COE 561
Digital System Design & Synthesis
Library Binding

Dr. Aiman H. El-Maleh

Computer Engineering Department

King Fahd University of Petroleum & Minerals

[Adapted from slides of Prof. G. De Micheli: Synthesis & Optimization of Digital Circuits]

Outline

- Modeling and problem analysis
- Rule-based systems for library binding
- Heuristic Algorithms for library binding
- Decomposition and partitioning
- Structural matching/covering
- Tree-based matching
- Tree-based covering
- Boolean matching/covering

Library Binding

- **Given an unbound logic network and a set of library cells**
  - Transform into an interconnection of instances of library cells.
  - Optimize area, (under delay constraints.)
  - Optimize delay, (under area constraints.)
  - Optimize power, (under delay constraints.)
- **Called also *technology mapping***
  - Method used for re-designing circuits in different technologies.

Library Models

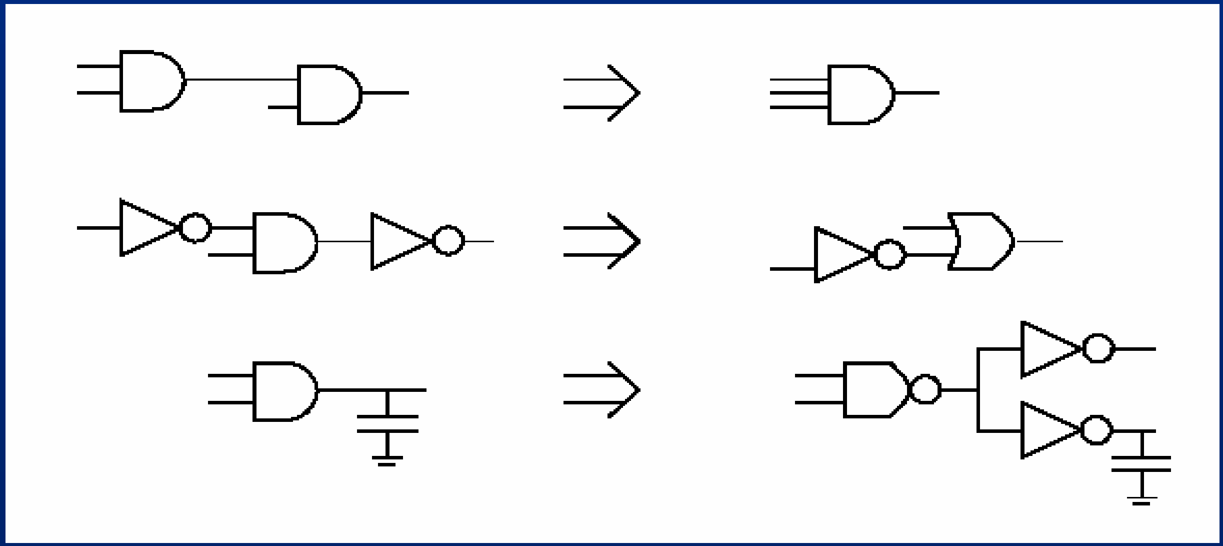
- **A cell library is a set of primitive gates including combinational, sequential, and interface elements.**
- **Each cell is characterized by**
  - Its function.
  - Input/output terminals.
  - Area, delay, capacitive load.
- **Combinational elements**
  - Single-output functions: e.g. AND, OR, NAND, NOR, INV, XOR, XNOR, AOI.
  - Compound cells: e.g. adders, encoders.
- **Sequential elements**
  - Flip-flops, registers, counters.
- **Miscellaneous**
  - Tri-state drivers.
  - Schmitt triggers.

Major Approaches

- **Rule-based systems**
  - Mimic designer activity.
  - Handle all types of cells including multiple-output, sequential and interface elements.
  - Requires creation and maintenance of set or rules.
  - Slower execution.
- **Heuristic algorithms**
  - Restricted to single-output combinational cells.
  - Implementation of registers, input/output circuits and drivers straightforward.
- **Most tools use a combination of both.**

Rule-Based Library Binding

- **Binding by stepwise transformations.**
- **Data-base**
  - Set of patterns associated with best implementation.
- **Rules**
  - Select subnetwork to be mapped.
  - Handle high-fanout problems, buffering, etc.



Rule-Based Library Binding

- **Execution of rules follows a priority scheme.**
- **Search for a sequence of transformations.**
- **A greedy search is a sequence of rules each decreasing cost.**
- **Search space**
  - Breadth (options at each step).
  - Depth (look-ahead).
- **Meta-rules determine dynamically breadth and depth.**
- **Advantages**
  - Applicable to all kinds of libraries.
- **Disadvantages**
  - Large rule data-base
    - Completeness issue.
  - Data-base updates.

Algorithms for Library Binding

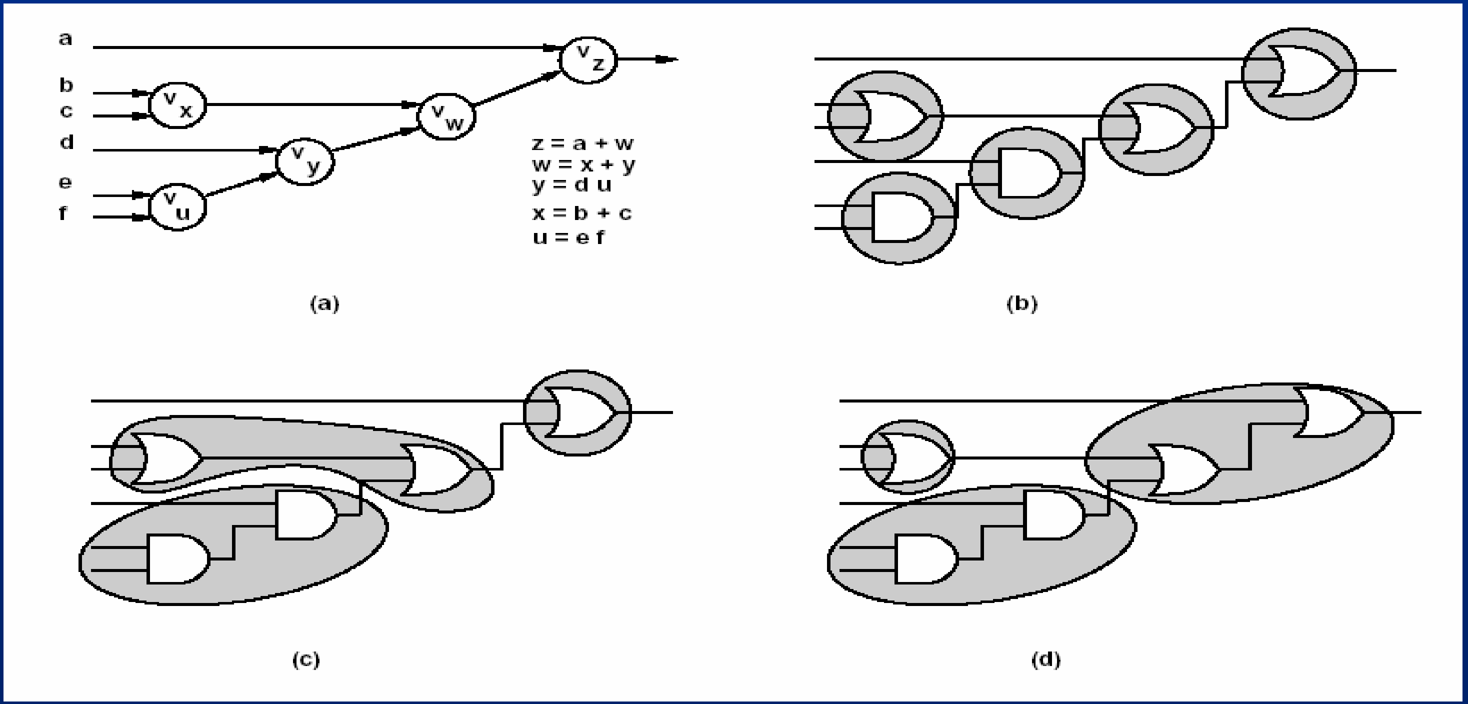
- **Mainly for single-output combinational cells.**
- **Fast and efficient**
  - Quality comparable to rule-based systems.
- **Library description/update is simple**
  - Each cell modeled by its function or equivalent pattern.
- **Involves two steps**
  - Matching
    - A cell matches a subnetwork if their terminal behavior is the same.
    - Input-variable assignment problem.
  - Covering
    - A cover of an unbound network is a partition into subnetworks which can be replaced by library cells.

