fuzzy-eqsat: Equality saturation with approximations

Jeff Zhang
McGill University
Montreal, Canada

***Abstract*—Equality saturation is a program optimization technique that has recently gained popularity in domains such as compiler optimization, program synthesis, and datapath optimization. It employs a graph data structure known as an e-graph to compactly represent a large amount of equivalent programs derived from a set of rewrite rules.**

We present `fuzzy-eqsat`, a novel form of equality saturation that employs approximate rewrite rules to optimize programs at the cost of accuracy. Namely, we introduce truncated singular value decomposition and matrix pruning rewrites in order to speed up matrix multiplication. This paper evaluates the efficacy of the approach on two basic neural networks for handwritten digit classification using the MNIST dataset. Experiments show that `fuzzy-eqsat` yields substantial cost reductions with only a slight accuracy penalty under the chosen cost model; however, it remains unclear whether these reductions translate into practical speedups.

I. Introduction

Equality saturation has emerged as a promising alternative to traditional program optimization techniques, becoming popular in domains such as compiler optimization [1], program synthesis [2], and datapath optimization [3]. It uses a data structure called an *e-graph* to compactly represent a large amount of equivalent programs derived from a set of rewrite rules, avoiding the pass ordering problem associated with sequential optimization passes [4].

This paper explores the feasibility of introducing rewrite rules representing **approximations** for linear algebra operations, such as matrix multiplication, using equality saturation. The concept of compilers automatically inserting approximations for operations, trading precision for performance, is not new. A prominent example of this is the `-ffast-math` flag in GCC [5], which speeds up floating point operations by allowing the compiler to make many assumptions that do not respect the IEEE standard.

The approach explored in this paper requires that values in the expression being optimized are known ahead of time, which is highly restrictive. Nevertheless, we explore its applicability to the task of optimizing neural network inference, as the model parameters are all known after training is complete.

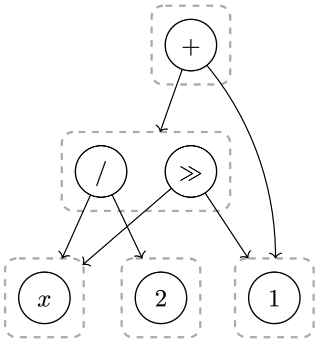


Fig. 1. E-graph for the expression $\frac{x}{2} + 1$ with the rewrite $\frac{x}{2} \to x \ll 1$ applied. Dotted boxes represent e-classes, and circles represent e-nodes. This e-graph represents both $\frac{x}{2} + 1$ and $(x \ll 1) + 1$.

II. Background

A. Equality saturation

An e-graph consists of a set of **e-classes** uniquely identified by an ID. E-classes are equivalence classes each containing a set of equivalent **e-nodes**. Each e-node represents a term of some language, with child terms being e-classes [6].

Rewrite rules grow the e-graph by introducing equivalencies. For example, we could have a rewrite rule that replaces integer division by 2 by a bitshift:

$$\frac{x}{2} \to x \gg 1$$

Equality saturation works by repeatedly applying these rewrite rules until:

1) The graph is saturated (applying further rewrites would not result in any changes)
2) A timeout is reached

Note that a saturated e-graph represents all possible equivalent expressions reachable from the initial expression using the given rewrite rules.

Then, the best possible equivalent expression can be extracted from the e-graph. The notion of "best" is defined by some **cost function** which outputs a cost value given an e-node. However, the extraction problem is known to be NP-Hard [7], as such computing an exact solution is intractable for large e-graphs.

Common solutions are simple bottom-up greedy approximation algorithms [6] which takes the minimum cost e-node from each e-class, and integer linear programming [8], [9] which computes an exact solution, but can be very slow on large e-graphs.

Neither of these approaches are compatible with the cost model used in this paper, so we employ a genetic algorithm to tackle the extraction problem.

B. Genetic algorithms

A genetic algorithm is a heuristic search method for optimization problems inspired by the biological process of evolution. [10]

The set of parameters that encode a solution are called a *chromosome*, and a single parameter is called a *gene*. The algorithm begins by randomly generating a *population* of initial solutions. These solutions are evaluated according to some *fitness function*. Then, it uses the fitness value to repeatedly sample pairs of parent solutions to "reproduce" by combining their chromosomes via some *crossover* method to produce the next population, with the hope being that combining two good solutions will produce an even better solution. Finally, random *mutation* is applied gene by gene in the new offspring to introduce more diversity in the solutions.

