Efficient E-Graph Extraction using Linear Programming

COS 521 FA 22 Final Project Report

HAICHEN DONG, Princeton University, USA
DEYUAN MIKE HE, Princeton University, USA

Equality saturation (EqSat) today is widely used to solve program optimization problems with respect to a set of syntactic rewrite rules by exploring a potentially infinite space of equivalent representations. Even though recent works have dramatically improved the usability and integration, EqSat still does not scale well partly due to the optimal term extraction procedure. In this project, we formulate the extraction problem first as an integer linear program. Then we relax it to a linear program. Finally, we devise a rounding scheme for the fractional optimal extraction to obtain an approximated optimal term.

1 INTRODUCTION

Compiler optimizations usually involve term rewriting. However, deciding the order of the rewrite passes is very subtle because some rewrites may disable others, resulting in a half-optimized program. For a simple example: suppose the compiler is to optimize two the matrix-vector arithmetic expressions $(M_1 \cdot M_2) \cdot M_3$ and $(M_1 \cdot M_2) \cdot M_4$. If the optimizer chooses to perform common subexpression elimination (CSE) first, it may end up eliminating the subexpression $M_1 \cdot M_2$ by creating a fresh variable M_{12} holding its value, then multiply M_3 and M_4 with M_{12}. However, multiplying M_2 with the other two matrices M_3, M_4 first could be much more efficient than doing the $M_1 \cdot M_2$ matrix multiplication. On the other hand, if $M_1 \cdot M_2$ is cheaper, performing the associativity rewrites would disable the CSE rewrite. This issue is known as the *Phase Ordering Problem*.

Equality saturation (EqSat) [Tate et al. 2011] is a technique to remove this blocker by maintaining all the equivalences discovered by the rewrite rules, usually equalities or axioms already known, and then extracting the optimal one from the explored space. An implementation of EqSat is using E-Graph [Nelson and Oppen 1980], an efficient data structure that maintains congruence relations of terms with respect to a set of syntactic rewrite rules. The recent egg [Willsey et al. 2021] implementation of E-Graph integrates more seamlessly with optimizations for Linear Algebra (LA) [Wang et al. 2020], Machine Learning (ML) [Smith et al. 2021; Yang et al. 2021], rewrite rule synthesis [Nandi et al. 2021] and so on. Even though egg uses a compact representation to efficiently maintain exponentially many equivalences, the optimal term extraction procedure still poses a major bottleneck that drastically worsens the performance of the entire optimization flow. Some research proposes to use a "Sketch" to filter out a large portion of the terms that are not likely to be the optimal candidate [Koehler et al. 2021], which remarkably improved both term rewriting, extraction speed, and memory usage. However, this would require the user to provide domain-specific sketches, making the entire process fully automated to semi-automated.

In this project, we explore the use of linear programming to solve the extraction bottleneck, which speeds up the extraction while requiring no manual input from the user. We formulate the extraction problem, first as integer programs, and then relax to linear programs. After solving the linear program, we devise a rounding scheme to obtain an approximated optimal extraction.

The report is organized as follows:

- In Section 2, we give some background of E-Graph, term rewriting on E-Graph, and equality saturation on E-Graph.

Authors' addresses: Haichen Dong, Department of Computer Science, Princeton University, USA, hd5234@cs.princeton.edu; Deyuan Mike He, Department of Computer Science, Princeton University, USA, dh7120@cs.princeton.edu.

- In Section 3, we identify some current issues of E-Graphs term extraction algorithm and outline our solution using linear programming.
- In Section 4, we give the formulation of the constraints-solving problem for E-Graph extraction procedure and the rounding scheme for obtaining a program from the fractional extraction.
- In Section 5, we describe the insufficiency of the greedy algorithm and provide some analysis of the rounding scheme. We specifically focus on the drawbacks of our algorithm and give adversarial constructions that lead LP rounding to extract the worst solution.
- In Section 6, we demonstrate the effectiveness and drawback of our algorithm by integrating it into the evaluation of Tensat [Yang et al. 2021]. We utilize the cost model of Tensat's infrastructure and compare the minimum costs of programs extracted by ILP and LP algorithms. We also compare the time spent by the solver during the constraint-solving phase.
- In Section 7, we provide some discussions on potential improvements and possible future research.

