DESIGN AUTOMATION CONFERENCE

59

JULY 10 - 14, 2022

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Functionality Matters in Netlist Representation Learning

Ziyi Wang[1], Chen Bai[1], Zhuolun He[1], Guangliang Zhang[2], Qiang Xu[1], Tsung-Yi Ho[1], Bei Yu[1], Yu Huang[2]

[1]The Chinese University of Hong Kong
[2]HiSilicon

Introduction

Background

- Recently, there is a surge in incorporating **graph learning** in electronic design automation (EDA).
- Most existing works follow a **representation learning paradigm** consisting of two steps: first, learn low-dimensional representations from the high-dimensional raw data and then conduct classification or regression based on the learned representations.
- The learned representations play a **dominant** role in improving model performance.

Problem Definition

focus on netlist: basic data structure used in several steps of the EDA flow.

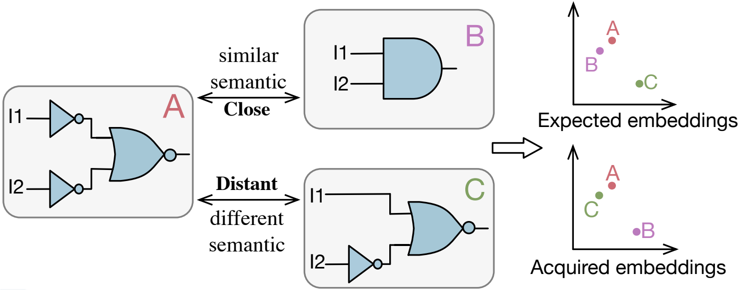
Netlist Representation Learning

design a general learning methodology that automatically discovers gate/netlist representations capturing their basic underlying semantics.

- We hope the representation can facilitate multiple downstream netlist tasks

Defect of previous works

- Previous works only focus on the graph structural information, which varies greatly across netlists.
- **We should extract general knowledge!**



Previous Structural methods fail to capture the underlying semantic

Methodologies

Question:

- What is the **universal** and **transferable** knowledge that is shared across different netlists?
- Can we **extract** the shared prior knowledge to enhance the ability of graph learning models?

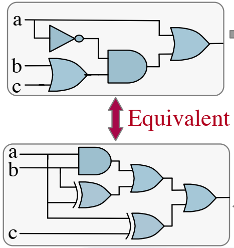
Gate Functionality and Boolean Equivalence

Logic functionality: keep the same for a specific gate type across different designs.

- Can be transfered and generalized to unseen netlists, even with totally different topology!

Can we extract this information?

- **Yes**! –> **Key**: Boolean Equivalence



a + a'(b+c)

a(b+b'+c') + a'(b+c)

ab + (ab'+a'b) + (ac'+a'c)

ab + xor(a,b) + xor(a,c)

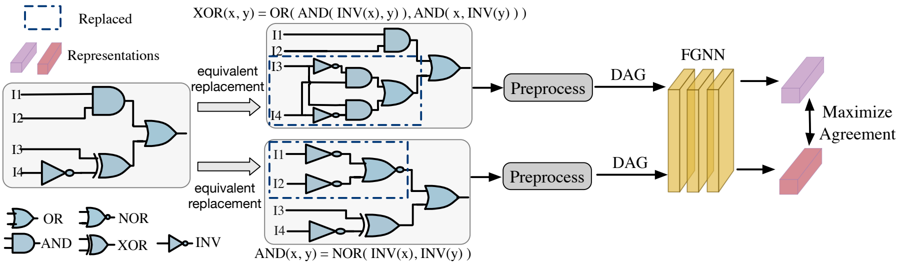
example of Boolean equivalence

Contrastive Learning

Main Idea: capture statistical dependencies by separating positive samples from negative samples in the embedding space. **Goal**: learn an encoder $f : x \rightarrow \boldsymbol{e}, \boldsymbol{e} \in \mathbb{R}^n$ that for any sample x:

$$score(f(x), f(x^+)) >> score(f(x), f(x^-)). \tag{1}$$

- Positive sample: augmentation of input sample
  - Augmentation method is **critical**!
    Key to the success of CL: generating augmented views that involves **enough variance** while avoiding any **semantic changes**.



