An Island-Style Multi-Objective Evolutionary Framework for Synthesis of Memristor-Aided Logic

Umar Afzaal, **Seunggyu Lee**, and Youngsoo Shin

School of Electrical Engineering, KAIST, Korea

Outline

- Introduction
- Motivation
- Proposed method
- Experimental results
- Summary

Logic Operation

- Conventional method is executed within a processing unit (CPU, GPU)
  - Processed data is transferred to the memory for storage
  - Data transfer degrades both performance and energy
- Processing-in-memory (PIM) enables logic operations to be performed within the memory
  - Data transfer is significantly reduced
  - Repetitive operations (e.g., machine learning model) on large dataset can be accelerated

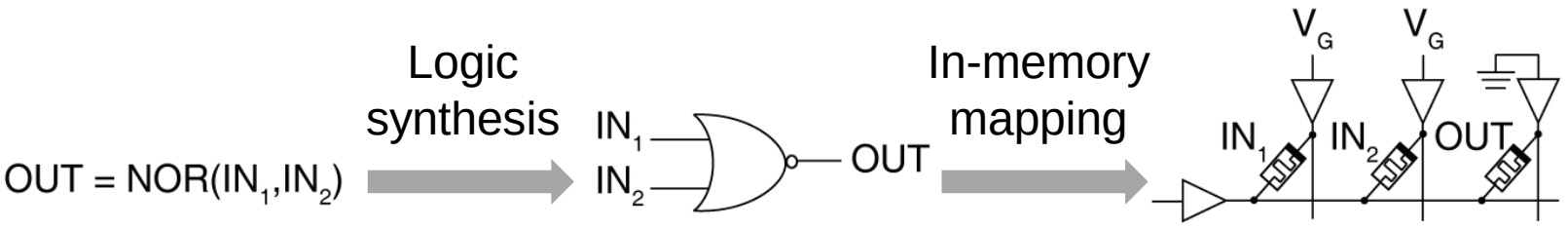
Memristor for PIM Logic Operation

- Non-volatile resistive memory (memristor) is leveraged for PIM logic operation

- Memristor offers low (or high) resistance state for logic-1 (or -0)
  - Resistance state is determined by applied voltage (*V*)
    - $V \leq V_{on}$: switching to low resistance state (LRS)
    - $V_{on} < V < V_{off}$: remaining current resistance state (CRS)
    - $V \geq V_{off}$: switching to high resistance state (HRS)

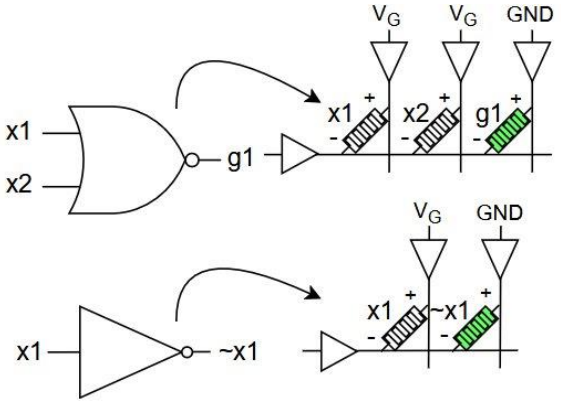
PIM Logic Operation

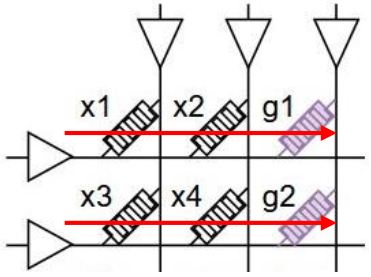
- After logic synthesis, in-memory mapping is performed [TCAS-I]
  - Logic function is synthesized into **2-input NOR and INV netlist**
    - #Nodes of AND-INV graph (AIG) is reduced
  - **Each gate** of netlist is **mapped** into memristive crossbar array
    - Gate input and output are stored in distinct memristors



In-Memory Mapping

- All memristors for gates should be aligned either in a row or a column
  - Each gate is operated in one clock cycle
- Multiple gates can be computed in the same clock cycle → parallel operation
  - All input memristors are aligned, along with their corresponding output memristors