Matching

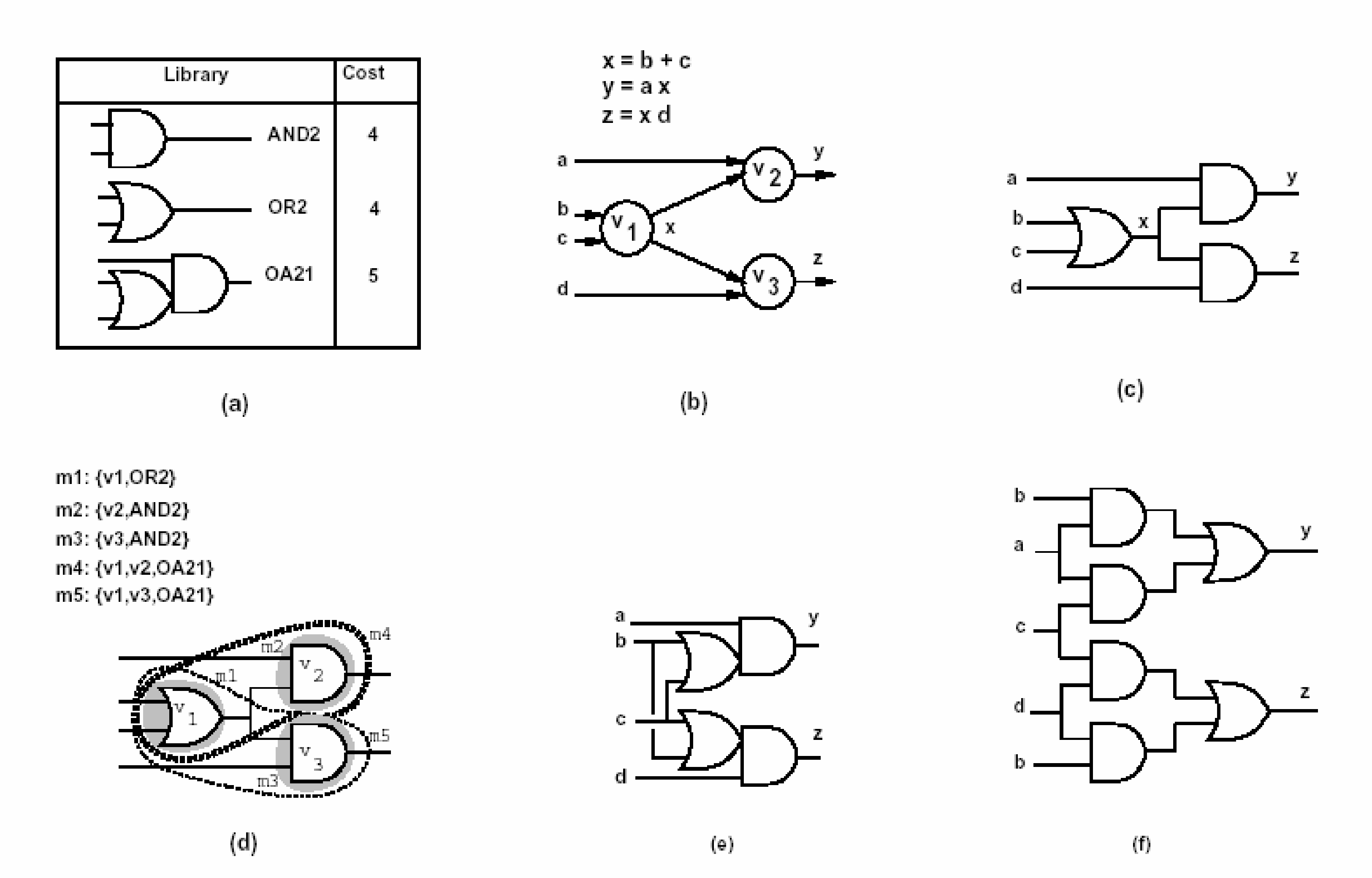
- **Given two single-output combinational functions *f(x)* and *g(x)* with same number of support variables.**
- ***f* matches *g* if there exists a permutation P such that *f(x) = g(*P *x)*.**
- **Example**
  - f = ab + c ;   g = p + qr.
  - By assigning {q, r, p} to {a, b, c}, f is equal to g.
  - f and g have a Boolean match.
  - By representing functions f and g by their AND-OR decomposition graphs, f and g have a structural match since their graphs are *isomorphic*.
- **Must ensure that vertices bound to inputs of a matched cell are outputs of other matched cells.**

Assumptions

- **Network granularity is fine.**
  - Decomposition into base functions: 2-input NAND, NOR, INV.
- **Trivial binding**
  - Replacement of each vertex by base cell.



