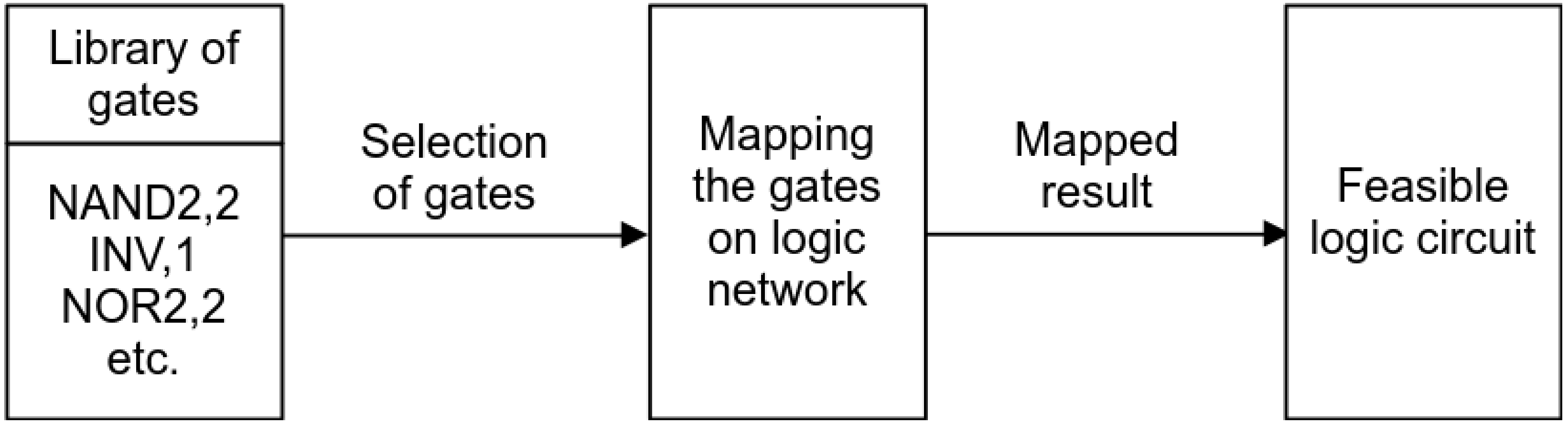
Technology Mapping, The Directed Acyclic Graph-Covering Problem and The Tree-Covering Problem

Presented by:	*Presented to:*
Rashed Hassan Siam *Class Roll: FH-2343* *Registration No: 2022-518-433* *Department of Computer Science and Engineering* *University of Dhaka*	*Professor Dr. Hafiz Md. Hasan Babu* *Department of Computer Science and Engineering* *University of Dhaka*