2 BACKGROUND

In this section, we provide some basic background of E-Graphs, including the formal definition and related operations.

2.1 Definition of E-Graphs

Definition 2.1. An E-Graph maintains congruence relations among a set of terms. Formally, an E-Graph is defined as $\mathcal{G}\langle C, \mathcal{F}\rangle$, where C is a set of union-finds [Tarjan 1975] (we call them *e-classes*) that maintains congruence relations ($\equiv_{id} :: T \times T$) among *e-nodes* $n \in \bigcup_{c\in C} c$. $\mathcal{F}$ is a set of edges between *e-nodes* and *e-classes*. Specifically, $\mathcal{F}$ is a set of $T \times C$ such that $e \in \mathcal{F} = \langle n, c\rangle$, where $n \in \bigcup_{c'\in C} c'$ and $c \in C$.

2.2 Terms and Congruence

An *e-node* itself is meaningless, instead, when discussing an *e-node* n, we are talking about the set of *terms* it represents, together with the *e-classes* $c_1, \cdots c_m$ such that

$$\langle n, c_i\rangle \in \mathcal{F} \quad \forall i \in [m]$$

Definition 2.2. Given an E-Graph $\mathcal{G}\langle C, \mathcal{F}\rangle$, for any node $n \in \bigcup_{c\in C} c$, denote the children of n by

$$\chi(n) = \{c \mid \langle n, c\rangle \in \mathcal{F}\}$$

Definition 2.3. Let $e_1, \cdots, e_m$ be some expressions (see Definition 2.4), let n be some nodes representing an m-ary operator, then $n(e_1, \cdots, e_m)$ is the expression of computing the operation taking $e_1, \cdots, e_m$ as the inputs.

Definition 2.4. Given an E-Graph $\mathcal{G}\langle C, \mathcal{F}\rangle$, for any *e-node* n, we define $[\![n]\!]_{\mathcal{G}}$, representing the set of expressions extracted using n as the root, as follows. For simplicity of notations, let $[\![c]\!] = \bigcup_{n\in c}[\![n]\!]$ where $c \in C$.

- $[\![n]\!]_{\mathcal{G}} = \{n\}$ if $\chi(n) = \emptyset$.
- $[\![n]\!]_{\mathcal{G}} = \bigcup_{e\in[\![c_1]\!]\times[\![c_2]\!]\times\cdots\times[\![c_m]\!]\forall c_i\in\chi(n)} n(\vec{e})$ otherwise

Corollary 2.5. *Given an E-Graph $\mathcal{G}\langle C, \mathcal{F}\rangle$, if $n_1, n_2 \in c$ for some $c \in C$, then*

$$\forall\langle e_1, e_2\rangle \in [\![n_1]\!]_{\mathcal{G}} \times [\![n_2]\!]_{\mathcal{G}},\ e_1 \simeq_{\mathcal{G}} e_2$$

meaning that $\mathcal{G}$ proves the equivalence between e_1 and e_2. Furthermore, for any E-Graph $\mathcal{G}$, if $e_1 \simeq_{\mathcal{G}} e_2$, then $\forall f, f(e_1) \simeq_{\mathcal{G}} f(e_2)$, which is the congruence axiom of E-Graphs.

2.3 Term Rewriting and Equality Saturation on E-Graphs

We are not able to explore the space of equivalent terms with an E-Graph alone. To do so, we also need to provide a set of **rewrite rules**, representing the set of **axioms** of the language we are working with. For a simple example, consider the language of integer arithmetic, axioms could be *commutativity* of addition and multiplication. In egg library, rewrite rules are encoded as *pattern* rewrites. Consider the commutativity of addition, the rewrite rule is encoded as