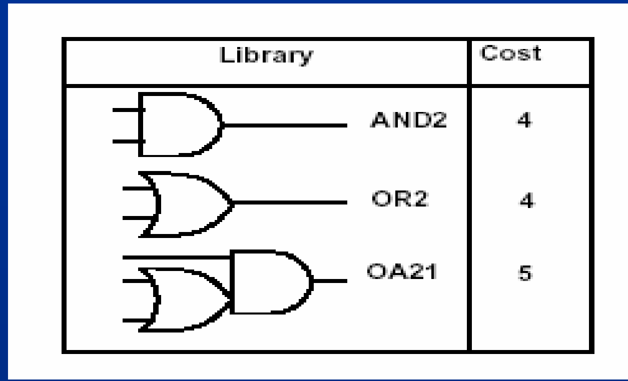
Example …



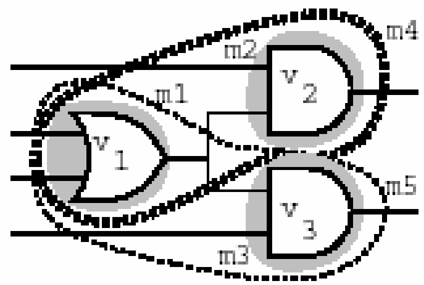
Library
Cost
AND2
4
OR2
4
OA21
5
(a)
x = b + c
y = a x
z = x d
a
b
c
d
v 1
v 2
v 3
x
y
z
(b)
a
b
c
d
x
y
z
(c)
m1: {v1,OR2}
m2: {v2,AND2}
m3: {v3,AND2}
m4: {v1,v2,OA21}
m5: {v1,v3,OA21}
m1
m2
m3
m4
m5
v 1
v 2
v 3
(d)
a
b
c
d
y
z
(e)
b
a
c
d
b
y
z
(f)

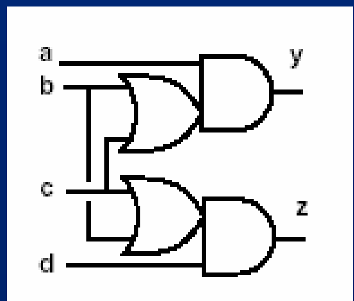
… Example

- **Vertex covering**
  - Covering v1: (m1 +m4 +m5).
  - Covering v2: (m2 +m4).
  - Covering v3: (m3 +m5).
- **Input compatibility**
  - Match m2 requires m1: (m2' +m1).
  - Match m3 requires m1: (m3' +m1).
- **Overall binate clause**
  - (m1 +m4 +m5)(m2 +m4)(m3 +m5)(m2' +m1)(m3' +m1) = 1
- **Optimum solution: m1'm2'm3'm4m5**
  - Cost=10



m1: {v1,OR2}
m2: {v2,AND2}
m3: {v3,AND2}
m4: {v1,v2,OA21}
m5: {v1,v3,OA21}





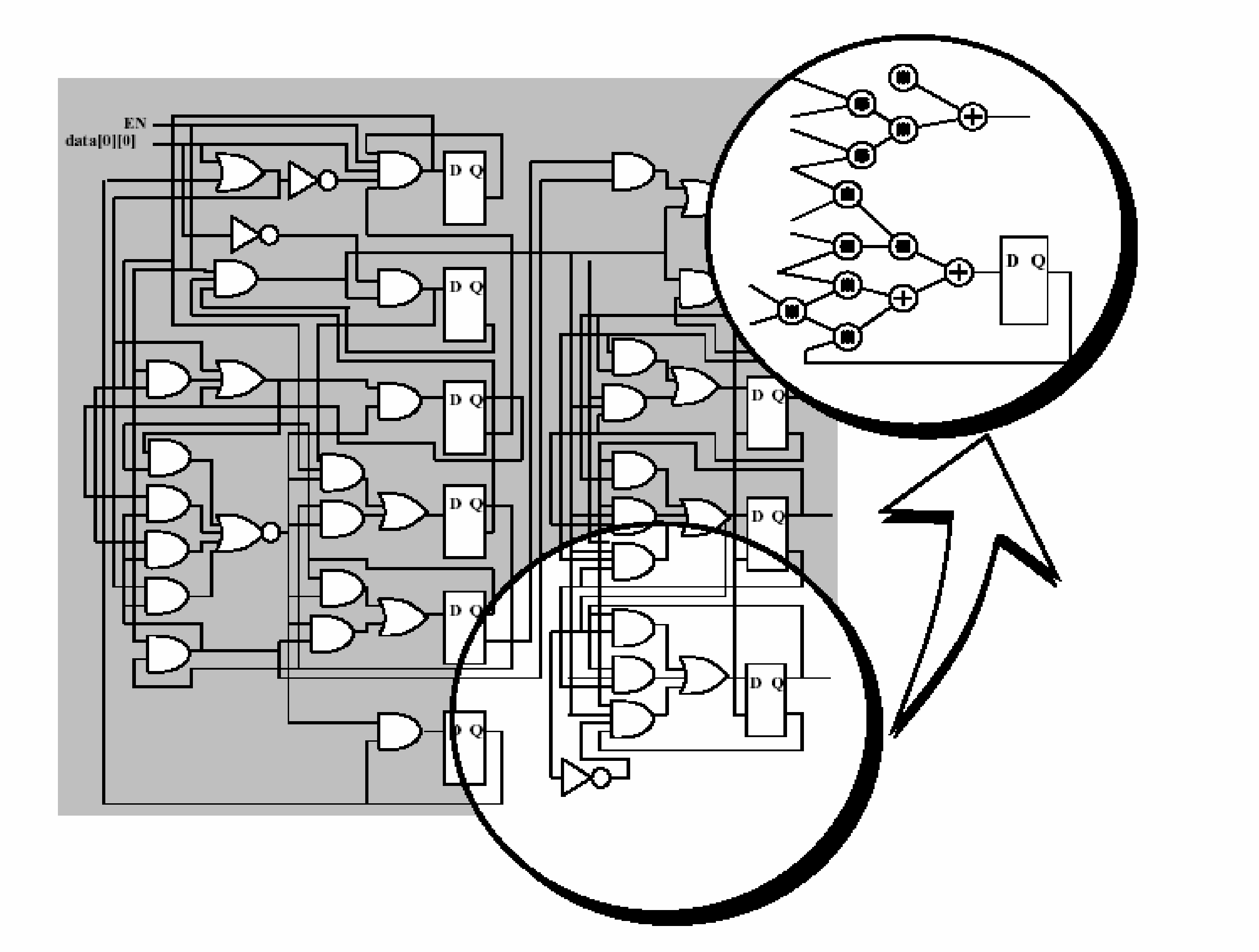
Heuristic Algorithms

- **To render covering problem tractable, network is decomposed and partitioned.**
- **Decomposition**
  - Cast network and library in standard form.
  - Decompose into base functions.
  - Example: NAND2 and INV.
  - Guarantees that each vertex is covered by at least one match.
- **Partitioning**
  - Break network into cones called *subject graphs*.
  - Reduce to many multi-input single-output subnetworks.
- **Covering**
  - Cover each subnetwork by library cells.