Contents

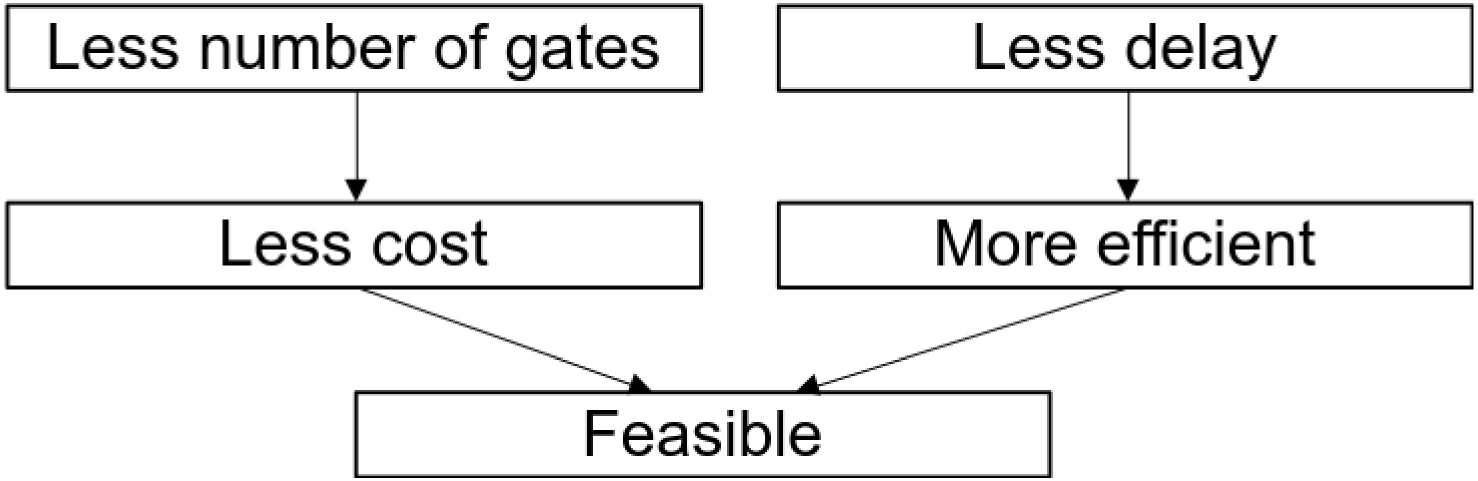
Technology Mapping

- What does it mean?
  - The process of synthesizing or simply implementing the logic circuit by performing the final gate selection from a particular library.



Feasible Circuits

- Why is the final circuit called "feasible"?
  - Optimal with respect to area: Least number of nodes/gates.
  - Satisfies a maximum critical-path delay: The longest path of the circuit along which the input results the output, is within a specified threshold.



Dos and Don’ts of Technology Mapping

- So, in general, Technology Mapping:
  - Doesn’t change the structure of the circuit radically.
  - Doesn’t reduce the number of levels of logic along the critical path.

- Instead, Technology Mapping:
  - Chooses the fastest gates along the critical path.
  - Uses the most area-efficient combination of gates off the critical path.

Technology Mapping Approaches

Two basic approaches

- **Rule-based techniques**

  Applies a set of rules (e.g., De Morgan’s law) to convert expressions into implementable gate-level circuits.

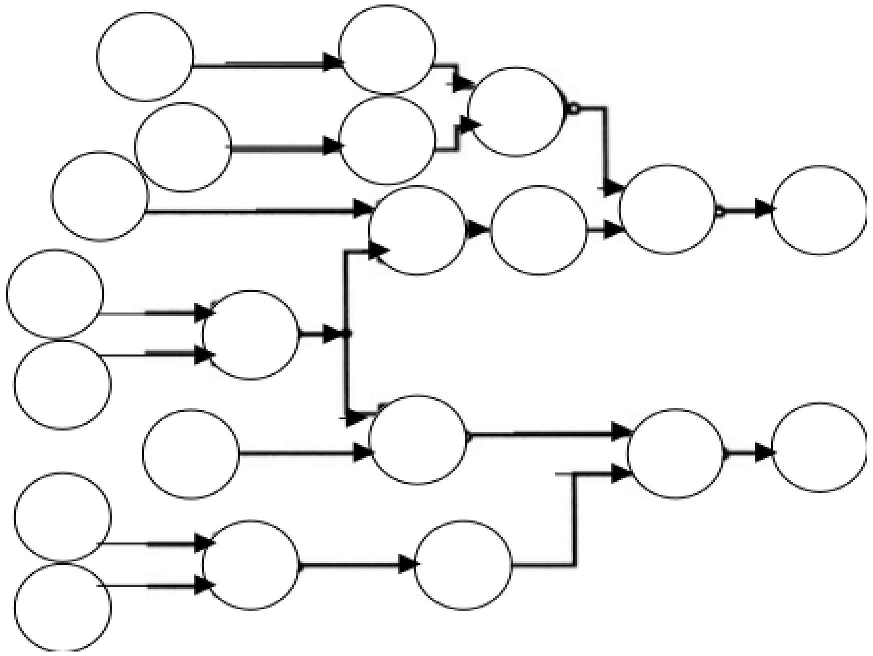
- **Graph covering techniques**

  Applies a set of base functions (e.g., NAND and inverter) which maps or covers all the graphs of the circuits and library gates.

