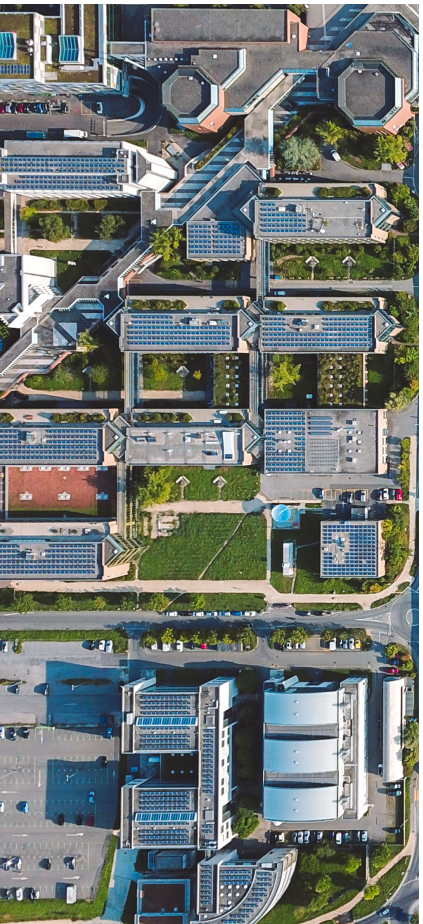
EPFL



A Versatile Mapping Approach for Technology Mapping and Graph Optimization

Alessandro Tempia Calvino[1] Heinz Riener[1] Shubham Rai[2]

Akash Kumar[2] Giovanni De Micheli[1]

[1] Integrated Systems Laboratory, EPFL, Lausanne, Switzerland

[2] Chair for Processor Design, TU Dresden, Dresden, Germany

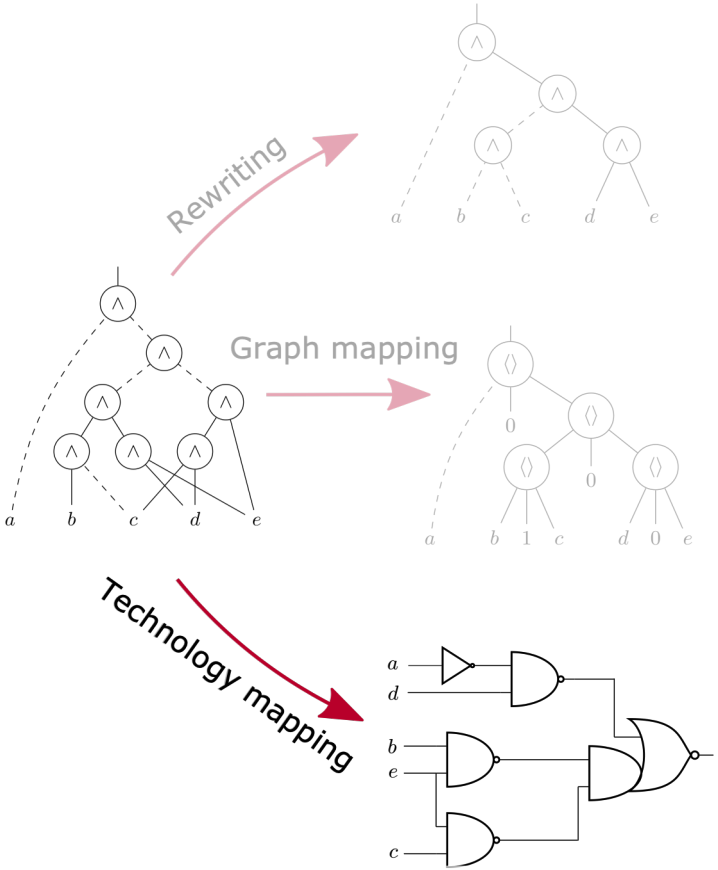
alessandro.tempiacalvino@epfl.ch

19th Jan 2022

Motivations

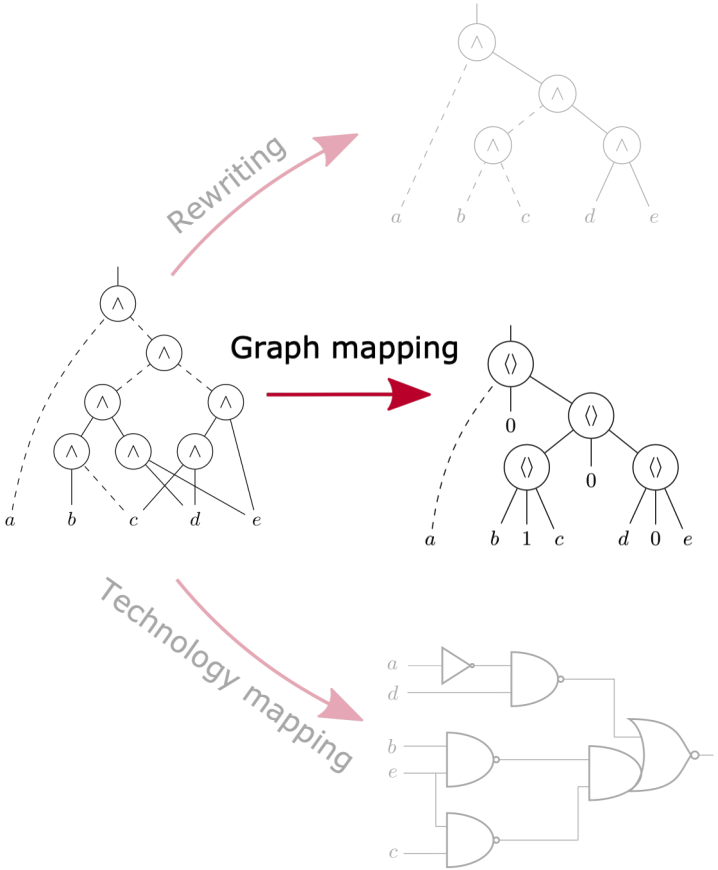
- **Multiple representations** are available in logic synthesis:
  - AIGs, MIGs, XMGs, etc.
  - Each one has its own advantages/applications
- Need of a way to not trivially map between representations
- Develop a generalized algorithm for **mapping** and **rewriting**
  - Previous methods have limitations
- Logic optimization tool
  - Especially for representations which **lack of ad-hoc approaches**

Goals



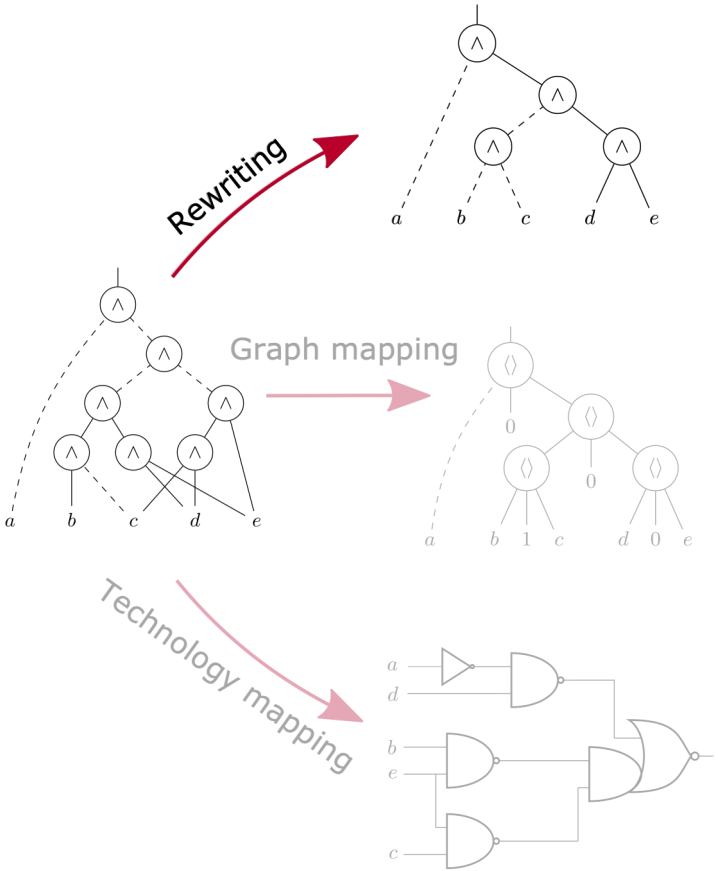
- Versatile mapping:
  - Map to a cell library