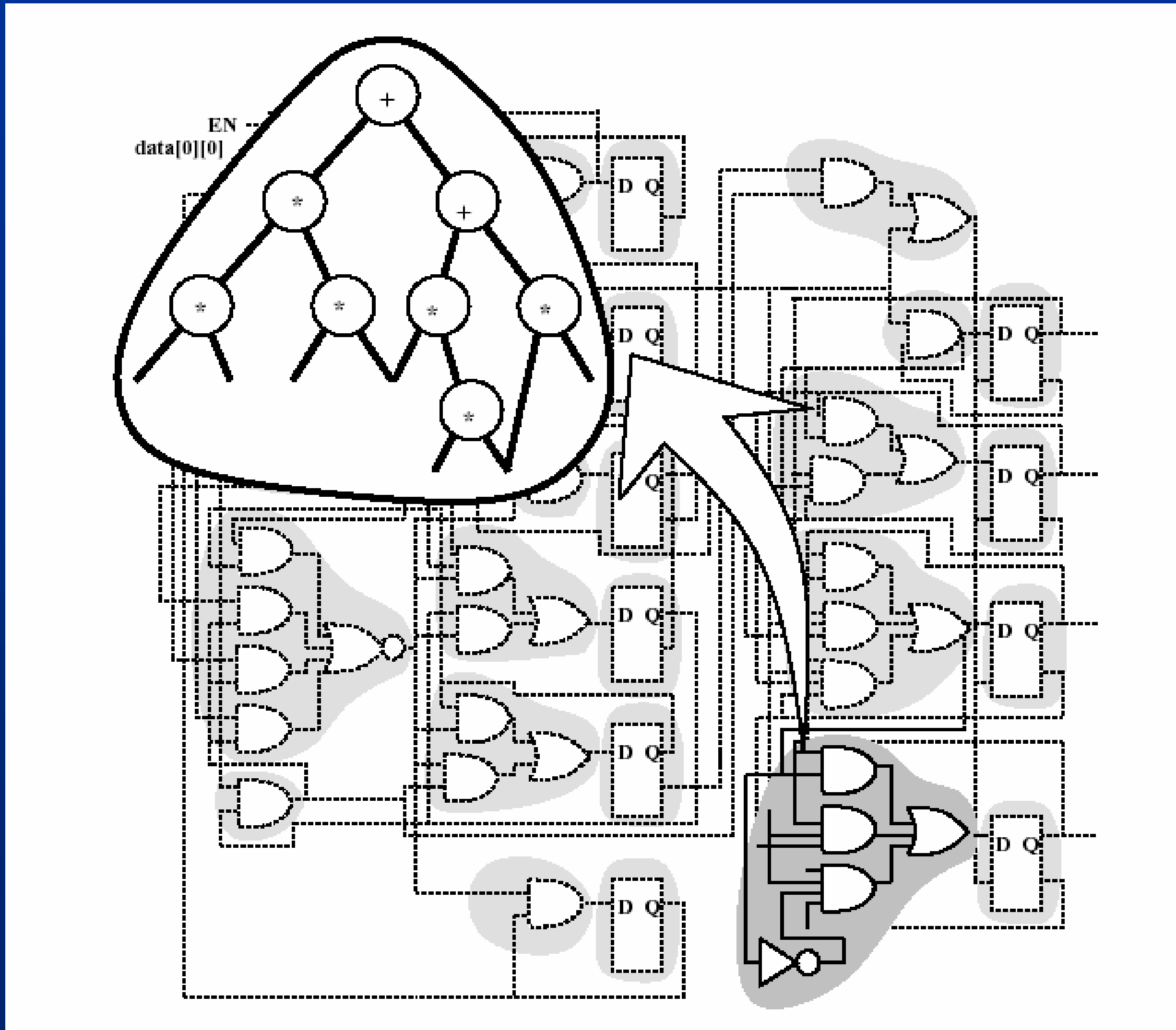
Partitioning

- **Rationale for partitioning**
  - Size of each covering problem is smaller.
  - Covering problem becomes tractable.
- **Used to isolate combinational portions from sequential elements and I/Os.**
- **Partitioning of combinational circuits**
  - Mark vertices with multiple fanout.
  - Edges whose tails are marked vertices define partition boundary.

