BoolGebra: Attributed Graph-learning for Boolean Algebraic Manipulation

Yingjie Li[1], Anthony Agnesina[2], Yanqing Zhang[3], Haoxing Ren[2], Cunxi Yu[1]
[1]*University of Maryland, College Park*, College Park, MD, USA
[2]*NVIDIA*, Austin, TX, USA
[3]*NVIDIA*, Santa Clara, CA, USA
{yingjiel, cunxiyu}@umd.edu, {aagnesina, yanqingz, haoxingr}@nvidia.com

I. Motivation and Introduction

Logic optimization is an essential stage in the design automation flow for digital systems as the performance of the system at logic level can have significant impacts on the final chip area, timing closure, and the power efficiency of the system. Logic optimization is a technology-independent circuit optimization at the logic level conducted on multi-level technology-independent representations such as And-Inverter-Graphs (AIGs) [1] and Majority-Inverter-Graphs (MIGs) [2] of the digital logic. Existing state-of-the-art (SOTA) Directed-Acyclic-Graphs (DAGs) aware Boolean optimization algorithms, such as structural *rewriting* (`rw`) [1], *resubstitution*(`rs`) [3], and *refactoring*(`rf`) [1] in ABC [4], are conducted on the AIG data structure with a graph-level single optimization concept, i.e., all nodes in the graph have one same fixed optimization opportunity, while overlooking other potential optimization opportunities. [5] proposes *orchestrated logic optimization*, which is a fine-grained node-level logic optimization method incorporating multiple optimization techniques within a single AIG traversal. However, the enlarged search space pose a significant challenge in searching optimal solutions without domain knowledge.

Our work proposes new machine learning aided solutions to AIG optimizations with domain knowledge discovery for orchestrated logic optimization [5]. Specifically, BoolGebra employs machine learning (ML), specifically Graph Neural Networks (GNNs) [6]–[8] as an aid to conventional design solutions. BoolGebra takes the feature embedded AIG as inputs and predicts the optimization results directly. Based on the fast-acquired inference result, the search space is thus shrunk, which aids the logic optimization to locate the optimized solution efficiently. Once the model is well trained, it shows its generalization capability for cross-design evaluations.

II. BoolGebra Framework

BoolGebra framework is illustrated with two parts: **dataset preparation**, where input graph data is embedded and normalized and **model construction**.

Dataset Preparation – The dataset preparation process is shown in Figure 1a – Figure 1d. Given an AIG, the node embedding for the PI nodes (node a, b, c in Figure 1a), which have no fanins, are all set as -99. We **first** generate the graph dataset with [9], where the logic graph is structurally augmented with node-level logic optimization for a single functionality. The generated information from [9] include the structural information, i.e., the edgelist for the AIG, and the functional information, i.e., edge features in Figure 1b, where if the input edge of the node is inverted (dashed line in Figure 1a), it will be encoded as 1; otherwise, encoded as 0. **Furthermore**, targeting the logic optimization predictions, the task-specific node embedding are in two categories:
(1) Static optimization features, where the features are static w.r.t the AIG structure, and will not alter with different optimization samplings. Specifically, the static optimization features are collected in a 6-dimensional vector including `rw`/`rs`/`rf` transformability and local optimization gain (third/fifth/seventh and fourth/sixth/eighth columns in Figure 1c). If optimization operation is applicable at the node, the transformability bit is set as 1 and the corresponding gain bit is set as the optimization gain, i.e., the number of AIG minimization with the optimization applied; otherwise, the transformability bit is set as 0, and the gain bit is set as -1. **(2) Dynamic sampling features**, where the features change as different optimization samplings are given, which introduce different optimization results. The dynamic feature embedding is shown in Figure 1d. For each logic node, it is a 4-bit one-hot representation, indicating which operation is practically applied to the node under the specific optimization sampling. The 4 bits indicate none of optimizations is applied, `rw` is applied, `rs` is applied, `rf` is applied, respectively. The practically applied operation is set as 1 in one-hot representation while others are set as 0. For example, sample 1 in Figure 1f and sample 2 in Figure 1e are two different graph structures for the same functional design. Their corresponding dynamic sampling features are thus different shown in Figure 1d, and its labels indicate different optimization performance.