$$\texttt{(+ ?x ?y)} \Rightarrow \texttt{(+ ?y ?x)}$$

where the pattern to match on the left-hand side, called the initial pattern, is an expression with a + operator. ?x is a pattern variable, standing as a named wildcard that matches any expression (and binds to the pattern variable). The target pattern is on the right-hand side, which refers to the pattern variables introduced in the initial pattern but altered the order. When executing this rewrite rule, a pattern matcher will match the initial pattern over the given E-Graph. When it finds a match with a + node, whose children are *e-classes* c_l and c_r, in some *e-class* c_+, then it will bind c_l to ?x and c_r to ?y, and instantiate a new *e-node* using the target pattern by substituting in ?x and ?y. In this case, the new *e-node* is $+(c_r, c_l)$. Recall that C is a union-find of *e-classes*. In this case, E-Graph will create a new *e-class* $c_{+'}$ for the new *e-node*, and then union c_+ and $c_{+'}$ into the same union find set. In this way, E-Graph maintains the equality after discovering new expressions. Notice that E-Graph re-uses the children of the matched term instead of creating new ones, which saves a lot of space while memoizing all the equivalences that have been already discovered. An E-Graph is called **saturated** if no new expression could be discovered by applying any (combinations) of the provided rewrite rules any number of times.

The egg library also supports more powerful rewrites such as conditional rewrites and multi-pattern rewrites, but they are orthogonal to the goal of this report and can be used together with the extraction algorithm proposed by this project. Thus, to focus on our goal, we do not diverge into those functionalities here.

2.4 Extraction

E-Graphs maintain an exponentially large set of equivalent terms, but we finally want some optimal one(s). For instance, when considering LA, we might want to extract the term with the least number of vector dots with the tiling size best fitting the target platform, which gives optimal efficiency. The common practice of term extraction is using a cost model for the language the E-Graph is working with. The cost model assigns a cost for each *e-node*, and the cost of an *e-node* possibly depends on its children. The extraction aims to extract a term(s) such that the cost of the term(s) (which is usually equal to the sum of the *e-nodes* in the term) is minimized.

Definition 2.6. A cost model $M : \bigcup_{c \in C} c \to \mathbb{R}^*$ is a mapping from *e-nodes* to non-negative real numbers, representing the cost of selecting a node to include in the extracted program. The cost model is provided as input to the E-Graph by the users.

A generic extraction procedure (extracting an arbitrary term) proceeds as follows:

1. Starting with a **provided** root *e-class* C_r.
2. Pick an *e-node* $n \in C_r$. If $\chi(n) = \emptyset$, return n as the result.
3. Otherwise, run step 1. by setting $C_r = C_i$, for each $C_i \in \chi(n)$. Denote the result by e_i. Return $n(e_1, \cdots, e_m)$ as the result.

Suppose the picked *e-nodes* are $n_1 \cdots n_k$, then, straightforwardly, the cost of the extracted term is $\sum_{i=1}^{k} M(n_i)$.

3 PROBLEM DESCRIPTION

Currently, the most frequently used extraction algorithm is *greedy* which tries to minimize the cost in each layer of the E-Graph. However, this strategy does not always give us the optimal cost term, since the path of the minimum cost could accumulate and exceed the cost on other branches. We will discuss this in more detail in Section 5. An alternative is to use *dynamic programming* (DP). Nevertheless, the rewrite rules applied by the E-Graph can be expansive, which grows the size of the E-Graph dramatically, and the memory can be insufficient for running a DP algorithm.

In this case, using integer linear programming (ILP) is a good choice since it introduces *sharing* automatically when the extraction is encoded as an integer program [Wang et al. 2020]. However, since ILP is NP-Complete, constraint solving can be very slow due to the large number of variables in the integer programs.

3.1 Project Proposal

Inspired by the course material, we noticed that the problem can possibly be relaxed to linear programs (LP), and then round the fractional solutions. We propose to use linear programming to solve the extraction bottleneck, which speeds up the extraction while requiring no manual input from the user. We formulate the extraction problem, first as integer linear programs, and then relax to a linear program. After solving the linear program, we devise a rounding scheme to obtain an approximated optimal extraction. We will give some theoretical analysis of our approximation algorithm.

We will first provide a theoretical analysis of the extraction performance and running time of the algorithm we propose, with comparisons with traditional ILP.

To obtain empirical results, we implemented the extraction algorithm in the egg library and adapted the interface for Tensat [Yang et al. 2021]. We compared the runtime costs of optimized programs yielded by ILP and LP rounding solutions and the time spent by constraint solving.