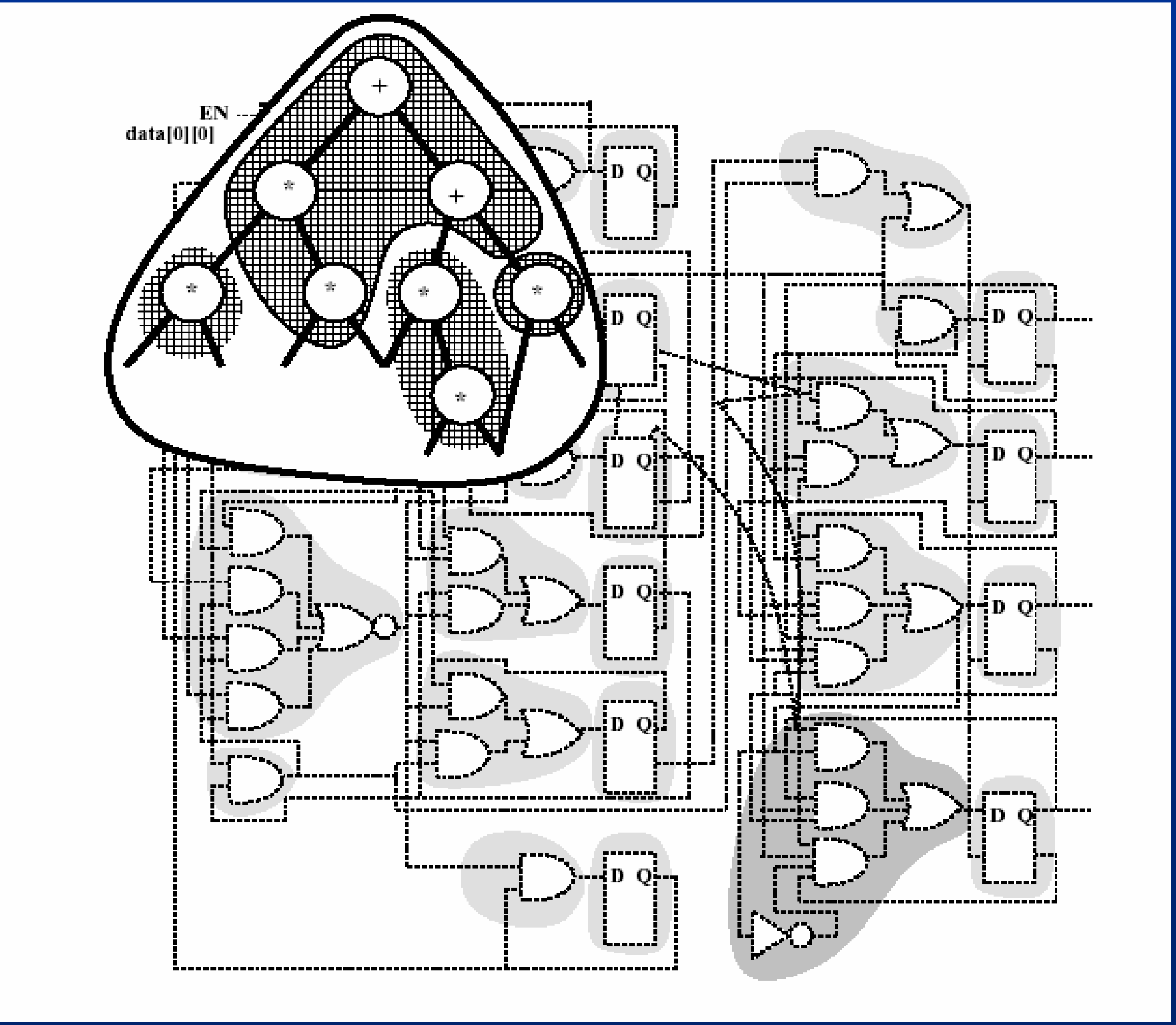
Decomposition



Partitioning



Covering

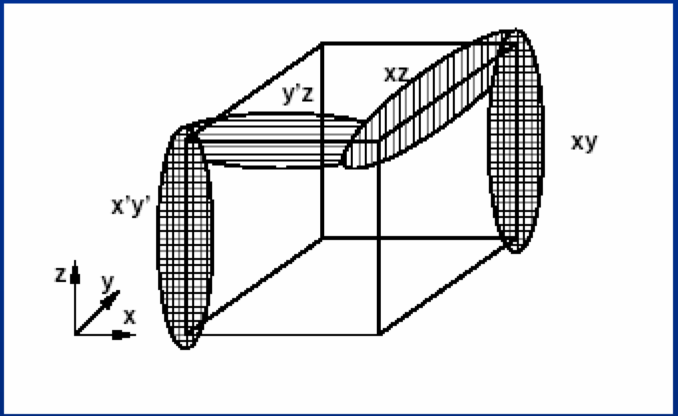


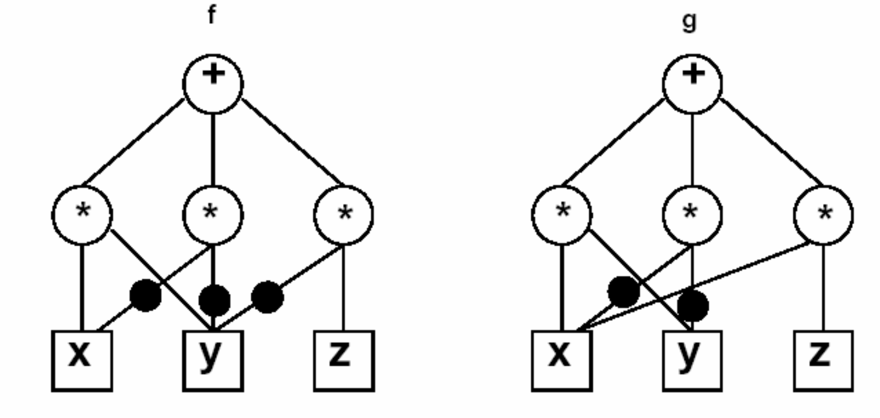
Matching

- **Structural matching**
  - Model functions by patterns.
    - Example: trees, dags (fanout only at the inputs).
  - Both subject graph and library cells cast into comparable form (subject and pattern graphs).
  - Rely on pattern matching techniques.
  - Some library cells may have more than one pattern graph.
- **Boolean matching**
  - Use Boolean models.
  - Solve *tautology* problem to check equivalence of two functions.
  - More powerful.

Boolean versus Structural Matching

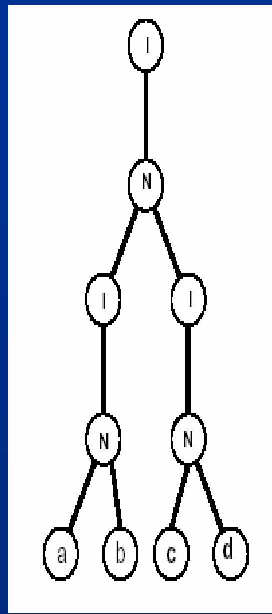
- **Example**
  - f = xy +x'y' +y'z
  - g = xy +x'y' +xz
- **Function equality is a tautology**
  - Boolean match.
- **Patterns are different**
  - No structural match.



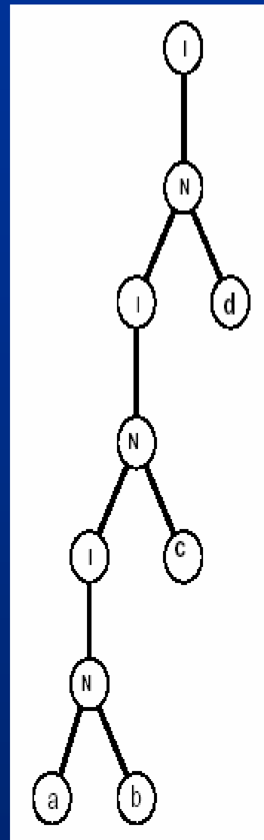


Structural Matching and Covering

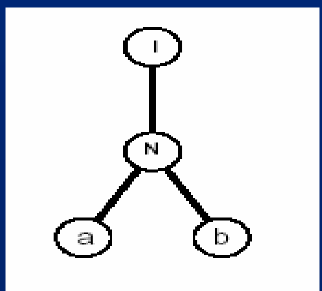
- **Expression patterns**
  - Represented by dags using a decomposition of 2-inp NAND and INV.
- **Identify pattern dags in network**
  - Matching by sub-graph isomorphism.
- **Simplification**
  - Use tree patterns.
  - Most library cells can be represented as trees.
  - Tree matching & tree covering is linear.