Goals



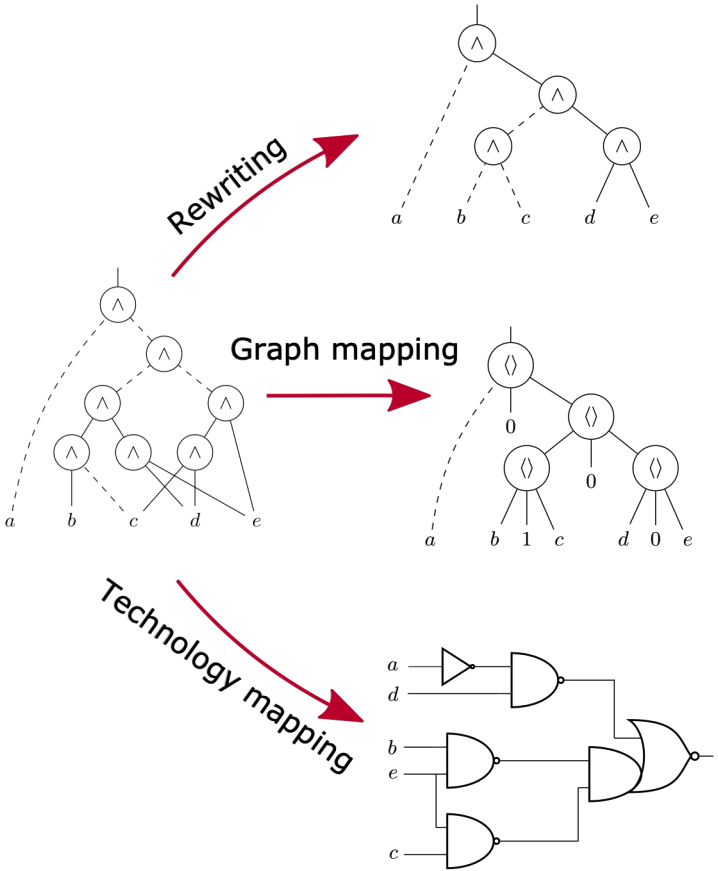
- Versatile mapping:
  - Map to a cell library
  - Map **from** and **to** a generic graph representation (e.g., AIG, XAG, MIG)

Goals



- Versatile mapping:
  - Map to a cell library
  - Map **from** and **to** a generic graph representation (e.g., AIG, XAG, MIG)
  - Logic optimization method

Goals



- Versatile mapping:
  - Map to a cell library
  - Map **from** and **to** a generic graph representation (e.g., AIG, XAG, MIG)
  - Logic optimization method
- **Independent** of the underlying graph representation

Applications

- Technology mapping
- Obtain graph abstractions for different applications:
  - **MIG**:
    - Adiabatic Quantum Flux Parametron (AQFP)
    - Quantum-dot Cellular Automata (QCA)
  - **XMG**:
    - Reconfigurable-FET nanotechnologies
  - **XAG**:
    - Cryptography and security applications
    - Quantum compilation
- Logic Optimization

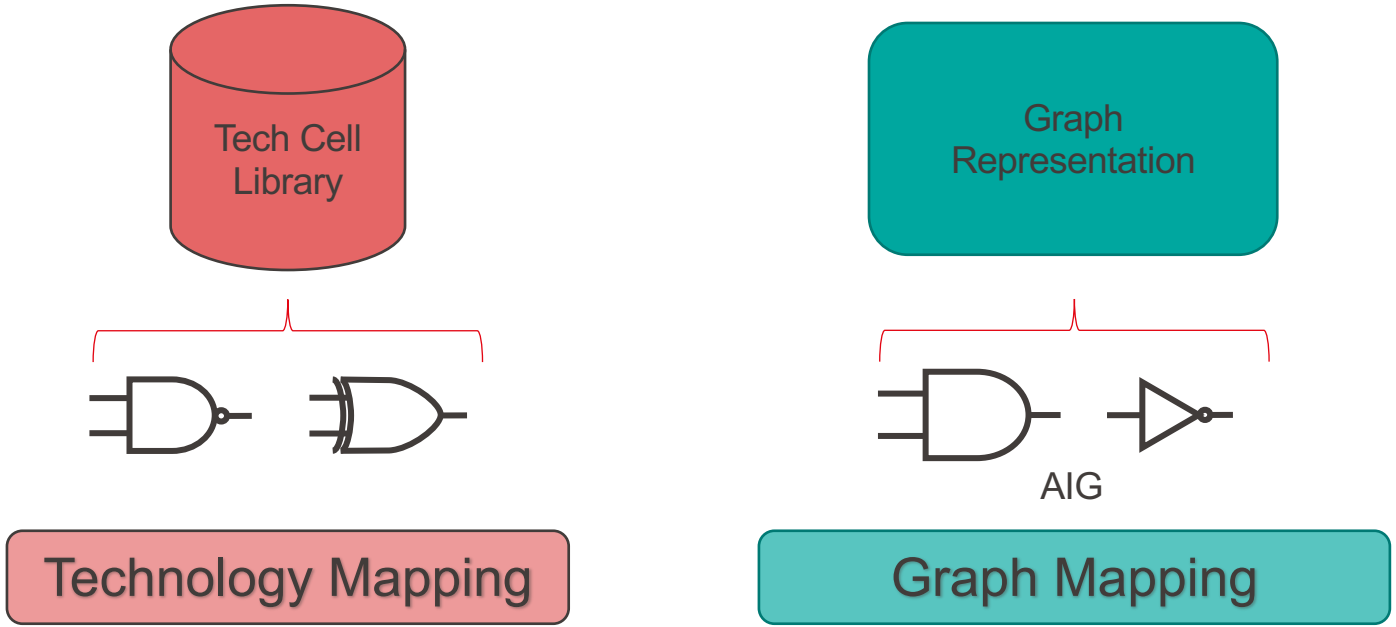
Outline

- **Background and Related Work**
  - Graph mapping
  - Logic restructuring
- **Versatile Mapper**
- **Experiments**
- **Conclusions**