We design a netlist augmentation scheme to generate positive samples, which is based on Boolean Equivalence.

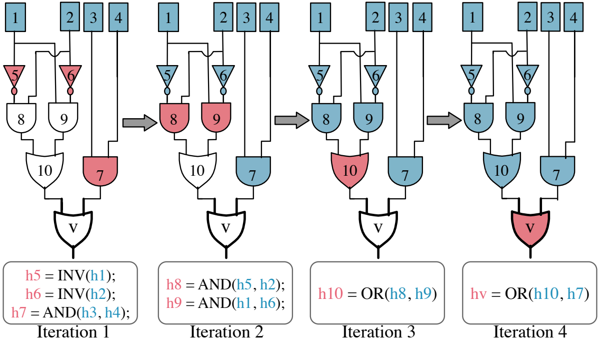
- Iterative random sub-netlist replacement.
- Positive sample pair share the **same functionality**, while having totally **different topology**.
- Maximizing agreement between positive samples: embedding of netlists with **similar semantic** (functionality) tend to be **close**

Customized Graph Neural Network: FGNN

- **Heterogeneous:** learn an individual aggregator for each gate type

  In practice, we learn 8 basic gate (cell) functions including `AND, OR, INV, MAJ, MUX, NAND, NOR` and `XOR`.

- **Asynchronous** message passing scheme: mimic the logic computation



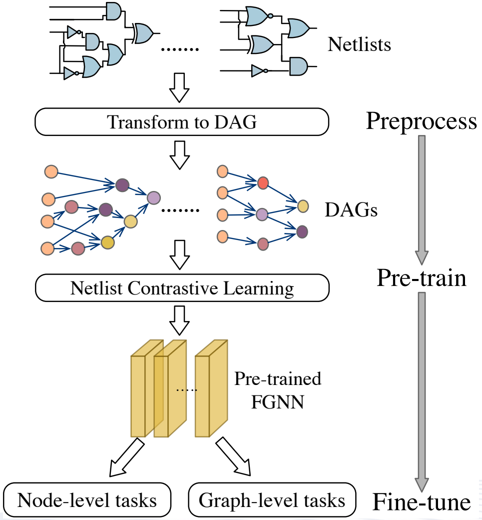
Curriculum Learning

Mimic the learning procedure of human beings: from easy to hard.

- first train the model on a small number of easy cases, and then train on **successively more complex** cases with increased batch size.

- Two difficulty dims:
  (1) netlist complexity (scale)
  (2) topological similarity between positive samples (times of replacement)

Overall Flow



Experimental Results

Experimental Setting

We evaluate our proposed framework on two different downstream netlist tasks covering both **local** and **global** scenarios.

i **Arithmetic Block Boundary Detection**:
  - identify the boundary wires of adders from a large-scale flatten netlist
  - node-level **local** task

ii **Circuit Classification**:
  - distinguish between circuits with different functionality, e.g., adder, multiplexer, etc.
  - circuit-level **global** task

Application 1: local scenario

- Evaluated on open-source RISC-V CPU designs

Table: Statistics of the dataset for sub-netlist identification with 6 different types of adders.

Architecture	*Rocket (test)*		*BOOM (train)*	
	#gates	#wires	#gates	#wires
Brent-Kung	24340	58124	139526	366280
Cond-sum	24737	57708	138358	360455
Hybrid	25491	60287	141319	369622
Kogge-Stone	24540	57726	139005	361962
Ling	26179	62864	143903	378354
Sklansky	25208	59567	141093	369774

Application 1: result

- Previous works are subjected to **sharp performance degradation** when generalizing to unseen data.
- Our method shows superior generalization ability.

Table: Performance of different models on adder output boundary prediction in terms of recall and F1-score. Best results are emphasized with **boldface**. Our proposed FGNN + NCL framework outperforms other models in all the test cases.

