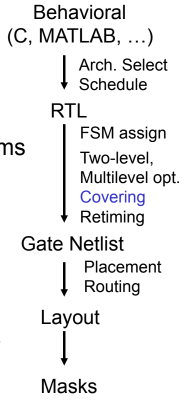
ESE535: Electronic Design Automation

Day 2: January 19, 2011
Covering

Work preclass exercise

Today: Covering Problem

- Implement a "gate-level" netlist in terms of some library of primitives
- General Formulation
  - Make it easy to change technology
  - Make it easy to experiment with library requirements
    - Evaluate benefits of new cells…
    - Evaluate architecture with different primitives



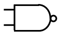
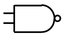
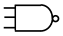
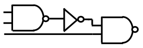
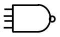
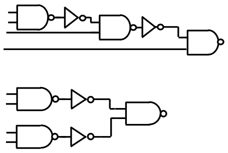
Behavioral
(C, MATLAB, …)
Arch. Select
Schedule
RTL
FSM assign
Two-level,
Multilevel opt.
Covering
Retiming
Gate Netlist
Placement
Routing
Layout
Masks

Input

1. netlist (logical circuit)
2. library

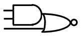
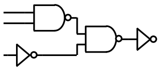
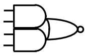
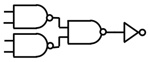
- represent both in normal form:
  - nand gate
  - inverters

Elements of a library - 1

Element/Area Cost			Tree Representation (normal form)
INVERTER	2		
NAND2	3		
NAND3	4		
NAND4	5		

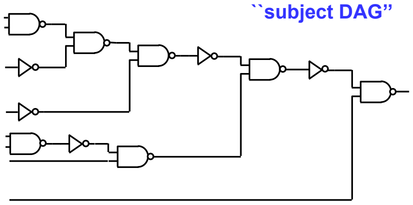
Example: Keutzer

Elements of a library - 2

Element/Area Cost			Tree Representation (normal form)
AOI21	4		
AOI22	5		

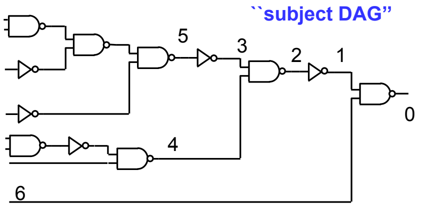
Input Circuit Netlist

``subject DAG"



•Each wire is a network (net).
•Each net has a single source (the gate that drives it).
•In general, net may have multiple sinks (gates that take as input)

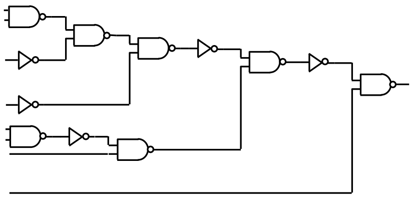
Input Circuit Netlist



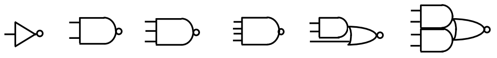
- A list of the nets (netlist) fully describes the circuit
  0 nand 1 6
  1 inv 2
  2 nand 3 4

Problem Statement

Find an ``optimal” (in area, delay, power) mapping of this circuit (DAG)



into this library

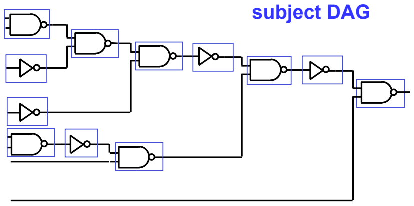


Why covering now?