Background and Related Work

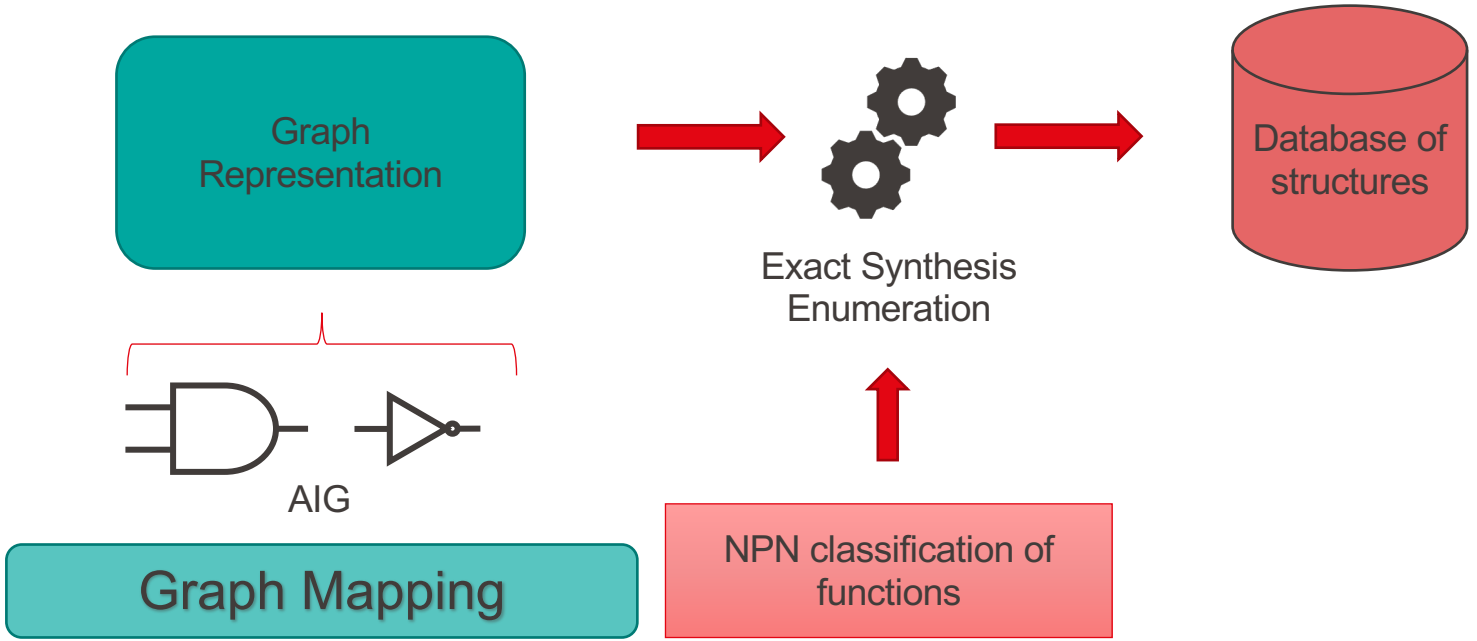
Background

- Expressing a Boolean circuit with a set of primitives



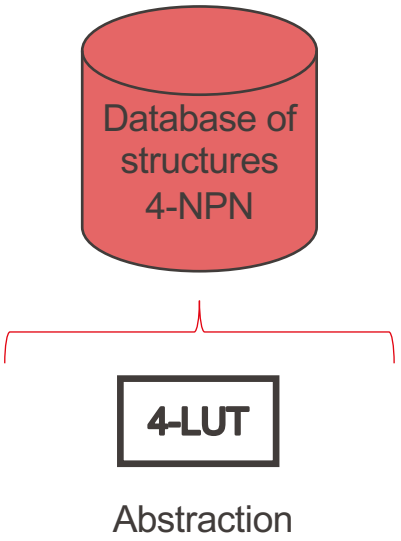
Background: Graph Mapping

- Use graph primitives to generate structures for better optimization



Related Work: Graph Mapping

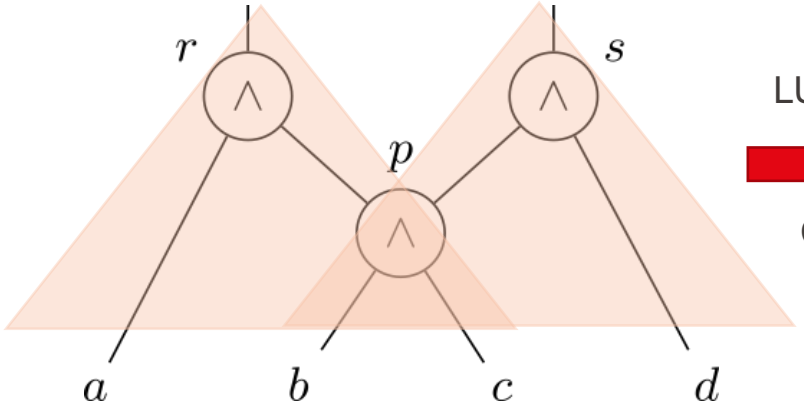
- LUT-based mapping [1]
  1. LUT mapping [2]
  2. LUT decomposition
     - Database of structures
     - Exact synthesis on-the-fly
- Area mapping by minimizing the #LUTs
- Repeat for logic restructuring



[1] W. Haaswijk, et al. "A novel basis for logic rewriting," *ASP-DAC*, 2017, pp. 151-156
[2] J. Cong and Y. Ding, "FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *Trans. CAD*, 1994

Related Work: Graph Mapping

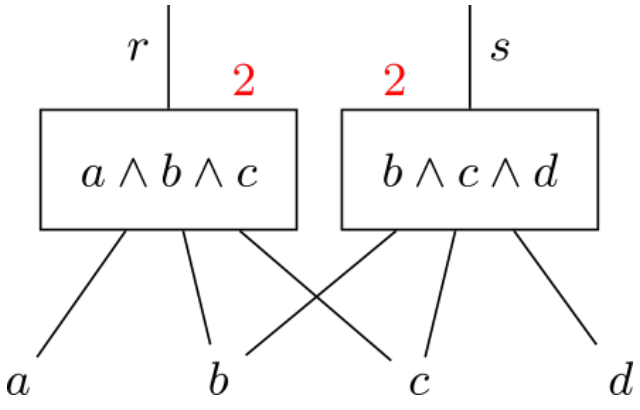
LUT-based mapping **logic sharing** limitation:



Original AIG

LUT mapping

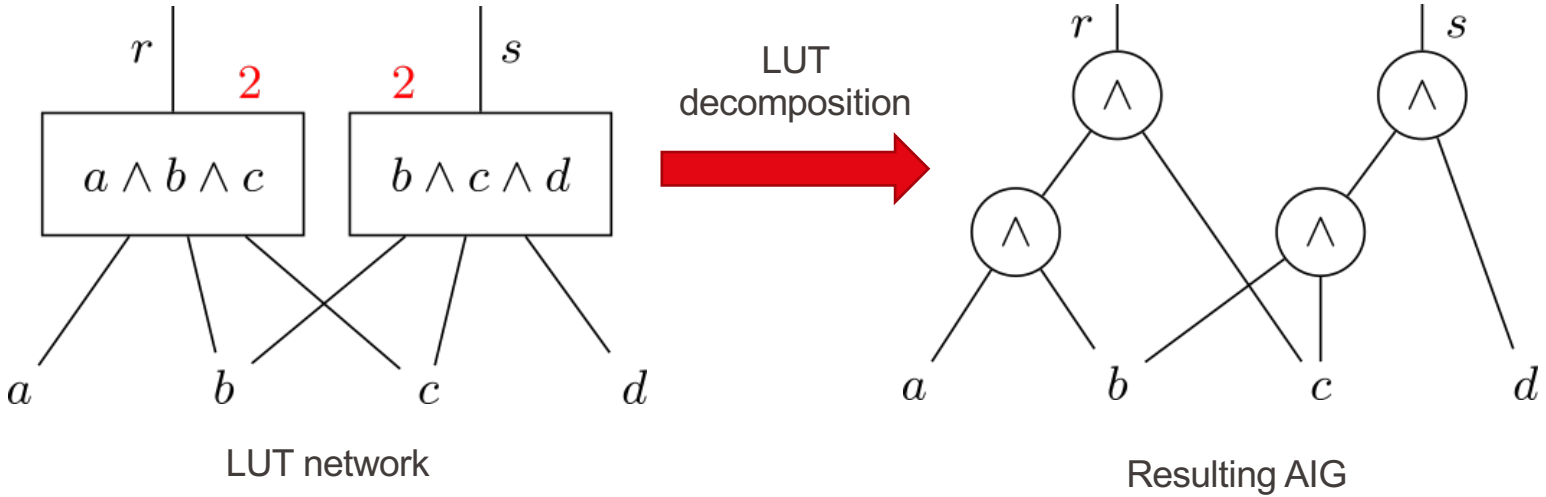
Cut size 3



LUT network

Related Work: Graph Mapping

LUT-based mapping **logic sharing** limitation:



One more node created!