Finally, the training labels are normalized, where the AIG minimization is normalized w.r.t the highest node reduction number in the optimization samples, i.e., we set ratio of the gap between the node reduction in the current sample to the most node reduction of all the samples in the dataset as the label for the current sample. For example, in Figure 1e and Figure 1f, the best optimization for the dataset with these two samples reduces the AIG for 3 nodes in Figure 1f, thus, the label for Sample 1 (Figure 1f) is 0 and the label for Sample 2 (Figure 1e) is 0.66. Thus, the optimization labels are normalized to $[0, 1]$.

Model Construction – As shown in Figure 1g, the model consists of two parts: the graph embedding part utilizing GraphSAGE for attributed graph feature extraction and aggregation, and the prediction part employing multi-layer fully connected neural networks for predictions. In the first part, the feature and structure embedding of the AIG are fed into the model, and three GraphSAGE Convolution (Conv) layers are utilized. Each Conv layer with hidden dimension of 512 and output dimension of 64 is accompanied by a dropout layer with a dropout rate of 0.1 and a nonlinear layer (ReLU6). The downstream model consists of three dense layers with output dimensions of 1000, 200, and 1 respectively. Finally, the last linear layer is connected to a sigmoid layer to produce prediction results in the range of $[0, 1]$.

The aforementioned technical steps are integrated to form the BoolGebra flow. The BoolGebra flow comprises three key steps: **(1)** Random sampling of a large batch of Boolean manipulation decisions on a given AIG. **(2)** Pruning the randomly sampled design space using the BoolGebra model, where samples with smaller predictions are better ones. **(3)** Evaluation of the top solutions based on the prediction results obtained in step 2 to derive the final AIG reduction outcomes.

979-8-3503-4859-0/DATE24/© 2024 EDAA

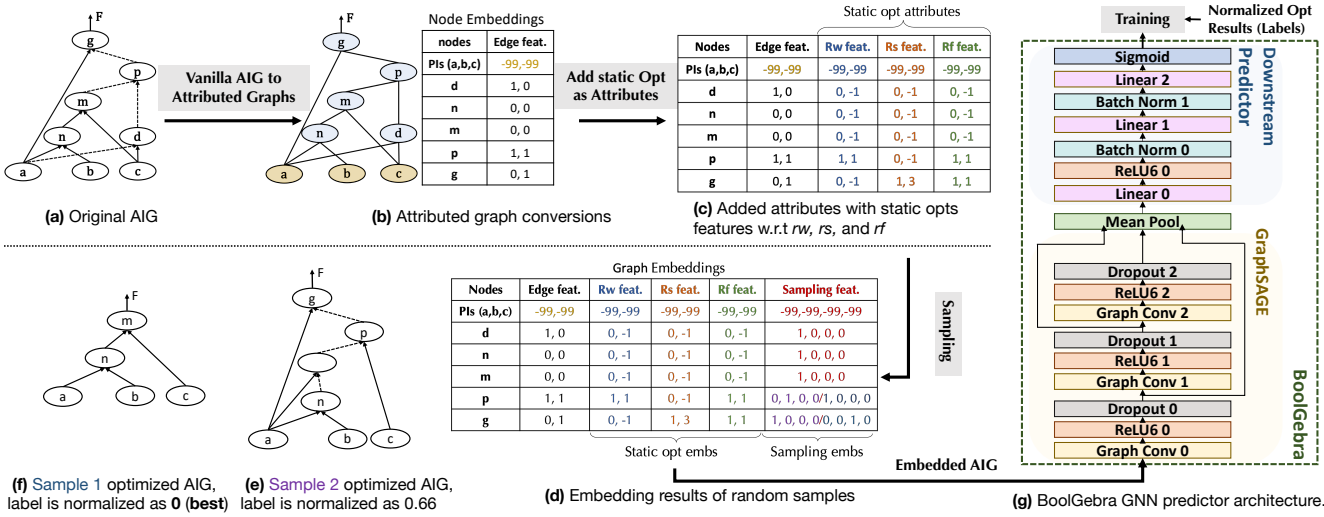
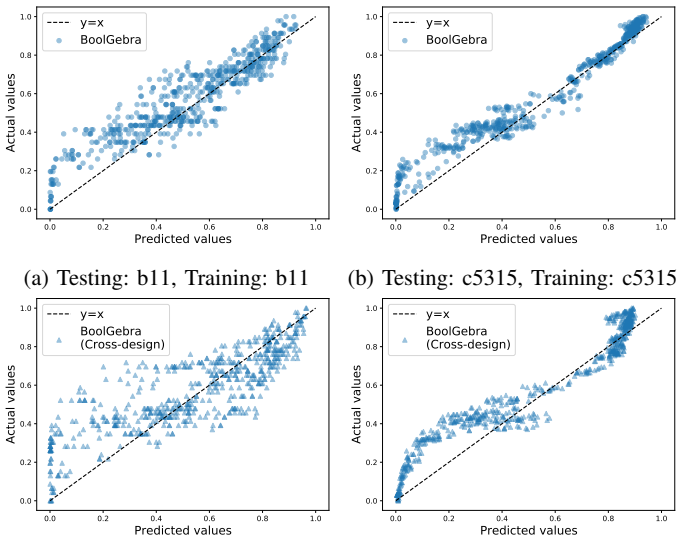