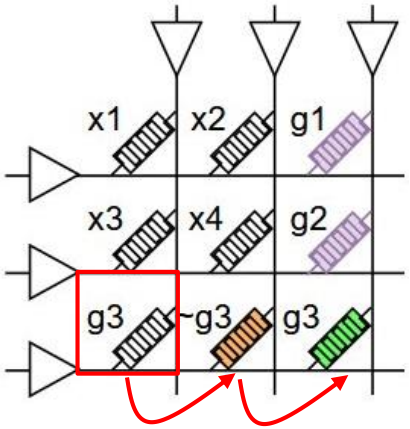


Parallel operation of $g1$ and $g2$

In-Memory Mapping

- Copying a value from one memristor to another can be executed → copy operation
  - Two INV operations are performed in two clock cycles

- #Copy operations depends on mapping methods



$g3$ copy from (2,0) to (2,2)

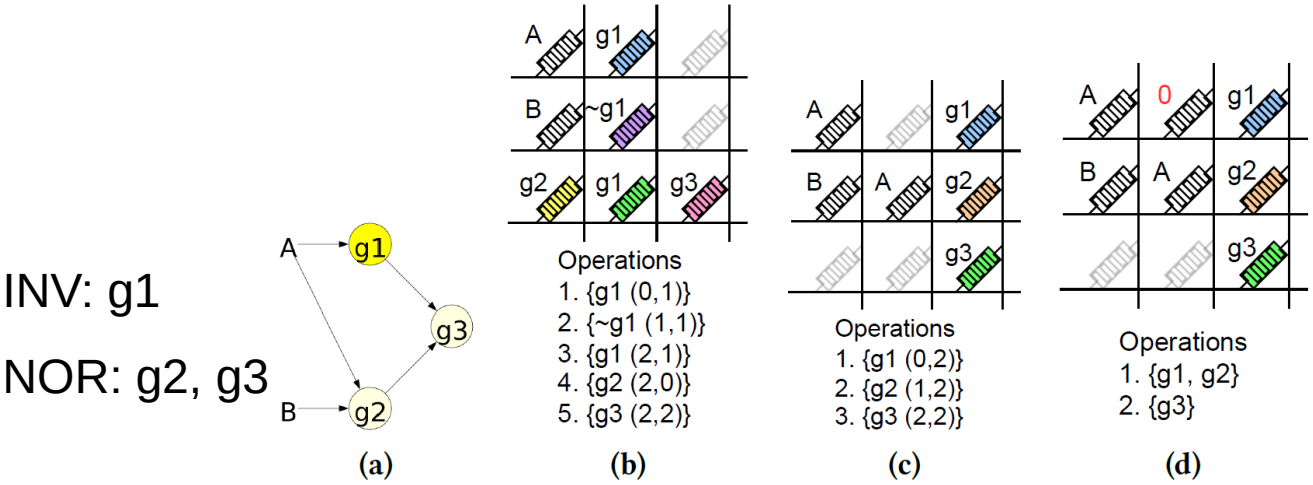
In-Memory Mapping

- Objective is to minimize #clock cycles and #memristors

- Key decision variables include:
  - Gate location
  - Gate alignment
  - #Memristors on the crossbar
  - Gates calculated in parallel

Mapping Example

- For a NIG illustrated in Fig.(a), there are 3 example mappings
  - Fig. (b): 5 cycles and 7 memristors
  - Fig. (c): 3 cycles and 6 memristors
  - **Fig. (d): 2 cycles and 6 memristors**
    - INV is treated as 2-input NOR with one input tied to logic-0



Previous Works

- **Integer linear programming (ILP) [ICCAD17]**
  - Optimization problem is formulated using coordinates of crossbar
- **Simple heuristic [TVLSI18]**
  - INV gates and NOR gates are mapped eastward and southward, respectively
- **Staircase-structure heuristic [ASPDAC19]**
  - Gates at each level are alternatively mapped in rows and columns
- **Multiple ILP for a partitioned netlist [ICCAD19]**
  - ILP formulation is individually solved for each partition