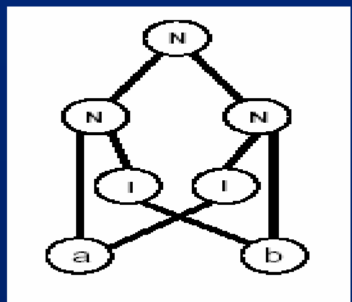
F = a b c d



F = a b c d



F = a b

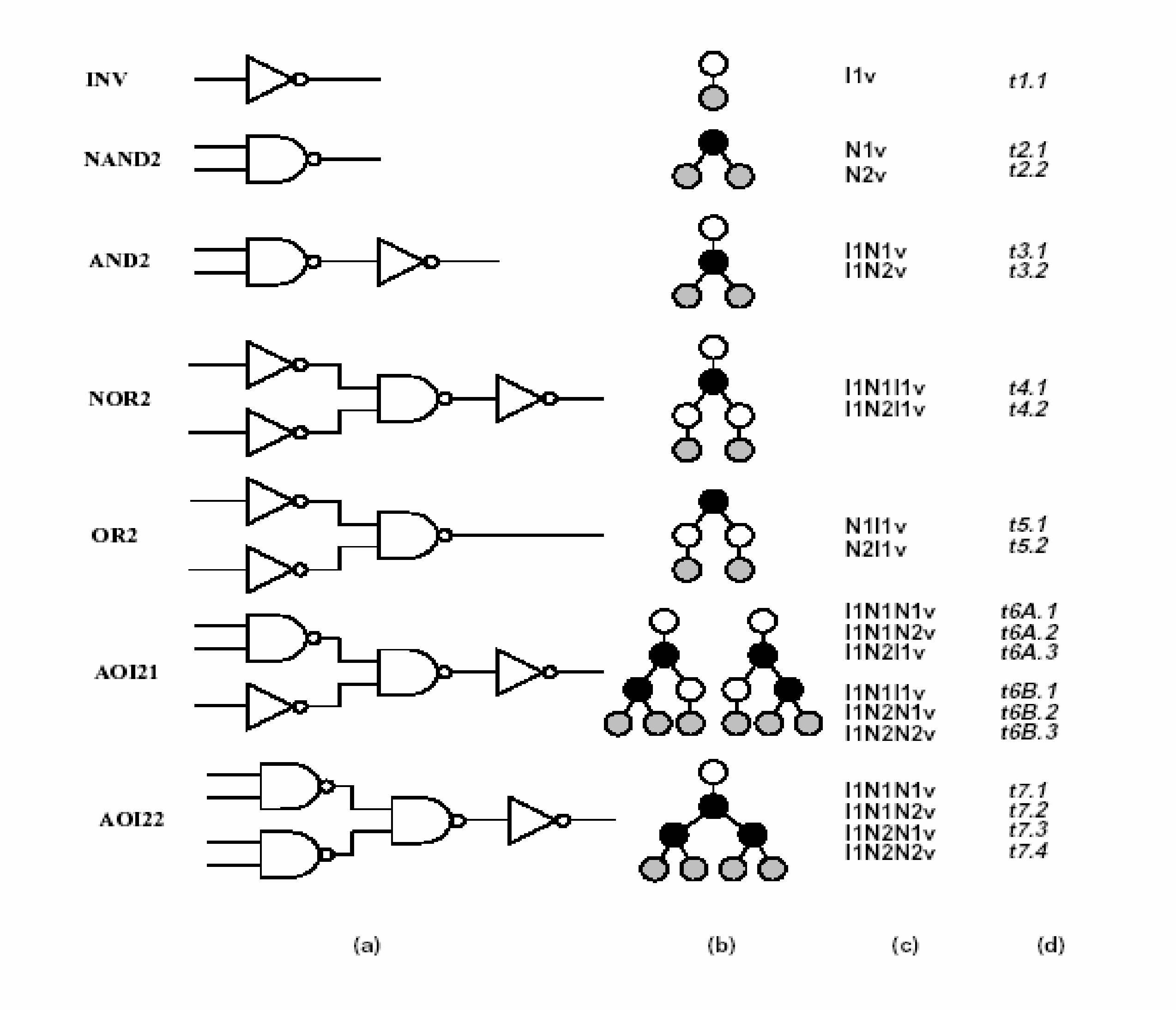


F = a $\oplus$ b

Tree-Based Matching …

- **Network**
  - Partitioned and decomposed
    - NOR2 (or NAND2) + INV.
    - Generic base functions.
  - Each partition called Subject tree.
- **Library**
  - Represented by trees.
  - Possibly more than one tree per cell.
  - Pattern recognition
    - Simple binary tree match.
    - Aho-Corasick automaton.

Simple Library



… Tree-Based Matching

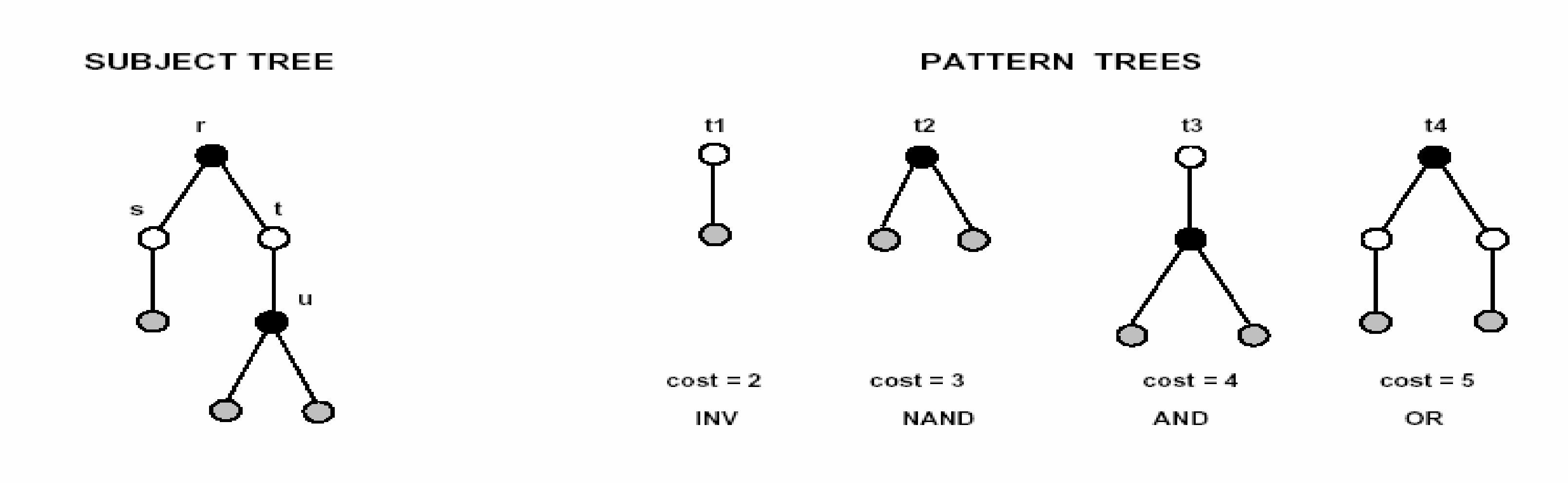
```
MATCH(u, v) {
    if (u is a leaf) return (TRUE);                    /* Leaf of the pattern graph reached */
    else {
        if (v is a leaf) return (FALSE);               /* Leaf of the subject graph reached */
        if (degree(v) ≠ degree(u)) return(FALSE);      /* Degree mismatch */
        if (degree(v) == 1) {                          /* One child each: visit subtree recursively */
            u_c = child of u ; v_c = child of v ;
            return (match(u_c, v_c) )
        }
        else {                                         /* Two children each: visit subtrees recursively */
            u_l = left-child of u ; u_r = right-child of u ;
            v_l = left-child of v ; v_r = right-child of v ;
            return (MATCH(u_l, v_l) · MATCH(u_r, v_r) + MATCH(u_r, v_l) · MATCH(u_l, v_r));
        }
    }
```

