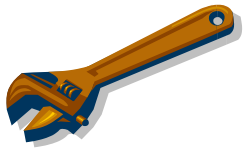
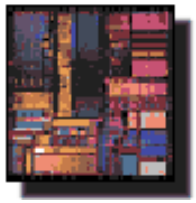






ECE 595Z
Digital VLSI Design Automation

Module 5 (Lectures 14-20): Multi-level Synthesis
Lecture 20



Anand Raghunathan

MSEE 348

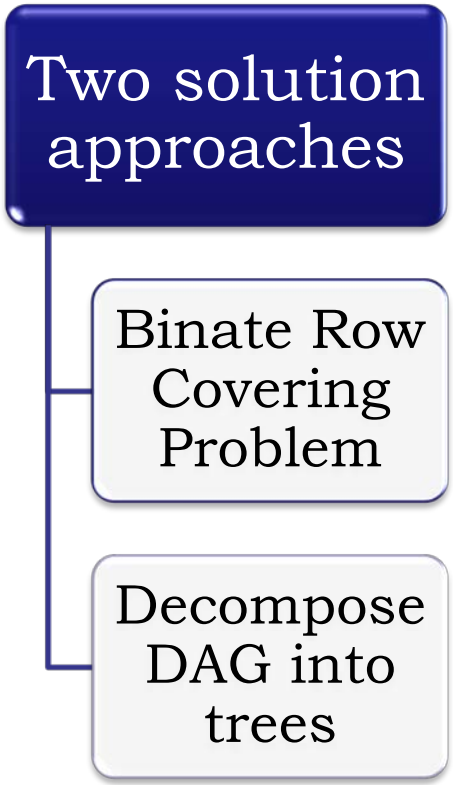
raghunathan@purdue.edu

Lecture #19: Re-cap

- Technology mapping realizes a Boolean circuit using gates or cells from a library
- Technology mapping can be formulated as a graph covering problem
  - Subject graph (represents circuit to be mapped)
  - Pattern graphs (represent library)

Technology Mapping Using Graph Covering

- General Approach
  - Construct a subject DAG (Directed Acyclic Graph) for the Boolean network
  - Represent each gate in the target library by pattern DAGs
  - Find an optimal-cost covering of subject DAG using the collection of pattern DAGs
- Challenge: Complexity of DAG covering
  - NP-hard
  - Remains NP-hard even when all nodes have in-degree ≤ 2



If subject graph and pattern graph are trees (each vertex has an out-degree of 1), then an efficient algorithm exists!

Background: Unate and Binate Covering Problems

- Recall covering from two-level minimization?
  - Covering table
    - Rows = minterms
    - Columns = primes
    - Select a set of columns such that all rows are "covered"
    - Column j covers a row i if entry (i,j) in table is 1
- This represents a **Unate Covering Problem**
  - A row is covered only by **inclusion** of a column
  - All literals in the equivalent CNF formula are in un-complemented form

Covering Table

	p1	p2	p3	p4	p5	p6	p7	p8
m1	1	0	0	0	0	0	0	0
m2	1	1	0	0	0	0	0	1
m3	0	1	1	0	0	0	0	0
m4	0	0	1	1	1	0	0	0
m5	0	0	0	0	1	0	1	1
m6	0	0	0	0	0	1	1	0
m7	0	0	0	0	1	1	0	1
m8	0	0	0	0	1	1	1	0

Equivalent CNF Formula

P1(P1 + P2 + P8)(P2 + P3)
(P3 + P4 + P5)(P5 + P7 + P8)
(P6 + P7)(P5 + P6 + P8)
(P5 + P6 + P7)

Background: Binate Covering Problems

- Also model constraints that require **exclusion** of some columns
- Rows may be covered by **including or excluding** a column
- Table entries are 0,1,-
  - 1 ⇒ including the column covers the row
  - 0 ⇒ excluding the column covers the row
  - - ⇒ including or excluding the column has no bearing on whether the row is covered
- Equivalent CNF formula has literals in true and complemented form