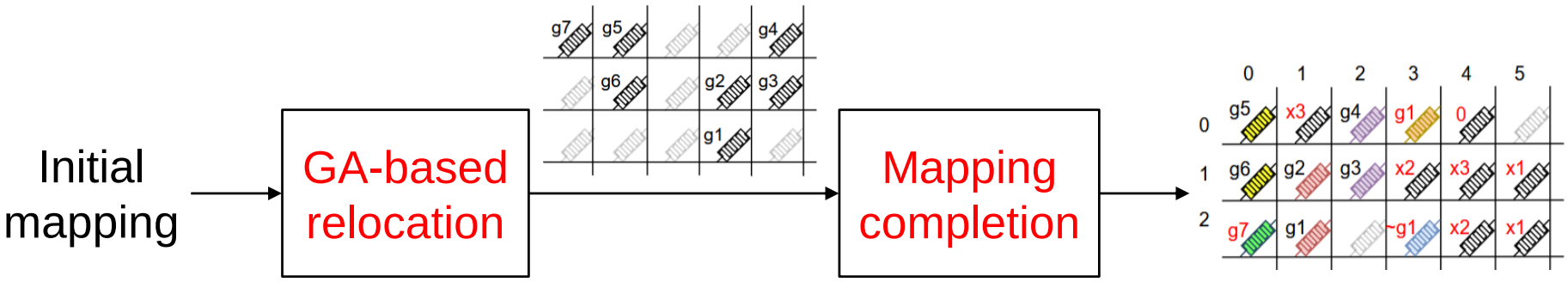
Motivation

- ILP solver is very time-consuming
- Heuristic methods target only a single objective

- **Proposed: multi-objective optimization using genetic algorithm (GA)**
  - Gate locations are modified by GA to satisfy multi-objective

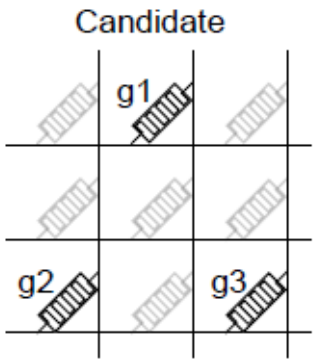
Overall Flow

- Given: initial mapping from any heuristic methods
  - GA relocates the gate locations using mutation and crossover
  - Mapping rule violation is corrected by mapping completion algorithm

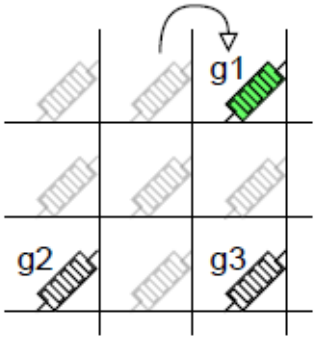


Mutation

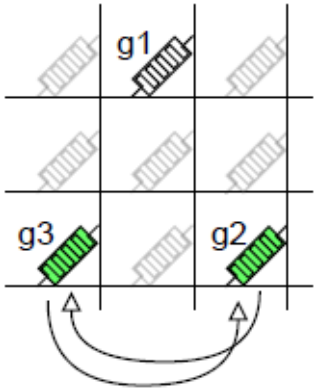
- **Gates** are **moved to the same row or column**
  - Random #gates is selected from a Levy probability distribution
  - If the new location is empty, gate is moved to there
  - Otherwise, a swap occurs between two gate outputs