4 LINEAR PROGRAMMING

4.1 LP Formulation

Following the notations and definitions introduced in Section 2, we create variables to denote the activation status of every e-class by $w_c (c \in C) \in \{0, 1\}$ and every node by $w_n (n \in \bigcup_{c \in C} c) \in \{0, 1\}$.

The Integer Linear Programming implemented in egg libeary can be formulated as:

$$
\begin{aligned}
\min \quad & \sum_{c \in C} \sum_{n \in c} M(n) \cdot w_n && && \text{(ILP)} \\
\text{subject to} \quad & \sum_{n \in c} w_n \geq w_c && \forall c \in C \\
& w_{c'} \geq w_n && \forall n \in \bigcup_{c \in C} c, \forall c' \in \chi(n) \\
& w_c, w_n \in \{0, 1\} && \forall c \in C, \forall n \in \bigcup_{c \in C} c \\
& w_R = 1.
\end{aligned}
$$

In Linear Programming formulation, the constraints are relaxed to range $[0, 1]$. Furthermore, the variables $w_c (c \in C)$ can be eliminated by combining the first two constraints. We also fix the *e-class* (denoted by R) where we pick the root node for the extracted expression. We implemented the following linear optimization:

$$\begin{aligned} \min \quad & \sum_{c\in C}\sum_{n\in c} M(n)\cdot w_n && && \text{(LP)}\\ \text{subject to} \quad & \sum_{n\in c} w_n \geq w_{n'} && \forall n' \in \bigcup_{c\in C} c, \forall n \in \bigcup_{c'\in\chi(n')} c' \\ & 0 \leq w_n \leq 1 && \forall n \in \bigcup_{c\in C} c \\ & \sum_{n\in R} w_n \geq 1. \end{aligned}$$

4.2 LP Rounding

We first remove the integral solution constraints on all the variables. After getting a fractional solution, for each *e-class*, we normalize the solutions to 1.

Algorithm 1 Term extraction via LP rounding

Require: $\{w_n\}_{n\in\bigcup_{c\in C} c}$: Solutions to (LP)

$A \leftarrow \emptyset$ ▷ Memorize rounding results
procedure ROUND(R: current *e-class*;)
  $m \leftarrow$ Sample from R such that $r' \in R$ is chosen with probability $w_{r'}/\sum_{r\in R} w_r$
  if $\exists x, (R, x, e) \in A$ **then**
    return e
  else if $\chi(m) = \emptyset$ **then**
    return m
  else
    for $c' \in \chi(m)$ **do**
      $e_i \leftarrow$ ROUND(c')
    end for
    $A \leftarrow A \cup \{(R, m, m(\vec{e}))\}$
    return $m(\vec{e})$
  end if
end procedure

As shown in Algorithm 1, the rounding scheme is based on the observation that in each *e-lass*, exactly one node should be activated, and it is natural to perform probabilistic allocation based on the LP score. However, LP tends to split the weights sometimes due to cost sharing, making the rounding scheme suboptimal, and we will provide a detailed analysis in Section 5.

5 ANALYSIS

5.1 Insufficiency of Greedy

Since the E-Graph without cycles has a DAG structure, one natural thought is that we can find the optimal allocation for e-classes in reverse topological order. However, due to possible sharing between nodes, i.e., multiple nodes can use the same e-class with no additional cost, the greedy algorithm cannot produce the optimal solution [Yang et al. 2021].

