Technology Mapping

Example:

```
t1 = a + b c;
t2 = d + e;
t3 = a b + d;
t4 = t1 t2 + f g;
t5 = t4 h + t2 t3;
F = t5';
```

This shows an unoptimized set of logic equations consisting of sixteen literals.

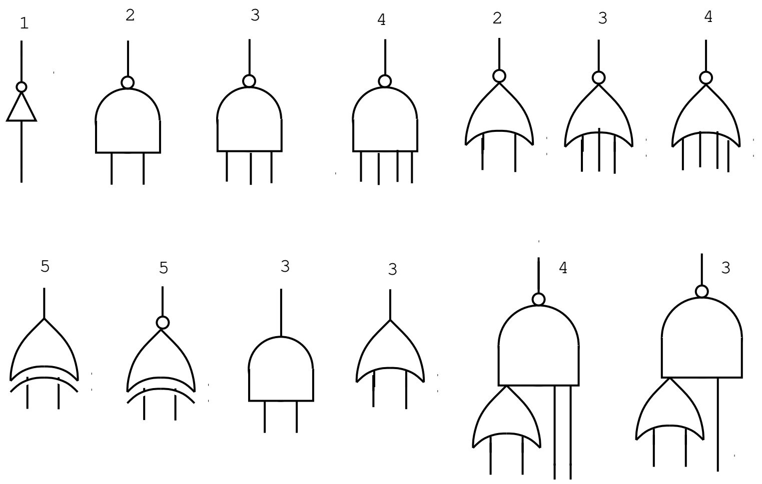
Optimized Equations

- Using technology-independent optimization, these equations are optimized using only fourteen literals:

```
t1 = d + e;
t2 = b + h;
t3 = a t2 + c;
t4 = t1 t3 + f g h;
F = t4';
```

- Implement this network using a set of gates which form a *library*. Each gate has a cost (its area, delay etc.).

Part of a Cell Library



Technology Mapping

Two approaches

1. Algorithmic [DAGON, MISII]

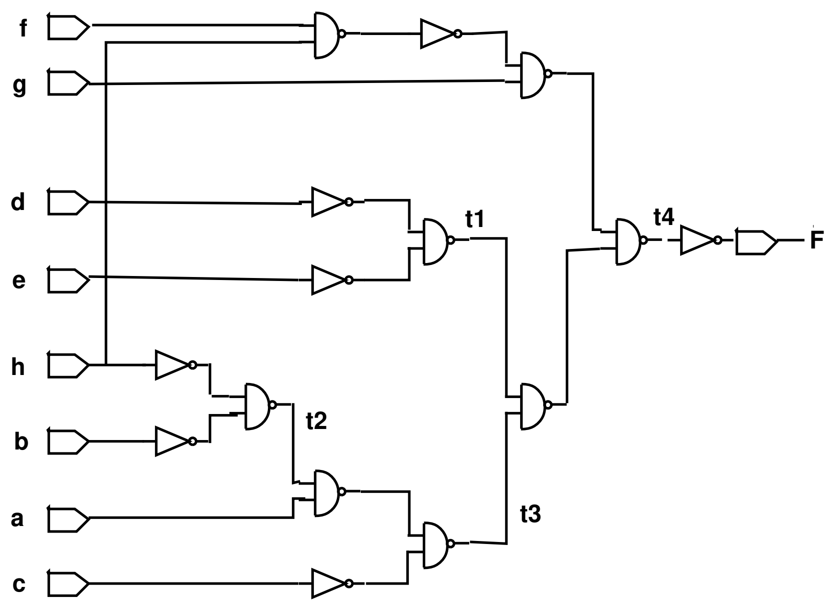
2. Rule-based [LSS]