Initial mapping



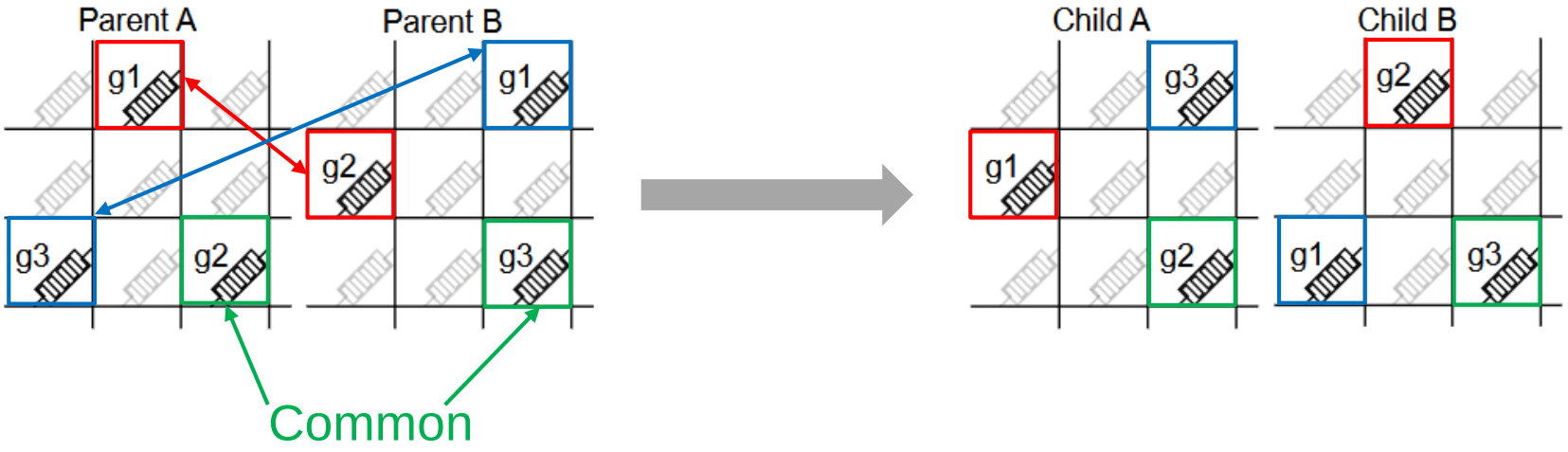
Move $g1$



Swap $g2$ and $g3$

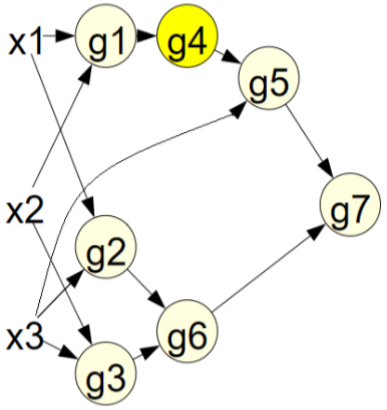
Crossover

- **Gate locations** from two different mappings are **exchanged**
  - Step 1) In each crossbar, list the gate locations
  - Step 2) Exclude common locations (e.g., g_2^A, g_3^B)
  - Step 3) Swap remaining gate locations in reverse order

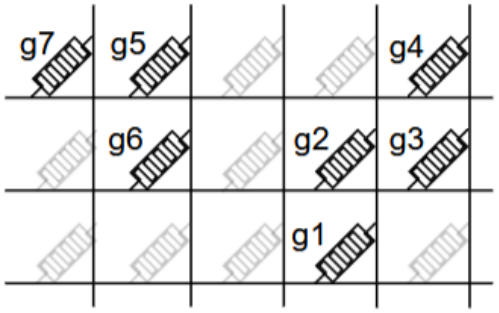


Mapping Completion Algorithm

- After performing genetic algorithm, **gate locations are corrected based on mapping rules**
  - Step 1) Find aligned gates in each row or column
    - Row-oriented: $\{g_4, g_5, g_7\}\{g_2, g_3, g_6\}\{g_1\}$
    - Column-oriented: $\{g_7\}\{g_5, g_6\}\{g_1, g_2\}\{g_3, g_4\}$
  - Step 2) Remove subsets of other groups: $\{g_1\}, \{g_7\}$
    - Set = $\{g_4, g_5, g_7\}\{g_2, g_3, g_6\}\{g_5, g_6\}\{g_1, g_2\}\{g_3, g_4\}$



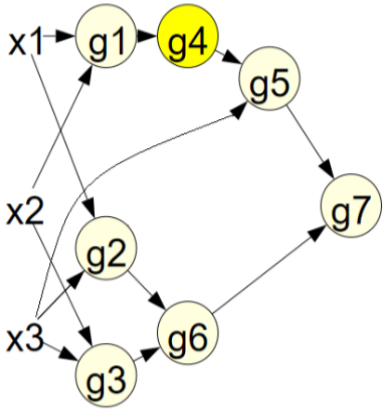
Example NIG

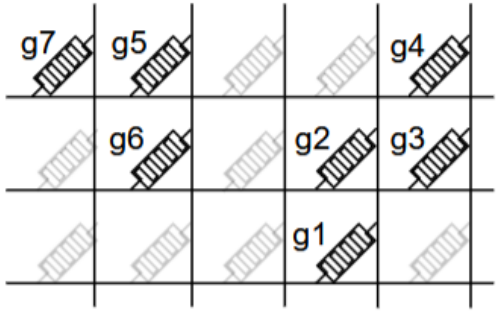


Relocated mapping from GA

Mapping Completion Algorithm

- Step 3) Resolve data dependency
  - If two gates are sequentially connected in the same path, they cannot simultaneously operate
    - $\{g_4, g_5, g_7\} \rightarrow \{g_4\}\{g_5\}\{g_7\}$
    - $\{g_2, g_3, g_6\} \rightarrow \{g_2, g_3\}\{g_6\}$
- Step 4) Remove subsets of other groups: $\{g_4\}\{g_5\}\{g_6\}$
  - Set = $\{g_7\}\{g_2, g_3\}\{g_5, g_6\}\{g_1, g_2\}\{g_3, g_4\}$