LEMMA 5.1. *The Greedy Algorithm is suboptimal.*

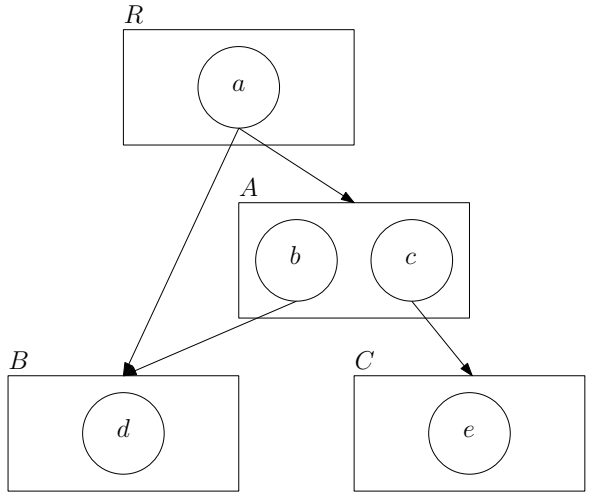


Fig. 1. An E-graph on which the Greedy Algorithm will fail

Proof. Consider the E-Graph illustrated in Figure 1 in which costs $M(n) = 2$ for all $n \neq e$ while $M(e) = 1$. Using reverse topological order, class A can be processed after the allocation for trivial classes B and C. At this time a greedy algorithm would compare $cost(B) = 2$ and $cost(C) = 1$, and choose to activate node c with the use of class C.

As a result, such allocation results in a total cost of $M(\{a, c, d, e\}) = 7$, while the optimal solution only needs $M(\{a, b, c\}) = 6$.

□

5.2 Insufficiency of LP Rounding

When solutions to LP (given by Algorithm. 1) is not limited to integers, nodes might be partially activated, and the sharing of costs can help to reduce the overall activation cost.

Lemma 5.2. *LP Rounding scheme 1 has an arbitrarily large approximation ratio.*

Proof. Here we illustrate a counter-example in Figure 2 on which the LP rounding scheme will fail. In this case, the E-Graph consists of $2k+1$ classes $C = \{root, S_1, T_1, \cdots, S_k, T_k\}$ and $4k+2$ nodes, and we assume that the activation cost of every node is 1. For every e-class, we have $root = \{x, y\}$ and $S_j = \{a_j, b_j\}$ and $T_j = \{c_j, d_j\}$.

The optimal integer solution is straightforward, it suffices to pick exactly one node in every layer, therefore the total cost $OPT_{ILP} = k + 1$.

When the ILP is relaxed to Linear Programming, one possible solution would be activating every node partially and splitting the activation factor w_n on every layer. Formally, we have

$$w_{a_0} = w_{b_0} = 2^{-1}$$
$$w_{a_j} = w_{b_j} = w_{c_j} = w_{d_j} = 2^{-j-1}$$

The total cost of this solution can be calculated as

$$ALG_{LP} = \sum_{c \in C} \sum_{n \in c} M(n) \cdot w_n = 2 \cdot \frac{1}{2} + \sum_{i=1}^{k} 4 \cdot \frac{1}{2^{k+1}} = 3 - 2^{-k+1}$$

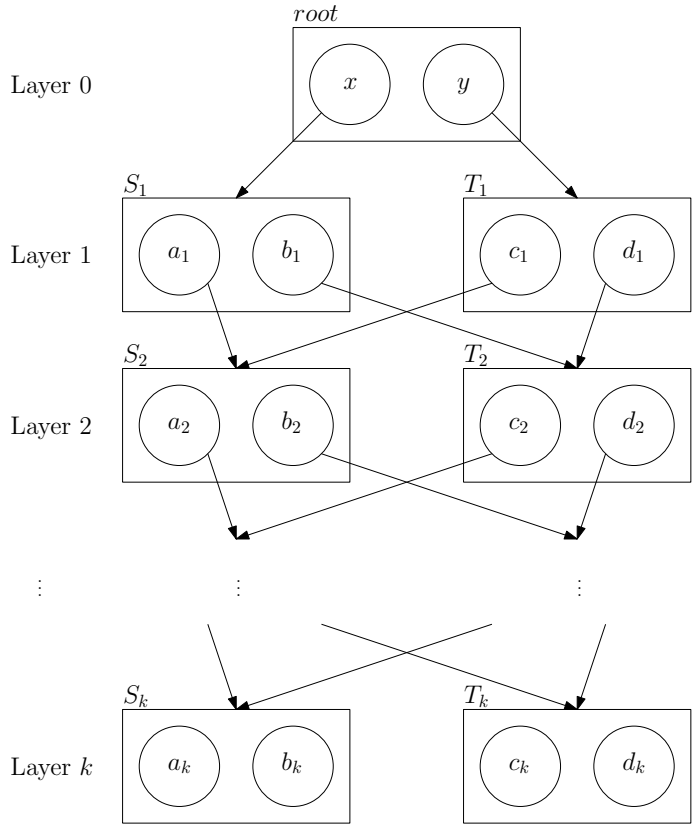


Fig. 2. An E-Graph on which LP with rounding scheme by Algorithm 1 will fail

As a result, we have $OPT_{ILP} = k + 1$ while $OPT_{LP} \leq ALG_{LP} < 3$. The approximation factor of the rounding scheme is as bad as $\frac{k+1}{3}$, which can be arbitrarily large. □