This process is repeated for some number of iterations, and the best solution is returned.

III. Equality saturation and approximations

A. Approximations as rewrite rules

There are many opportunities for approximations when performing matrix multiplication. We choose two to implement as rewrite rules: truncated singular value decomposition (SVD) and pruning. In addition to these, we also apply a basic rewrite rule for matrix multiplication associativity since this can give small cost reductions for free:

$$A(BC) \to (AB)C$$

a) ***Truncated SVD:*** The singular value decomposition of an $m \times n$ matrix A is a decomposition of the form

$$A = U\Sigma V^T$$

where U is an $m \times r$ matrix, Σ is an $r \times r$ diagonal matrix, and V is an $n \times r$ matrix, and $r = \min(m, n)$.

The truncated SVD is obtained by taking only the first k columns of each matrix, for some $1 \leq k < r$, yielding an approximation for A.

$$A_k = U_k \Sigma_k V_k^T$$

This gives the **best rank-k** approximation for A by the Eckart-Young theorem [11]. Given a matrix multiplication AB, we rewrite A into its truncated SVD.

$$AB \to U_k \Sigma_k V_k^T B$$

If A is $m \times n$, and B is $n \times p$, then the total cost is mnp multiplications. When multiplying by the truncated SVD instead, the total cost is

- $V_k^T B$: knp multiplications
- $\Sigma_k\left(V_k^T B\right)$: kp multiplications since Σ_k is diagonal
- $U_k\left(\Sigma_k\left(V_k^T B\right)\right)$: kmp multiplications

$knp + kp + kmp$ can be much smaller than mnp if k is sufficiently small. In the rewrite implementation, the truncated SVD for several k are inserted in a linearly decreasing manner. The step size is equal to $\lfloor \log_2(\min(m, n)) \rfloor$. For example, for a 784×100 matrix, $\lfloor \log_2(\min(784, 100)) \rfloor = 6$, thus the truncated SVDs for $k = 94, 88, 82, ...$ are merged into the node's eclass. An example of an e-graph with this rewrite applied is shown in Fig. 4.

b) ***Pruning:*** Pruning truncates small values in a matrix to 0. If Z is the number of zeroes in matrix A, then to compute AB where A is $m \times n$, and B is $n \times p$, the number of multiplications required is $(mn - Z)p$, since the zero entries in A can be ignored when using a sparse matrix representation.

The pruning rewrite rule is applied to simple terms such as plain variables or terms of their SVD. It simply introduces a new variable for the pruned value. This is done instead of adding a prune term to the language because the rewrite

$$A \to \text{prune}(A, k)$$

introduces a cycle into the e-graph, which complicates extraction significantly. The actual rewrite is

$$A \to P_{i,k}$$

where $P_{i,k}$ is a fresh variable name that has the value of matrix A where all $x_{ij} \in A$ such that $x_{ij} < 10^k$ are truncated to 0. Similar to the truncated SVD, multiple pruning thresholds are inserted into the eclass. The values of k chosen were −3, −2, and −1.

B. Cost model

We first compute a *canonical* expression value for each e-class, which is derived from the variable values. The root e-class's canonical value corresponds to the actual model output given the test set.

The cost function implemented not only outputs an integer cost, but also the actual value of the e-node and its relative error compared to the e-class's canonical value. The relative error η between the canonical value A and its approximation A' is computed in terms of the Frobenius norm, which is a matrix norm analagous to the standard Euclidean norm on vectors:

$$\eta = \frac{\|A - A'\|_F}{\|A\|_F}$$

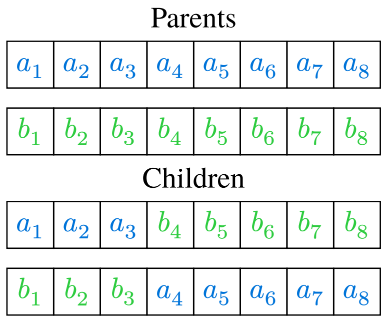

Parents
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
Children
a_1 a_2 a_3 b_4 b_5 b_6 b_7 b_8
b_1 b_2 b_3 a_4 a_5 a_6 a_7 a_8

Fig. 2. Single point crossover, with crossover point 3.