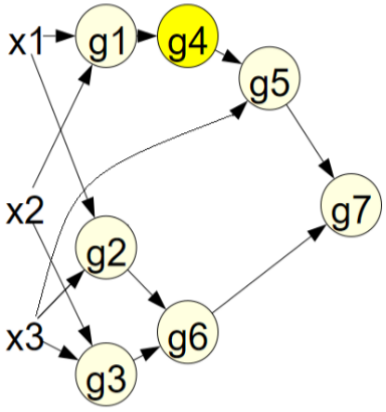


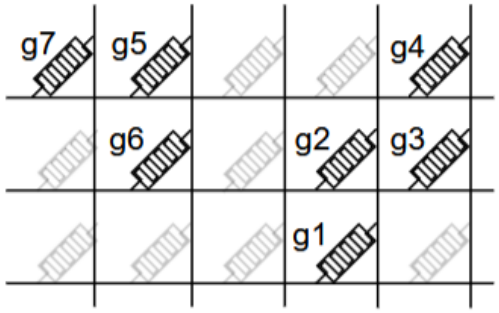


Mapping Completion Algorithm

- Step 5) Assign a mapping direction *D* to each gate based on fanin locations

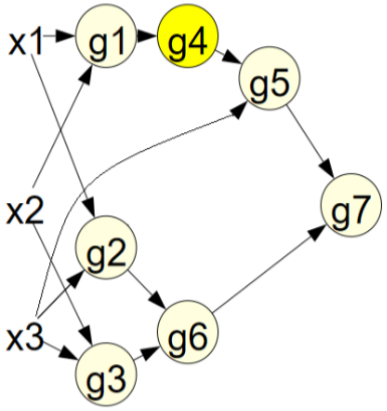
$$D(g) = \begin{cases} \text{EAST} & \text{if } fanin(g) : \text{EAST} \\ \text{WEST} & \text{if } fanin(g) : \text{WEST} \\ \text{NORTH} & \text{if } fanin(g) : \text{NORTH} \\ \text{SOUTH} & \text{if } fanin(g) : \text{SOUTH} \vee \\ & (fanin(g)[0] \vee fanin(g)[1]) : \text{NORTH} \\ \text{EAST} & \text{otherwise} \end{cases}$$





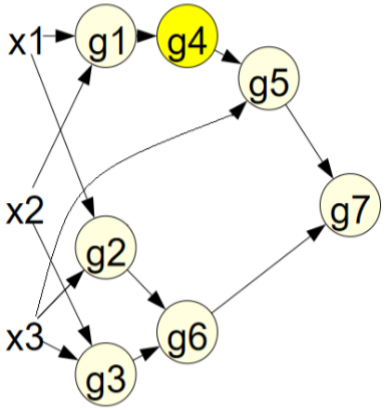
Mapping Completion Algorithm

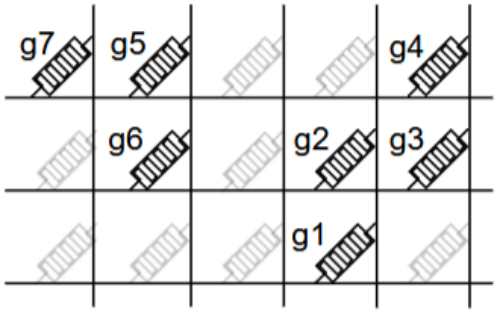
- Step 6) Ungroup gates with different directions
  - In this example, no change
- Step 7) Remove subsets of other groups
  - Set = $\{g_7\}\,\{g_2, g_3\}\,\{g_5, g_6\}\,\{g_1, g_2\}\,\{g_3, g_4\}$