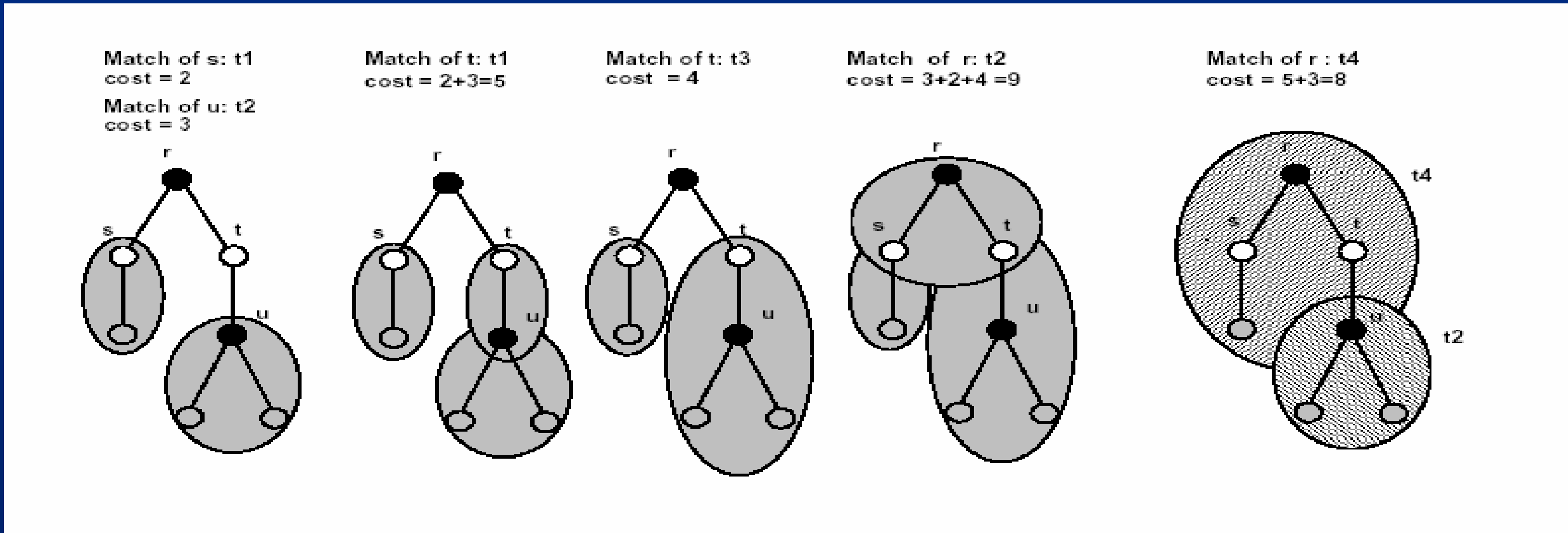
Tree-Based Covering

- **Dynamic programming**
  - Visit subject tree bottom-up.
- **At each vertex attempt to match**
  - Locally rooted subtree.
  - Check all library cells for a match.
- **Optimum solution for the subtree.**

$TREE\_COVER(T(V,E))$ {
  Set the cost of the internal vertices to -1;
  Set the cost of the leaf vertices to 0;
  while (some vertex has negative weight) **do** {
    Select a vertex $v \in V$ whose children have all nonnegative cost;
    M = set of all matching pattern trees at vertex v;
    $\text{cost}(v) = \min_{m \in M(v)} \left(cost(m) + \sum_{u \in L(m)} cost(u) \right);$
  }
}

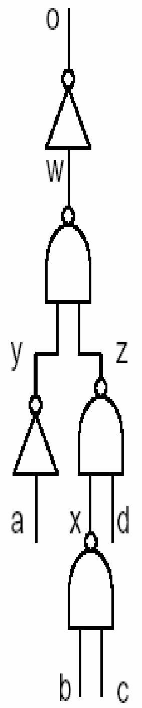
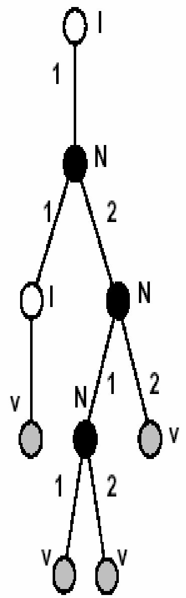
Example


SUBJECT TREE
r
s
t
u
PATTERN TREES
t1
t2
t3
t4
cost = 2
INV
cost = 3
NAND
cost = 4
AND
cost = 5
OR


Match of s: t1
cost = 2
Match of u: t2
cost = 3
Match of t: t1
cost = 2+3=5
Match of t: t3
cost = 4
Match of r: t2
cost = 3+2+4 =9
Match of r : t4
cost = 5+3=8
t4
t2

Minimum Area Cover Example

- **Minimum-area cover.**
- **Area costs**
  - INV:2; NAND2:3; AND2:4; AOI21:6.
- **Best choice**
  - AOI21 fed by a NAND2 gate.

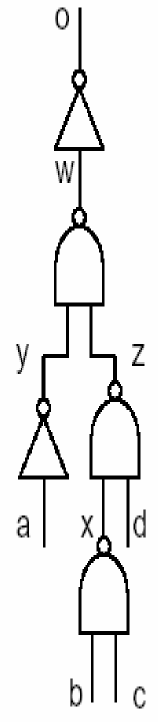
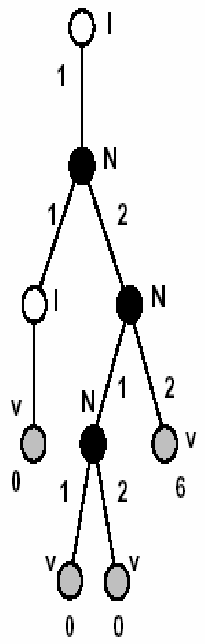
Network	Subject graph	Vertex	Match	Gate	Cost
		x	t2	NAND2(b,c)	3
		y	t1	INV(a)	2
		z	t2	NAND2(x,d)	2* 3 = 6
		w	t2	NAND2(y,z)	3 * 3 + 2 = 11
		o	t1	INV(w)	3 * 3 + 2 * 2 = 13
			t3	AND2(y,z)	2 * 3 + 4 + 2 = 12
			t6B	AOI21(x,d,a)	3 + 6 = $\boxed{9}$

Minimum Delay Cover

- **Dynamic programming approach.**
- **Cost related to gate delay.**
- **Delay modeling**
  - Constant gate delay: straightforward.
  - Load-dependent delay
    - Load fanout unknown.
- **Minimum delay cover with constant delays**
  - The cell tree is isomorphic to a subtree with leaves L.
    - The vertex is labeled with the cell cost plus the maximum of the labels of L.

Minimum Delay Cover Example

- Inputs data-ready times are 0 except for $t_d = 6$
- Constant delays
  - **INV:2; NAND2:4; AND2:5; AOI21:10.**
- Compute data-ready times bottom-up
  - $t_x = 4$; $t_y = 2$; $t_z = 10$; $t_w = 14$.
- Best choice
  - **AND2, two NAND2 and an INV gate.**

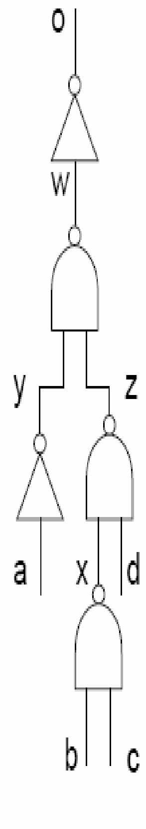
Network	Subject graph	Vertex	Match	Gate	Cost
		x	t2	NAND2(b,c)	4
		y	t1	INV(a)	2
		z	t2	NAND2(x,d)	6 + 4 = 10
		w	t2	NAND2(y,z)	10 + 4 = 14
		o	t1	INV(w)	14 + 2 = 16
			t3	AND2(y,z)	10 + 5 = $\boxed{15}$
			t6B	AOI21(x,d,a)	10 + 6 = 16

