Improving Term Extraction with Acyclic Constraints

Mike He
Princeton University
Princeton, NJ USA
dh7120@cs.princeton.edu

Haichen Dong
Princeton University
Princeton, NJ USA
hd5234@cs.princeton.edu

Sharad Malik
Princeton University
Princeton, NJ USA
sharad@princeton.edu

Aarti Gupta
Princeton University
Princeton, NJ USA
aartig@cs.princeton.edu

Abstract

Term extraction is a crucial workload in egg for determining the desired terms to be extracted. Some prior works have formulated term extraction as integer linear programming (ILP) problems in order to cope with common sub-expressions for optimality that could not be handled by greedy algorithms. Although ILP-based extraction algorithms ensure optimality, these formulations offload a topological sorting problem to the ILP solver to avoid extracting cyclic terms, which does not scale well with the complexity of cycles in the e-graph. Instead of enforcing topological orders with constraints, we propose to explicitly identify the cycles and encode them to *Acyclic constraints*. This approach enables us to formulate term extraction problems in terms of Weighted Partial MAXSAT problems and improve the solving speed of the current ILP formulation. Our evaluation of term extraction for equality saturation on real-world Deep Learning (DL) workloads shows that using the improved formulation yields up to ~3x speed-up on the total extraction time compared with using the state-of-the-art ILP encoding.

CCS Concepts: • **Theory of computation** → *Constraint and logic programming*.

Keywords: E-Graphs, Extraction, Optimization

1 Introduction

Equality saturation [11] and egg [14] are commonly used for compiler optimizations. In practice, these applications have a high demand for both efficiency and optimization quality. Prior work has improved the efficiency of equality saturation in egg by filtering out e-nodes that are not likely to appear in the optimal term but requires sketches from the programmers to guide the filtering [5]. Some works proposed encoding term extraction as integer linear programming (ILP) problems [9, 15]. While solving an ILP problem promises to extract an optimal term, the current formulation does not scale when put into practice.

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00

We observe that in the current formulations implemented in Glenside [9] and Tensat [15], constraints that prevent extracting cyclic terms are encoded as topological sorting. These constraints are hard to solve when the cycles in the input e-graph become complicated, and the topological order variables are only bounded by the size of the e-graph, which may yield an exponentially large search space.

Some recent updates made to egg [14] also implemented ILP-based extraction. Instead of solving a topological sorting problem, egg avoids extracting cycles by blacklisting *all e-nodes* that potentially lead to a cyclic term. This over-approximation may filter out the optimal term, and, in a more extreme case, it blacklists all the e-nodes in the root e-class, making the entire problem infeasible.

In this paper, inspired by these prior works, we propose breaking cycles by identifying the cycles. The cycles detected in a given e-graph will be encoded as *Acyclic constraints*, which guide the solver to avoid extracting the cycles. This approach enables us to formulate term extraction in terms of weighted partial MAXSAT problems (WPMAXSAT) and derive another version of the ILP formulation.

We implement both formulations and evaluate them on Glenside [9] with term rewriting workloads on real-world DL applications and compare the term extraction speed with the state-of-the-art ILP encoding implemented in Tensat [15]. Our evaluations show that the extraction time using WP-MAXSAT and the modified ILP formulation have approximately ~3x speed up.

We also discuss some efforts we made on the LP relaxation of the ILP problems and some optimizations to our new formulations.

2 Notations

For ease of understanding and formalizing later, we adopt the following set of notations,

- Denote an *e-graph* by $G\langle N, C, E\rangle$, where let N be the set of *e-nodes*, C be the set of *e-classes*, and E be the set of edges $e : (n, c)$ betweeen $n \in N$ and $c \in C$.
- Denote the *e-class* where an *e-node* n resides in by $\kappa(n)$, i.e. $\kappa(n) = c \Leftrightarrow n \in c$.

- Denote the children of a node $n \in N$ by $\chi(n)$, which is a set of *e-classes*. Formally, $c \in \chi(n)$ if and only if $\exists e \in E, e = \langle n, c\rangle$
- Denote the cost model by a total mapping $M : N \to \mathbb{R}^+$, which assigns each *e-node* a positive cost.