Mapping Completion Algorithm

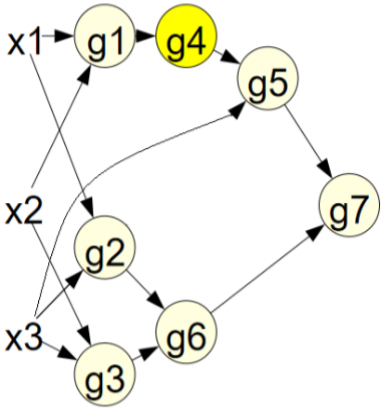
- Step 8) If a gate belongs to multiple groups, move the gate to the largest group
  - If multiple groups have the same size, move the gate in a way that preserves other groups
    - $g_2 \in \{\boldsymbol{g_2}, \boldsymbol{g_3}\}\{g_1, g_2\},\ g_3 \in \{\boldsymbol{g_2}, \boldsymbol{g_3}\}\{g_3, g_4\}$
  - Set = $\{g_7\}\{g_5, g_6\}\{g_1, g_2\}\{g_3, g_4\}$

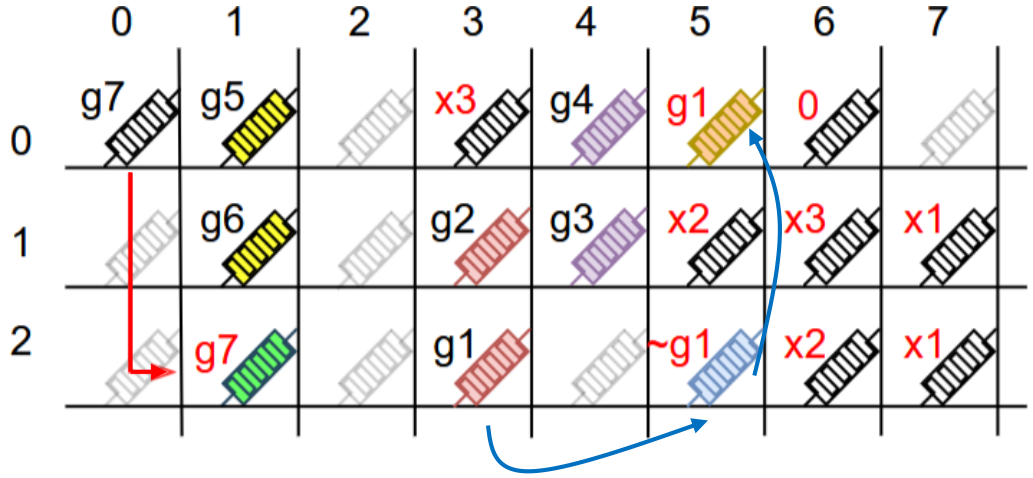




Mapping Completion Algorithm

- Step 9) Relocate gate outputs which are single member in each group
  - $\{g_7\}$ is aligned with fanins $\{g_5, g_6\}$
- Step 10) Copy gate outputs which are not aligned with fanins
  - g_1 is copied from (2,3) to (0,5) for fanin of g_4

