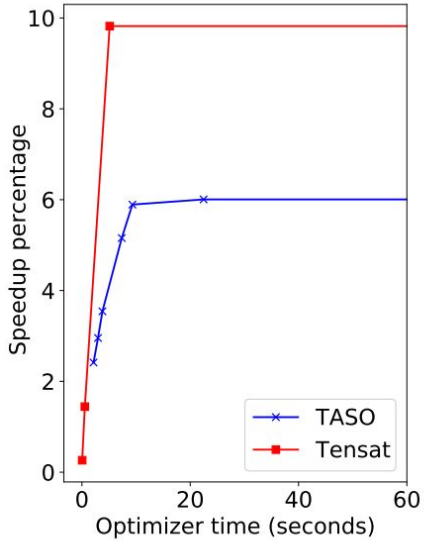
Novel e-graph extraction speedups: cycles slow down the ILP solver. Prevent cycles forming in the first place during e-graph exploration!

- Pre-filter (prevent cycles, sound but not complete)
- Run 1 step
- Post-process (detect and break any new cycles)

Results


TASO
Tensat
Speedup percentage
0
10
20
30
40
50
60
70
NasRNN
BERT
ResNeXt
NasNet-A
Squeeze.
VGG
Incept.
Incept. k=2


TASO total
TASO best
Tensat
Optimizer time (seconds)
10^0
10^1
10^2
10^3
379.4x
9.5x
34.6x
115.7x
59.9x
20.0x
127.4x
13.4x
NasRNN
BERT
ResNeXt
NasNet-A
Squeeze.
VGG
Incept.
Incept. k=2


Speedup percentage
0
2
4
6
8
10
TASO
Tensat
0
20
40
60
Optimizer time (seconds)

Related Work

1. TASO (Jia et. al.) — cost-based backtracking search on a tensor graph
    a. Relatively slower at search than this work, also worse optimization decisions
2. Optimizing DNN Computation Graph using Graph Substitutions (Fang et. al.) — sampling-based approximation with limited search
    a. This paper did not aim to improve runtime performance/optimization decisions
3. Superoptimization (many papers) — Find fastest equivalent program with same outputs
    a. Often too slow to be practical on larger programs
4. Equality saturation in other compilers