The cost function itself can be configured by passing a maximum allowed relative error $\eta_{\max}$. The ordering between costs is in part determined using this value. Specifically, let C_a and C_b be the integer cost values for enodes a and b, and η_a and η_b be their respective errors. An ordering for the costs of a and b are determined as follows:

- If $\eta_a > \eta_{\max}$ and $\eta_b > \eta_{\max}$ then compare by η_a and η_b.
- Otherwise if only one of η_a or η_b exceed $\eta_{\max}$ then the one that does not exceed $\eta_{\max}$ is smaller.
- Otherwise compare by C_a and C_b.

C. Genetic algorithm implementation

The e-graph extraction problem can be formulated as follows: Given an e-graph, choose an enode from each e-class such that the resulting expression's cost is minimized.

Solution encoding: A solution to the extraction problem is simply a mapping from e-class IDs to indices, with the indices corresponding to a specific e-node in that e-class. This is represented using a hashmap.

Population size: The population size used was 100, which is somewhat small however the cost function is quite expensive to compute due to the size of the matrices. We choose 100 to strike a balance between extraction time and solution quality.

Convergence: The algorithm stops once 5 iterations have passed without finding a new best solution, or after a maximum of 20 total iterations.

Selection: Since costs only have a relative order and not a single fitness value, rank-based selection is most suitable. Exponential rank selection is employed to more heavily favor the best solutions. The probability of choosing rank i (where rank 1 is the best solution) is given by

$$P(i) = \frac{0.9^{i-1}}{\sum_{k=1}^{n} 0.9^{k-1}}$$

where n is the population size (in this case 100).

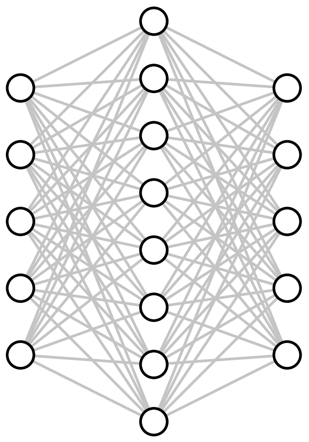


Fig. 3. A multilayer perception with 1 hidden layer.

Crossover: Crossover is done using simple single point crossover. The 2 parent solutions are flattened into an array, and a random index is chosen to be the crossover point. The two slices past the crossover point are then swapped, producing 2 children, as shown in Fig. 2.

Mutation: We chose a mutation rate of 5%, this rate is rather arbitrary, but seems to produce good results. Mutation simply randomly selects a different index for an e-class.

IV. Applications

Our approach to equality saturation comes with the restriction that all variable values in the expression are known ahead of time, since we rely on computing an approximation error relative to some canonical value. We use `fuzzy-eqsat` to optimize inference for two classification models.

1) A simple multilayer perception with 1 hidden layer (784 x 100 x 10 neurons)
2) LeNet 5 [12], one of the earliest convolutional neural networks

Both of these models are trained on the task of handwritten digit recognition using the MNIST dataset [13].

The two neural network architectures used are both feedforward neural networks, as such they can easily be converted into a single expression for optimization.

`fuzzy-eqsat` is implemented using the `egg` equality saturation library written in Rust. We define the following grammar for expressions:

Num	n		*integer literals*	
Expr	e	::=	x	*variable*
		|	$e + e$	*matrix addition*
		|	$e * e$	*matrix multiplication*
		|	$\mathrm{diag\_mul}(e_1, e_2)$	*diagonal matrix mult.*

$\mid$	$\text{svd\_u}(x, n)$	*U matrix of truncated SVD*
$\mid$	$\text{svd\_d}(x, n)$	*Σ matrix of truncated SVD*
$\mid$	$\text{svd\_vt}(x, n)$	*V^T matrix of truncated SVD*
$\mid$	$\text{ReLU}(e)$	*ReLU function*
$\mid$	$\tanh(e)$	*hyperbolic tangent function*
$\mid$	$\text{softmax}(e)$	*softmax function*

The grammar includes standard matrix addition and multiplication, `diag_mul` is a special case of matrix multiplication where e_1 is a diagonal matrix. This is faster to compute than normal matrix multiplication so it is discounted in the cost model.