Covering Table

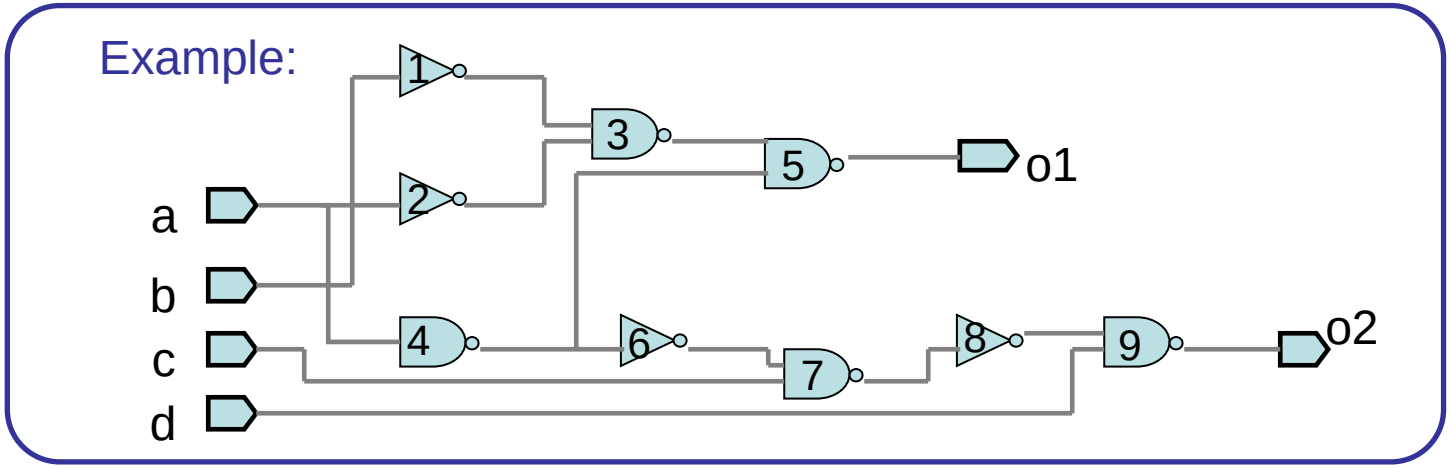
	p1	p2	p3	p4	p5
m1	1	0	-	-	0
m2	1	1	-	-	-
m3	0	1	-	0	-
m4	0	0	1	-	1
m5	1	-	-	-	1
m6	0	-	-	1	0
m7	-	-	-	0	1

Equivalent CNF Formula

(P1 + P2' + P5') (P1 + P2)
(P1' + P2 + P4')
(P1' + P2' + P3 + P5)
(P1 + P5)(P1' + P4 + P5')
(P4' + P5)

Technology Mapping as Binate Covering

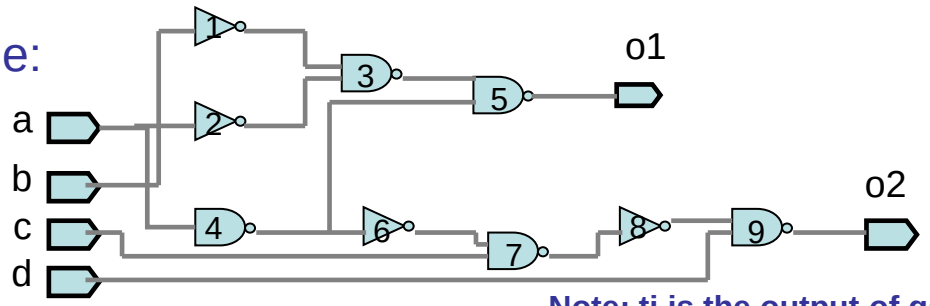
- Compute all possible matches in the subject graph using the pattern graphs
- For each match, record the following
  - Cost
  - Which nodes in the subject graph it covers
  - Inputs
  - Outputs



Technology Mapping as Binate Covering

- Compute all possible matches in the subject graph using the pattern graphs
- For each match, record
  - Cost
  - Which nodes in the subject graph it covers
  - Inputs
  - Outputs

Example:

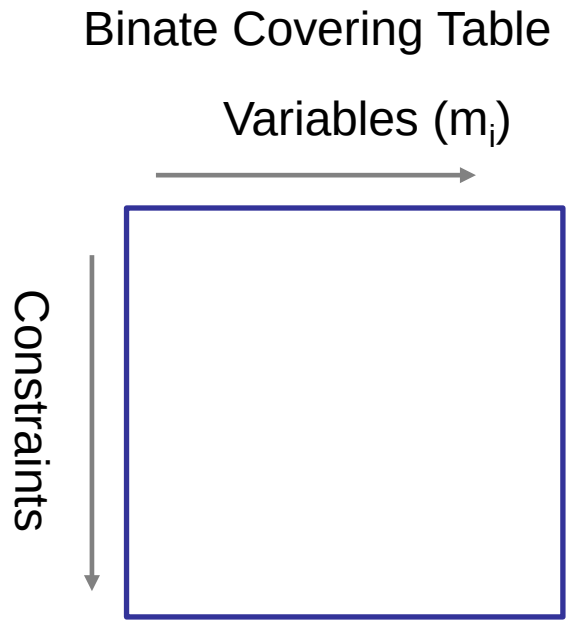


Note: ti is the output of gate i

Match	Gate	Cost	Inputs	Produces	Covers
m1	inv	1	b	t1	g1
m2	inv	1	a	t2	g2
m3	nand2	2	t1, t2	t3	g3
m4	nand2	2	a, b	t4	g4
m5	nand2	2	t3, t4	t5	g5
m6	inv	1	t4	t6	g6
m7	nand2	2	t6, c	t7	g7
m8	inv	1	t7	t8	g8
m9	nand2	2	t8, d	t9	g9
m10	nand3	3	t6, c, d	t9	g7, g8, g9
m11	nand3	3	a, b, c	t7	g4, g6, g7
m12	xnor2	5	a, b	t5	g1, g2, g3, g4, g5
m13	nand4	4	a, b, c, d	t9	g4, g6, g7, g8, g9
m14	oai21	3	a, b, t4	t5	g1, g2, g3, g5

