iRw: An Intelligent Rewriting

Haisheng Zheng ♠ Haoyuan Wu ♡ Zhuolun He ♡ Yuzhe Ma ♣ Bei Yu ♡

♠ Shanghai AI Laboratory ♡ CUHK ♣ HKUST (GZ)

Highlights

- We propose **iRw**, which effectively minimizes node counts in AIGs.
- We introduce a subgraph extraction algorithm that efficiently extracts subcircuits with optimization potential.
- We propose a method of machine learning-guided optimization process, ensuring efficient optimization with minimal runtime overhead.
- Our method outperforms SOTA methods both in quality and runtime.

Related Works

Subcircuit Extraction.
- Typically tailored to the needs of rewriting algorithms.

- **Single-Output Subcircuits.**
  (e.g., *Cut Enumeration* [1], *Maximum Fanout-Free Cones* [2])
  - Simplifies the optimization process.
- **Multi-Output Subcircuits.**
  (e.g., *Reconvergence-driven Window* [3], *Maximum Fanout-Free Window* [4])
  - Enables exploration of shared logic between outputs, enhancing efficiency.

Subcircuit Optimization.
- The trade-off between optimization quality and computational overhead.

Method	Advantages & Limitations
Pre-Computed Library [1, 5]	✓ Fast; May provides optimal optimization. ✗ Limited to small subcircuits.
Heuristic Resynthesis [2, 3]	✓ Fast; Can handle larger subcircuits. ✗ May not deliver optimal optimization.
Exact Resynthesis [6, 7]	✓ Provides the best solutions; Handles large subcircuits. ✗ Requires substantial computational resources.

ML-Guided Optimization [8, 9].
- Optimization strategies are adapted to the input circuit to enhance efficiency.

References

[1] Alan Mishchenko et al. DAG-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis. In *Proc. DAC*, 2006.

[2] Alan Mishchenko et al. Scalable Logic Synthesis using a Simple Circuit Structure. In *Proc. IWLS*, 2006.

[3] Heinz Riener et al. Boolean Rewriting Strikes Back: Reconvergence-Driven Windowing Meets Resynthesis. In *Proc. ASPDAC*, 2022.

[4] Xuliang Zhu et al. A Database Dependent Framework for K-Input Maximum Fanout-Free Window Rewriting. In *Proc. DAC*, 2023.

[5] Wenlong Yang et al. Lazy Man's Logic Synthesis. In *Proc. ICCAD*, 2012.

[6] Heinz Riener et al. On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis. In *Proc. DATE*, 2019.

[7] Heinz Riener et al. Exact DAG-Aware Rewriting. In *Proc. DATE*, 2020.

[8] Walter Lau Neto et al. LSOracle: a Logic Synthesis Framework Driven by Artificial Intelligence. In *Proc. ICCAD*, 2019.

[9] Xing Li et al. EffiSyn: Efficient Logic Synthesis with Dynamic Scoring and Pruning. In *Proc. ICCAD*, 2023.

[10] Will Hamilton et al. Inductive Representation Learning on Large Graphs. In *Proc. NIPS*, 2017.

Methodology

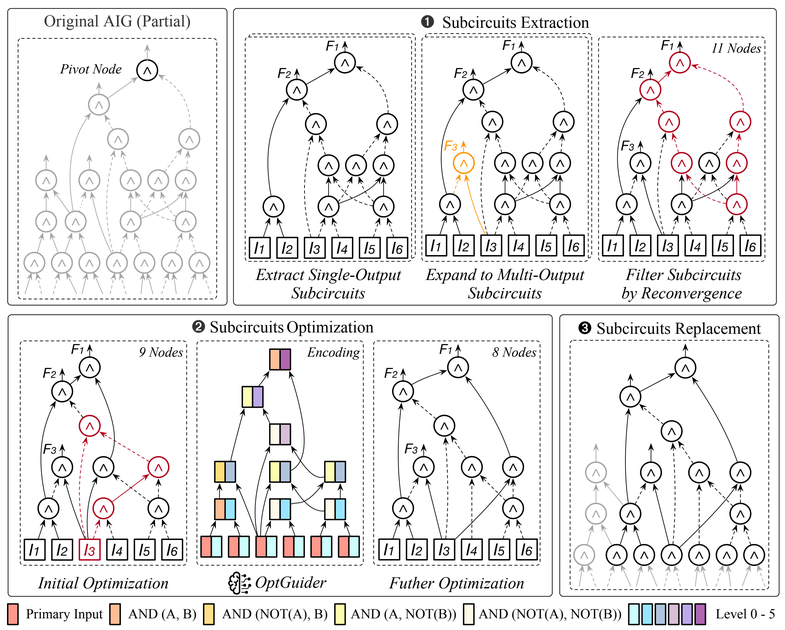


Figure 1. Overview of the iRw.

iRw iteratively processes each node of a given AIG G, excluding primary inputs, as the pivot node P, with a subcircuit input size limit K, following these stages:

❶ **Subcircuit Extraction.**

- **Single-Output Subcircuits:** $S = TFI(G, K, P)$
  Extracts the Transitive Fan-In (TFI) of the pivot node P, ensuring that the number of inputs in S does not exceed the limit $|\text{Inputs}(S)| \leq K$.
- **Expansion to Multi-Output Subcircuits:** $S' = \{S_i \cup TFO(S_i) \mid S_i \in S\}$
  For each single-output subcircuit S_i, explores its Transitive Fan-Out (TFO) to extend it into a multi-output subcircuit.
- **Filter by Reconvergence:** $S_{\text{filtered}} = \{S' \mid S' \text{ contains reconvergent paths}\}$
  Retains subcircuits that contain reconvergent paths, as these offer optimization potential for observability-based optimizations (e.g., resubstitution).

❷ **Subcircuit Optimization.**

- **Initial Optimization:** *Balance*, *Resubstitution*.
- **Further Optimization:** *Rewrite*, *Refactor*.
- **OptGuider:** Identifies subcircuits that can be optimized by the futher optimization stage, minimizing computational overhead.

Node Feature Encoding.
- Edge information is integrated into *AND* gate representations.
- The logical level of node v in the AIG is encoded as:

$$PE_{(v,2i)} = \sin\left(\frac{\text{level}(v)}{10000^{2i/d}}\right), \quad PE_{(v,2i+1)} = \cos\left(\frac{\text{level}(v)}{10000^{2i/d}}\right). \tag{1}$$

Lightweight GNN.
- GraphSAGE [10] is used as the GNN model due to its computational efficiency.

Cost-Sensitive Learning.
- Helps the classifier prioritize subcircuits that impact performance:
  - Misclassifying non-optimizable subcircuits increases overhead without improving optimization.
  - Overlooking optimizable subcircuits impedes the optimization process.
  - Adjusts the weights of classes (*Optimizable* ↑, *Non-Optimizable* ↓) in the Binary Cross Entropy loss function.

Experimental Results

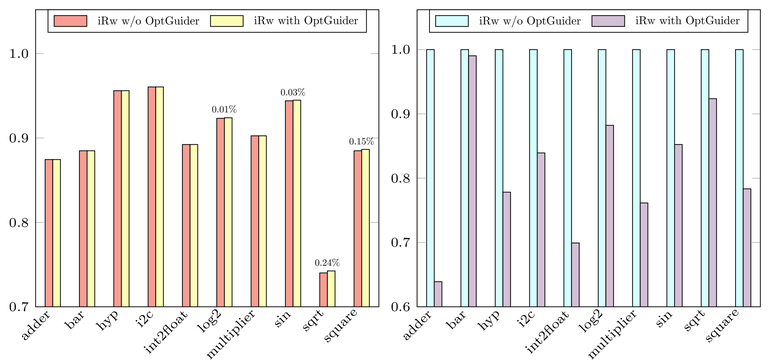
- **Performance Comparison Between iRw and SOTA Rewriting.**

Benchmark		Window Rewriting [3]		iRw			
		Until Convergence		First Iteration		Until Convergence	
Name	# Nodes	# Nodes	Time	# Nodes	Time	# Nodes	Time
adder	1,020	892	0.04	892	0.13	892	0.28
bar	3,336	2,952	3.76	2,952	0.62	2,952	1.17
hyp	214,335	204,926	20.28	204,926	19.90	204,926	40.38
i2c	1,342	1,291	0.10	1,289	0.14	1,273	0.58
int2float	260	239	0.02	232	0.09	226	0.36
log2	32,060	29,700	6.59	29,603	5.19	29,556	28.55
multiplier	27,062	24,566	3.89	24,426	3.93	24,426	8.80
sin	5,416	5,132	1.85	5,115	1.05	5,095	7.73
sqrt	24,618	18,325	2.95	18,279	2.42	18,236	16.17
square	18,484	16,606	2.72	16,386	2.04	16,316	6.47
Average	-	9.79%	-	10.32%	-	10.78%	-
Total	-	-	42.20		35.51	-	110.49

- **Impact of Multi-Output Subcircuit Extraction.**

Method	# Initial Nodes	# Optimized Nodes	Average Improvement
b; rs -K 6; rw; rf;	327,933	314,793	4.01%
iRw		304,100	10.32%

- **Performance Comparison Between iRw with and without OptGuider.**



(a) Node Reduction comparison. (b) Runtime Comparison.

- **Settings:**
  - *Experiments were run on a 2.6 GHz AMD EPYC 7H12 CPU.*
  - *Input size $K = 6$ was set for all configurable algorithms to ensure fair comparison.*

Conclusion

- iRw outperforms Window Rewriting [3] both in node reduction and runtime.
- iRw achieves better node reduction, demonstrating the effectiveness of multi-output subcircuit extraction.
- iRw guided by OptGuider achieves significant improvements in runtime, with only a slight impact on node reduction.