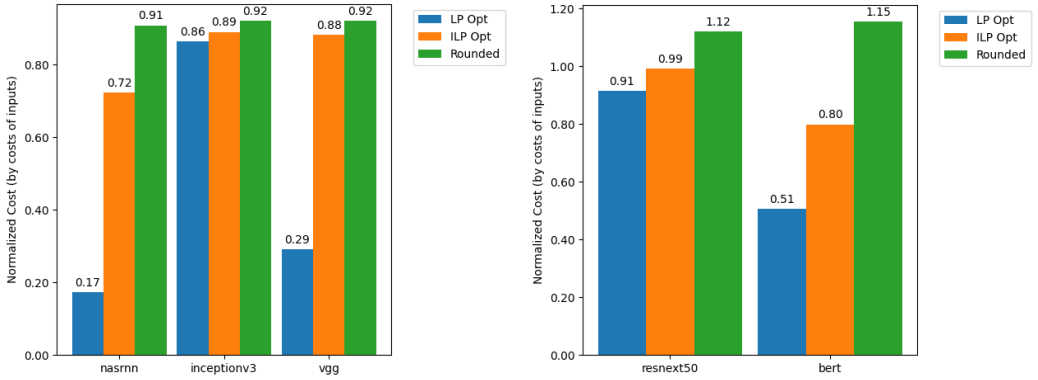
6 EVALUATION

We implement[1] the LP extraction algorithm and the rounding scheme in the egg [Willsey et al. 2021] library as an alternative extractor and utilize the Rust library of the CPLEX solver for defining and solving constraints. We evaluated our algorithm by adapting the pipeline for Tensat [Yang et al. 2021]. Due to time constraints, we reuse Tensat's evaluations, which include VGG-16 [Simonyan and Zisserman 2014], InceptionV3 [Szegedy et al. 2015], ResNexT-50 [Xie et al. 2016], BERT [Devlin et al. 2018] and NasRNN [Zoph and Le 2016]. Our evaluation mainly experiments on (1) comparisons of minimum costs given by ILP, LP, and rounded solutions; (2) cases where LP rounding fails to find better extractions, and (3) comparisons of solver time on ILP and LP problems.

6.1 Experiment Setup

We use the rewrite rules provided by Tensat [Yang et al. 2021], which are re-implementations of the graph substitutions from TASO [Jia et al. 2019]. As explained in Section 5, our LP formulation

[1]Custom egg: https://github.com/AD1024/egg
Custom Tensat: https://github.com/AD1024/tensat



(a) Comparison between the cost of programs extracted by ILP and LP rounding solutions.

(b) Some structures can lead the rounding scheme to extract a program worse than the input.

Fig. 3. Comparison between the costs of programs extracted using different algorithms. (a) shows that the rounding scheme could yield an optimized program, but slightly more inefficient than the optimal solution extracted using ILP solutions. (b) shows that some inputs, potentially adversarial ones, may fail the rounding scheme by worsening the program, as analyzed in Section 5.

does not work well with E-Graphs with cycles, we, therefore, eliminate cycles leveraging Tensat's `no-cycle` mode *e-class* analysis.

The egg library has its own built-in ILP extractor; however, it has a different formulation from that of Tensat. Therefore, we re-implement the ILP formulation using a Rust front-end of the CPLEX solver, and then we implement the rounding scheme as described in Section 4 to find an extraction that minimizes the cost. We implement an algorithm that takes the fractional solution and recursively reconstructs the extracted expression by rounding the solution online. During reconstruction, we use the cost model provided by Tensat to compute the cost of the extracted expression. Since the cost model is obtained from online measurement running each individual operator on GPU [Yang et al. 2021], the final cost closely aligns with the real run time on the hardware.