- Nice/simple cost model
- Problem can be solved well
  - somewhat clever solution
- General/powerful technique
- Show off special cases
  - harder/easier cases
- Show off things that make hard
- Show off bounding

What’s the problem? Trivial Covering

subject DAG

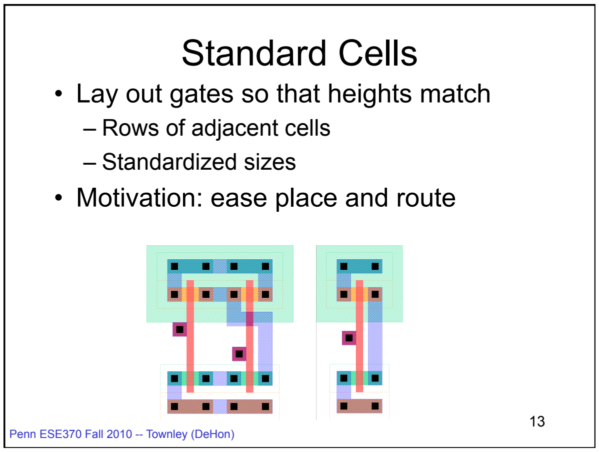


7	NAND2 (3) =	21
5	INV (2) =	10
	Area cost	31

Cost Models

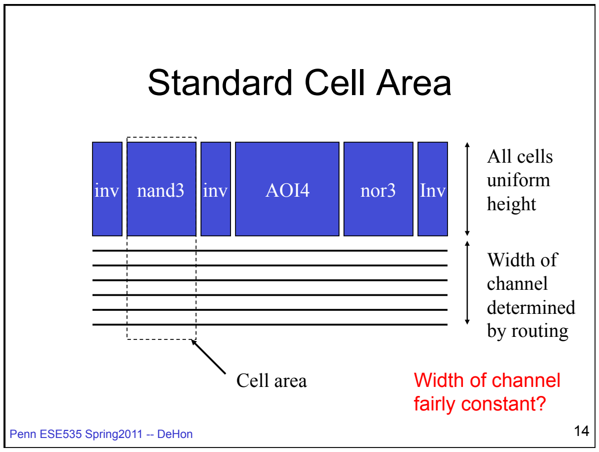
Cost Model: Area

- **Assume:** Area in gates
- or, at least, can pick an area/gate
  - so proportional to gates
- *e.g.*
  - Standard Cell design
  - Standard Cell/route over cell
  - Gate array



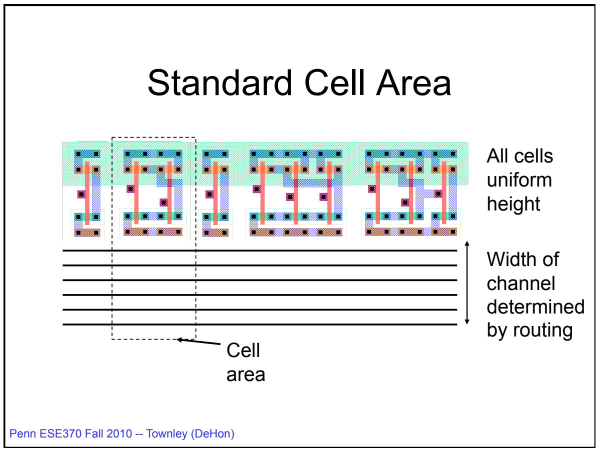
Standard Cells

- Lay out gates so that heights match
  - Rows of adjacent cells
  - Standardized sizes
- Motivation: ease place and route

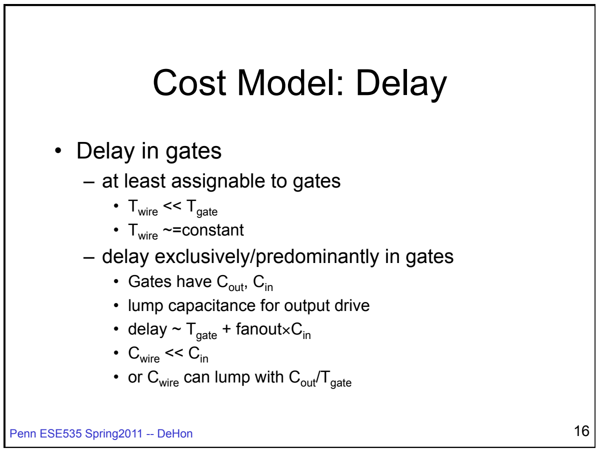


Standard Cell Area

Width of channel fairly constant?