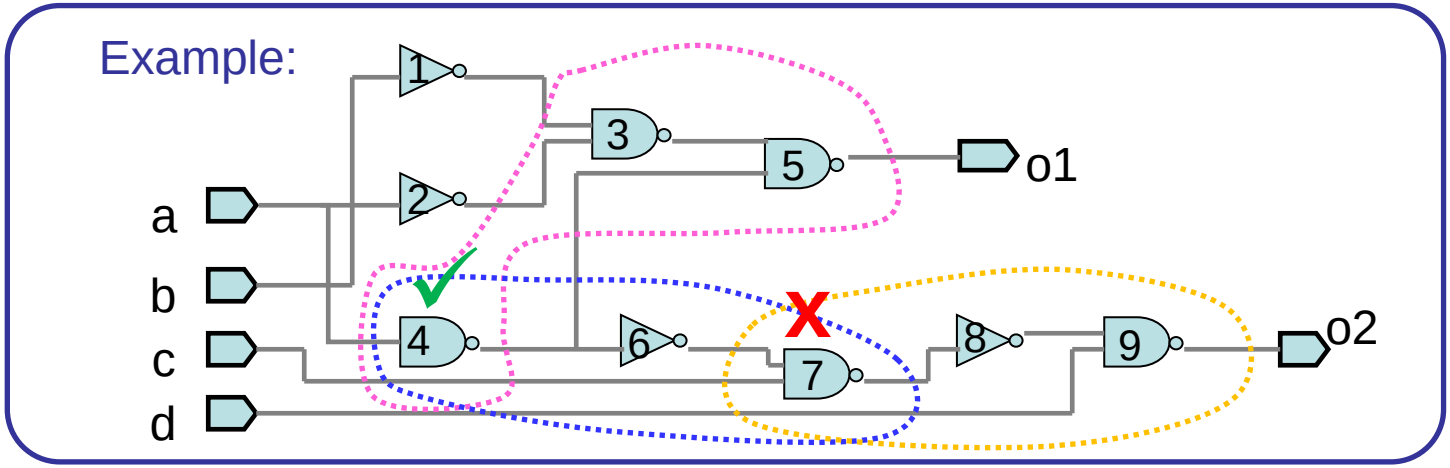
Technology Mapping as Binate Covering

- Consider each match as a Boolean variable
  - Let $m_i \in \{0,1\}$, indicate the exclusion or the inclusion of match i in the cover.
    - $m_i = 0 \Rightarrow$ match i is excluded.
    - $m_i = 1 \Rightarrow$ match i is included
- Problem reduces to determining cost-optimal assignment to m_i subject to constraints



Technology Mapping as Binate Covering

- Constraints ensure that the solution to the covering problem leads to a valid circuit
  1. Need to cover each node in the subject graph with a match (one or more).
     1. Multiple matches = logic duplication (may actually reduce cost)
  2. Need to make sure that the inputs of the matches in the cover are either primary inputs or outputs of other matches in the cover.
  3. Need to make sure that each primary output is produced as the output of a match in the cover.



Example

Each node must be covered by at least one match:

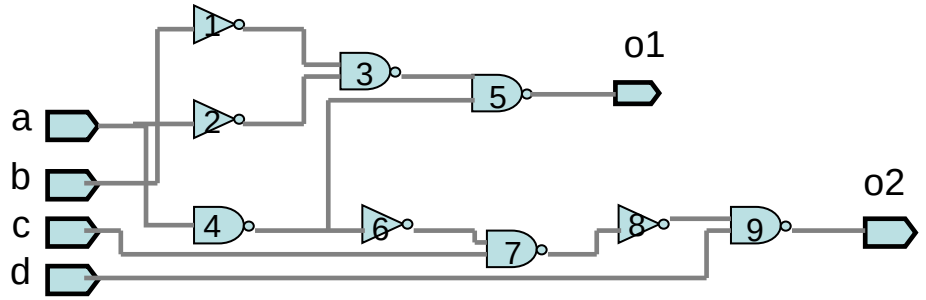
(m1+m12+m14)
(m2+m12+m14)(m3+m12+m14)
(m4+m11+m12+m13)
(m5+m12+m14)(m6+m11+m13)
(m7+m10+m11+m13)
(m8+m10+m13)(m9+m10+m13)

Each input of a selected match must be a primary input or the output of some other selected match:

(m3'+m1)(m3'+m2)
(m5'+m3)(m5'+m4)(m6'+m4)
(m7'+m6)(m8'+m7)(m9'+m8)
(m10'+m6)(m14'+m4)

The two primary outputs must be outputs of matches:

(m5+m12+m14)
(m9+m10+m13)



Match	Gate	Cost	Inputs	Produces	Covers
m1	inv	1	b	t1	g1
m2	inv	1	a	t2	g2
m3	nand2	2	t1, t2	t3	g3
m4	nand2	2	a, b	t4	g4
m5	nand2	2	t3, t4	t5	g5
m6	inv	1	t4	t6	g6
m7	nand2	2	t6, c	t7	g7
m8	inv	1	t7	t8	g8
m9	nand2	2	t8, d	t9	g9
m10	nand3	3	t6, c, d	t9	g7, g8, g9
m11	nand3	3	a, b, c	t7	g4, g6, g7
m12	xnor2	5	a, b	t5	g1, g2, g3, g4, g5
m13	nand4	4	a, b, c, d	t9	g4, g6, g7, g8, g9
m14	oai21	3	a, b, t4	t5	g1, g2, g3, g5