We evaluated our algorithm on an Intel i9 12900k CPU and compared it with running the same set of constraints but restricting all variables to be binary, which effectively is the ILP formulation. The E-Graph saturates on all inputs with at most 7 iterations of term rewriting.

6.2 Minimum Cost Comparison

We run the end-to-end procedure, from input model to optimized program, 10 times and measure the average cost for both ILP and LP extraction. Figure 3 shows the comparison between the costs of extracted programs given by ILP solution and by LP rounding. The costs are normalized with respect to the cost of the input program, measured by Tensat's cost model. We categorize them into two groups. Fig. 3a shows the case where the rounding scheme works well on the E-Graph after equality saturation, whereas Fig. 3b shows that when taking BERT and ResNexT-50 as input models, the program extracted using rounded solutions are even worse than the input program. This attributes to the insufficiency discussed in Section 5.

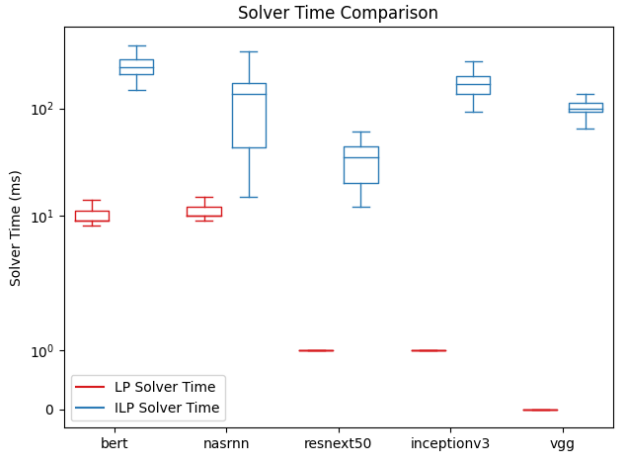


Fig. 4. Boxplots of solver time on solving ILP v.s. LP constraints. The red boxes are LP solver time and the blue boxes are ILP solver time. It shows that even on small examples, solving LP constraints is at least an order of magnitude faster than solving ILP constraints, which is as expected.

6.3 Solver time comparison

We additionally compared the time spent on solving the constraints. Fig. 4 shows the solver times (in milliseconds) of solving ILP and LP constraints. As we expected, LP extraction is at least an order of magnitude faster than ILP extraction. In fact, it mostly terminates with an optimal solution instantly. It is worth noticing that in Tensat evaluations, E-Graphs are saturated for all the inputs. Therefore, the size of E-Graphs are not large in these cases: running EqSat for NasRNN gives the largest E-Graph with 1223 *e-classes* and 6714 *e-nodes*. However, in other domains, for instance, hardware optimizations at tensor level [Smith et al. 2021], the size of the E-Graph can be blown up very quickly and drastically because expressions in tensor programs are much more granular than operator-level expressions, which is the main focus of Tensat. In this case, solving the ILP constraints is very likely to time out.

7 DISCUSSION & FUTURE DIRECTIONS

7.1 Critical drawback of LP extraction

Even though solving LP constraints is a lot faster than solving ILP constraints, it does not work well with various adversarial inputs, as discussed in Section 5. Moreover, in ILP formulation, we are actually able to encode the constraint to get rid of cycles leveraging the solver instead of via an ad hoc cycle eliminator in Tensat. Specifically, we additionally assign each *e-class* a variable representing its topological order and constrain parents to have larger/smaller topological order than their children if there is a node picked in the *e-class*. This is not feasible in LP formulation, since whether the topological order constraints take effect depends on whether an *e-node* is picked. In integral solutions, this is a binary value, thus can be directly encoded as a linear constraint (e.g. $v_i \rightarrow t_i > t_c \Rightarrow t_i - t_c + \Delta(1 - v_i) > 0$ for some $\Delta > 0$). On the other hand, LP solutions are fractional, which means an *e-node* with a non-zero solution is potentially, but not definitely, picked, and thus we cannot encode the constraint for topological ordering in the same way. This is also the reason we eliminate cycles before feeding the E-Graph for LP extraction.

7.2 Other formulations

While egg implements ILP extraction, there are other formulations that are also NP, but we may have better heuristics and pruning strategies for solving these problems. Here we mainly discuss 2 directions: non-convex optimization and weighted partial MaxSAT.