Definition 2.1. A *cycle* in an e-graph $G\langle N, C, E\rangle$ is an ordered set of e-nodes $\Phi = (n_0, n_1, \ldots, n_{L-1}) \subseteq N$ such that $\kappa(n_j) \in \chi(n_i)$ for all $i + 1 \equiv j (\text{mod } L)$.

Definition 2.2. Given a cycle $\Phi \subseteq N$, the *class cycle* of Φ is $\Phi_c = \{\kappa(n) \mid n \in \Phi\}$. Given an ordered set of e-classes $\Psi = (c_1, c_2, \ldots, c_L)$, define its corresponding cycles by $\theta(\Psi) = \{\Phi \mid \Phi_c = \Psi\}$. Denote the set of e-nodes involved in the cycles in some e-class $c_i \in \Psi$ as $\mathcal{N}_\Psi^{c_i} = c_i \cap \bigcup_{\Phi\in\theta(\Psi)} \Phi$.

Denote the set of all class cycles by

$$\mathcal{A} = \{\Psi \subseteq C \mid \theta(\Psi) \neq \varnothing\}.$$

3 Existing ILP Formulations

Glenside. Glenside [9] is a framework for exploring deep learning accelerator design by composing general-purpose tensor-level rewrite rules and customizable hardware-specific rewrite rules.

Given an e-graph $G\langle N, C, E\rangle$ and a cost model M where $M(n) = w_n$ for all $n \in N$, Glenside implements the following formulation,

- Node variables: for each e-node n, create a **binary** variable w_n. If $w_n = 1$ then n is selected in the extracted term, otherwise it is not selected.
- Class variables: for each e-class c, create a **binary** variable w_c. If $w_c = 1$ then some *e-node(s)* in c are selected.
- Topological ordering variables: for each e-class c, create an **integral** variable t_c (bounded by a sufficiently large value).

Let R be the root of G.

$$\min \sum_{n\in N} M(n)\cdot w_n \qquad \text{(ILP-Glenside)}$$

$$\text{subject to} \quad \sum_{n\in c} w_n \geq w_c \qquad \forall c \in C \quad \text{(Class constraints)}$$

$$w_{c'} \geq w_n \qquad \forall n \in N, \forall c' \in \chi(n) \quad \text{(Children constraints)}$$

$$w_R = 1 \qquad \text{(Root constraint)}$$

$$t_{\kappa(n)} + \sigma(1 - w_n) \geq t_c + 1 \qquad \forall n \in N, \forall c \in \chi(n), \quad \text{(Topological constraints, where } \sigma \text{ is large enough)}$$

$$w_c, w_n \in \{0, 1\} \qquad \forall c \in C, \forall n \in N.$$

The objective is to minimize the sum of the cost of e-nodes in the extracted term. *Class constraints* enforce the requirement that if an e-class c is picked, then at least one e-node $n \in c$ must also be picked. *Children constraints* say that if an e-node is picked, all of its children (e-classes) must also be picked. *Root constraint* ensures that there is at least 1 e-node in the root e-class picked. Finally, the *Topological constraints* impose a strict topological order on the selected e-classes. The topological ordering is done on e-classes: when an e-node n in some e-class p is chosen, the topological order variable t_p must be greater than $t_{q_1}, t_{q_2}, \cdots, t_{q_m}$ where q_i are children of n. On the other hand, if n is not chosen, the topological ordering constraint should always be satisfiable. Therefore, an arbitrary offset (large enough) is added to t_p when n is not chosen ($w_n = 0$). Having these constraints will ensure the acyclicity of the extracted term.

Tensat. Tensat [15] is a re-implementation of TASO [3] using equality saturation for tensor algebra super-optimization.

Tensat's formulation has a similar set of constraints but optimizes away the class variables and the class constraints in Glenside's formulation by representing w_c as $\sum_{n\in C} w_n$. We refer to this formulation as ILP-Topo.

Cycle breaking in egg's ILP extractor. The ILP-based extractor in egg has been updated to implement an algorithm that traverses the e-graph and identifies all e-nodes that create cycles. These nodes are then added to a blacklist to prevent the extraction of cyclic terms before constraint construction. The extractor uses a similar encoding as *ILP-Topo* but employs the blacklist to set the weight (w_n) of blacklisted e-nodes to 0, thus providing an *over-approximation* to the extraction problem.