Example

Each node must be covered by at least one match:

(m1+m12+m14)
(m2+m12+m14)(m3+m12+m14)
(m4+m11+m12+m13)
(m5+m12+m14)(m6+m11+m13)
(m7+m10+m11+m13)
(m8+m10+m13)(m9+m10+m13)

Each input of a selected match must be a primary input or the output of some other selected match:

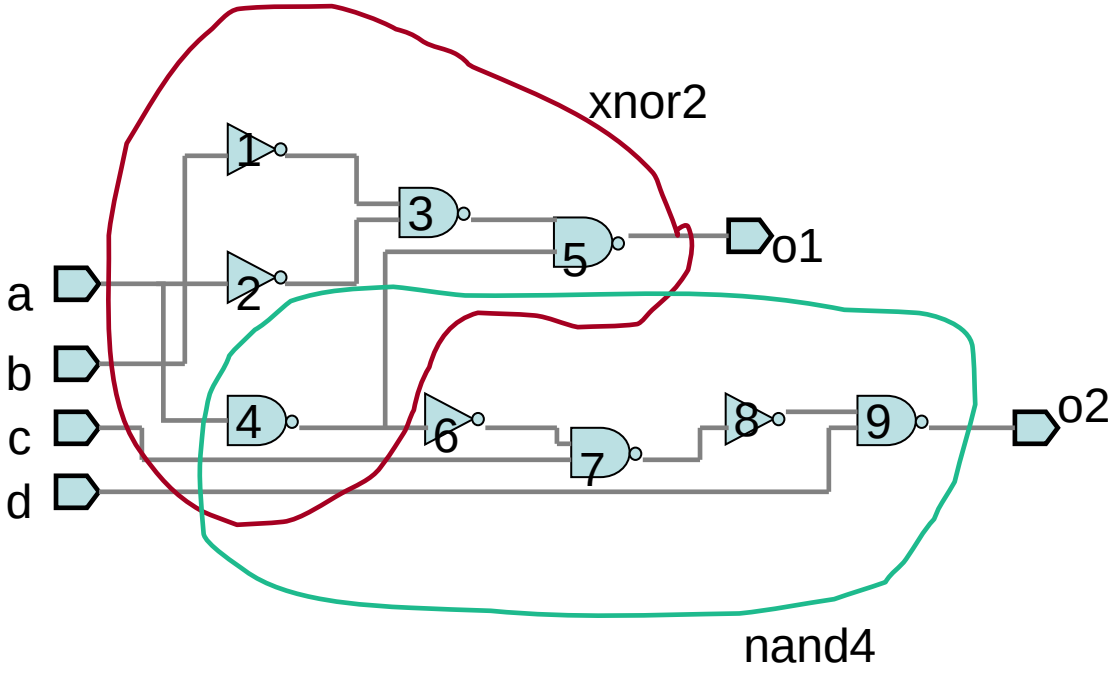
(m3'+m1)(m3'+m2)(m5'+m3)
(m5'+m4)(m6'+m4)(m7'+m6)
(m8'+m7)(m9'+m8)(m10'+m6)
(m14'+m4)

The two primary outputs must be outputs of matches:

(m5+m12+m14)
(m9+m10+m13)

Least cost solution is:
m3' m5' m6' m7' m8' m9' m10' m12 m13 m14'

Uses two gates for a cost of 9.
m12: xnor (5)
m13: nand4 (4)



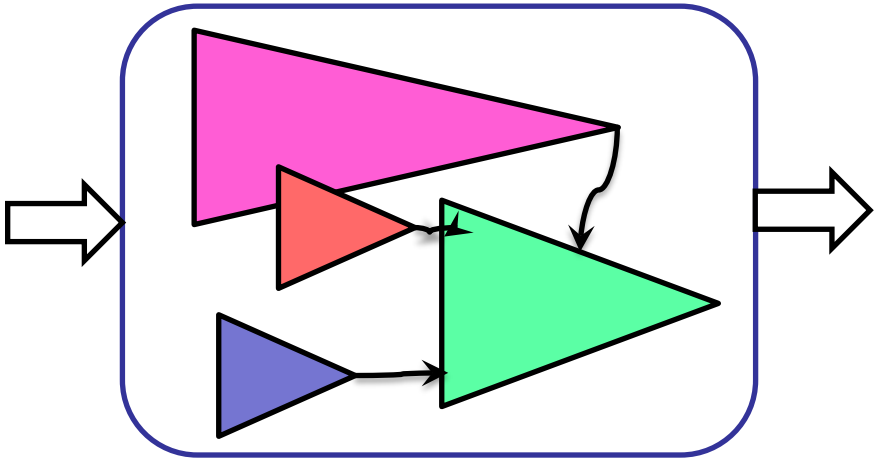
Problems with Binate Covering Formulation of Technology Mapping

- Intrinsically a hard problem.
  - Even finding a feasible solution for a binate covering problem is NP-complete (SAT) – very easy in the case of unate covering
  - Search techniques seem to work well for large unate covering problems, but not for binate covering problems ☹
- Problem size very large.
  - Large number of matches.
  - Large number of constraints (rows).
- Cost function is not very flexible.
  - Cost for a match must be independent of the cost of other matches.
    - Works for area as a cost, but not for delay and power (as we will see).