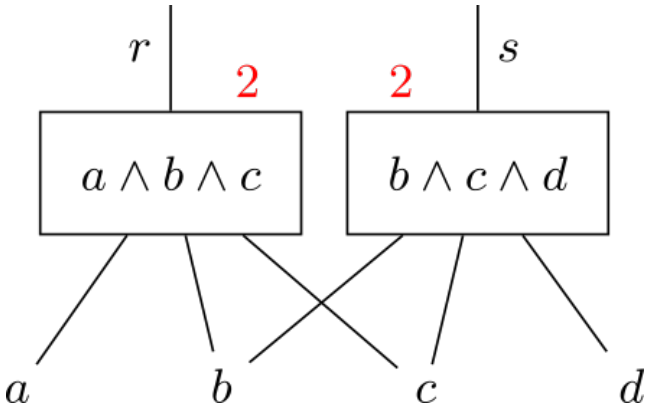
Related Work: Graph Mapping

- Logic sharing:
  - Minimize #LUTs
  - Prefer **bigger** LUTs to cover more logic
  - Lose **shared** nodes information
- Depth is unpredictable

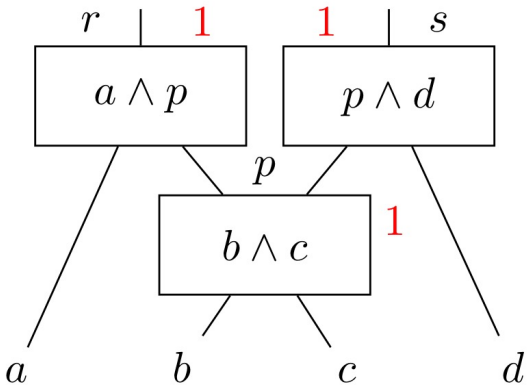
Proposed idea → **structurally** preserve the **shared nodes**

- Use **decomposition costs** to drive the mapping
- Pre-compute size and depth weight of each structure
- Minimize the total weight

Related Work: Graph Mapping

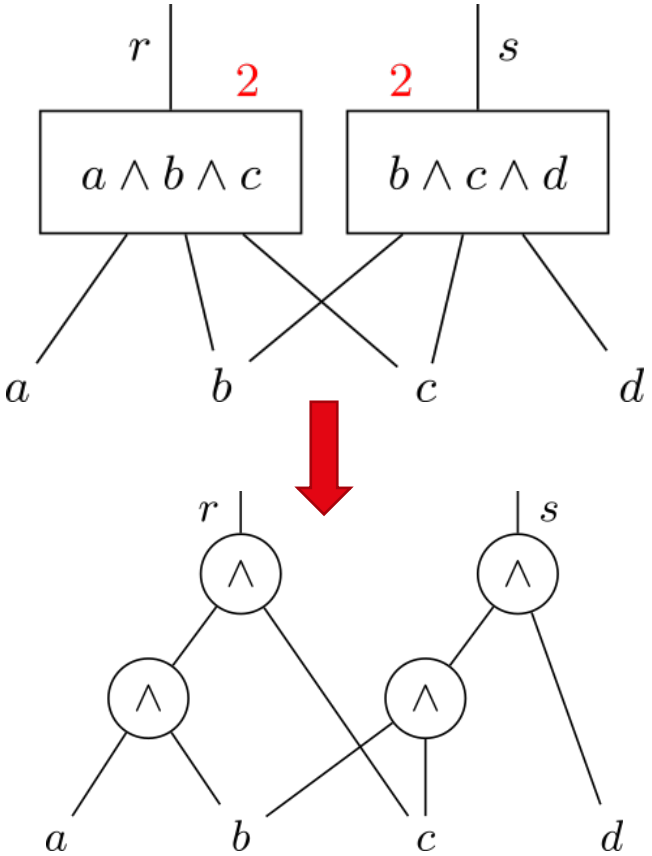


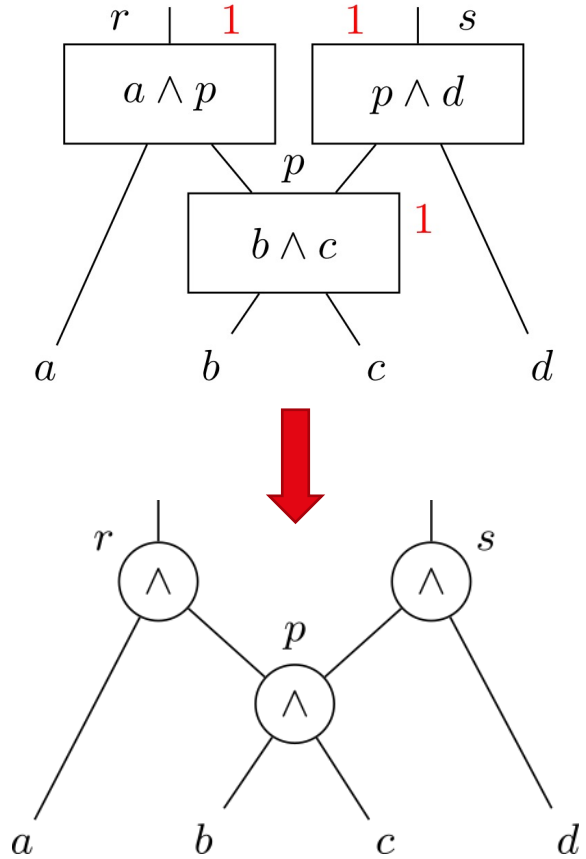
Total decomposition cost of **4**



Total decomposition cost of **3**

Related Work: Graph Mapping





Related Work: Logic restructuring

Rewriting[2]

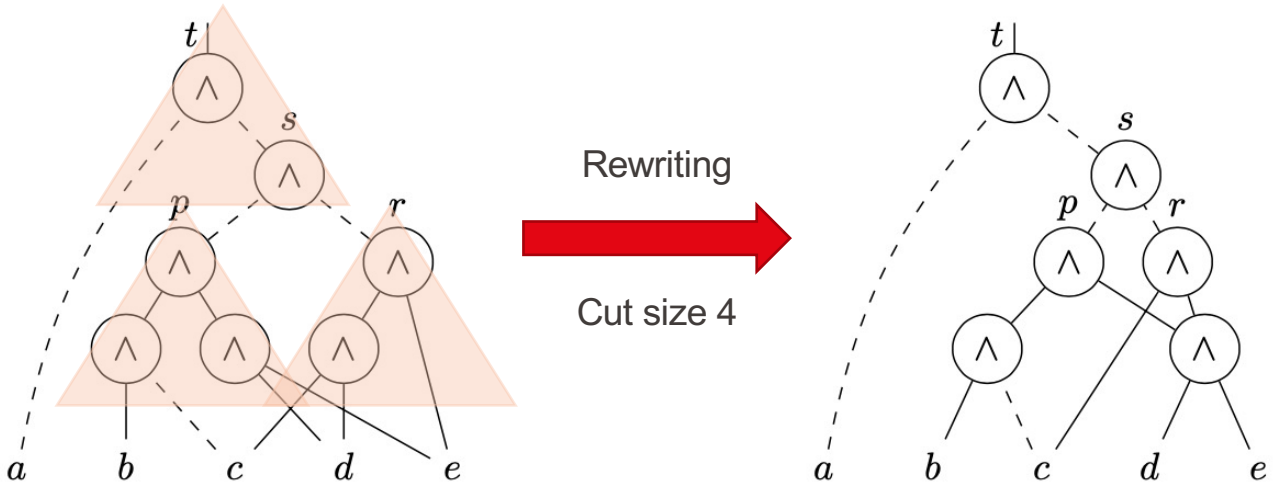
- Replace small parts with structures
- **Graph-aware**
- Exploits **structural hashing**
- Use a database of structure
- **Limitation**: replacement conflicts → limited to local view

[2] A. Mishchenko, S. Chatterjee and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," DAC, 2006, pp. 532-535.