Since this approach assumes a root for a single-root e-graph, the disabled e-nodes may reside in the root e-class the user wants to extract a term from. Therefore, it can miss the true optimal term, particularly if all e-nodes in the root e-class provided by the user are part of cycles in the e-graph. In such cases, the constraints become contradictory, leading to an UNSAT result even when non-cyclic terms could be extracted. As an example, consider the e-graph in Figure 1, where the e-class containing n_1 and n_2 is the root e-class provided by the user. The *Root Constraint* yields $w_{n_1} + w_{n_2} \geq 1$. However, the algorithm may start with the other two e-classes, resulting in setting $w_{n_1} = 0$ and $w_{n_2} = 0$. This will cause a contradiction with the Root constraints, leading to an UNSAT. One may argue that this issue can be fixed by starting with the root provided by the user. However, fixing the order of traversal is equivalent to assuming a topological order of the e-classes, which may not be the correct order.

4 WPMAXSAT and ILP-ACyc Encodings

We propose to encode the *Acyclic constraints* by identifying class cycles in the given e-graph. This approach enables us to formulate term extraction as WPMAXSAT problems and devise an alternative ILP encoding.

4.1 WPMAXSAT

In WPMAXSAT problems, constraints are divided into two parts: hard constraints and soft constraints. Hard constraints

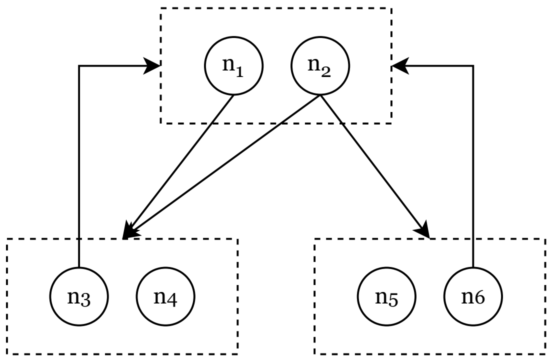

Figure 1. An example e-graph that causes an UNSAT in egg's ILP extractor since *both* n_1 and *and* n_2 are creating a cycle if the algorithm starts from other two e-classes.

are those that must be satisfied by the truth assignments, while soft constraints could be unsatisfied. Each soft constraint is associated with a positive weight and the objective of solving WPMAXSAT is to find a truth assignment to the variables that maximizes the sum of the weights of *satisfied* soft constraints while satisfying all the hard constraints.

Given an e-graph $G\langle N, C, E\rangle$, a cost model M, and a root e-class R to extract, we create a Boolean variable w_n for each e-node $n \in N$. Then the WPMAXSAT constraints for term extraction problems are defined such that w_n is assigned to true if n is in the extracted term.

Hard Constraints.

- *Root constraint*:
$$\bigvee_{n \in R} w_n$$
- *Children constraints*:
$$w_n \to \bigvee_{n' \in c} w_{n'} \qquad \forall n \in N, c \in \chi(n)$$
- *Acyclic constraints*:
$$\bigvee_{c \in \Phi_c} \bigwedge_{n \in N_\Psi^c} \neg w_n \qquad \forall \Psi \in \mathcal{A}$$

Soft Constraints. $\neg w_n$ for each $n \in N$. The weight of $\neg w_n$ is set to $M(n)$ given by the cost model.

As in *Hard Constraints*, the *Root constraint* and *Children constraints* are naturally translated from the ILP-Topo. *Acyclic constraints* are added using class cycles in the e-graph. We rely on existing algorithms (e.g. Johnson's algorithm [4]) to detect class cycles. For each class cycle Ψ, we encode Acyclic constraints to eliminate cycles in $\theta(\Psi)$. An Acyclic constraint is a disjunction of conjunctions of negated e-node variables corresponding to the e-nodes involved in any cycle. This simply disallows the solver from extracting all of the e-nodes in some cycle at the same time. To efficiently encode the Acyclic constraints, we apply Algorithm 1.