Directed Acyclic Graph

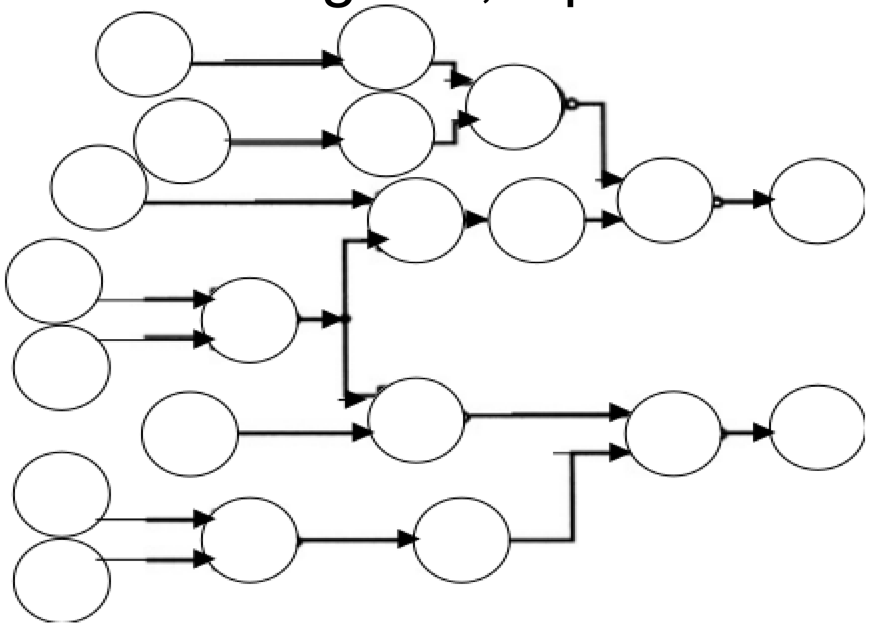
- What does it mean?
  - It is simply a graph made of nodes connected by directed edges.
  - There are no cycles, meaning we can't start at a node and return to it by following the edges.
  - In short, it is also called DAG.

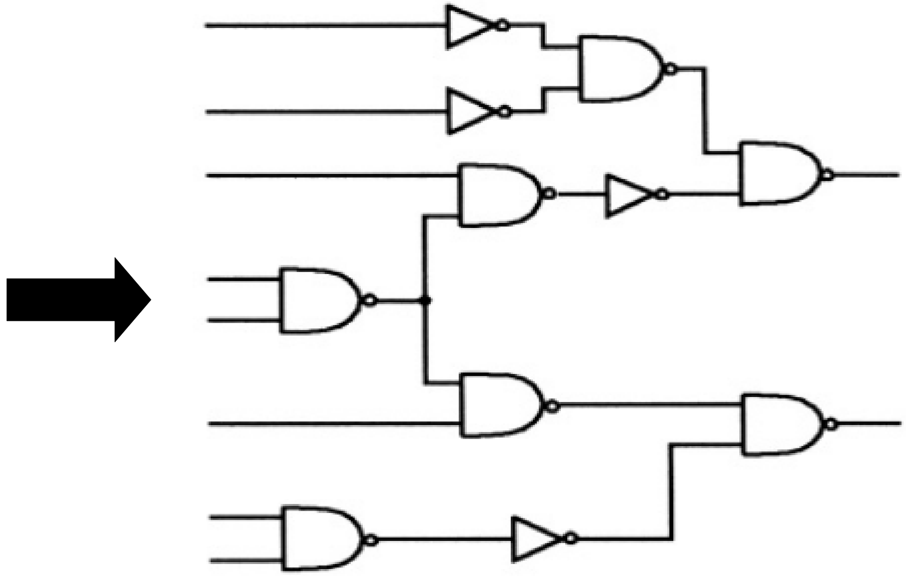
A directed acyclic graph, where the circles are nodes and arrows are the directed edges.



Logic Network in DAG Form

➢ If we consider the previous DAG example, and replace its nodes with various gates, inputs and outputs, we get:





So, combinational logic networks are basically DAGs!

