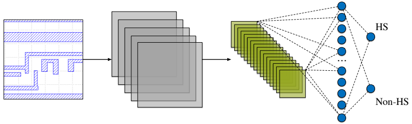
High Performance Graph Convolutional Networks with Applications in Testability Analysis

Yuzhe Ma[1], Haoxing Ren[2], Brucek Khailany [2], Harbinder Sikka[2], Lijuan Luo[2], Karthikeyan Natarajan[2], Bei Yu[1]

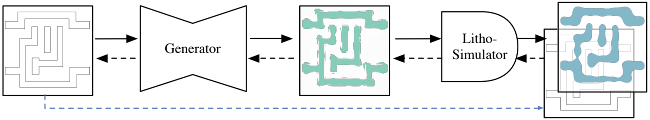
[1]The Chinese University of Hong Kong
[2]NVIDIA

Learning for EDA

- Verification [Yang et.al TCAD'2018]

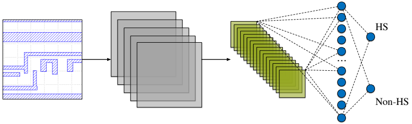


- Mask optimization [Yang et.al DAC'2018]

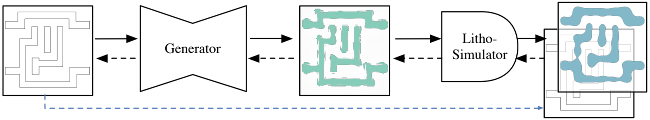


Learning for EDA

- Verification [Yang et.al TCAD'2018]



- Mask optimization [Yang et.al DAC'2018]

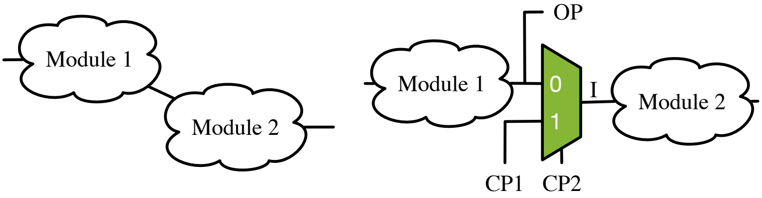


More Considerations

- Existing attempts still rely on regular format of data, like images;
- Netlists and layouts are naturally represented as graphs;
- Few DL solutions for graph-based problems in EDA.

Test Points Insertion

- Fig. (a): Original circuit with bad testability. Module 1 is unobservable. Module 2 is uncontrollable;
- Fig. (b): Insert test points to the circuit;
- (CP1, CP2) = (0, 1) $\rightarrow$ line I = 0; (CP1, CP2) = (1, 1) $\rightarrow$ line I = 1;
- CP2 = 0 $\rightarrow$ normal operation mode.



(a) (b)

Problem Overview

Problem

Given a netlist, identify where to insert test points, such that:

- Maximize fault coverage;
- Minimize the number of test points and test patterns.

* (Focus on observation points insertion in this work.)

Problem Overview

Problem

Given a netlist, identify where to insert test points, such that:

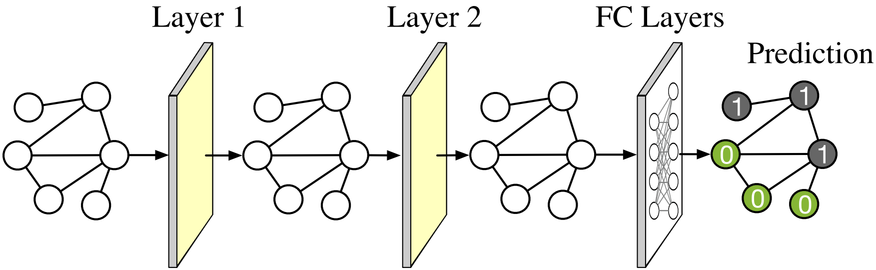
- Maximize fault coverage;
- Minimize the number of test points and test patterns.

* (Focus on observation points insertion in this work.)

- It is a binary classification problem from the perspective of DL model;
- A classifier can be trained from the historical data.
- Need to handle graph-structured data.
- Strong scalability is required for realistic designs.

Node Classification

- Represent a netlist as a directed graph. Each node represents a gate.
- Initial node attributes: SCOAP values [Goldstein et. al, DAC'1980].
- Graph convolutional networks: compute node embeddings first, then perform classification.