Related Work: Logic restructuring

Rewriting[2]

- **Limitation**: replacement conflicts → limited to local view



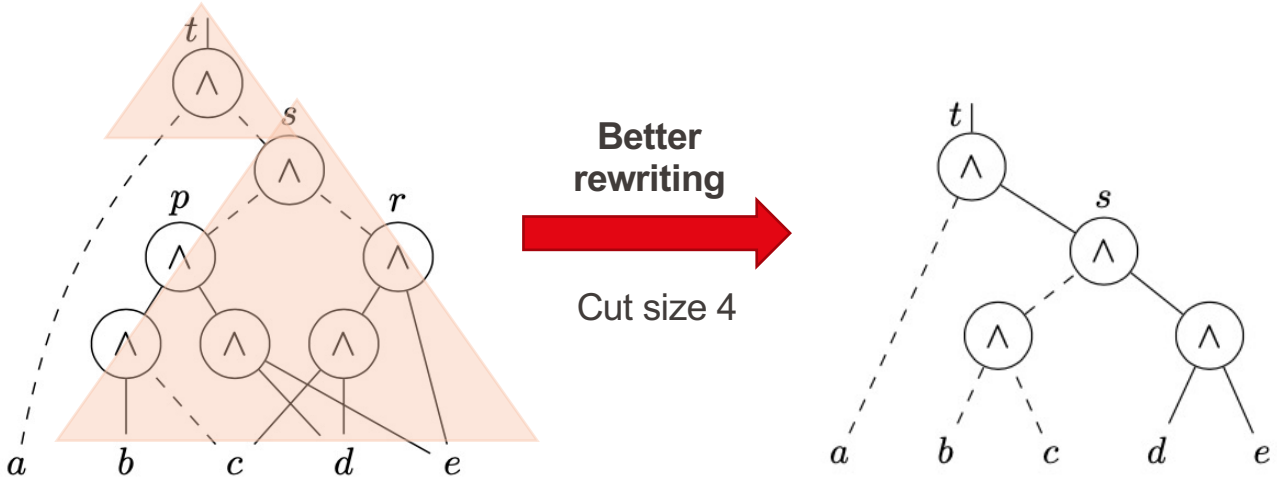
Only one node saved!

[2] A. Mishchenko, S. Chatterjee and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," DAC, 2006, pp. 532-535.

Related Work: Logic restructuring

Rewriting

- **Limitation**: replacement conflicts → limited to local view



Three nodes saved!

Related Work: Logic restructuring

Rewriting[2]

- Good for local refinement but bad for global optimization

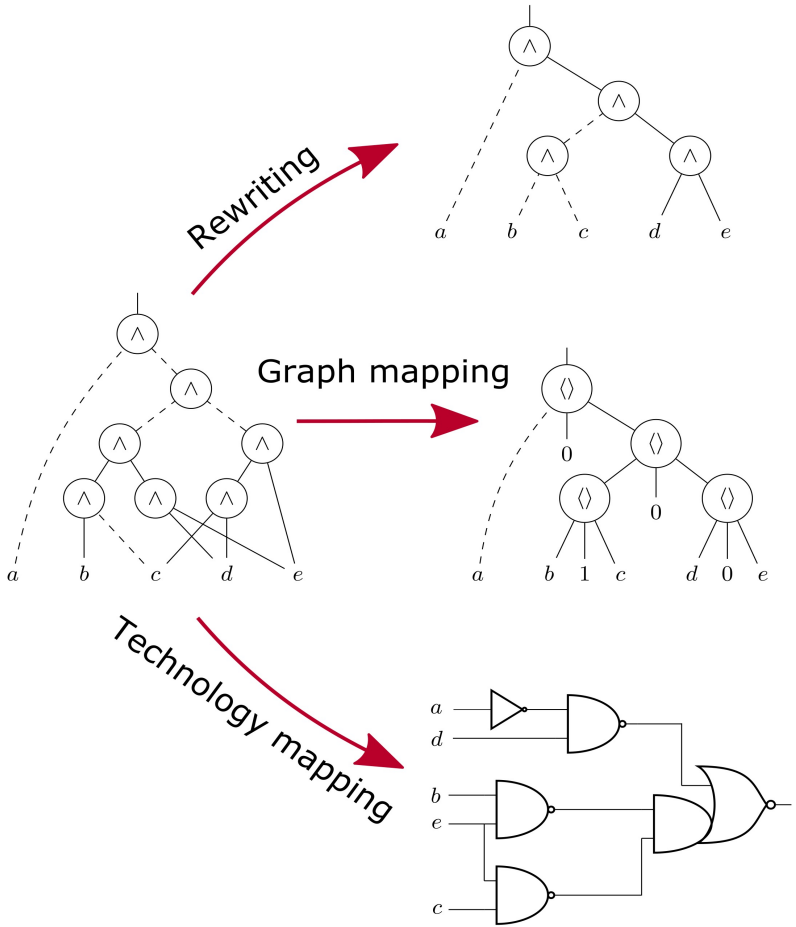
Proposed idea:

- Integrate a rewriting approach using structural hashing
- Use it as a local refinement after global mapping
  - Better global replacements

[2] A. Mishchenko, S. Chatterjee and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," DAC, 2006, pp. 532-535.

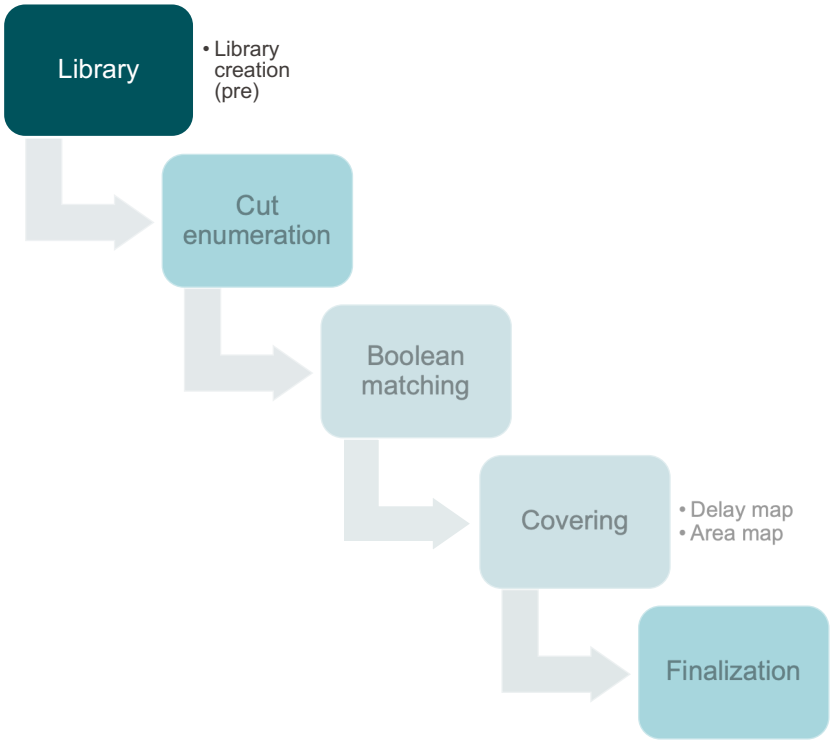
Versatile Mapper

Versatile Mapper



- Map to a cell library
- Map to a graph representation
  - Improve logic sharing
- Logic restructuring
  - Structural hashing
- Available as a command *map* in *mockturtle*: https://github.com/lsils/mockturtle