Algorithmic Approach

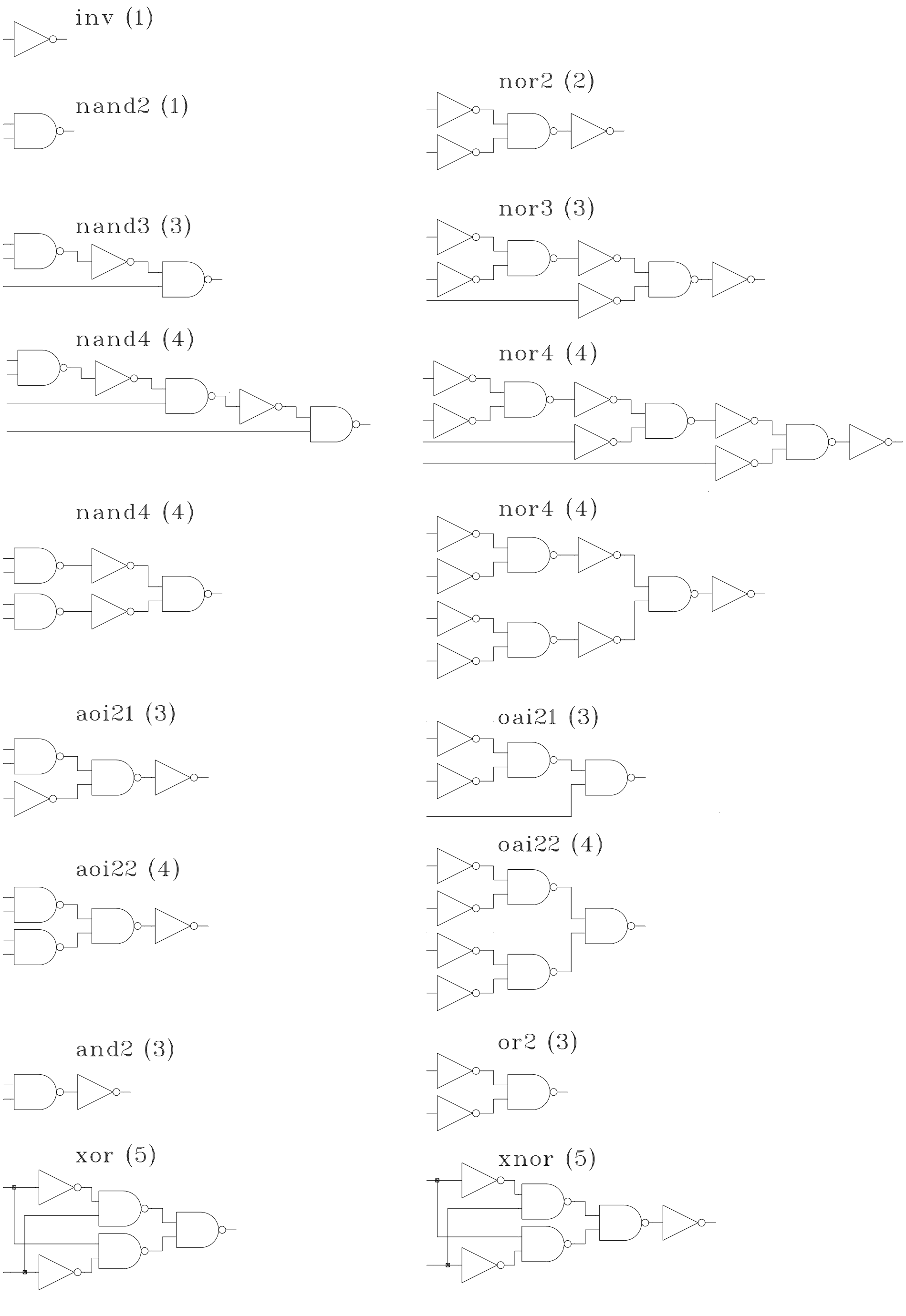
- Represent each function of the network using a set of *base functions*. This representation is called the *subject graph*.
  - Typically the base set is 2-input NANDs and inverters [MISII]. Generic nodes.
  - The set should be functionally complete.
- Each gate of the library is likewise represented using the base set. This generates *pattern graphs*.
  - Represent each gate in all possible ways.
- A *cover* is a collection of pattern graphs such that every node of the subject graph is contained in one (or more) of the pattern graphs. The cover is further constrained so that each input required by a pattern graph is actually an output of some other pattern graph.
- For minimum area, cost of cover is the sum of areas of gates in the cover.
- *The technology mapping problem is the optimization problem of finding a minimum cost covering of the subject graph by choosing from the collection of pattern graphs for all gates in the library.*

Subject Graph

```
t1 = d + e;
t2 = b + h;
t3 = a t2 + c;
t4 = t1 t3 + f g h;
F = t4';
```