Prerequisites of the DAG-Covering Problem

- Base Functions:
  - A set of base functions is chosen (e.g., a two-input NAND-gate and an inverter), using which the logic function and library gates will be designed as graphs.
- Subject Graph:
  - The optimized/simplified Boolean equation is converted into the subject graph, where each node is restricted to one of the base functions.
- Pattern Graph:
  - The logic function for each library gate is also represented by a pattern graph, where each node is restricted to one of the base functions.

Goals of the DAG-Covering Problem

- The main goal is:
  - To find a “minimum cost covering” of the subject graph by choosing from the collection of pattern graphs for all gates in the library.

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.

- Cost of the cover is defined in two ways:
  - For area optimization: The cost of the cover is defined as the sum of the areas of the individual gates.
  - For minimum delay optimization: The cost of a cover is defined as the critical path delay of the resulting circuit.

Choice of Base Functions

- The choice of a set of base functions is arbitrary as long as the base function set is functionally complete.
  - A functionally complete set is a set of functions that can express any other function (e.g., two-input NANDs and inverters).
- Different combinations of base functions can produce different patterns of the same logic function and library gates.
- The goal is to find the base-function set which provides the highest level of optimization and produces a small set of patterns.

$$f = (abcd + efgh + ijkl + mnop)'$$

  - Base function set with:
    - Two-input, three-input, and four-input NAND-gates: 1 pattern.
    - Two-input NAND-gates and inverters: 18 patterns.

Creating the Subject Graph

- A logic network has many representations as graphs of components from the base function set.
- Each representation is a potential subject graph for DAG-covering, having different costs.
- Every one of these starting points (subject graphs) should be considered for an optimum solution.
- Heuristics (e.g., algebraic decomposition, Boolean simplification techniques, etc.) are used to find an optimal form for the subject graph.

The DAG-Covering Problem

- This approach has a major issue: DAG-covering-by-DAGs is NP-hard!
- It is NP-hard even with only three pattern graphs (inverter, two-input NAND, two-input NOR) and if each subject graph node has no more than two incoming and outgoing edges.
- An exact covering algorithm was proposed based on a branch-and-bound procedure, but the complexity of the algorithm is very very large!
- So, heuristics can be a more effective approach, but this is still an open problem.
  - L. Lavagno at U.C. Berkeley has experimented with a number of heuristics with some degree of success.

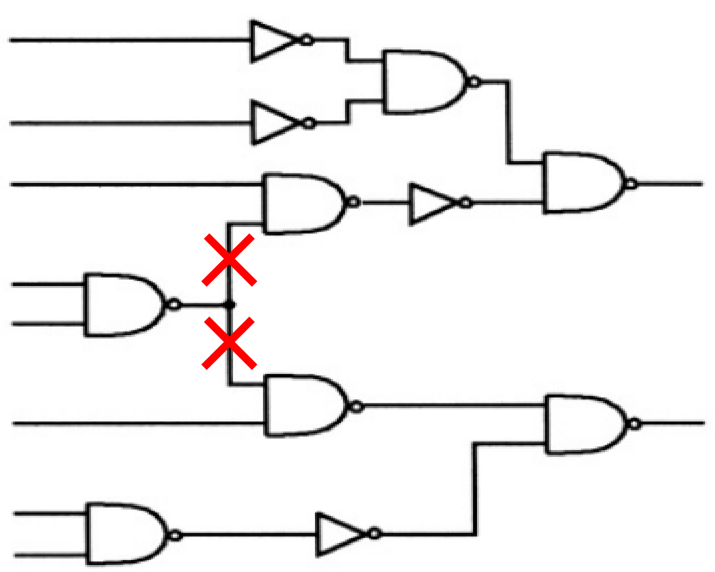
Alternative Approach: Tree-Covering Problem

- Keutzer has proposed reducing the DAG-covering problem to a set of tree-covering-by-trees problems.
- The steps are:
  - Partition the subject graph into trees by splitting it at the fanout points.
  - Cover each tree optimally.
  - Piece the tree-covers into a cover for the subject graph.