Non-convex Optimization. The key problem of the linear programming method is the fractional solutions that cause partial selection. To overcome this challenge, we can think of adding a regularization term in the minimization objective function. Formally, we set the objective function as $f(n, c) = \sum_{i\in[M],j\in[l_i]}(a_{i,j}n_{i,j} - \lambda(n_{i,j} - \frac{1}{2})^2)$, in which $\lambda > 0$ would be tunable parameter. Intuitively, when λ increases, the solution focuses more on producing integer solutions rather than minimizing the total cost. We can search for the λ that best balances the linear programming cost and the rounding cost.

Weighted Partial MaxSAT. Weighted partial MaxSAT is a variant and generalization of SAT. There are two kinds of constraints in this problem: hard constraints ($\mathcal{H}$) and soft constraints ($\mathcal{S}$). Hard constraints are those that must be satisfied by the assignment to variables, whereas soft constraints are not required to be satisfied. For each $s \in \mathcal{S}$, there is an associated non-negative weight W_s. The goal is to maximize $\sum_{i\in SAT(\mathcal{S})} W_i$, where $SAT(\mathcal{S})$ is the set of soft constraints being satisfied by the truth assignment, i.e. we are maximizing the sum of weights of satisfied soft constraints.

We can translate the ILP constraints shown in Section 4 one-to-one to hard constraints in weighted partial MaxSAT except for the topological ordering constraint. Then, instead of getting rid of all cycles, we add some additional constraints: for each cycle $C \in Cycle(G)$, we add a constraint $\bigvee_{v_i\in C} \neg v_i$, which also offloads the cycle elimination problem to the constraint solver. Finally, soft constraints are the negation of variables for each *e-node*, namely $(\neg v_i)$, and the cost for each $(\neg v_i)$ is given by the cost model for *e-node* n_i. This is equivalent to minimizing the cost of ILP formulations: we are maximizing $\sum W_i$ for all satisfied $(\neg n_i)$, which means we are minimizing the number of n_i that are assigned to $\top$. This essentially is minimizing the cost of the extracted expression.

8 CONCLUSION

In this project, we aim at addressing the inefficiency of E-Graph extraction. We explore the problem by encoding the extraction procedure as linear constraints, by relaxing the integer program proposed in Tensat.

We give some theoretical analyses of the approximation ratio and prove that adversarial inputs can drastically degrade the performance of programs extracted by the rounding scheme, demonstrated by our manual construction. In addition to theoretical results, we implement our proof-of-concept algorithm in the egg library and integrated our customized LP extractor with Tensat. We evaluate our algorithm on five applications. The empirical results show that our LP rounding scheme gives good approximations on three of the models while having at least an order of magnitude speed up. However, for the other two cases, the rounding scheme results in expressions even worse than the inputs, which matches our theoretical analysis.

REFERENCES

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805

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

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

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, and Zachary Tatlock. 2021. Rewrite Rule Inference Using Equality Saturation. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 119 (oct 2021), 28 pages. https://doi.org/10.1145/3485496

Greg Nelson and Derek C. Oppen. 1980. Fast Decision Procedures Based on Congruence Closure. *J. ACM* 27, 2 (apr 1980), 356–364. https://doi.org/10.1145/322186.322198

Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. https://doi.org/10.48550/ARXIV.1409.1556

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

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. https://doi.org/10.48550/ARXIV.1512.00567

Robert Endre Tarjan. 1975. Efficiency of a Good But Not Linear Set Union Algorithm. *J. ACM* 22, 2 (apr 1975), 215–225. https://doi.org/10.1145/321879.321884

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

Yisu Remy Wang, Shana Hutchison, Jonathan Leang, Bill Howe, and Dan Suciu. 2020. SPORES: Sum-Product Optimization via Relational Equality Saturation for Large Scale Linear Algebra. https://doi.org/10.48550/ARXIV.2002.07951

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

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2016. Aggregated Residual Transformations for Deep Neural Networks. https://doi.org/10.48550/ARXIV.1611.05431

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

Barret Zoph and Quoc V. Le. 2016. Neural Architecture Search with Reinforcement Learning. https://doi.org/10.48550/ARXIV.1611.01578