DAG Covering Using Tree Covering

- If subject graph and pattern graphs are trees, then efficient covering algorithms exist.
- Uses a dynamic programming algorithm.
  - Same technique first used for code generation in compilers. (Aho and Johnson, ACM STOC, 1975)

Idea: Partition any given subject DAG into a **forest** of **trees**, and use efficient algorithm on each tree



K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," in Proc. 24th ACM/IEEE Conf. on Design Automation, pp. 341-347, 1987

Dynamic Programming

- A method of solving problems that exhibit the properties of **overlapping sub-problems** and **optimal sub-structure**
  - **Optimal sub-structure**: optimal solutions to sub-problems can be composed to find the optimal solution to the overall problem
  - **Overlapping sub-problems**: same sub-problems are (re-)used to solve many different larger problems
- Originally described by Bellman
  - *Principle of Optimality: An optimal sequence of decisions has the property that whatever the initial state and decisions are, the remaining decisions must constitute an optimal decision sequence w.r.t the state resulting from the first decision.*
- Objective function (or cost) typically represented as a recursive equation:

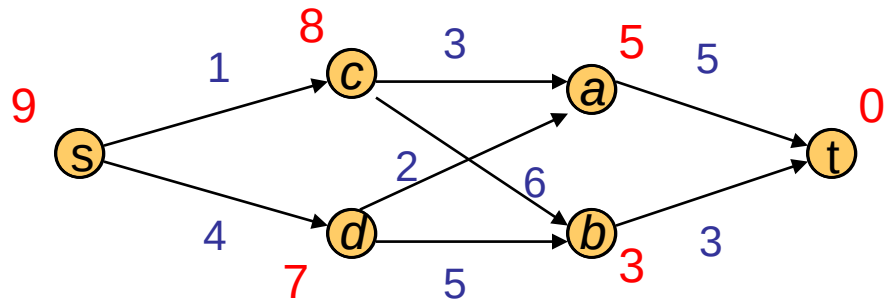
$$min\_cost(problem) = min_{all\ local\ choices\ i}\{ local\_cost(choice_i) + min\_cost(sub\text{-}problems\ resulting\ from\ choice\ i) \}$$

Dynamic Programming – Simple Example

- Shortest path in a directed graph

$$cost(i) = min_k\{\, d_{ik} + cost(k) \,\}$$

Example:



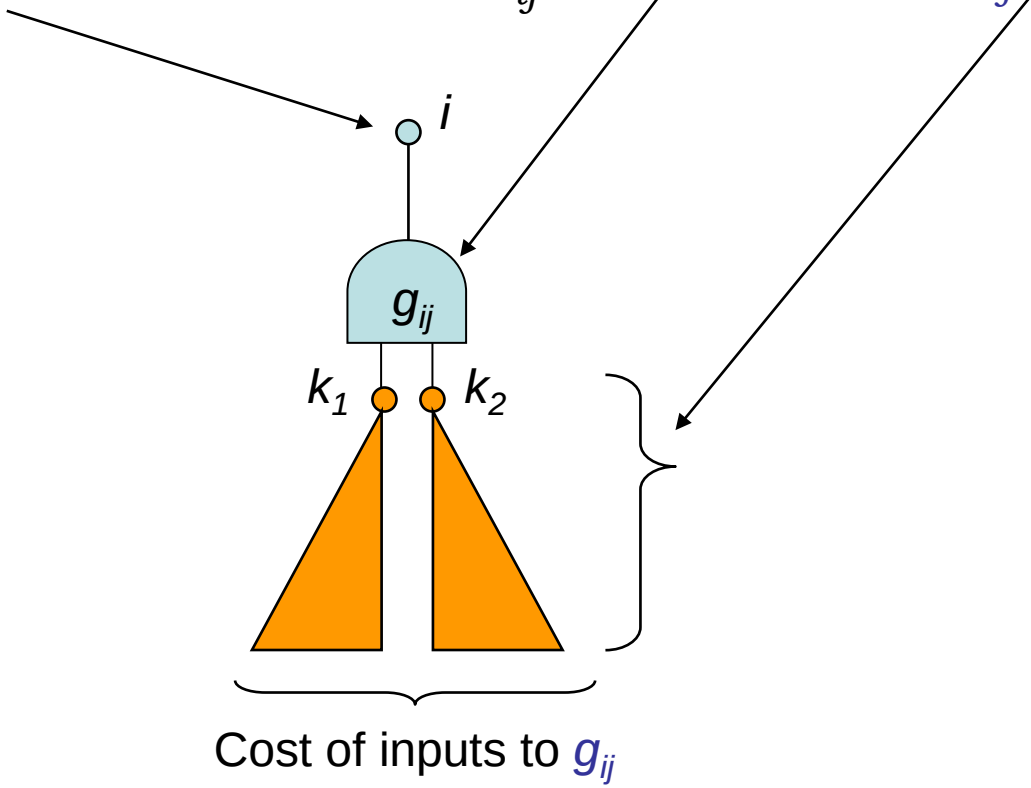
cost(v) = length of shortest path from v to destination vertex

cost(s) = min(1+cost(c), 4+cost(d))
cost(c) = min(3+cost(a), 6+cost(b))
cost(d) = min(2+cost(a), 5+cost(b))
cost(a) = 5+cost(t)
cost(b) = 3+cost(t)