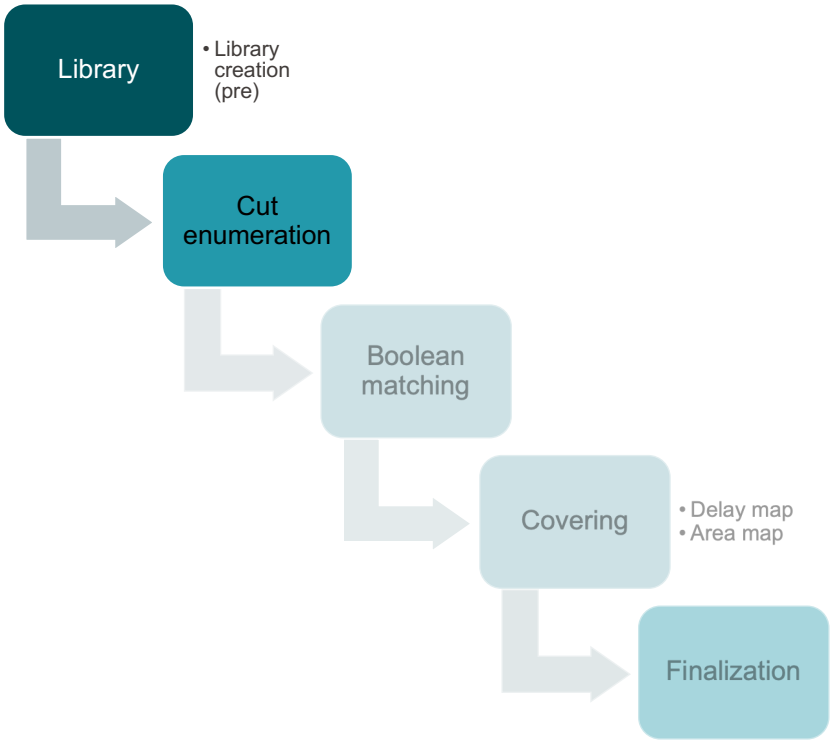
Versatile Mapper



Algorithm 1: Versatile Mapper

```
1  Input  : Boolean network N, cut size k, library,
            cut_sorting_func, constraints, skip_delay, AreaGlobalIter,
            AreaLocalIter, AreaStrashIter, rw_limit
2  Output: mapped network M
3  cuts ← compute_cuts(N, k, cut_sorting_func);
4  match_cuts(cuts, library);
5  if !skip_delay then
6  |   delay_oriented_map(N, cuts);
7  end
8  for i ← 1 to AreaGlobalIter do
9  |   compute_required_times(N, cuts, constraints);
10 |   global_area_oriented_map(N, cuts);
11 end
12 for i ← 1 to AreaLocalIter do
13 |   compute_required_times(N, cuts, constraints);
14 |   local_area_oriented_map(N, cuts);
15 end
16 M ← new_network();
17 foreach primary input i ∈ N do
18 |   create_input(M, i);
19 end
20 if graph mapping and AreaStrashIter then
21 |   compute_required_times(N, cuts, constraints);
22 |   local_area_strash_oriented_map(N, cuts, M, rw_limit);
23 |   remove_dangling(M);
24 else
25 |   finalize_network(N, cuts, M);
26 end
27 return M;
```

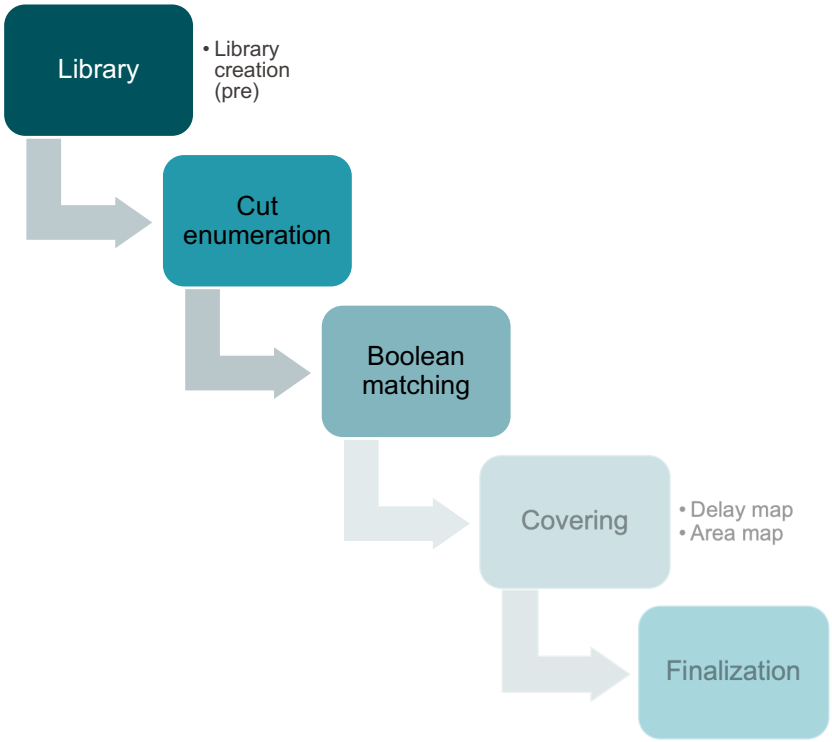
Versatile Mapper



Algorithm 1: Versatile Mapper

```
1  Input  : Boolean network N, cut size k, library,
         cut_sorting_func, constraints, skip_delay, AreaGlobalIter,
         AreaLocalIter, AreaStrashIter, rw_limit
2  Output: mapped network M
3  cuts ← compute_cuts(N, k, cut_sorting_func);
4  match_cuts(cuts, library);
5  if !skip_delay then
6  |   delay_oriented_map(N, cuts);
7  end
8  for i ← 1 to AreaGlobalIter do
9  |   compute_required_times(N, cuts, constraints);
10 |   global_area_oriented_map(N, cuts);
11 end
12 for i ← 1 to AreaLocalIter do
13 |   compute_required_times(N, cuts, constraints);
14 |   local_area_oriented_map(N, cuts);
15 end
16 M ← new_network();
17 foreach primary input i ∈ N do
18 |   create_input(M, i);
19 end
20 if graph mapping and AreaStrashIter then
21 |   compute_required_times(N, cuts, constraints);
22 |   local_area_strash_oriented_map(N, cuts, M, rw_limit);
23 |   remove_dangling(M);
24 else
25 |   finalize_network(N, cuts, M);
26 end
27 return M;
```