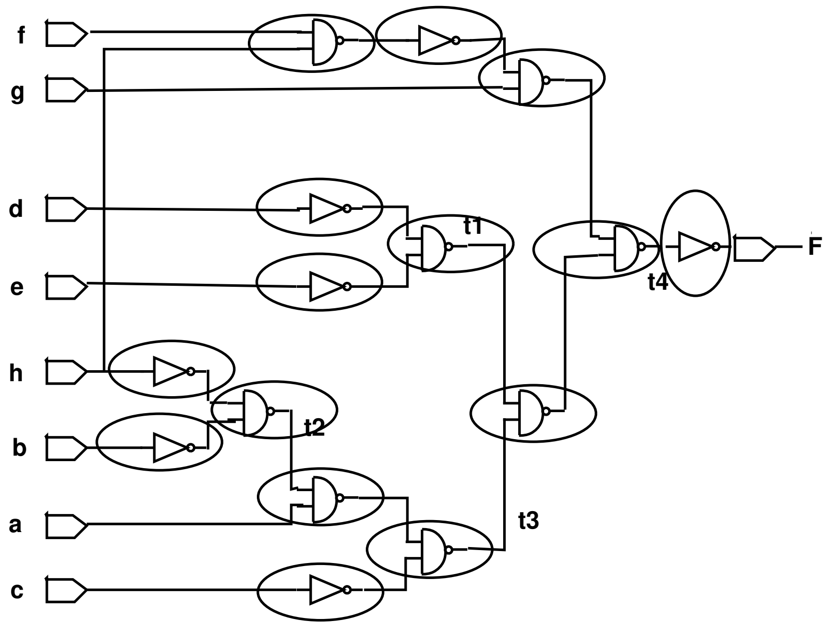


Pattern Graphs for the IWLS Library



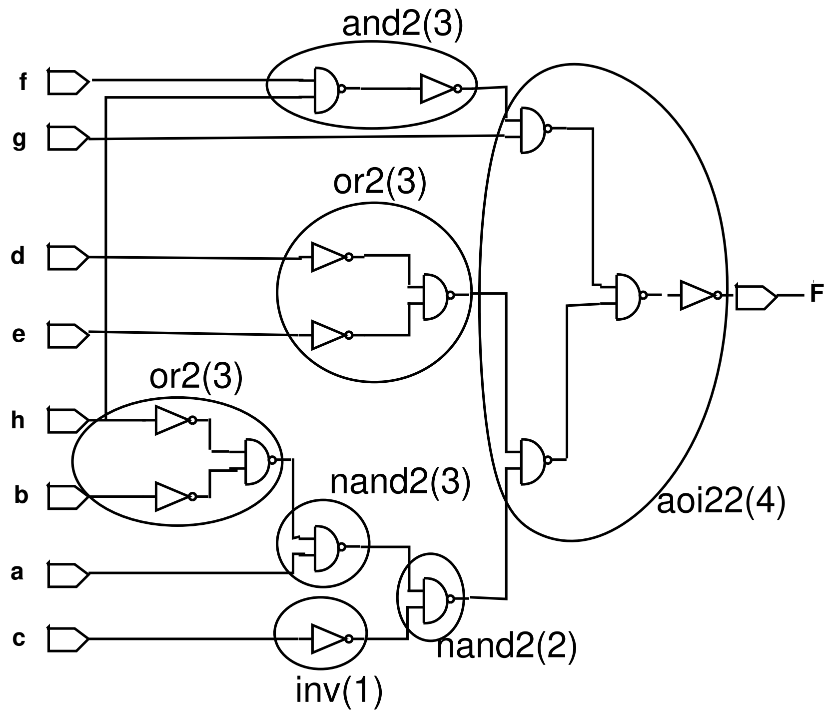
Trivial Covering

```
t1 = d + e;
t2 = b + h;
t3 = a t2 + c;
t4 = t1 t3 + f g h;
F = t4';
```



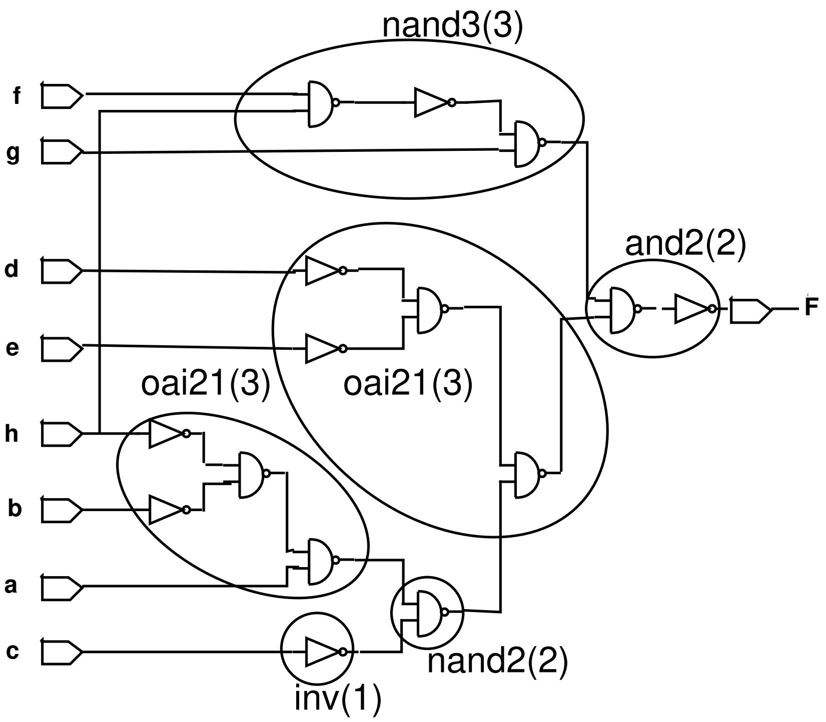
8 two-input NAND-gates and 7 inverters for an area cost of 23.

Better Covering



A better covering with an area of 19.

Alternate Cover



An alternate covering with an area of 14.

Technology Mapping using DAG covering

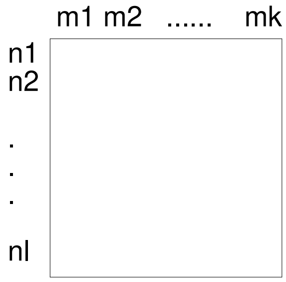
- Input:
  - Technology independent, optimized logic network.
  - Description of gates in library, with their costs.
- Output:
  - Netlist of gates(from library) which minimizes total cost.
- General Approach:
  - Construct a subject DAG for network.
  - Represent each gate in target library by pattern DAG's.
  - Find an optimal-cost covering of subject DAG using the collection of pattern DAG's.

Technology Mapping using DAG covering

- Complexity of DAG covering:
  - NP-hard
  - Remains NP-hard even when the nodes have degree ≤ 2.
  - If subject DAG and pattern DAG's are trees, an efficient algorithm exists [Keutzer].

DAG covering as a binate covering problem

- Compute all possible matches $\{m_k\}$ (ellipses in previous figures).
- Using a variable m_i for each successful match of a pattern graph in the subject graph, write a clause for each node of the subject graph indicating which matches cover this node.
  - For example, if a subject node is covered by matches m_2, m_5 and m_{10}, then the clause would be $(m_2 + m_5 + m_{10})$.
- Repeat for each subject node and take the product over all subject nodes. (CNF)



- Any satisfying assignment guarantees that all subject nodes are covered, **but** does not guarantee that *another match actually creates as an output the input signal needed for a given match.* Rectify this by adding additional clauses.