Node Classification

Node embedding: two-step operation:

- Neighborhood feature aggregation: weighted sum of the neighborhood features.

$$\boldsymbol{g}_d^{(v)} = \boldsymbol{e}_{d-1}^{(v)} + w_{pr} \times \sum_{u \in \mathrm{PR}(v)} \boldsymbol{e}_{d-1}^{(u)} + w_{su} \times \sum_{u \in \mathrm{SU}(v)} \boldsymbol{e}_{d-1}^{(u)}$$

- Projection: a non-linear transformation to higher dimension.

$$\boldsymbol{e}_d = \sigma(\boldsymbol{g}_d \cdot \boldsymbol{W}_d)$$

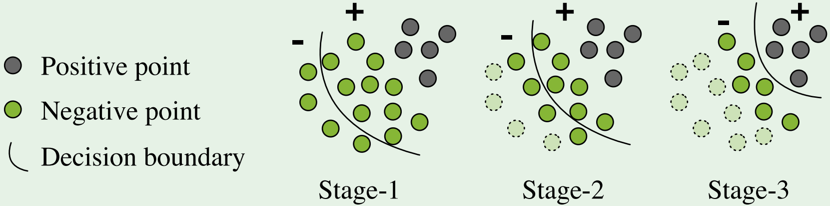
Classification: A series of fully-connected layers.

Imbalance Issue

- High imbalance ratio: much more negative nodes than positive nodes in a design;
- Poor performance: bias towards majority class;

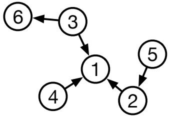
Solution: multi-stage classification.

- Impose a large weight on positive points.
- Only filter out negative points with high confidence in each stage.



Efficient Inference

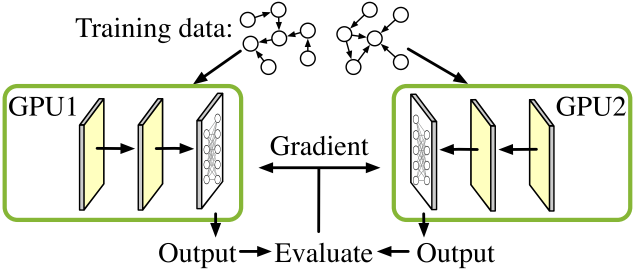
- Neighborhood overlap leads to duplicated computation $\rightarrow$ poor scalability.
- Transform weighted summation to matrix multiplication.
- Potential issue: adjacency matrix is too large.
- Fact: adjacency matrix is highly sparse! It can be stored using compressed format.



$$\boldsymbol{G}_d = \boldsymbol{A} \cdot \boldsymbol{E}_{d-1} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & w_1 & w_1 & w_1 & 0 & 0 \\ w_2 & 1 & 0 & 0 & w_1 & 0 \\ w_2 & 0 & 1 & 0 & 0 & w_2 \\ w_2 & 0 & 0 & 1 & 0 & 0 \\ 0 & w_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & w_1 & 0 & 0 & 1 \end{bmatrix} \end{array} \times \begin{bmatrix} \boldsymbol{e}_{d-1}^{(1)} \\ \boldsymbol{e}_{d-1}^{(2)} \\ \boldsymbol{e}_{d-1}^{(3)} \\ \boldsymbol{e}_{d-1}^{(4)} \\ \boldsymbol{e}_{d-1}^{(5)} \\ \boldsymbol{e}_{d-1}^{(6)} \end{bmatrix}$$

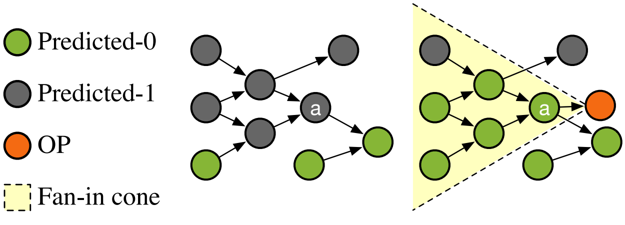
Efficient Training

- Adjacency matrix cannot be split as conventional way.
- A variant of conventional data-parallel scheme.
  - Each GPU process one graph instead of one "chunk";
  - Gather all to calculate the gradient.



Test Point Insertion Flow

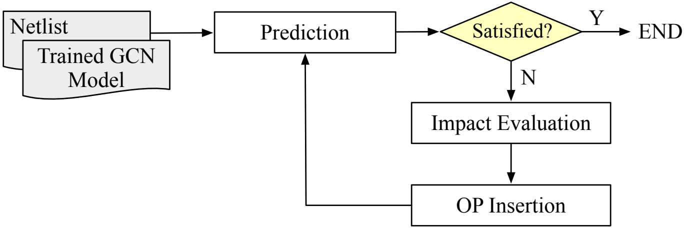
- Not every difficult-to-observe node has the same impact for improving the observability;
- Select the observation point locations with largest impact to minimize the total count.
- Impact: The positive prediction reduction in a local neighborhood after inserting an observation point.
- E.g., the impact of node a in the figure is 4.