Algorithm 1 Encoding of Acyclic Constraints

Input: *e-classes* that form a cycle in the egraph

```
 1: procedure EncodeCycle(P: a list of e-classes that forms
    a cycle)
 2:     $K \leftarrow []$
 3:     for $C \in P$ do
 4:         $K' \leftarrow []$
 5:         for $n \in C$ do
 6:             if $\chi(n) \cap P \neq \emptyset$ then
 7:                 $K'.push\_back(w_n)$
 8:             end if
 9:         end for
10:         $K.push\_back(\bigwedge_{c \in K'} \neg c)$
11:     end for
12:     return TseitinTransform($\bigvee_{c \in K} c$)
13: end procedure
```

Note that when we detect a cycle of e-classes, if we naively encode *all* cycles from $\theta(\Psi)$, the number of constraints could increase exponentially. Therefore, as shown on Line 12 of Algorithm 1, we apply Tseitin transformation [13] of the Acyclic constraints. Specifically, for each e-class c in some class cycle Ψ, we create an auxiliary variable v_Ψ^c. Taking v_Ψ^c as boolean variables, by Tseitin transformation,

$$v_\Psi^c \leftrightarrow \bigwedge_{n \in N_\Psi^c} \neg w_n$$

If direction:

$$v_\Psi^c \to \bigwedge_{n \in N_\Psi^c} \neg w_n \Leftrightarrow \bigwedge_{n \in N_\Psi^c} \neg v_\Psi^c \vee \neg w_n \tag{1}$$

Only if direction:

$$\bigwedge_{n \in N_\Psi^c} \neg w_n \to v_\Psi^c \Leftrightarrow \bigvee_{n \in N_\Psi^c} w_n \vee v_\Psi^c \tag{2}$$

Then the Acyclic constraints for Ψ simply become

$$\bigvee_{c \in \Psi} v_\Psi^c \tag{3}$$

The objective of WPMAXSAT formulation is intuitive: the *soft constraints* naturally encode the objective of extracting the lowest-cost term by *maximizing the weight of negated node variables*, which is equivalent to minimizing the weight of extracted e-nodes hence optimizing the extracted term.

4.2 ILP-ACyc

A benefit of using $\mathcal{A}$ to encode constraints is that we are able to get rid of the topological sorting constraints introduced by the formulations in Section 3. We could instead encode the Acyclic constraints, for all cycles $\{\Phi \mid \Phi \in \Psi, \Psi \in \mathcal{A}\}$ in the given e-graph, as

$$\sum_{n \in \Phi} w_n \leq |\Phi| - 1$$

These constraints restrict the solver to extract *at most* $|\Phi| - 1$ e-nodes from a cycle Φ hence breaking cycles but not filtering out all e-nodes in the cycles. However, as discussed in Section 4.1, the naive encoding may introduce exponentially many constraints if we simply pick an e-node per e-class in the cycle. To reduce the number of constraints, we convert the encoding of Acyclic constraints in WPMAXSAT into equivalent ILP constraints by treating v^c_Ψ as binary variables.

To translate Formula (1), we encode the following constraint for each e-node $n \in \mathcal{N}^c_\Psi$,

$$(1 - v^c_\Psi) + (1 - w_n) \geq 1. \tag{4}$$

Formula (2) simply becomes the ILP constraint

$$v^c_\Psi + \sum_{n \in \mathcal{N}^c_\Psi} w_n \geq 1. \tag{5}$$

Finally, we encode the Acyclic constraint (3) for Ψ using these auxiliary variables:

$$\sum_{c \in \Psi} v^c_\Psi \geq 1 \tag{6}$$

We modify the constraints from ILP-Topo and call this alternative form of formulation *ILP-ACyc*.

$$\begin{aligned}
\min \quad & \sum_{n \in \tau} M(n) \cdot w_n && \text{(ILP-ACyc)} \\
\text{subject to} \quad & \sum_{n' \in c} w_{n'} \geq w_n && \forall n \in N, c \in \chi(n) \\
& \sum_{n \in R} w_R = 1 && \\
& (1 - v^c_\Psi) + (1 - w_n) \geq 1 && \forall \Psi \in \mathcal{A}, c \in \Psi, n \in \mathcal{N}^c_\Psi \\
& v^c_\Psi + \sum_{n \in \mathcal{N}^c_\Psi} w_n \geq 1 && \forall \Psi \in \mathcal{A}, c \in \Psi \\
& \sum_{c \in \Psi} v^c_\Psi \geq 1 && \forall \Psi \in \mathcal{A}, c \in \Psi \\
& w_n, v^c_\Psi \in \{0, 1\} && \forall \Psi \in \mathcal{A}, c \in \Psi, n \in N.
\end{aligned}$$