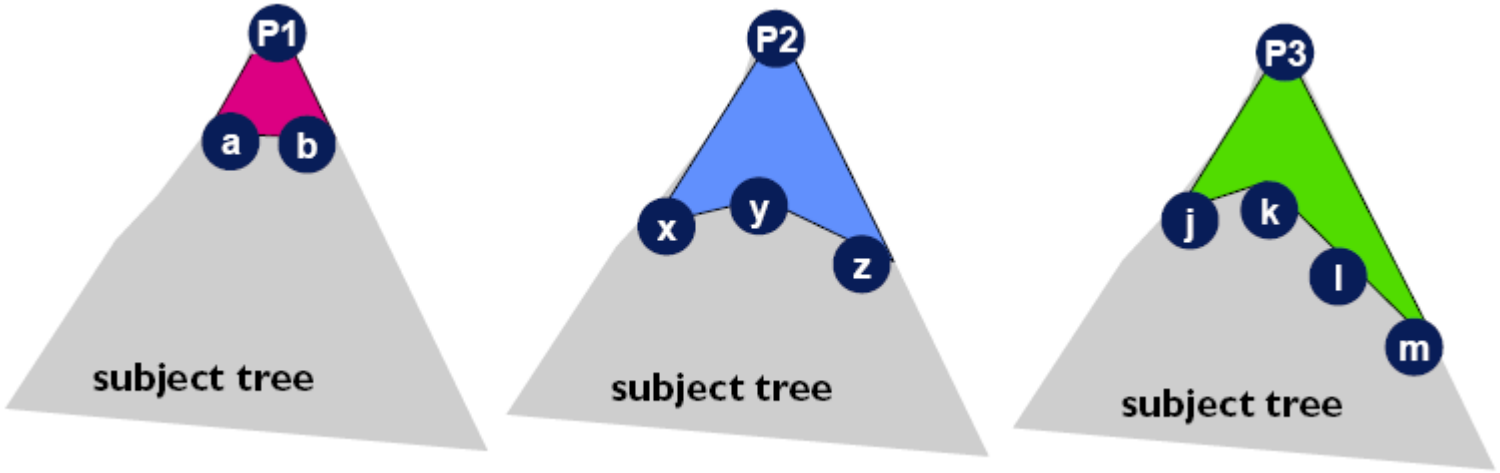
Tree Covering Using Dynamic Programming

- For node i in a subject tree

$$min\_cost(i) = min_{\text{all matches } g_{ij} \text{ at } i}\{cost(g_{ij}) + \Sigma_{k_j} min\_cost(k_j)\}$$



Tree Covering Using Dynamic Programming



min_cost(root) = min{ (cost(P1) + min_cost(a) + min_cost(b)),
(cost(P2) + min_cost(x) + min_cost(y) + min_cost(z))
(cost(P3) + min_cost(j) + min_cost(k) + min_cost(l) + min_cost(m))

Principle of Optimality: The optimal cover for a tree consists of a match at its root, plus the optimal covers for the sub-trees starting at each input of the match

Tree Covering Using Dynamic Programming

- Naïve recursive algorithm for tree covering

```
int min_area(v, P){
/* v is a vertex in the tree, P is the set of pattern graphs */
        best_cost = infinity;
        foreach(m = match(v, P)) {
                cost(m) = area(m) + Σ_{v_i ∈ inputs(m)} min_area(v_i, P);
                if(cost(m) < best_cost){
                        match(v) = m;
                        best_cost = cost(m);
                }
        }
        return best_cost;
}
```

Consider all matches at v

Recursive call

- What's wrong with this algorithm?

Tree Covering Using Dynamic Programming

- Better recursive algorithm: Avoid re-computing best match at a node
- Save the best cost the first time you visit each node

```
int min_area(v, P){
/* v is a vertex in the tree, P is the set of pattern graphs */
        if (best_cost[v] < infinity) return best_cost[v];
        foreach(m = match(v, P)){
                cost(m) = area(m) + Σ_{vi ∈ inputs(m)} min_area(vi, P);
                if(cost(m) < best_cost[v]){
                        match(v) = m;
                        best_cost[v] = cost(m);
                }
        }
        return best_cost[v];
}
```