The `svd_u`, `svd_d` and `svd_vt` terms correspond to the U_k, Σ_k, V_k^T matrices of the truncated SVD respectively, where the truncation factor k is given by the second argument n. `ReLU`, `tanh`, and `softmax` are included since they are common activation functions. `ReLU` and `tanh` are applied element-wise, and `softmax` is applied column-wise to matrices.

The network shown in Fig. 3 matches the architecture of the first classification model tested, albeit with fewer neurons. In `egg`, which uses an s-expression syntax, this network corresponds to the expression:

$$(\text{softmax } (+ \ (* \ w_1 \ (\text{ReLU } \ (+ \ (* \ w_0 \ x) \ b_0)) \ b_1))$$

where w_i and b_i correspond to the weight matrix and bias vector for the i-th layer respectively, and x corresponds to the model input.

The model parameters are constants, but the model input obviously can't be known ahead of time. We resolve this by treating x as a random variable which is defined by some distribution. The value assigned to x is the test set matrix from the classification problem's data set, since this approximates the distribution of possible model inputs. When evaluating the expression, this is functionally the same as batch inference.

V. Evaluation

A. *Experimental setup*

We tested many different configurations for both neural networks by:

- Varying the maximum allowed relative error $\eta_{\max}$ (2%, 5%, 10%, 25%)
- Applying only the truncated SVD rewrite, only the pruning rewrite, and both rewrites

Extraction is run 3 times on each configuration, taking the best solution found out of the 3 runs, in order to minimize noise due to the randomness inherent in genetic algorithms.

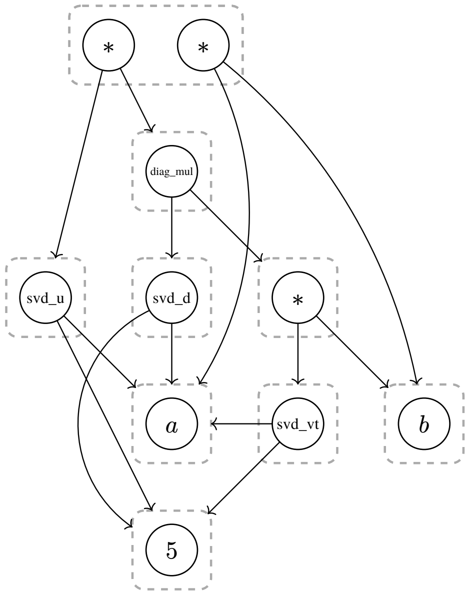


Fig. 4. E-graph for the expression (* a b) after applying the truncated SVD rewrite, with $k = 5$

Note that in the case of LeNet 5, the equality saturation includes no rewrites for convolutions. As such, the expression being optimized is only the last three fully connected layers of the network, since those are the relevant parts that can be optimized. Accordingly, the value used for the model input is the output of the original model's convolutional layers on the test set.

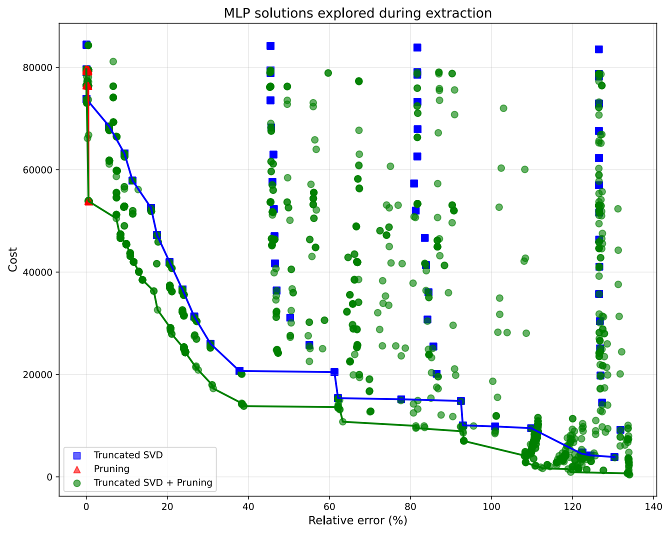
The experiments were run on a Lenovo Thinkbook G16 with a AMD Ryzen 7 8845HS, running Arch Linux 6.17. Extraction time is not included as they were fairly consistent, but runs for the MLP took 15-30 seconds, while LeNet took 1-2 minutes.