5 Analysis

We briefly analyze our formulations and compare them with those in Section 3. We say an e-graph $G\langle N, C, E\rangle$ is (n, m, d, k)-*bounded* if $|N| \leq n$, $|C| \leq m$, the maximal congruent degree $\max_{c \in C} |c| \leq d$, and the maximal number of operands $\max_{n \in N} |\chi(n)| \leq k$.

Analysis of WPMAXSAT and ILP-ACyc. First, we analyze the number of acyclicity constraints used in MPMAXSAT and ILP-ACyc. Given a class cycle Ψ of length k with each class containing at most d nodes, a straightforward algorithm would restrict all corresponding node cycles $\theta(\Psi)$, which would add $O(d^n)$ constraints. However, In MPMAXSAT and ILP-ACyc, Tseitin encoding is applied and only polynomial constraints are required.

Lemma 5.1. *For an (n, m, d, k)-bounded e-graph G, the acyclicity constraints for a class cycle $\Psi = (c_1, c_2, \cdots, c_l)$ will add $O(l)$ variables and $O(dl)$ hard clauses (or constraints) to WP-MAXSAT (or ILP-ACyc).*

Proof Sketch. Let $\Psi' = \{c \in \Psi : |\mathcal{N}^c_\Psi| > 1\}$ be the e-classes in Ψ with more than one corresponding e-node. According to Tseitin encoding, $|\Psi'|$ variables will be created. The constraints for conditions (1), (2) and (3) can be counted as

$$\sum_{c \in \Psi'} |\mathcal{N}^c_\Psi| + \sum_{c \in \Psi'} 1 + 1 \leq (d+1)|\Psi'| + 1 = O(dl)$$

□

Definition 5.2. *Cycle complexity* of G is defined by

$$\alpha(G) = \sum_{\Psi} |\{c \in \Psi : |\mathcal{N}^c_\Psi| > 1\}|.$$

Corollary 5.3. *For an (n, m, d, k)-bounded e-graph G, let $\mathcal{A}$ to be the set of all class cycles, WPMAXSAT and ILP-ACyc use $O(n + \alpha(G))$ variables and $O(nk + d\alpha(G)))$ constraints.*

Worst-case Analysis. In ILP-Glenside and ILP-Tensat, there are $O(nk)$ constraints, while, due to the acyclic constraints, the number of constraints in WPMAXSAT and ILP-ACyc is $O(nk + d\alpha(G)))$.

Lemma 5.4. *For an (n, m, d, k)-bounded e-graph G, the number of class cycles $\alpha(G) \leq O(m2^m)$.*

Proof Sketch. In the worst case, each e-nodes has all the e-class as their children, and picking any set of e-nodes such that they are from different e-classes will result in a cycle. Thus the number of class cycles is $2^{|C|} = O(2^m)$. Also, every class cycle contains at most $O(m)$ e-nodes. Therefore, $\alpha(G) \leq O(m2^m)$. □

This means there could be exponentially many constraints in WPMAXSAT and ILP-ACyc (we will discuss some optimization on the number of constraints and variables in Section 7). This is not because of the encoding, but because there can be **exponentially many cycles** of *e-classes* in an e-graph.

6 Evaluation

We have implemented both ILP-Topo, WPMAXSAT and ILP-ACyc as language-agnostic extractors (~900 LoC of Rust) in egg and evaluated on Glenside [9] evaluations. For ILP-Topo and ILP-ACyc, we offload the constraints to the CPLEX ILP solver, and for WPMAXSAT, we feed the constraints to MaxHS [1]. The experiments are conducted on a platform with an Intel i9 processor and 64 GB memory.