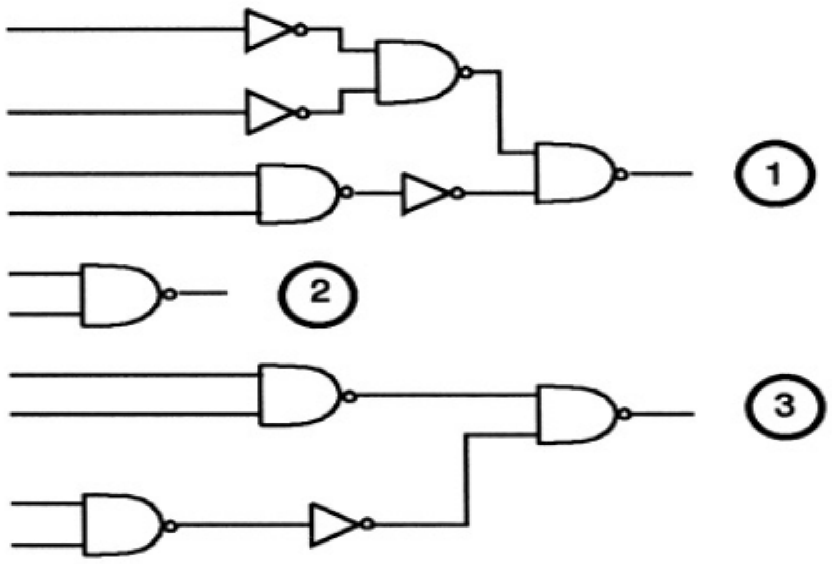
A tree circuit is a single output circuit in which each gate, except the output, feeds exactly one other gate, i.e., no fanout/branching at the outputs of any of the gates.

Splitting Subject Graph into Trees

- Let’s consider the previous subject graph and split it:



(a)



(b)

The circuit on the left is not a tree, because there is one gate that feeds two other gates (marked as red crosses). So splitting it at these two locations, we obtained three trees (1, 2 and 3) on the right.

The Tree Mapping Procedure

Two phases

- Pattern matching

  Finds all possible ways in which a library pattern may cover some nodes of the subject tree.

- Tree covering

  Selects one optimum matching (minimum cost) for each node.

Some Assumptions

- Let’s consider the tree-3 for pattern matching and the library containing three gates (INV,1; NAND2,2; NAND3,3) as shown below:

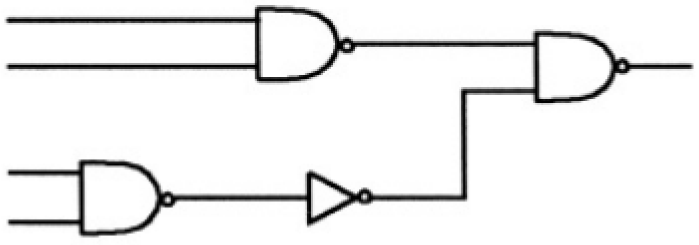


Figure: Tree-3 of the subject graph

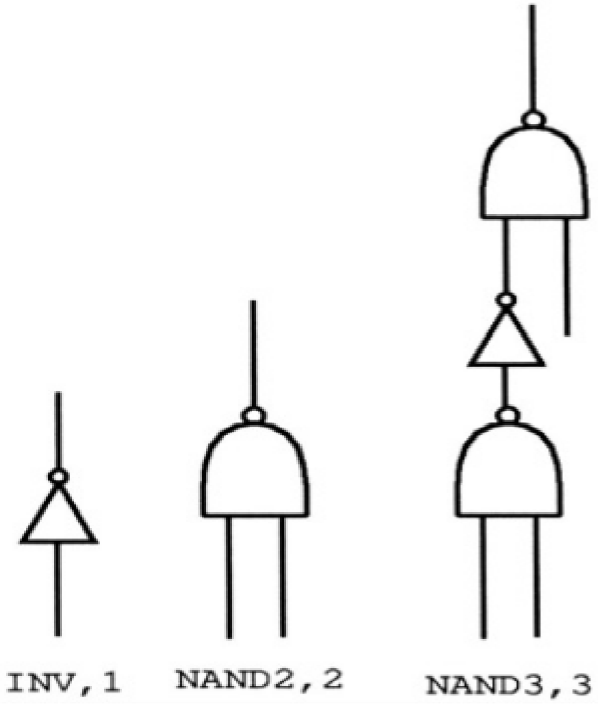


Figure: Library gates

Pattern Matching

- If we select the three-input NAND gate to match the output node, then we cover gates f and g, besides h; hence, we do not need matches for f and g.
- If we choose the two-input NAND gate, then we need to select gates to cover f and g.