DAG covering as a binate covering problem

- Let match m_i have subject nodes $s_{i_1}, \ldots, s_{i_n}$ as n inputs. If m_i chosen, one of the matches which realizes s_{i_j} must also be chosen for each input j, j not a primary input.
- Let S_{i_j} be the disjunctive expression in the variables m_k giving the possible matches which realize s_{i_j} as an output node. Selecting match m_i implies satisfying each of the expressions S_{i_j} for $j = 1 \ldots n$. This can be written as

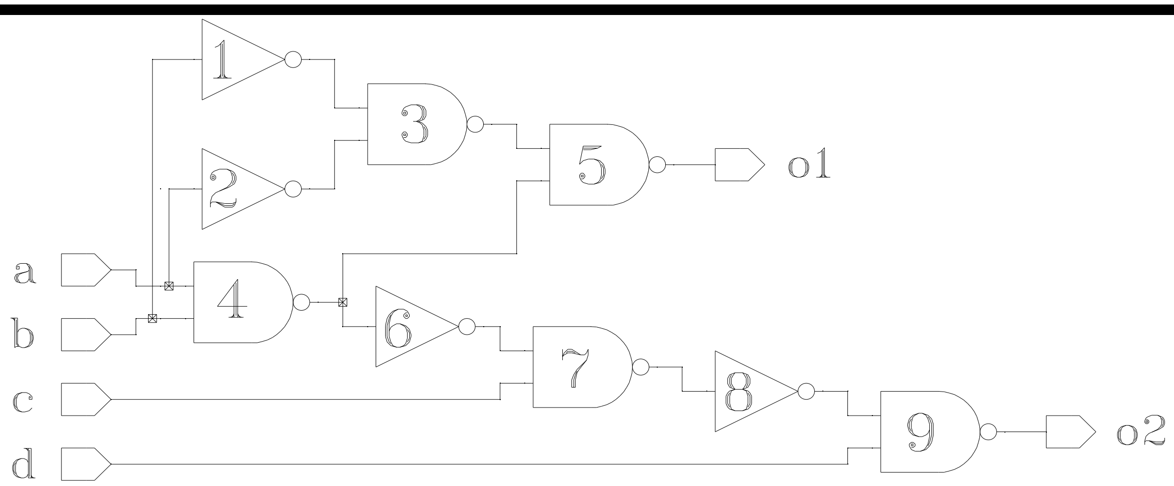
$$\begin{aligned} & m_i \Rightarrow (S_{i_1} \cdots S_{i_n}) \\ \Leftrightarrow \quad & \overline{m_i} + (S_{i_1} \cdots S_{i_n}) \\ \Leftrightarrow \quad & (\overline{m_i} + S_{i_1}) \cdots (\overline{m_i} + S_{i_n}). \end{aligned}$$

- Also, one of the matches for each primary output of the circuit must be selected.
- An assignment of values to variables m_i that satisfies the above covering expression is a legal graph cover.
- For area optimization, each match m_i has a cost c_i which is the area of the gate the match represents.
- The goal is a satisfying assignment with the least total cost.
  - Find a least cost prime: if a variable m_i occurs in complemented form in the prime, its cost is 0, else its cost is c_i.

Binate Covering

- This problem is more general than unate-covering for two-level minimization because
  - variables are present in the covering expression in both their true and complemented forms.
- The covering expression is a binate logic function, and the problem is referred to as the *binate-covering problem*.
- This problem has appeared before
  1. As state minimization of incompletely-specified finite-state machines [Grasselli-65]
  2. As a problem in the design of optimal three-level NAND-gate networks [Gimpel-67]
  3. In the optimal phase-assignment problem [Rudell-86].

Binate Covering: Example



	gate	cost	inputs	produces	covers
m_1	inv	1	b	g_1	g_1
m_2	inv	1	a	g_2	g_2
m_3	nand2	2	g_1, g_2	g_3	g_3
m_4	nand2	2	a, b	g_4	g_4
m_5	nand2	2	g_3, g_4	g_5	g_5
m_6	inv	1	g_4	g_6	g_6
m_7	nand2	2	g_6, c	g_7	g_7
m_8	inv	1	g_7	g_8	g_8
m_9	nand2	2	g_8, d	g_9	g_9
m_{10}	nand3	3	g_6, c, d	g_9	g_7, g_8, g_9
m_{11}	nand3	3	a, b, c	g_7	g_4, g_6, g_7
m_{12}	xnor2	5	a, b	g_5	g_1, g_2, g_3, g_4, g_5
m_{13}	nand4	4	a, b, c, d	g_9	g_4, g_6, g_7, g_8, g_9
m_{14}	oai21	3	a, b, g_4	g_5	g_1, g_2, g_3, g_5

Binate Covering: Example

1. Generate constraints that each node g_i be covered by some match.

$$
\begin{aligned}
&(m_1 + m_{12} + m_{14})(m_2 + m_{12} + m_{14})(m_3 + m_{12} + m_{14}) \\
&(m_4 + m_{11} + m_{12} + m_{13})(m_5 + m_{12} + m_{14}) \\
&(m_6 + m_{11} + m_{13})(m_7 + m_{10} + m_{11} + m_{13}) \\
&(m_8 + m_{10} + m_{13})(m_9 + m_{10} + m_{13}).
\end{aligned}
$$

2. To ensure that a cover leads to a valid circuit, extra clauses are generated. For example, selecting m_3 requires that a match be chosen which produces g_1 as an output, and a match be chosen which produces g_2 as an output. The only match which produces g_1 is m_1, and the only match which produces g_2 is m_2.

3. The primary output nodes g_5 and g_9 must be realized as an output of some match. The matches which realize g_5 as an output are m_5, m_{12}, and m_{14}; the matches which realize g_9 as an output are m_9, m_{10} and m_{13}.

Note: A match which requires a primary input as an input is satisfied trivially. Matches $m_1, m_2, m_4, m_{11}, m_{12}$ and m_{13} are driven only by primary inputs, and hence do not require additional clauses.