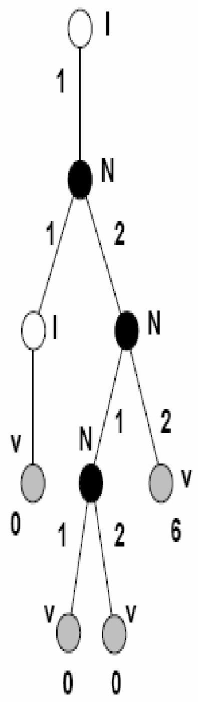
Minimum Delay Cover
Load-Dependent Delays

- **Model**
  - For most libraries, input capacitances are a finite small set.
  - Label each vertex with all possible load values.
- **Dynamic programming approach**
  - Compute an array of solutions for each vertex corresponding to different loads.
  - For each match, arrival time is computed for each load value.
  - For each input to a matching cell the best match for the given load is selected.
- **Optimum solution, when all possible loads are considered.**

Example

- **Inputs data-ready times are 0 except for $t_d = 6$**
- **Load-dependent delays**
  - INV:1+l; NAND2:3+l; AND2:4+l; AOI21:9+l; SINV:1+0.5l.
- **Loads**
  - INV:1; NAND2:1; AND2:1; AOI21:1; SINV:2.
- **Assume output load is 1**
  - Same solution as before.
- **Assume output load is 5**
  - Solution uses SINV cell.

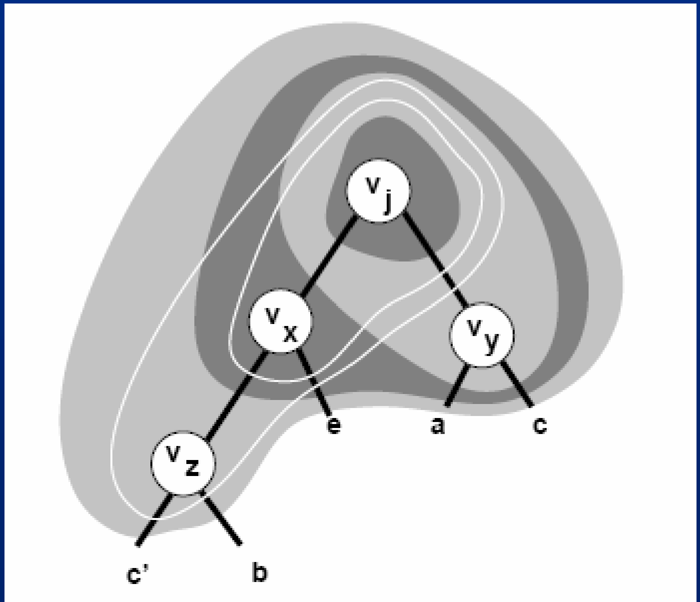




Network	Subject graph	Vertex	Match	Gate	Cost Load=1	Cost Load=2	Cost Load=5
		x	t2	NAND2(b,c)	4	5	8
		y	t1	INV(a)	2	3	6
		z	t2	NAND2(x,d)	10	11	14
		w	t2	NAND2(y,z)	14	15	18
		o	t1	INV(w)			20
			t3	AND2(y,z)			19
			t6B	AOI21(x,d,a)			20
				SINV(w)			$\boxed{18.5}$

Boolean Matching/Covering

- **Decompose network into base functions.**
- **When considering vertex v_i**
  - Construct clusters by local elimination.
  - Several functions associated with v_i.
- **Limit size and depth of clusters.**



$j = x\,y;$ $x = e + z;$
$y = a + c;$ $z = c' + b;$

$f_{j,1} = x\,y;$
$f_{j,2} = x\,(a + c);$
$f_{j,3} = (e + z)\,y;$
$f_{j,4} = (e + z)\,(a + c)$
$f_{j,5} = (e + c' + b)\,y;$
$f_{j,6} = (e + c' + b)\,(a + c);$

Boolean Matching: P-Equivalence

- **Cluster function *f(x):* sub-network behavior.**
- **Pattern function *g(y):* cell behavior.**
- **P-equivalence**
  - Exists a permutation operator P, such that $f(x) = g(P\ x)$ is a tautology?
- **Approaches**
  - Tautology check over all input permutations.
  - Multi-rooted pattern ROBDD capturing all permutations.

Signatures and Filters …

- **Drastically reduce the number of permutations to be considered.**
- **Capture some properties of Boolean functions.**
- **If signatures do not match, there is no match.**
- **Used as filters to reduce computation.**
- **Signatures**
  - Unateness.
  - Symmetries.
- **Any pin assignment must associate**
  - unate (binate) variables in f(x) with unate (binate) variables in g(y).
- **Variables or groups of variables**
  - that are interchangeable in f(x) must be interchangeable in g(y).

… Signatures and Filters …

- **Cluster and pattern functions must have the same number of unate and binate variables to match.**
- **If there are *b* binate variables, un upper bound on number of variable permutations is b! (n-b)!**
  - (instead of n!)
- **Example**
  - g = s1 s2 a + s1 s2’ b + s1’ s3 c + s1’s3’ d.
  - n=7 variables; 4 unate and 3 binate.
  - Only 3! 4! = 144 variable orders and corresponding OBDDs. need to be considered in worst case.
  - Compare this with overall number of permutations
    - 7!=5040

… Signatures and Filters …

- A **symmetry set** is a set of variables that are pairwise interchangeable.
- A **symmetry class** is an ensemble of symmetry sets with the same cardinality.
- A **symmetry class** C_i has symmetry sets with cardinality i.
- A necessary condition for two functions to match is having symmetry classes of the same cardinality for each i.
- **Example**
  - F = x1 x2 x3 + x4 x5 + x6 x7
  - Symmetry sets: (x1, x2, x3), (x4, x5), (x6, x7)
  - C_2= {(x4, x5), (x6, x7)}; $|C_2|$=2
  - C_3= {(x1, x2, x3)}; $|C_3|$=1

… Signatures and Filters …

- **Symmetry classes can be used to determine non-redundant variable orders**
  - All variables in a given symmetry set are equivalent.
  - Number of permutations required is $\prod_{i=1 \text{ to } n} (|C_i|!)$.
- **Example**
  - F = x1 x2 x3 + x4 x5 + x6 x7
  - Number of permutations = 2! = 2 variable orders.
  - (x1, x2, x3, x4, x5, x6, x7)
  - (x1, x2, x3, x6, x7, x4, x5)
- **Cluster function: f = abc.**
  - Symmetries: {(a, b, c)} – 3 unate.
- **Pattern functions**
  - g1 = a+b+c
    - Symmetries: {(a, b, c)} – 3 unate.
  - g2 = ab+c
    - Symmetries: {(a, b) (c)} -- 3 unate.
  - g3 = abc +a'b'c'
    - Symmetries: {(a, b, c)} -- 3 binate.

… Signatures and Filters

- Taking advantage of both symmetric classes and unate-binate properties

$$Let\ C_i = C_i^b + C_i^u$$

$$\left|C_i\right| = \left|C_i^b\right| + \left|C_i^u\right|$$

- Number of non-redundant permutations

$$\prod_{i=1}^{n} \left|C_i^b\right|! * \left|C_i^u\right|!$$