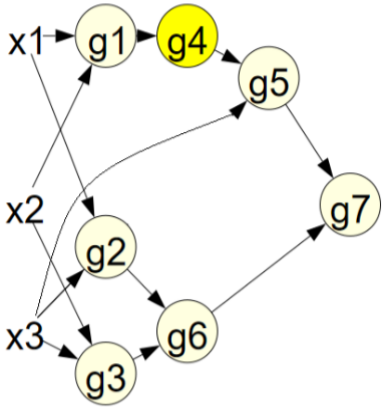


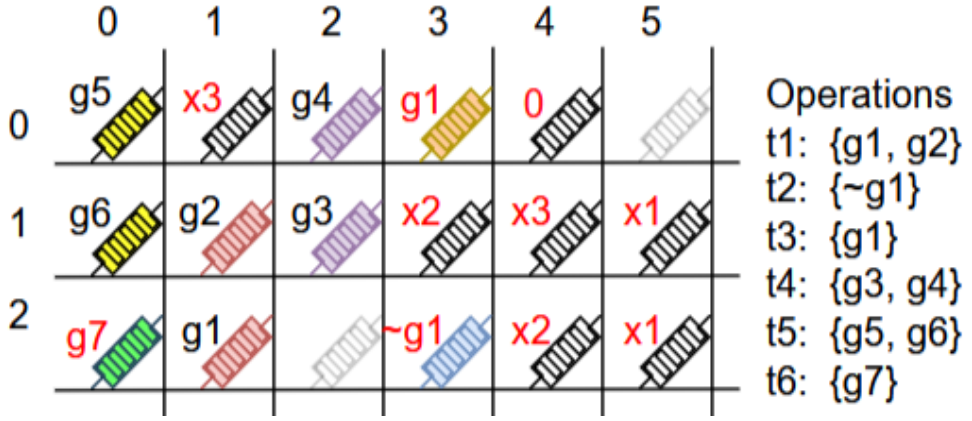


Relocated mapping result

Mapping Completion Algorithm

- Step 11) Empty rows or columns are removed, and in the corrected mapping result,
  - #Memristors is counted → 13
  - #Cycles = #groups + 2x#copies → 4 + 2x1





Final mapping result

Experimental Setup

- Test circuits: 8 benchmarks from IWLS-93
  - Format "#PI/#PO(#NIG nodes from ABC)"

5xp1 :	**7/10(131),**	**misex1 :**	**8/7(81),**
b12 :	**15/9(94),**	**misex2 :**	**25/18(159),**
clip :	**9/5(133),**	**rd73 :**	**7/3(157),**
cordic :	**23/2(93),**	**inc :**	**7/9(149)**

- Logic synthesis is conducted using Berkeley-ABC tool
- Hyperparameter values for GA:

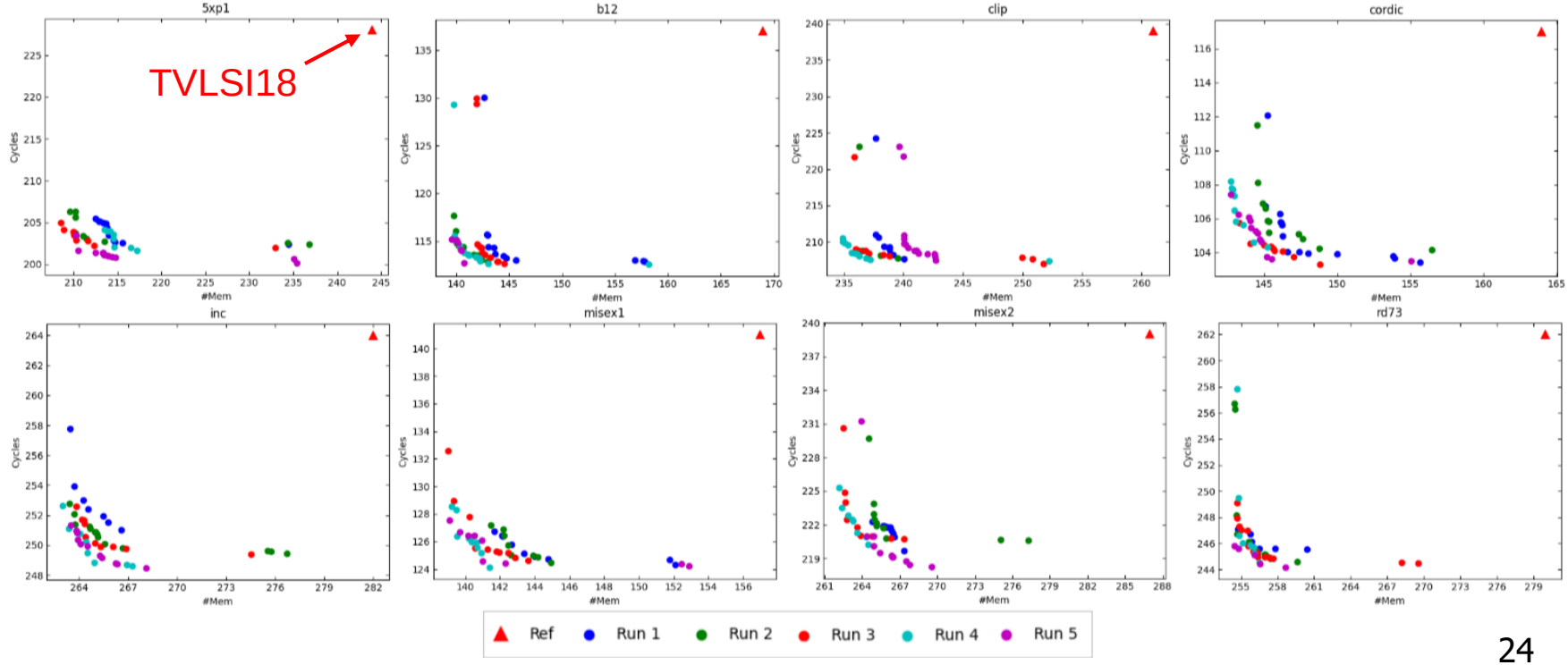
n_process :	32,	*cxpb* :	0.7,
n_pops :	50,	*mutpb* :	0.3,
n_ind :	50,	*mig_rate* :	0.1,
mig_interval :	10		