Binate Covering: Example

- Finally, we get

$$(\overline{m}_3 + m_1)(\overline{m}_3 + m_2)(\overline{m}_5 + m_3)(\overline{m}_5 + m_4)(\overline{m}_6 + m_4)$$
$$(\overline{m}_7 + m_6)(\overline{m}_8 + m_7)(\overline{m}_9 + m_8)(\overline{m}_{10} + m_6)$$
$$(\overline{m}_{14} + m_4)(m_5 + m_{12} + m_{14})(m_9 + m_{10} + m_{13})$$

- The covering expression has 58 prime implicants

- The least cost prime implicant is $\overline{m_3}\ \overline{m_5}\ \overline{m_6}\ \overline{m_7}\ \overline{m_8}\ \overline{m_9}\ \overline{m_{10}}\ m_{12}\ m_{13}\ \overline{m_{14}}$

- This uses two gates for a cost of nine gate units. This corresponds to a cover which selects matches m_{12} (xor2), and m_{13} (nand4).
  - Note that the node g_4 is covered by both matches.

Complexity of DAG-covering

- *Given a subject graph*, the binate covering provides the exact solution to the technology-mapping problem.
  - Better results may be obtained with a different initial decomposition into 2-input NANDs and inverters.
- Methods to solve the binate covering formulation:
  - Branch and bound [Thelen].
  - BDD-based [Lin and Somenzi].
- Even for moderate-size networks, difficult to solve exactly.
- More general than unate covering
  - Finding least-cost prime of a *binate* function. Even finding a feasible solution is NP-complete. For unate covering, finding a feasible solution is easy.
  - From DAG-covering viewpoint: generating covering + implication constraints.

Optimal Tree Covering by Trees

- If the subject DAG and primitive DAG's are trees, then an efficient algorithm to find the *best* cover exists
- Based on dynamic programming algorithm
- First proposed for optimal code generation in a compiler

Given:

- Subject Trees (networks to be mapped)
- Forest of patterns (gate library)

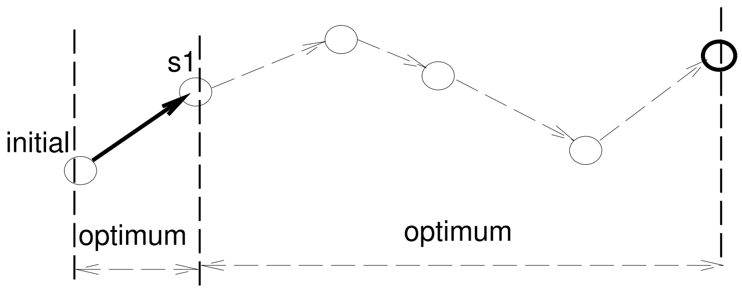
Consider a node $\mathcal{N}$ of the subject trees

- **Recursive Assumption:** For all children of $\mathcal{N}$ a best match which implements the node is known
- Cost of a leaf is 0
- Consider each pattern tree which matches at $\mathcal{N}$; compute cost as the cost of implementing each node which the pattern requires as an input **plus** the cost of the pattern
- Choose lowest cost, matching pattern to implement $\mathcal{N}$

Tree-Covering Approximation

- Partition subject graph into forest of trees.
- Cover each tree optimally using dynamic programming.

Principle of Optimality An optimum sequence of decisions has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.



Proposition: *The minimum area cover for a tree T can be derived from the minimum area covers for every node below the root of T.*

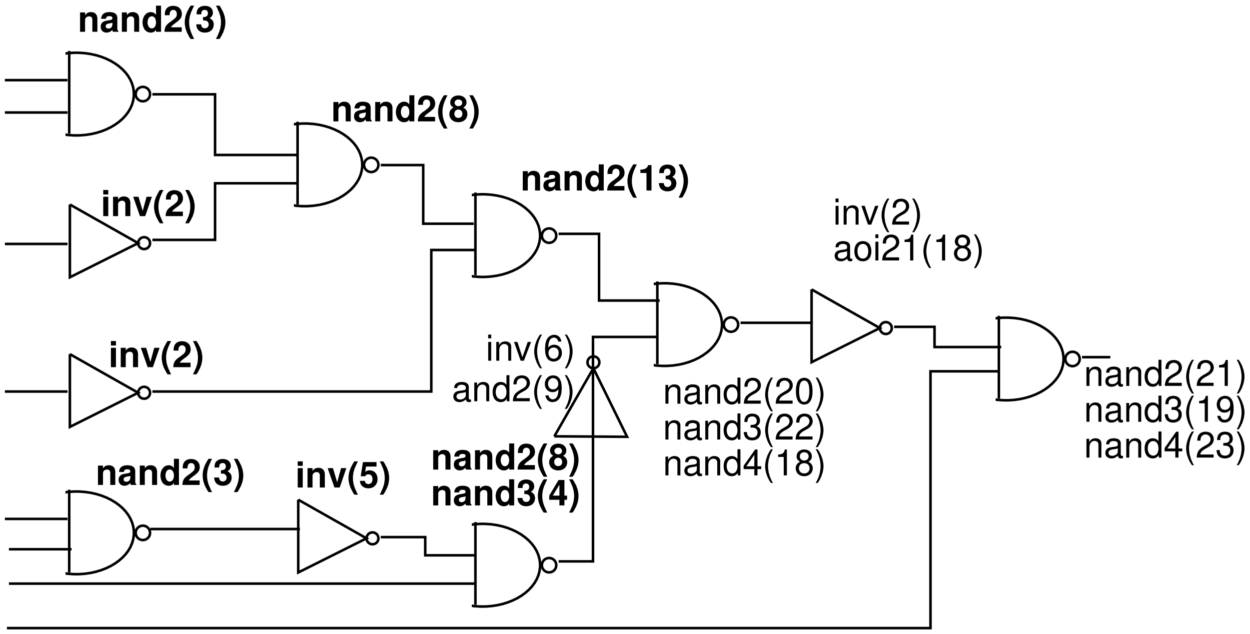
Optimum Area Algorithm

```
optimal_area_cover(node) {

      /* Find optimal cover for all nodes below "node" */
      foreach input of node {
            optimal_area_cover(input);
      }

      /* Using these, find the best cover at "node" */
      node→area = INFINITY;
      node→match = 0;
      foreach match at node {
            area = match→area;
            foreach pin of match {
                  area = area + pin→area;
            }
            if (area < node→area) {
                  node→area = area;
                  node→match = match;
            }
      }
}
```