Note: Assume all entries in array best_cost[] are initialized to infinity

Tree Covering Using Dynamic Programming

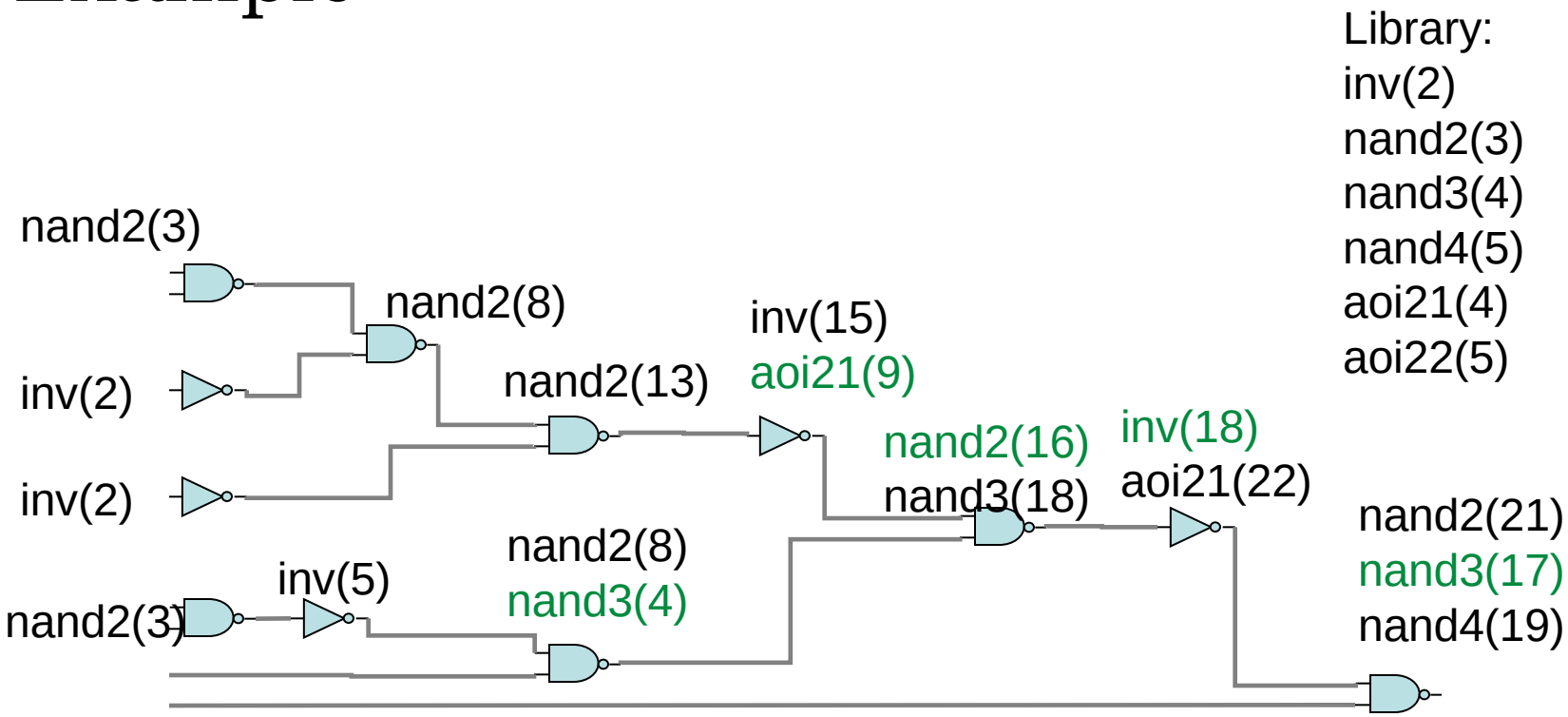
- Iterative algorithm
  - Visit nodes in topological order from inputs to outputs

```
int min_area(T, P){
/* T is the tree, P is the set of pattern graphs */
   foreach (v ∈ T in topological order) {
         best_cost[v] = infinity;
         foreach(m = match(v, P)) {
               cost(m) = area(m) + Σ_{vi ∈ inputs(m)} best_cost[vi];
               if(cost(m) < best_cost[v]){
                     match(v) = m;
                     best_cost[v] = cost(m);
               }
         }
   }
   return best_cost[root of T];
}
```

Topological order guarantees that best_cost[vi] will be available when we get here

Tree Covering in Action

- Example



Complexity of Tree Covering

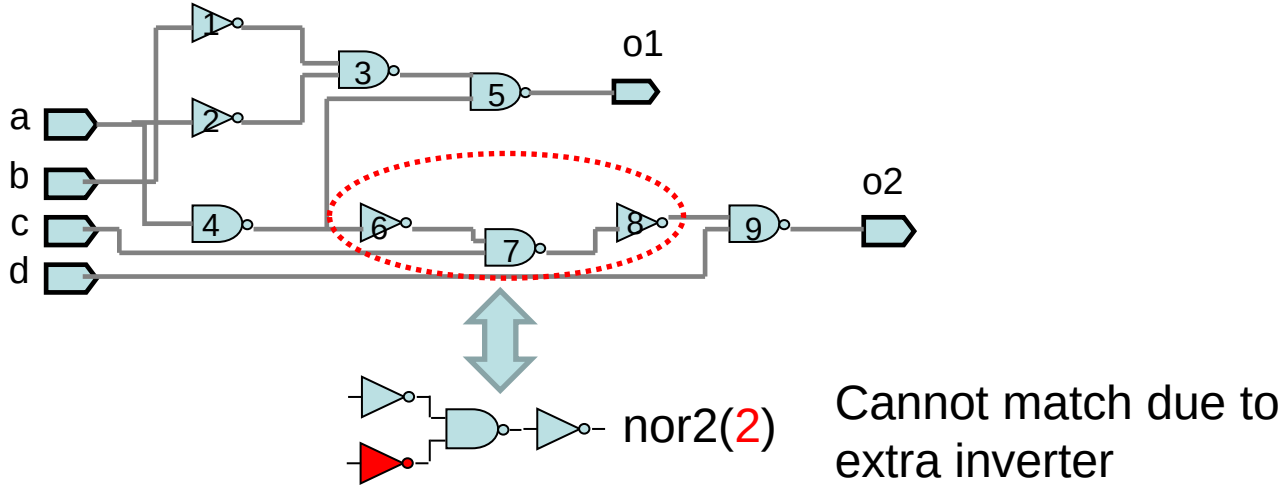
- Complexity of tree covering is *linear* in both size of subject tree and size of collection of pattern trees
- But: for the overall mapping, must add complexity of *matching*
  - *Complexity = O(nodes)*(complexity of matching)*