Fig. 1: The design flow of BoolGebra including the feature embedding with static and dynamic sampling embedding ((a) – (f)), and the model construction ((g)).

III. Experiments



(a) Testing: b11, Training: b11 (b) Testing: c5315, Training: c5315

(c) Testing: b11, Training: c5315 (d) Testing: c5315, Training: b11

Fig. 2: Inference evaluation for predicting AIG minimization performance for given manipulation decisions. (a)-(b): Design-specific evaluations; (c)-(d): cross-design evaluations.

The experimental results are presented with: **(1)** Model inference with design-specific evaluation, where the model are trained and tested with the samples with the **same** logic functionality. For example, in Figure 2(a-b), the predicted logic optimization results of design `b11/c5315` are from the inference of the model trained with design `b11/c5315`, respectively. Both results show the learnt correlations between the actual values and the predicted values. **(2)** Model inference with cross-design evaluations, where the model are trained with a design functionality while tested with designs with **different** logic functionalities. For example, in Figure 2(c-d), we present the cross-design evaluation of the model trained with `b11/c5315` while tested with `c5315/b11`(Figure 2c and Figure 2d). **(3)** The comparative analysis between the exact node optimization results and the original And-Inverter Graph (AIG) node size. We picked the top 10 optimized samples, as predicted by the model, which were **exclusively trained on a single design 'b11', i.e., cross-design inference for other designs**. The corresponding outcomes are documented in Table I. Specifically, we have included both the average performance (BG-Mean) of the predicted top 10 samples and the best result (BG-Best). The performance enhancements compared with `rw`, **rs**, and `rf` are measured at 3.6%, 5.3%, and 5.5% respectively, underscoring the potential of the BoolGebra flow. Additionally, the runtime of evaluating BoolGebra to all designs consumes **inference time only** (≤ 0.05 sec) while the model is trained offline only with the design 'b11'.

TABLE I: Boolean minimization evaluations compared to state-of-the-art (SOTA) synthesis methodologies. The results are presented in optimized AIG sizes (%) over the original AIG sizes. **BG**: BoolGebra.

Designs	`rw`	`rs`	`rf`	BG (Mean)	BG (Best)
b07	0.981	0.975	0.959	0.940	0.934
b08	0.935	0.923	0.987	0.917	0.910
b09	0.978	0.971	0.993	0.956	0.956
b10	0.978	0.950	0.978	0.937	0.933
b11	0.895	0.897	0.881	0.834	0.828
b12	0.968	0.964	0.988	0.950	0.950
c2670	0.824	0.895	0.862	0.798	0.794
c5315	0.836	0.958	0.893	0.804	0.801
Avg	0.925	0.942	0.943	0.892	0.888
Impr.	**3.6%**	**5.3%**	**5.5%**	-	-

References

[1] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: A fresh look at combinational logic synthesis," in *DAC*, 2006.

[2] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on CAD*, vol. 35, no. 5, pp. 806–819, 2015.

[3] A. M. R. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

[4] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *http://www. eecs.berkeley. edu/alanmi/abc*, vol. 17, 2007.

[5] Y. Li, M. Liu, M. Ren, A. Mishchenko, and C. Yu, "DAG-aware Synthesis Orchestration," *arXiv preprint arXiv:2310.07846*, 2023.

[6] Z. He *et al.*, "Graph learning-based arithmetic block identification," in *Proc. ICCAD*, 2021.

[7] N. Wu, *et al.*, "Gamora: Graph Learning based Symbolic Reasoning for Large-Scale Boolean Networks," *DAC*, 2023.

[8] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for fpga hls using graph neural networks," in *ICCAD*, 2020.

[9] Y. Li, *et al.*, "Verilog-to-PyG-A Framework for Graph Learning and Augmentation on RTL Designs," in *2023 ICCAD*. IEEE, 2023, pp. 1–4.