Algorithm **optimal_area_cover**

Tree Covering in Action

nand2=3
inv=2
nand3=4
nand4=5
and2=4
aio21=4
oai21=4



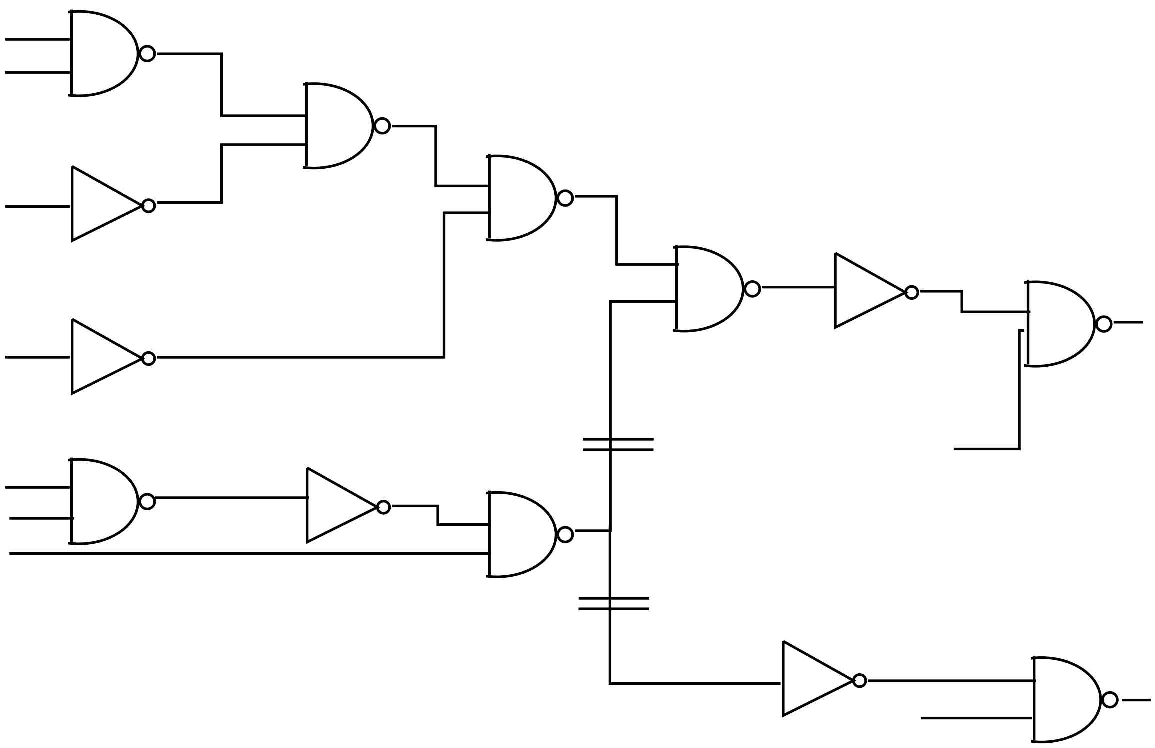
Complexity of Tree Covering

- Complexity is controlled by finding all subtrees of the subject graph which are isomorphic to a pattern tree.

- Linear complexity in both size of subject tree and size of collection of pattern trees.