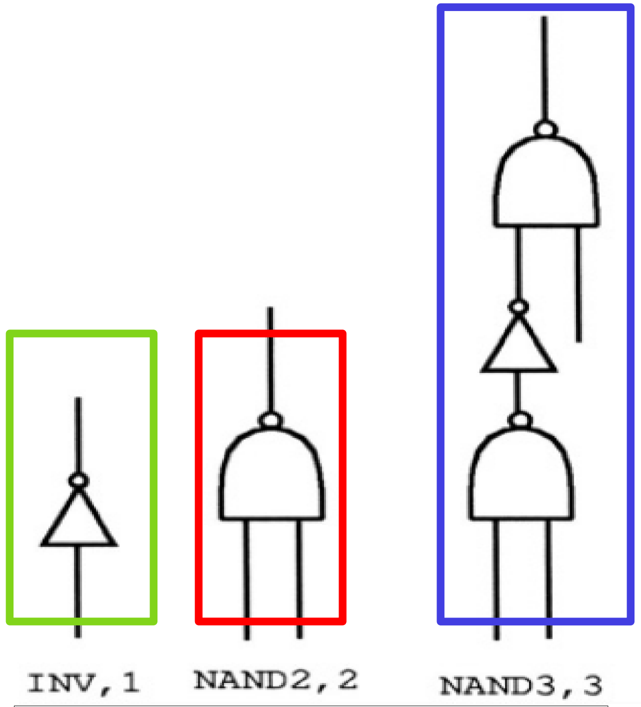


Figure: Library gates

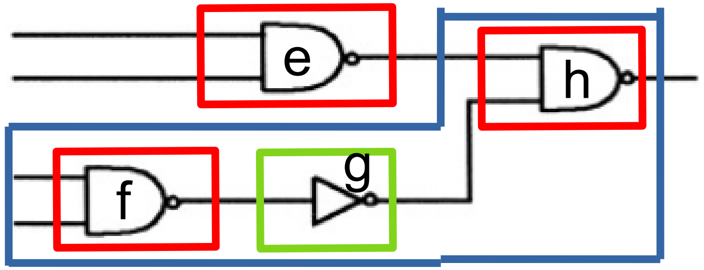


Figure: Tree-3 of the subject graph

Tree Covering

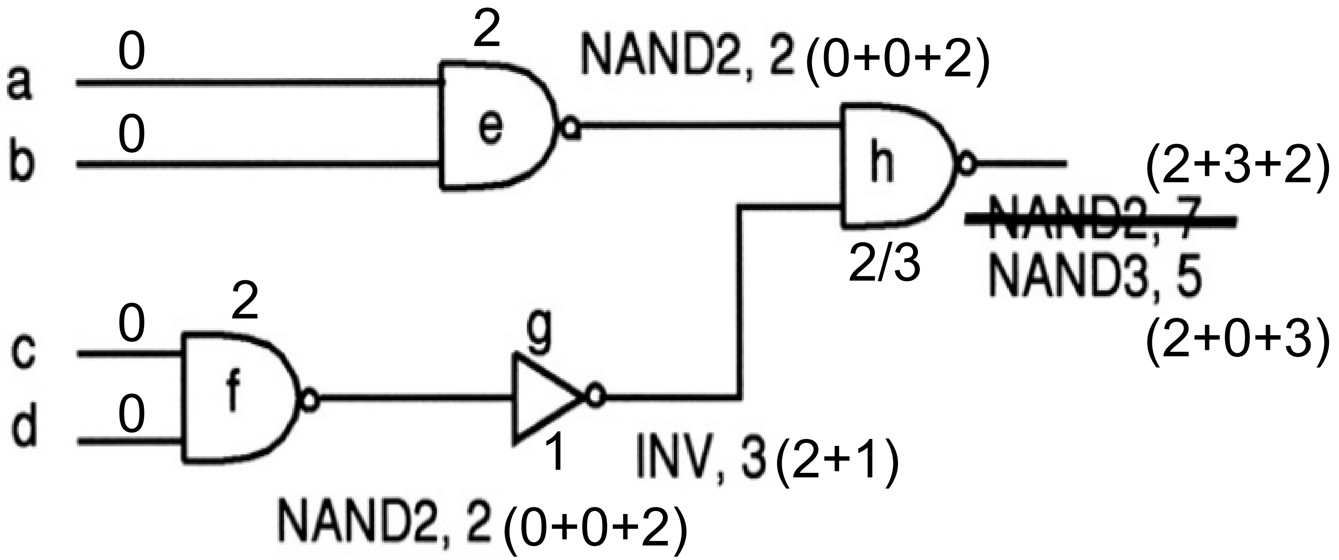


Figure: Optimum cover selection

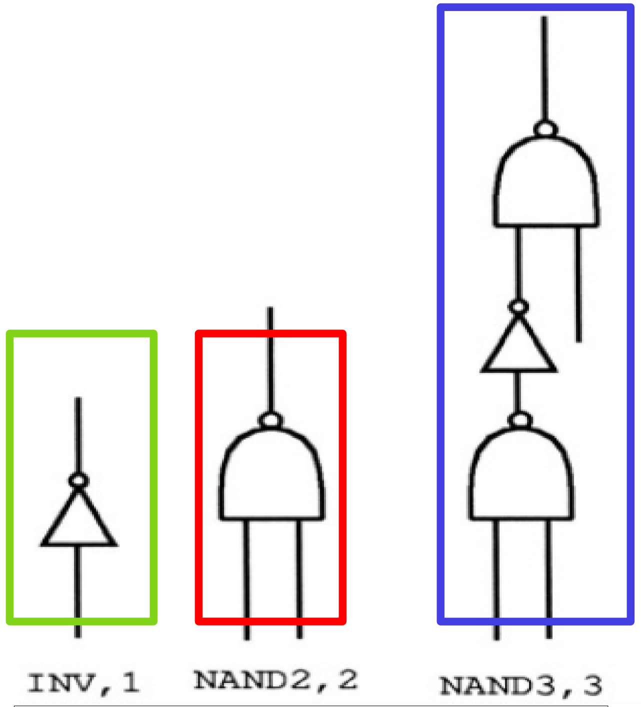


Figure: Library gates

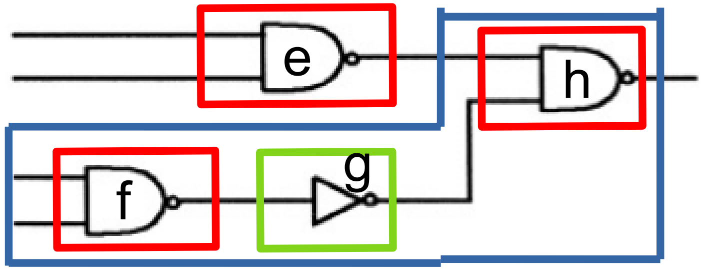


Figure: Tree-3 of the subject graph

Advantages of Tree-Covering Problem

- The NP-hardness of DAG-covering is removed.
- Linear complexity can be achieved due to splitting of the subject graph.
- Pattern matching is easier due to the reduction of search space.
- This method has proven to be quite effective.

Limitations of Tree-Covering Problem

- Loss of global view due to the step of partitioning into trees.
- Covers across partition boundaries are not allowed.
- Inefficient reduction of search space may affect the quality of the final solution.
- Does not always provide the exact optimum solution.

References

- ➢ Logic Synthesis and Verification Algorithms, Gary D. Hachtel and Fabio Somenzi.
- ➢ DAGON: Technology Binding and Local Optimization by DAG Matching, K. Keutzer.

Thank You!