B. *Effect of optimizations*

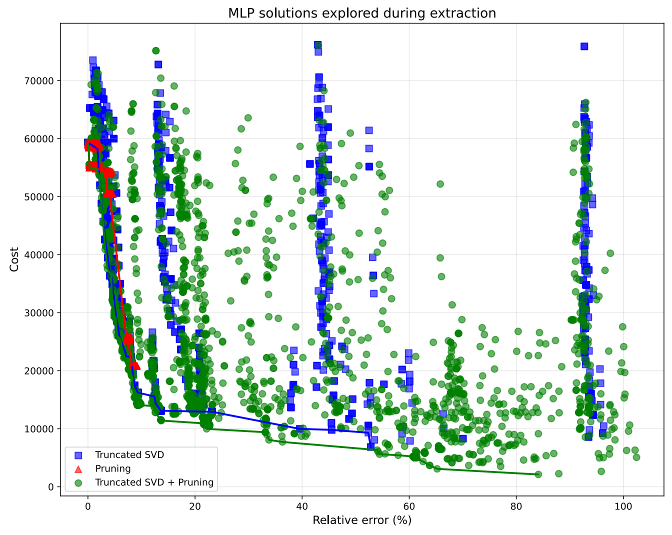
Looking at Table I, in the MLP case, classification accuracy sees no decrease at all until the relative error surpasses 10%. The truncated SVD rewrite by itself is fairly effective, giving a 20.7% cost reduction without compromising on accuracy, and a 54% reduction with only a 1% accuracy loss.

Pruning only achieves the same solution on each error threshold: a 32.4% cost reduction with 0.6% relative error. This indicates that there are many parameters in the MLP that have little impact on the final result, so pruning can be applied freely without tradeoffs. Pruning does not make the e-graph much bigger, so the solution space is quite small. This can be seen in Fig. 5 a), there are only a few triangles on the very left side.

The optimization is most effective when the two rewrite rules are combined, since a matrix can be decomposed into its SVD and the resulting decomposition



a) MLP expression



b) LeNet-5 expression

Fig. 5. Explored solutions for MLP and LeNet-5 with each rewrite configuration. Points joined with a line mark the Pareto front.

TABLE I
COST AND ERROR OF OPTIMIZED MLP EXPRESSION FOR EACH REWRITE RULE CONFIGURATION
BASE CLASSIFICATION ACCURACY: 97.02%
BASE COST: 79620

	Truncated SVD				**Pruning**				**Truncated SVD + Pruning**			
$\eta_{\max}$	Cost	Rel. error	Rel. cost	Accuracy	Cost	Rel. error	Rel. cost	Accuracy	Cost	Rel. error	Rel. cost	Accuracy
2%	73790	0.0%	-7.3%	97.02%	53806	0.6%	-32.4%	97.03%	53806	0.6%	-32.4%	97.03%
5%	73790	0.0%	-7.3%	97.02%	53806	0.6%	-32.4%	97.03%	53806	0.6%	-32.4%	97.03%
10%	63170	9.4%	-20.7%	97.03%	53806	0.6%	-32.4%	97.03%	45424	9.9%	-43.0%	97.05%
25%	36620	23.7%	-54.0%	96.01%	53806	0.6%	-32.4%	97.03%	24338	24.4%	-69.4%	95.93%

TABLE II
METRICS FOR OPTIMIZED LENET EXPRESSION FOR EACH REWRITE RULE CONFIGURATION
BASE CLASSIFICATION ACCURACY: 98.56%
BASE COST: 59348