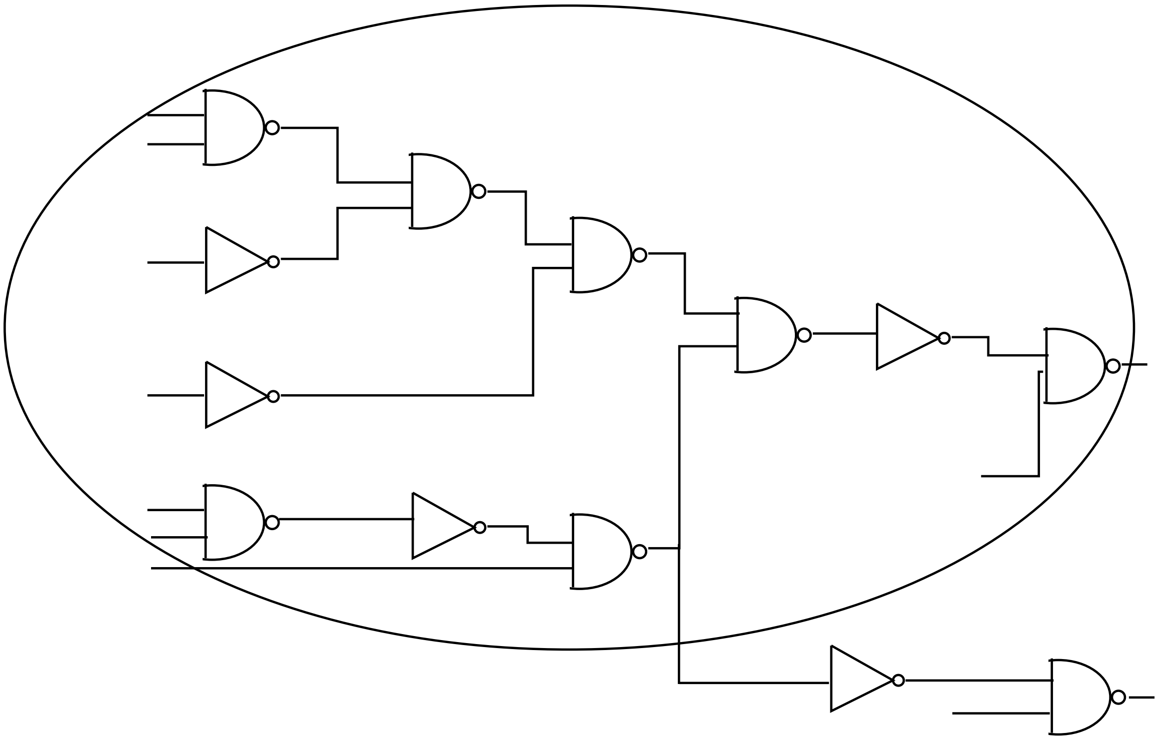
Partitioning the subject DAG into trees

1. Trivial partition: break the graph at all multiple-fanout points.
   - leads to no 'duplication' or 'overlap' of patterns.
   - Drawback - sometimes results in a lot of small trees.



Partitioning the subject DAG into trees

2. Single-cone partition: From a single output, form a large tree back to the primary inputs; map successive outputs until they hit the cone formed from previous outputs.
   - Duplicates some logic (where the trees overlap).
   - Produces much larger trees, potentially even better area results.



Rule-Based Approach

- Algorithmic approach: drawbacks
  1. May not handle special features of the technology or library.
  2. Cannot use design-knowledge ("tricks") used by the designers.
  3. Not very flexible: have to come up possibly with a new algorithm for a new cost function.

- Rule-based approach is based on an *expert system* paradigm.
  - Rule-base
    * A set of rules to do local, peephole transformations - analogous to compilers
    * An example of a rule: "Replace NAND-gate followed by an inverter by an AND gate."
  - Find target patterns on which transformations can be applied.
  - Rule-selection: best rule to apply.

Rules

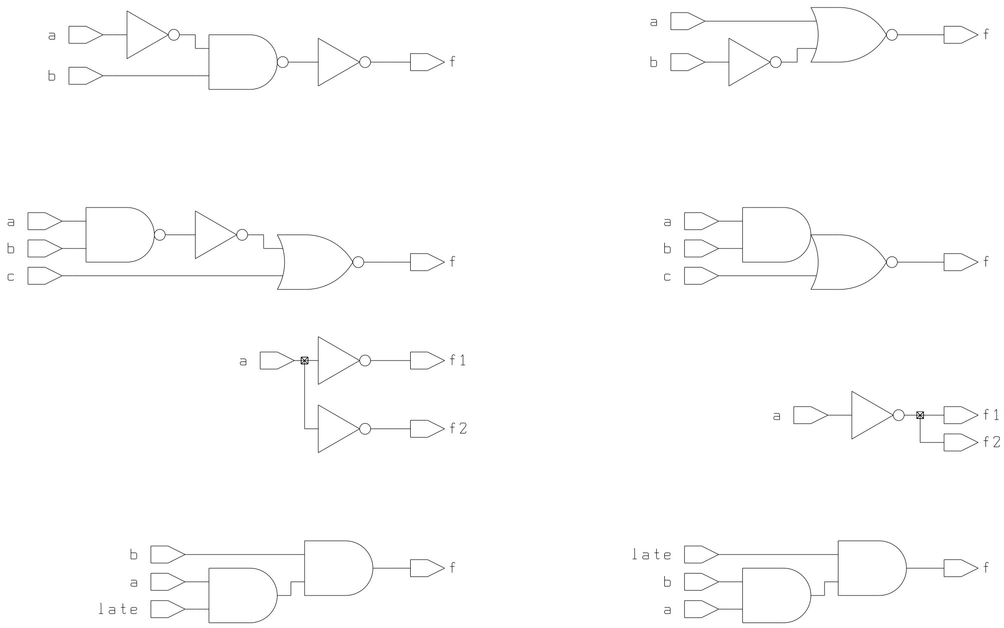
Rules: target configuration $\Rightarrow$ replacement configuration
Classes of Rules: (rules are ordered in each class)

- General rules
- Timing rules
- Area rules

Simple Strategy:

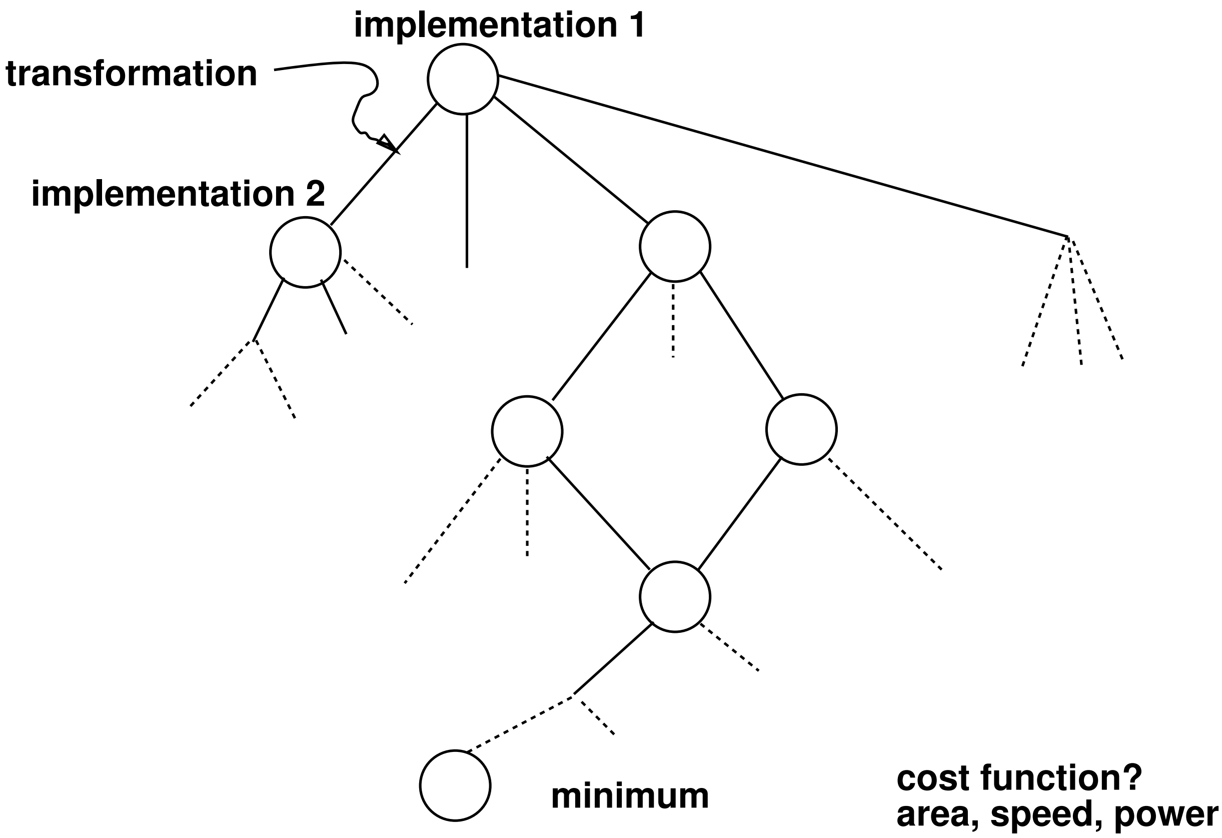
- Match - find all rules which apply
- Cost - evaluate the cost benefit of each applicable rule
- Select - select *best* rule
- Replace - replace target with replacement configuration

Rule-Based Approach



Example rules from SOCRATES

State Space Search



Greedy Search = Steepest Descent

Select Rule

Rules are ordered - always apply first (best) applicable rule

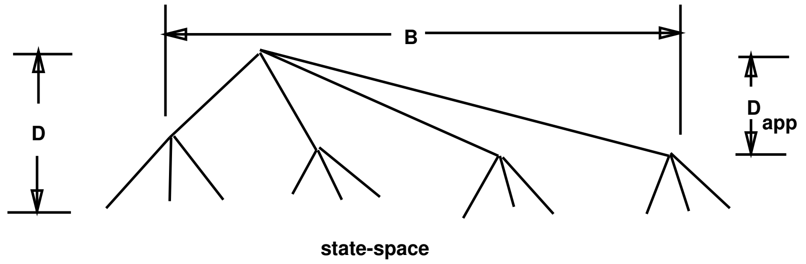
```
select_rule:
    foreach rule R in class C
        foreach gate G in circuit
            if R matches G
                apply R to G
                goto select_rule
```

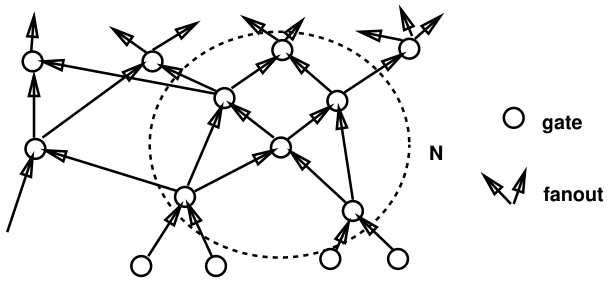
This works well for "area" but is poor for "timing"

Look-ahead Strategy

Parameters for pruning state-space search tree

- $B = \#$ of children explored (breadth)
- $D =$ depth of look-ahead
- $N =$ neighborhood (only look at gates in small subcircuit)
- $D_{app} =$ rule application depth





typical values: B=4, D=2, D_{app}=1, N=1

i.e. look ahead of 2 but apply only one step.

Rule Based Approach: Summing it up

- Before DAG-covering and tree-covering, rule-based approach was used {LSS, SOCRATES}.
- Disadvantages of rule-based approach:
  1. Creating, maintaining and changing the rule-base may be difficult: how a new rule will interact with existing set of rules.
  2. Poor technology portability: i.e. incorporating new gates in the library is difficult.
  3. Expensive in terms of CPU: related to # of nodes examined in state-space graph.
  4. Little notion of optimality: quality depends on completeness and selection of rules.
  5. Cannot use different rule-selection strategies for different problems.
- Now that covering idea has been discovered, almost all tools [DAGON, SIS, in industry] use algorithmic approach. But as a final step, to take advantage of special features of technology, library, *ad-hoc* methodology etc, rule-based approach is used to improve the mapping quality. It may also be used to improve performance or as a "clean-up".