Case	Ratio	EV-CNN [Fay+19]		GraphSage [Ham+17]		ABGNN [He+21]		FGNN		FGNN + NCL	
		Recall	F1-Score	Recall	F1-Score	Recall	F1-Score	Recall	F1-Score	Recall	F1-Score
1	1/6	0.602	0.575	0.643	0.656	0.657	0.682	0.684	0.715	**0.734**	**0.753**
2	2/6	0.612	0.605	0.758	0.757	0.734	0.74	0.784	0.788	**0.857**	**0.839**
3	3/6	0.633	0.615	0.854	0.865	0.877	0.881	0.916	0.914	**0.940**	**0.937**
4	4/6	0.662	0.637	0.883	0.889	0.921	0.917	0.931	0.933	**0.954**	**0.947**
5	5/6	0.738	0.648	0.905	0.898	0.927	0.922	0.952	0.944	**0.966**	**0.951**
6	6/6	0.768	0.655	0.919	0.917	0.945	0.941	0.963	0.952	**0.969**	**0.957**

Application 2: global scenario

Table: Statistics of the dataset for circuit classification, including adder, subtractor, multiplier, and divider. We try to avoid involving similar designs used for training in the test dataset.

Module	Train architectures	Train #	Validate / Test architectures	Validate / Test #
Adder	Brent-Kung, Cond-Sum, Hybrid, Koggle-Stone, Ling, Sklansky	450	Block Carry Look-head, Carry Look-head, Carry Select, Carry-skip, Ripple-Carry	100 + 300
Subtractor	Hybrid, Koggle-Stone, Ling	250	Brent-Kung, Cond-Sum, Sklansky	50 + 150
Multiplier	Array, Booth-Encoding	550	Wallace, Dadda, Overturned-stairs, (4,2) compressor, (7,3) counter, Redundant binary addition	150 + 500
Divider	Array	250	Array	50 + 200
Total	/	1500	/	350 + 1150

Application 2: result

- Our proposed framework shows substantial performance superiority over the baseline methods across all the cases.

Table: Summary of performance on netlist classification in terms of accuracy. The second column gives the ratio of the training data size to the testing data size. Our proposed FGNN + NCL framework achieves the **best** performance on all the cases and suffers from slighter degradation when the training data scale is reduced.

Case	Ratio	GIN [Xu+18]	EV-CNN [Fay+19]	DVAE [Zha+19]	Ours
1	1.3	0.762±0.020	0.904±0.011	0.913±0.005	**0.975±0.008**
2	1	0.745±0.026	0.896±0.009	0.902±0.007	**0.962±0.007**
3	0.7	0.737±0.022	0.884±0.003	0.895±0.009	**0.960±0.009**
4	0.5	0.730±0.015	0.877±0.006	0.885±0.010	**0.951±0.005**
5	0.3	0.725±0.028	0.859±0.015	0.871±0.003	**0.945±0.007**

Conclusion

- Learning feasible representations from raw gate-level netlists is critical for applying machine learning techniques to EDA.
- We need customization to fully utilize prior knowledge and achieve better performance, instead of simply applying the general GNN architectures.
- In this paper, we propose:
  - a contrastive learning based pre-training framework for extracting basic semantic of netlists.
  - a specialized GNN for netlist functionality learning.
- We conduct comprehensive experiments on several complex real-world designs to evaluate our methods.

Reference I

[1] A. Fayyazi, S. Shababi, P. Nuzzo, S. Nazarian, and M. Pedram, "Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations", in *Proc. DATE*, 2019, pp. 638–641.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs", in *Proc. NIPS*, 2017, pp. 1024–1034.

[3] Z. He, Z. Wang, C. Bai, H. Yang, and B. YU, "Graph learning-based arithmetic block identification", in *Proc. ICCAD*, 2021.

[4] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?", *Proc. ICLR*, 2018.

[5] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs", *Proc. NIPS*, 2019.

THANK YOU!