	Truncated SVD				**Pruning**				**Truncated SVD + Pruning**			
$\eta_{\max}$	Cost	Rel. error	Rel. cost	Accuracy	Cost	Rel. error	Rel. cost	Accuracy	Cost	Rel. error	Rel. cost	Accuracy
2%	55112	2.0%	-7.1%	98.51%	54825	1.5%	-7.6%	98.56%	54825	1.5%	-7.6%	98.56%
5%	32426	5.0%	-45.4%	98.37%	50461	4.0%	-15.0%	98.49%	29377	4.9%	-50.5%	98.40%
10%	16232	9.3%	-72.7%	98.19%	20823	8.8%	-64.9%	98.22%	14083	9.4%	-76.3%	98.22%
25%	12931	23.1%	-78.2%	94.86%	20823	9.1%	-64.9%	98.18%	9982	22.2%	-83.2%	95.38%

matrices can be pruned, further reducing cost. When $\eta_{\max} = 2\%$ or 5%, the solution is no better than pruning by itself, however when $\eta_{\max}$ is relaxed to 10% or 25%, it is able to find a solution that is cheaper than either of the individual rewrite cases with similar accuracy.

In the case of LeNet-5, the accuracies in Table II show that this model is more sensitive to changes in the output, as the accuracy begins to fall before reaching a 10% relative error, unlike the MLP. The model suffers heavy accuracy losses of more than 3% when the relative error exceeds 20%.

Truncated SVD is notably more impactful in this case, giving a 72.7% reduction in cost while only incurring a 0.32% accuracy loss. Pruning is also more effective though introduces more error, indicating that the individual parameters are more meaningful compared to the MLP. This is to be expected since LeNet-5 is a convolutional model with fewer parameters than the MLP, hence each neuron activation encodes more information.

Combining the two rewrites for LeNet-5 is noticeably less effective compared to the MLP, resulting only in a 4-5% cost reduction. Indeed, Fig. 5 b) shows that the Pareto fronts for truncated SVD and truncated SVD + pruning are very close together, unlike the MLP case where there is a sizable gap.

In all cases, it is evident that going past 10% error results in diminishing returns in terms of cost reduction. This can be seen in Fig. 5 b), the Pareto front has a very steep downwards slope up to 10% error before flattening out quickly. The MLP does not seem to hit these diminishing returns until around 40% error.

VI. Limitations

A small limitation of the experimental setup is the limited number of pruning thresholds used, resulting in very few data points for the pruning only configuration. This could easily be expanded, but was not done due to time constraints.

As for the approach, the most obvious limitation is the requirement that all values are known ahead of time. This greatly restricts the number of possible usecases, though it is unclear how this requirement could be relaxed. Additionally, it suffers from scaling issues in two ways:

1) The runtime of the cost function scales with the size of the model. As a result, extraction will be very slow for large models.
2) Rewrite rules like the truncated SVD rewrite insert several approximations with varying levels of accuracy. As more and more rewrite rules are introduced and compose together, it will lead to a combinatorial explosion in the size of the solution space.

The crossover method of the genetic algorithm could also be improved to be aware of the structure of the e-graph rather than simply swapping contiguous slices ordered by e-class IDs.

Finally, the cost values of the optimized expressions are simply part of the model and may not actually be faster in practice. Full performance data was not collected, but a few tests were run, comparing the runtime of a model that was optimized using the truncated SVD rewrite to the original model. We found that the "optimized" model was slightly slower.

This could be caused by a variety of factors, but the most likely explanation is the CPU cache. A single matrix is stored in a contiguous block of memory which is very cache-friendly, while multiplying by the decomposed matrices performs three separate matrix multiplications, resulting in more cache misses. For similar reasons, pruning is unlikely to result in a real speedup unless the resulting matrix is sufficiently sparse.

The models tested in this paper are very small. It is likely that the model being optimized would have to be much larger before a significant speedup could be observed.

VII. Related Work

Equality saturation has seen numerous uses in compilers and hardware synthesis. [1] uses equality saturation to optimize Julia programs. Cranelift [14] is a compiler backend for Rust and WebAssembly employing acyclic e-graphs (aegraphs) for its optimization phase. This is weaker than regular equality saturation, which can include cycles, but reduces compilation time. [3] uses equality saturation to automatically find optimal circuit architectures for floating point operations.