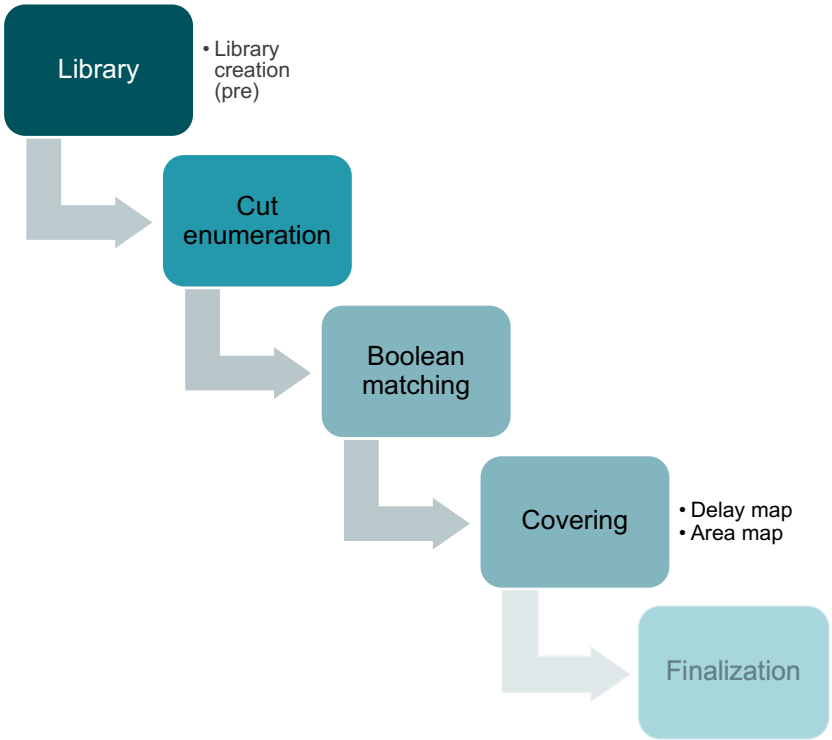
Versatile Mapper



Algorithm 1: Versatile Mapper

```
1  Input  : Boolean network N, cut size k, library,
           cut_sorting_func, constraints, skip_delay, AreaGlobalIter,
           AreaLocalIter, AreaStrashIter, rw_limit
2  Output: mapped network M
3  cuts ← compute_cuts(N, k, cut_sorting_func);
4  match_cuts(cuts, library);
5  if !skip_delay then
6  |   delay_oriented_map(N, cuts);
7  end
8  for i ← 1 to AreaGlobalIter do
9  |   compute_required_times(N, cuts, constraints);
10 |   global_area_oriented_map(N, cuts);
11 end
12 for i ← 1 to AreaLocalIter do
13 |   compute_required_times(N, cuts, constraints);
14 |   local_area_oriented_map(N, cuts);
15 end
16 M ← new_network();
17 foreach primary input i ∈ N do
18 |   create_input(M, i);
19 end
20 if graph mapping and AreaStrashIter then
21 |   compute_required_times(N, cuts, constraints);
22 |   local_area_strash_oriented_map(N, cuts, M, rw_limit);
23 |   remove_dangling(M);
24 else
25 |   finalize_network(N, cuts, M);
26 end
27 return M;
```

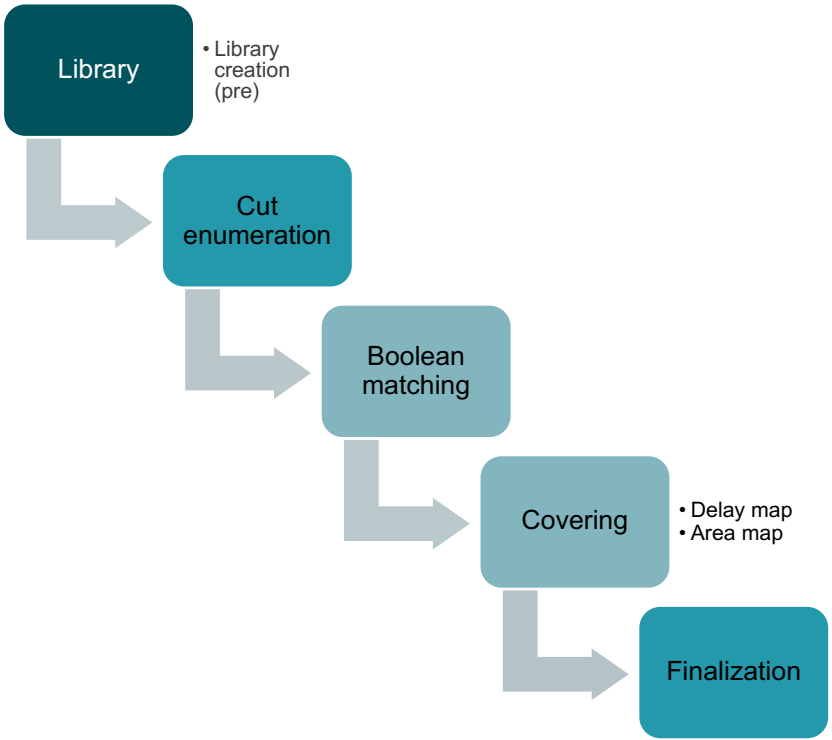
Versatile Mapper



Algorithm 1: Versatile Mapper

```
1  Input  : Boolean network N, cut size k, library,
            cut_sorting_func, constraints, skip_delay, AreaGlobalIter,
            AreaLocalIter, AreaStrashIter, rw_limit
2  Output: mapped network M
3  cuts ← compute_cuts(N, k, cut_sorting_func);
4  match_cuts(cuts, library);
5  if !skip_delay then
6  |   delay_oriented_map(N, cuts);
7  end
8  for i ← 1 to AreaGlobalIter do
9  |   compute_required_times(N, cuts, constraints);
10 |   global_area_oriented_map(N, cuts);
11 end
12 for i ← 1 to AreaLocalIter do
13 |   compute_required_times(N, cuts, constraints);
14 |   local_area_oriented_map(N, cuts);
15 end
16 M ← new_network();
17 foreach primary input i ∈ N do
18 |   create_input(M, i);
19 end
20 if graph mapping and AreaStrashIter then
21 |   compute_required_times(N, cuts, constraints);
22 |   local_area_strash_oriented_map(N, cuts, M, rw_limit);
23 |   remove_dangling(M);
24 else
25 |   finalize_network(N, cuts, M);
26 end
27 return M;
```

Versatile Mapper



Algorithm 1: Versatile Mapper

```
1  Input  : Boolean network N, cut size k, library,
           cut_sorting_func, constraints, skip_delay, AreaGlobalIter,
           AreaLocalIter, AreaStrashIter, rw_limit
2  Output: mapped network M
3  cuts ← compute_cuts(N, k, cut_sorting_func);
4  match_cuts(cuts, library);
5  if !skip_delay then
6  |  delay_oriented_map(N, cuts);
7  end
8  for i ← 1 to AreaGlobalIter do
9  |  compute_required_times(N, cuts, constraints);
10 |  global_area_oriented_map(N, cuts);
11 end
12 for i ← 1 to AreaLocalIter do
13 |  compute_required_times(N, cuts, constraints);
14 |  local_area_oriented_map(N, cuts);
15 end
16 M ← new_network();
17 foreach primary input i ∈ N do
18 |  create_input(M, i);
19 end
20 if graph mapping and AreaStrashIter then
21 |  compute_required_times(N, cuts, constraints);
22 |  local_area_strash_oriented_map(N, cuts, M, rw_limit);
23 |  remove_dangling(M);
24 else
25 |  finalize_network(N, cuts, M);
26 end
27 return M;
```

Versatile Mapper: Library

- Hash table for Boolean matching: Truth table → List of gates

Versatile Mapper: Library

- Hash table for Boolean matching: Truth table → List of gates

Technology Mapping

- Contain NP-configurations of gates
- Bail out symmetries
- Area
- Pin-to-pin delay

Graph Mapping

- Contain structures for NPN-classes
- Compute area (size)
- Compute pin-to-pin delay (depth)

Versatile Mapper: Library

- Hash table for Boolean matching: Truth table → List of gates

Technology Mapping	Graph Mapping
▪ Contain NP-configurations of gates ▪ Bail out symmetries ▪ Area ▪ Pin-to-pin delay	▪ Contain structures for NPN-classes ▪ Compute area (size) ▪ Compute pin-to-pin delay (depth)

Versatile Mapper: Cut Enumeration

- Working on a **generic** graphs data structure
- Cut prioritization:
  - Cell mapping:
    1. Size → certainty to have a feasible mapping (complete cell library)
    2. Depth, area (area flow)
  - Graph mapping:
    1. Area (area flow), depth
    2. Size
- Truth table support minimization
  - Check for internal don’t cares
  - Reduce the cut support

Versatile Mapper: Boolean matching

- Given a cut truth table → find gates in the library that implement the TT
- Dual-rail matching
  - Matching positive and negative polarity

Versatile Mapper: Boolean matching

- Given a cut truth table → find gates in the library that implement the TT
- Dual-rail matching
  - Matching positive and negative polarity