6.1 Setup and Workloads

For Glenside [9] we limit the maximum number of *e-nodes* in the e-graph to 100,000 and set a 5-second time-out for equality saturation. We pick 5 real-world DL applications for computer vision, including MobileNetV2 [7], ResMLP [12], EfficientNet [10], ResNet-18 and ResNet-50 [2]. We choose these applications since Glenside converts coarse-grained operators to fine-grained tensor-level expressions, which can give us e-graphs large enough for evaluating different extraction formulations. We make the set of rewrites applied to the e-graph configurable and performed experiments on 2 sets of rewrites:

- IM2COL: a set of rewrites that performs IM2COL transformation
- IM2COL+SIMPL: IM2COL transformation plus a set of simplification rules, which includes collapsing multiple reshapes, transposes and tensor accesses to the most recent one, bubble reshape operators through computations, etc.

We choose a simple cost model that assigns 1 to each e-node, which minimizes the size of the extracted term. We set a 5-minute (300 seconds) time-out for term extraction and measure the time spent for extraction (broken down to time spent by the solvers and the overhead of constructing the constraints).

6.2 Preliminary Results

We performed 5 independent runs for each set of the rewrite rules and computed the average time taken for term extraction. Figure 2 illustrates a comparison of the time required for term extraction using different formulations. Our WPMAXSAT and the improved ILA-ACyc formulations resulted in faster term extraction time: the solver yielded solutions within $\leq$ 5 seconds in all cases whereas the implementation using ILP-Topo takes more than 6 seconds on certain workloads. Notably, applying the WPMAXSAT formulation to term extraction after applying the IM2COL rewrite rules on ResMLP and ResNet-18 provided an $\sim$3x speedup compared to the current Tensat implementation [15]. Moreover, the solver timeouts on solving the ILP-Topo formulation for ResNet-50 after including SIMPL rewrites, but all of our new formulations could be solved within a few seconds.

We also record the statistics of e-graphs after running equality saturation on the workloads described above, shown in Figure 3 and Table 1. There are significantly more e-classes and e-nodes in the e-graph after running IM2COL without simplification rewrites, but the solver does not time out on all the formulations. After we put in SIMPL rewrites, the solver is not able to finish solving ILP-Topo formulation of ResNet-50. This is because SIMPL rewrites introduce complex cycle structures to the e-graph, which makes the topological ordering constraints too complicated to be solved efficiently. On the other hand, the encoding of ILP-ACyc and WPMAXSAT is much simpler since we have already identified the cycles explicitly before emitting the constraints. As shown in Figure 2, the solver is able to find the optimal solution *within a second* using the ILP-ACyc formulation in the case where using ILP-Topo timeouts.

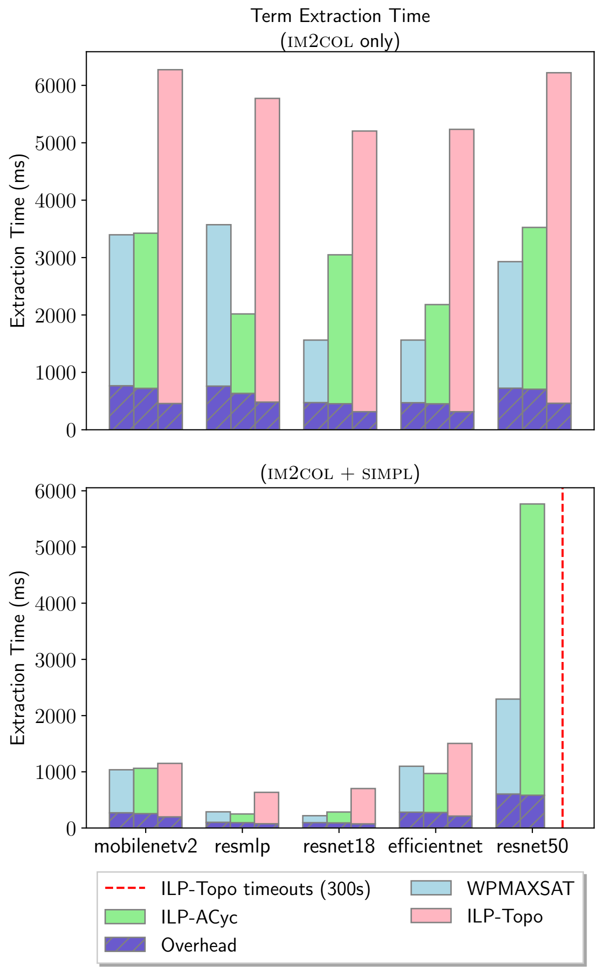

Figure 2. The comparison between the term extraction time (including time for constructing the constraints displayed as *Overhead*) of three different formulations. The upper figure displays the term extraction time for different formulations after running equality saturation with IM2OL rewrites. The lower figure shows the term extraction time after running equality saturation with IM2COL and a set of simplification rewrites (SIMPL). The overhead time (encoding constraints, building extracted term, etc.) is indicated at the bottom of each bar.