There has also been a fair amount of research devoted to tackling the e-graph extraction problem. [15] proposes a solution based on simulated annealing with solution space pruning which achieves impressive results on large e-graphs for hardware synthesis. [16] takes a similar approach also based on simulated annealing, but combined with ant colony optimization.

[17] formulates the e-graph problem probabilistically, converting the discrete optimization problem into a continuous differentiable form and applying gradient descent.

Both of these approaches could be promising alternatives to the genetic algorithm extractor used in this paper, though the difficulty lies in coming up with a suitable cost model formulation for either of them to be usable since `fuzzy-eqsat`'s cost model has multiple objectives.

VIII. Conclusion

This paper presents `fuzzy-eqsat` which introduces approximate rewrite rules to equality saturation, namely truncated SVD and matrix pruning for linear algebra expressions. Experimental results show large cost reductions on basic neural networks under the cost model used, though this does not lead to actual runtime improvement in practice.

For future work, the main improvement to be made is reformulating the cost model so that better extraction methods could be used instead of genetic algorithms, improving reliability and scalability. Additionally, generalizing the truncated SVD rewrite to tensor train decomposition [18] could also be useful for applying this approach to larger models that have higher dimensional parameter tensors.

References

[1] J. Merckx, T. Besard, and B. D. Sutter, “Equality Saturation for Optimizing High-Level Julia IR.” [Online]. Available: https://arxiv.org/abs/2502.17075

[2] G. H. Smith *et al.*, “Scaling Program Synthesis Based Technology Mapping with Equality Saturation.” [Online]. Available: https://arxiv.org/abs/2411.11036

[3] S. Coward, G. A. Constantinides, and T. Drane, “Automating Constraint-Aware Datapath Optimization using E-Graphs.” [Online]. Available: https://arxiv.org/abs/2303.01839

[4] R. Tate, M. Stepp, Z. Tatlock, and S. Lerner, “Equality Saturation: A New Approach to Optimization,” *Logical Methods in Computer Science*, Mar. 2011, doi: 10.2168/lmcs-7(1:10)2011.

[5] “Options That Control Optimization ,” 2025, [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[6] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, and P. Panchekha, “egg: Fast and Extensible Equality Saturation,” *Proc. ACM Program. Lang.*, vol. 5, no. POPL, Jan. 2021, doi: 10.1145/3434304.

[7] M. B. Stepp, “Equality Saturation: Engineering Challenges and Applications,” pp. 79–83, 2011.

[8] S. Coward, G. A. Constantinides, and T. Drane, “Automatic Datapath Optimization using E-Graphs.” [Online]. Available: https://arxiv.org/abs/2204.11478

[9] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, “IMpress: Large Integer Multiplication Expression Rewriting for FPGA HLS,” in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–10. doi: 10.1109/FCCM53951.2022.9786123.

[10] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[11] C. Eckart and G. Young, “The Approximation of One Matrix by Another of Lower Rank,” *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936, doi: 10.1007/BF02288367.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[13] L. Deng, “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012, doi: 10.1109/MSP.2012.2211477.

[14] C. Fallin, “Aegraphs: Acyclic e-Graphs for Efficient Optimization in a Production Compiler..” 2023.

[15] C. Chen, G. HU, C. Yu, Y. Ma, and H. Zhang, “E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis.” [Online]. Available: https://arxiv.org/abs/2504.11574

[16] R. Li, L. Li, H. Yu, and Y. Ha, “ESACO: Fast E-Graph Extraction via Orchestrated Simulated Annealing-based Local Search and Ant Colony Optimization-based Global Search,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, no. , p. 1, 2025, doi: 10.1109/TCAD.2025.3611101.

[17] Y. Cai, K. Yang, C. Deng, C. Yu, and Z. Zhang, “SmoothE: Differentiable E-Graph Extraction,” in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, in ASPLOS '25. Rotterdam, Netherlands: Association for Computing Machinery, 2025, pp. 1020–1034. doi: 10.1145/3669940.3707262.

[18] I. V. Oseledets, “Tensor-Train Decomposition,” *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011, doi: 10.1137/090752286.