(c) (d)

Test Point Insertion Flow

- Iterative prediction and OPs insertion.
- Once an OP is inserted, the netlist would be modified and node attributes would be re-calculated.
- Sparse representation enables incremental update on adjacency matrix.
- Exit condition: no positive predictions left.



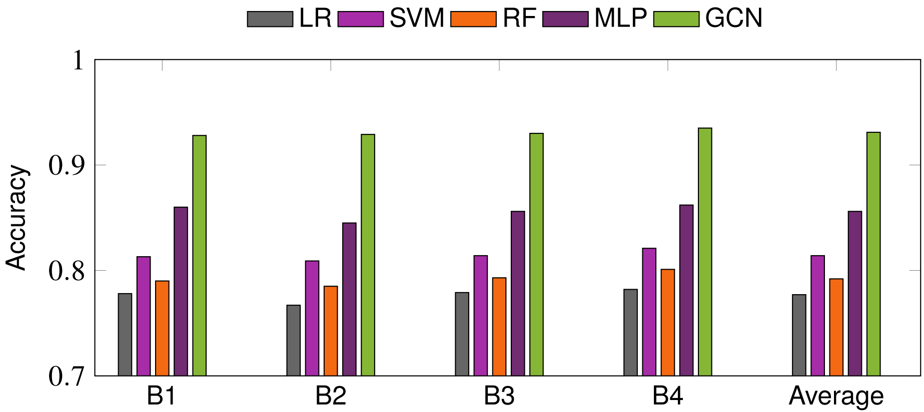
Benchmarks

- Industrial designs under 12nm technology node.
- Each graph contains > 1M nodes and > 2M edges.

Design	#Nodes	#Edges	#POS	#NEG
B1	1384264	2102622	8894	1375370
B2	1456453	2182639	9755	1446698
B3	1416382	2137364	9043	1407338
B4	1397586	2124516	8978	1388608

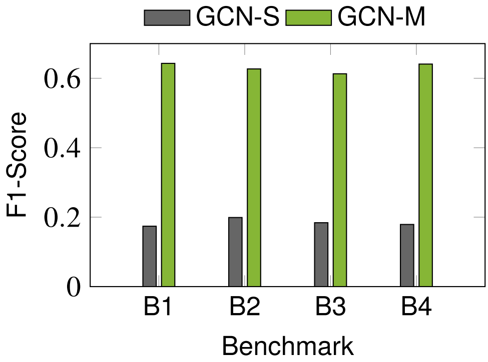
Classification Results Comparison

- Baselines: classical learning models with feature engineering in industry;
- GCN outperforms other classical learning algorithms.

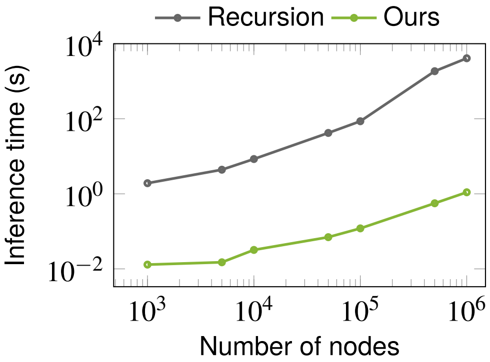


Multi-stage GCN Results

- Single-stage GCN vs. Multi-stage GCN ;



- Scalability: $10^3\times$ speedup on inference time for a design with > 1 million cells.



Testability Results Comparison

- Without loss on fault coverage, 11% reduction on test points inserted and 6% reduction on test pattern count are achieved.

Design	Industrial Tool			GCN-Flow		
	#OPs	#PAs	Coverage	#OPs	#PAs	Coverage
B1	6063	1991	99.31%	5801	1687	99.31%
B2	6513	2009	99.39%	5736	2215	99.38%
B3	6063	2026	99.29%	4585	1845	99.29%
B4	6063	2083	99.30%	5896	1854	99.31%
Average	6176	2027	**99.32%**	**5505**	**1900**	**99.32%**
Ratio	1.00	1.00	**1.00**	**0.89**	**0.94**	**1.00**

Thank You