7 Discussion and Ongoing Study

7.1 Linear Programming Approximation

We have made some efforts on devising rounding schemes for the relaxed ILP-Topo problem. Specifically, we relax all the ILP variables to fractional variables. After solving the relaxed problem, we get "fractional extraction solutions" where w_n for each e-node is a fractional between 0 and 1. Our rounding scheme is straightforward: we normalize the fractional solutions for e-nodes in each e-class. When choosing a child

	MobileNet	ResMLP	ResNet-18	EfficientNet	ResNet-50
IM2COL	17266	15819	14754	21016	16305
IM2COL + SIMPL	17320	4247	4466	10978	20294

Table 1. Class cycle count after equality saturation

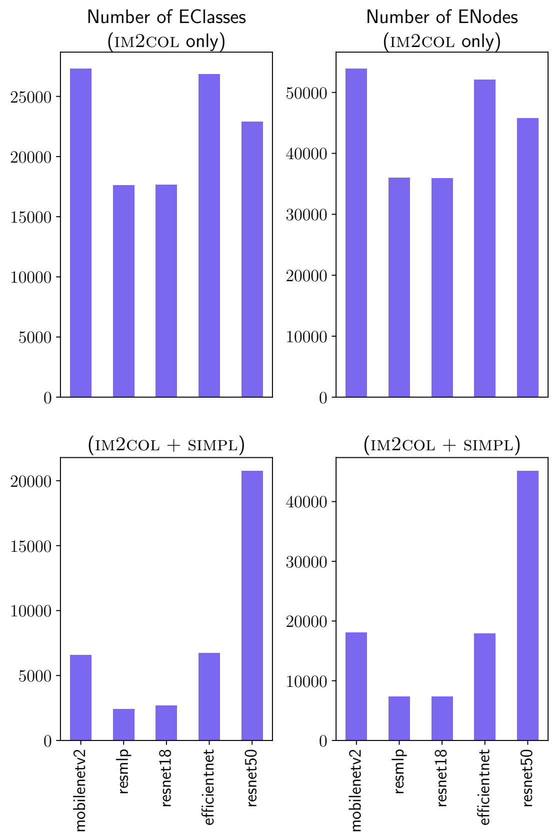

Figure 3. The number of e-classes and e-nodes result from equality saturation applied to different sets of rewrites. The upper figure shows the number of e-classes and e-nodes after applying IM2COL rewrites; the lower figure shows the corresponding numbers after applying IM2COL rewrites in conjunction with simplification rewrites (SIMPL).

for some operators, we take the normalized solution as the probability of choosing an e-node within the child e-class.

There are several challenges left to solve: first, in the relaxation, cycles are harder to handle because neither topological sorting nor Acyclic constraints work. Solving these formulations depends on whether an e-node is extracted in the ILP/WPMAXSAT solutions. However, if we relax the solutions to fractionals, then whether an e-node is extracted cannot be determined until we actually construct the extracted term from the fractional solutions. A more significant issue is that the approximation can be seriously bad. The ILP-Topo introduced in Section 3 only requires $\sum_{n' \in c} w_{n'} \geq w_n$ for each child c of an e-node. Since M will always give a positive weight, $\sum_{n' \in c} w_{n'}$ is at most w_n. Therefore, weights can be divided up and decrease exponentially with respect to the depth of the e-graph since some rewrite rules always bring terms with equal weights (e.g. commutativity). This can potentially lead to bad approximations with an arbitrarily large ratio[1]. To address this problem, we are trying to apply the Sherali-Adam Lift and Project Method [8] to strengthen the LP constraints. In addition to considering the probability of extracting a single e-node, this approach encodes variables that can represent the probability of extracting some e-nodes at the same time.