Technology Mapping	Graph Mapping
▪ Library lookup	▪ Matching by NPN-canonization ▪ Store permutations, input negations to apply ▪ Pin-to-pin delay is permuted during covering

Versatile Mapper: Boolean matching

- Given a cut truth table → find gates in the library that implement the TT
- Dual-rail matching
  - Matching positive and negative polarity

Technology Mapping	Graph Mapping
▪ Library lookup	▪ Matching by NPN-canonization ▪ Store permutations, input negations to apply ▪ Pin-to-pin delay is permuted during covering

Versatile Mapper: Covering

- **Delay-oriented** map
  - Select the best delay gates for each node
- **Area-recovery** or **area-oriented** map
  - Global area → Area flow
    - Estimate the area shared in the transitive fanin cone
    - Select the best area flow constrained by the required time
  - Local Area → Exact area

Versatile Mapper: Covering

- Exact Area
  - Refinement of the local cut selection
  - Driven by the area in the maximum fanout free cone (MFFC)
  - Based on recursive cut referencing/dereferencing
  - Select the cut that minimizes the local area
- Exact Area with structural hashing
  - Apply to the l best cuts
  1. **Add structures** into the network
  2. **Measure #added nodes** with recursive node referencing/dereferencing
  3. **Update the pre-computed decomposition costs**

Experiments

Experiments

i. Map to a cell library
ii. Mapping and rewriting MIGs
iii. Mapping and rewriting XAGs and XMGs

Experiments: Map to a Cell Library

- Better runtime than ABC *map* for similar or better quality
- Delay-oriented mapping

Benchmark	I/O	Baseline		ABC map			Versatile mapper		
		Size	Depth	Area	Delay	Time (s)	Area	Delay	Time (s)
adder	256 / 129	1020	255	1976	204.9	0.01	1975	204.9	0.01
bar	135 / 128	3336	12	5911	10.2	0.04	5911	10.2	0.05
div	128 / 128	57247	4372	124016	3516.5	1.20	127191	3516.5	1.34
hyp	256 / 128	214335	24801	435468	17520.6	7.35	429738	17520.6	5.59
log2	32 / 32	32060	444	55686	330.4	1.41	53778	329.8	1.02
max	512 / 130	2865	287	6186	208.4	0.06	5958	208.4	0.07
multiplier	128 / 128	27062	274	49597	210.9	1.05	47015	210.9	0.75
sin	24 / 25	5416	225	10690	154.3	0.24	10413	153.0	0.24
sqrt	128 / 64	24618	5058	44724	4235.8	0.58	44523	4235.8	0.71
square	64 / 128	18484	250	36321	199.4	0.74	35154	199.4	0.58
Total						12.68			10.36
Improvement							+1.75%	+0.10%	

Experiments: Mapping and Rewriting on MIGs

- Comparing to previous state-of-the-art methods
- Mapping for size minimization

Benchmark	Baseline		LUT-based rewriting [21]			Cut rewriting [24]			Versatile mapper		
	Size	Depth	Size	Depth	Time (s)	Size	Depth	Time (s)	Size	Depth	Time (s)
adder	1020	255	385	130	0.05	893	129	0.08	384	129	0.06
bar	3336	12	2940	14	0.15	2952	15	0.71	2588	13	1.79
div	57247	4372	48827	4288	22.77	41553	2276	157.27	36858	2235	14.95
hyp	214335	24801	163398	9168	15.80	178736	9330	93.93	137048	8885	28.83
log2	32060	444	25651	247	3.91	30056	420	8.88	24295	206	3.20
max	2865	287	2446	248	0.35	2346	240	0.85	2171	162	0.96
multiplier	27062	274	20309	138	3.07	24829	271	12.37	19299	142	2.97
sin	5416	225	4560	159	0.44	5049	201	3.66	4196	122	1.14
sqrt	24618	5058	21002	6132	2.29	23889	4941	11.69	17355	3846	45.75
square	18484	250	14050	155	1.24	17669	163	8.85	11924	126	2.39
Total					50.09			298.27			102.05
Improvement			+22.66%	+25.10%		+11.47%	+20.54%		+32.11%	+41.88%	

LUT mapping using ABC command: *&if –a –K 4*

[21] W. Haaswijk, et al. “LUT Mapping and Optimization for Majority-Inverter Graphs.”, Proc. IWLS, 2016
[24] H. Riener, et al “On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis.”, DATE, 2019

Experiments: Mapping and Rewriting on XAGs and XMGs

- Comparing to previous state-of-the-art methods
- Mapping for size minimization

Benchmark	XAG		XMG		XMG (k = 4) in [5]		XMG (k = 6) in [30]	
	Size	Depth	Size	Depth	Size	Depth	Size	Depth
adder	639	256	383	128	639	130	383	128
bar	3013	13	2944	14	3281	16	2149	14
div	29124	4316	17613	2300	29607	4371	37003	4243
hyp	158682	24912	114746	8984	155349	12507	99428	8755
log2	24330	327	21361	204	27936	275	22957	213
max	2766	234	1845	157	2296	296	1938	200
multiplier	18651	268	15642	134	17508	154	16357	133
sin	4259	175	3728	138	5100	176	3896	140
sqrt	12617	6122	9750	2431	20130	6031	17187	5169
square	13876	247	11250	126	15070	130	8325	156
Average	26,795.7	3.687.0	19,926.2	1,461.6	27961.6	2408.6	20,962.3	1,915.1
GeoMean	2,293.1		1,511.2		2,117.8		1,721.6	
Size · Depth	98,795,745.9		29,124,133.9		66,697,987.8		40,144,900.7	

LUT mapping using ABC command: *&if –a –K 4*

[5] W. Haaswijk, et al. "A novel basis for logic rewriting," *ASP-DAC*, 2017, pp. 151-156
[30] Z. Chu, M. Soeken, Y. Xia, and G. De Micheli, “Functional decomposition using majority,” in *ASP-DAC*, 2018, pp. 676–681

Conclusions

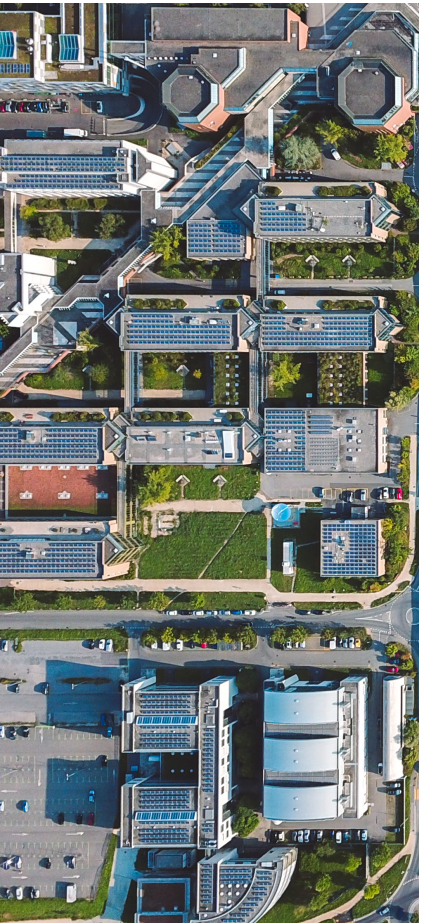
i. **Versatility**
   - Map to a cell library
   - Map to a graph representation
   - Logic restructuring

ii. **Applications** in recent technologies and logic optimization

iii. **Better runtime** in technology mapping compared to ABC *map*

iv. **Better results** in **graph mapping** and **logic restructuring**
   - Pre-compute structures info
   - Better logic sharing
   - Global optimization
   - Depth optimization
   - Structural hashing

Available as a command *map* in *mockturtle*
https://github.com/lsils/mockturtle



Thank you for your attention!

alessandro.tempiacalvino@epfl.ch

19th Jan 2022