OK, so that's the basic idea..

- What else?
  - Inverter heuristic: Improving matching quality
  - How do we optimize delay?
  - Partitioning a DAG into trees

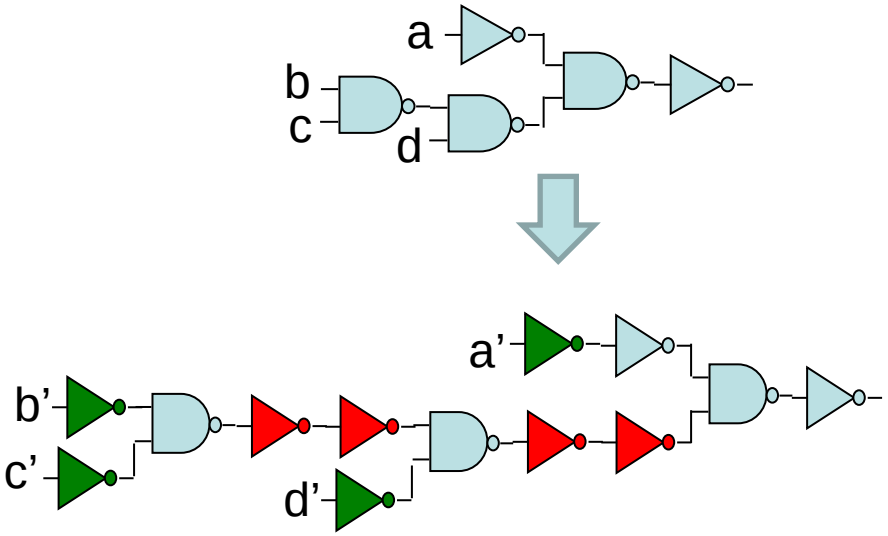
Inverter Heuristic

- Sometimes, you cannot match patterns due to additional inverters
- There is a simple way to enable such matches
  - Add additional "zero-cost" inverters in the subject graph (without changing functionality)



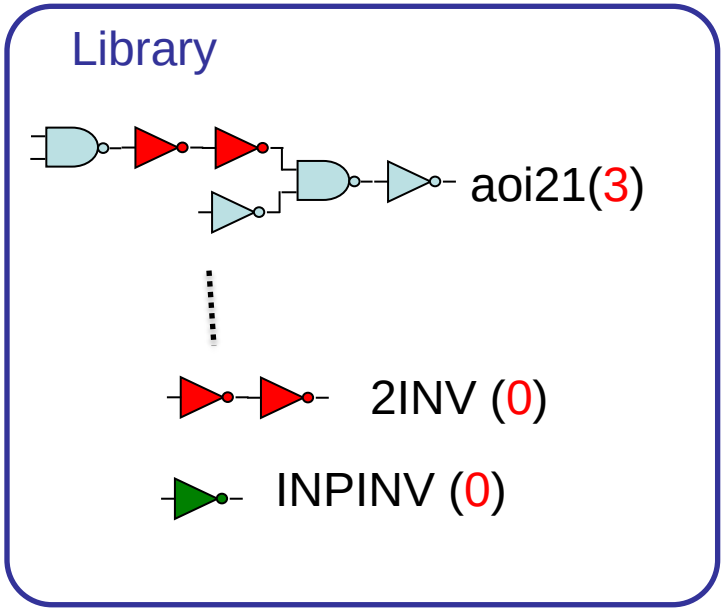
Inverter Heuristic

- Add "dummy" inverters to the subject tree
  - At every internal wire that connects two NAND2 nodes, add two back-to-back inverters
    - Does not change functionality of circuit
  - At every primary input, add a special "zero cost" single inverter
    - Equivalent to using complements of primary inputs



Inverter Heuristic

- Corresponding change to the library
  - For each pattern, at every internal wire that connects two NAND2 nodes, add two back-to-back inverters
  - Add back-to-back inverter pattern to the library with zero cost
  - Add a special "zero-cost" inverter to the library that only matches the inverters added at the primary inputs of the subject tree



Inverter Heuristic

- Added inverters enable additional matches to be considered
  - Extra inverters are absorbed by matching the zero-cost patterns in the library
  - Zero-cost matches be eliminated from the final mapped circuit