7.2 Optimizing number of constraints for WPMAXSAT and ILP-ACyc

By Lemma 5.4, $\alpha(G)$ can be exponential, which may introduce exponentially many constraints and variables (though unlikely in practice). However, we can reduce the number of constraints by encoding for cycle bases (fundamental cycles) [6] instead of all cycles since there are linearly dependent cycles. Given a graph with n nodes and e edges, there are $e - n + 1$ fundamental cycles, and the number of constraints for a cycle base is linear with respect to the number of *unique* edges[2]. Moreover, in practice, the degree of *e-nodes* in an e-graph is a constant and the lengths of cycles are almost constant. If these properties hold, the number of additional constraints will decrease to $\tilde{O}(|C|)$ where $|C|$ is the number of e-classes.

8 Conclusion

This paper proposes to improve the efficiency of constraint-solving-based term extraction for egg by using *Acyclic constraints* to avoid extracting cyclic terms. This approach enables the encoding of term extraction problems in terms of weighted partial MAXSAT problems and dramatically improves the solving speed of the ILP formulation. Our evaluation shows significant speed-ups, even enabling extraction within a few seconds where the solver could not finish solving in 5 minutes using the encoding introduced in prior works. We are taking additional steps to reduce the number of constraints in WPMAXSAT and ILP-ACyc formulations and enhance the practicality of the LP relaxation method.

9 Acknowledgement

We thank Sami Davies for discussing with us an iterative rounding scheme for LP approximation and pointing us to the Sherali-Adams Lift and Project Method for strengthening the LP formulations.

[1]We have proved this result but we will omit it here due to the space constraint

[2]The edges from *e-nodes* in the same e-class pointing to the same child are merged into a single edge, counted once.

References

[1] Jessica Davies. 2014. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. Ph. D. Dissertation. University of Toronto, Canada. http://hdl.handle.net/1807/43539

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[3] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 47–62. https://doi.org/10.1145/3341301.3359630

[4] Donald B. Johnson. 1975. Finding All the Elementary Circuits of a Directed Graph. *SIAM J. Comput.* 4, 1 (1975), 77–84. https://doi.org/10.1137/0204007 arXiv:https://doi.org/10.1137/0204007

[5] Thomas Koehler, Phil Trinder, and Michel Steuwer. 2021. Sketch-Guided Equality Saturation: Scaling Equality Saturation to Complex Optimizations of Functional Programs. https://doi.org/10.48550/ARXIV.2111.13040

[6] Keith Paton. 1969. An Algorithm for Finding a Fundamental Set of Cycles of a Graph. *Commun. ACM* 12, 9 (sep 1969), 514–518. https://doi.org/10.1145/363219.363232

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs.CV]

[8] Hanif D. Sherali and Warren P. Adams. 1990. A Hierarchy of Relaxations between the Continuous and Convex Hull Representations for Zero-One Programming Problems. *SIAM Journal on Discrete Mathematics* 3, 3 (1990), 411–430. https://doi.org/10.1137/0403036 arXiv:https://doi.org/10.1137/0403036

[9] Gus Henry Smith, Andrew Liu, Steven Lyubomirsky, Scott Davidson, Joseph McMahan, Michael Taylor, Luis Ceze, and Zachary Tatlock. 2021. Pure tensor program rewriting via access patterns (representation pearl). In *Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming*. ACM. https://doi.org/10.1145/3460945.3464953

[10] Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946 [cs.LG]

[11] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2011. Equality Saturation: A New Approach to Optimization. *Logical Methods in Computer Science* 7, 1 (mar 2011). https://doi.org/10.2168/lmcs-7(1:10)2011

[12] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. 2021. ResMLP: Feedforward networks for image classification with data-efficient training. arXiv:2105.03404 [cs.CV]

[13] G. S. Tseitin. 1983. *On the Complexity of Derivation in Propositional Calculus*. Springer Berlin Heidelberg, Berlin, Heidelberg, 466–483. https://doi.org/10.1007/978-3-642-81955-1_28

[14] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (jan 2021), 1–29. https://doi.org/10.1145/3434304

[15] Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. 2021. Equality Saturation for Tensor Graph Superoptimization. https://doi.org/10.48550/ARXIV.2101.01332