Experimental Setup

- Multi-objective: **minimize #memristors and #cycles**
- 5 independent runs are performed per benchmark
- The proposed method is compared with the simulated-annealing (SA) approach [TVLSI18]

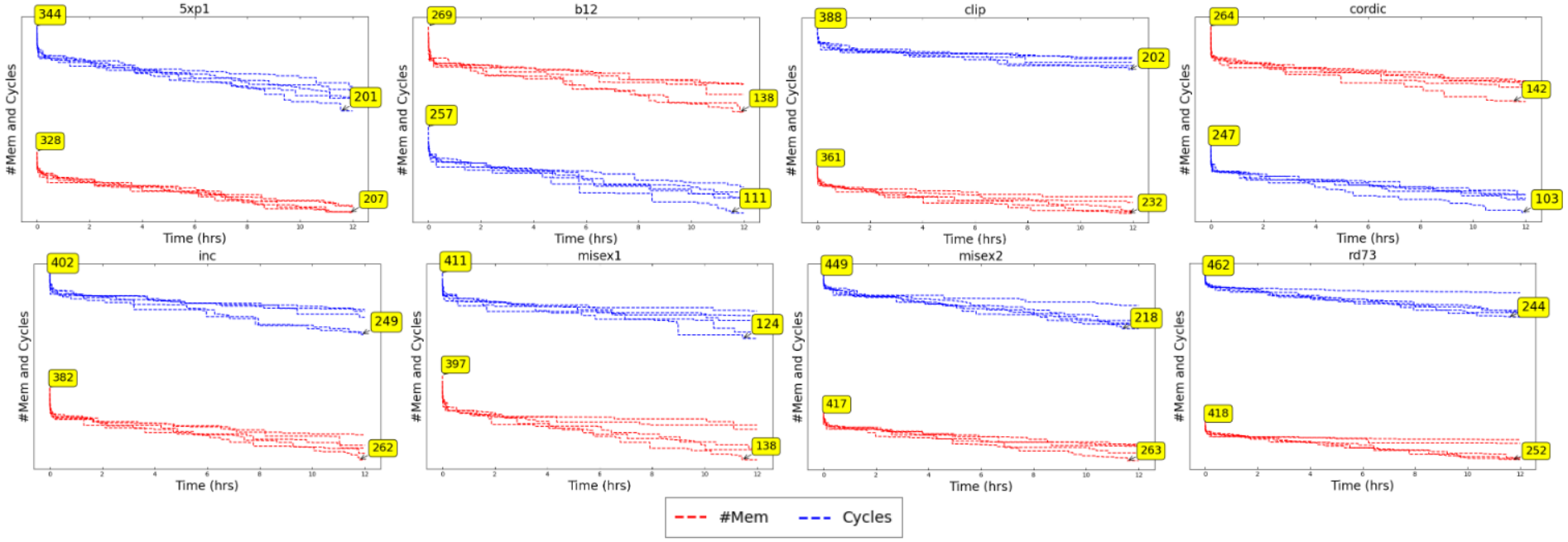
Experimental Results

- Better results are shown than SA method [TVLSI18]
  - For each run, multiple mapping results are generated
  - #Memristors and #cycles are reduced by more than 10%, respectively



Experimental Results

- Our method demonstrates consistent improvement across all benchmarks
  - The objective curve patterns are similar over runtime
    - A significant reduction occurs within 1 hour



Summary

- Multi-objective optimization approach is proposed
  - **Gate relocation** is conducted using **genetic algorithm**
  - **Mapping completion algorithm** corrects the relocated gates based **mapping rules**

- Experimental results exhibit **consistent reductions** of more than **10% in #memristors and #cycles** for all benchmarks