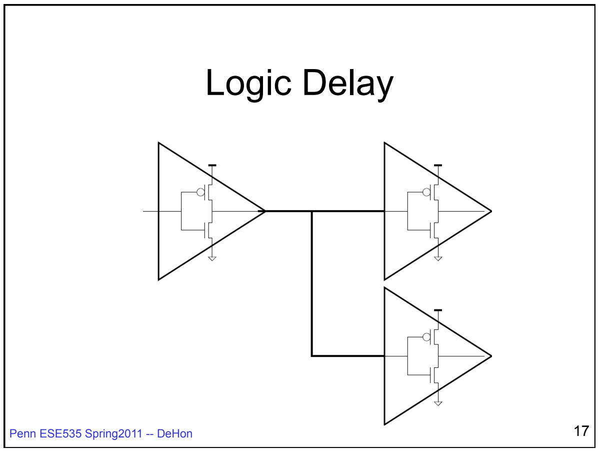


Standard Cell Area

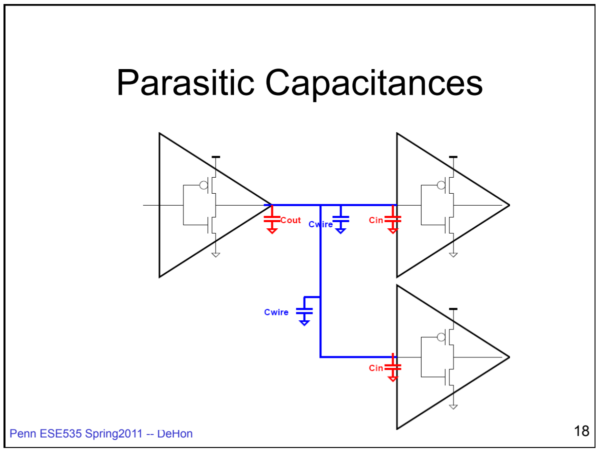


Cost Model: Delay

- Delay in gates
  - at least assignable to gates
    - $T_{wire} << T_{gate}$
    - T_{wire} ~=constant
  - delay exclusively/predominantly in gates
    - Gates have C_{out}, C_{in}
    - lump capacitance for output drive
    - delay ~ T_{gate} + fanout×C_{in}
    - $C_{wire} << C_{in}$
    - or C_{wire} can lump with C_{out}/T_{gate}

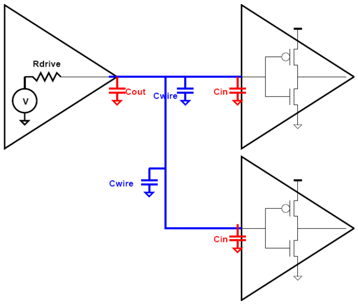


Logic Delay



Parasitic Capacitances

Delay of Net



Cost Model: Delay

- Delay in gates
  - at least assignable to gates
    - $T_{wire} << T_{gate}$
    - T_{wire} ~=constant
  - delay exclusively/predominantly in gates
    - Gates have C_{out}, C_{in}
    - lump capacitance for output drive
    - delay ~ T_{gate} + fanout×C_{in}
    - $C_{wire} << C_{in}$
    - or C_{wire} can lump with C_{out}/T_{gate}

F=22nm CMOS
T_{gate}(inv drive 4 inv)~=1ps
T_{wire}(300μm)~=1ps
W_{gate}~=0.3μm

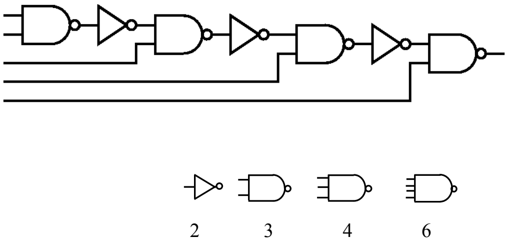
Cost Models

- Why do I show you models?
  - not clear there's one "right" model
  - changes over time
  - you're going to encounter many different kinds of problems
  - want you to see formulations so can critique and develop own
  - simple cost models make problems tractable
    - are surprisingly adequate
  - simple, at least, help bound solutions
  - may be wrong today…need to rethink

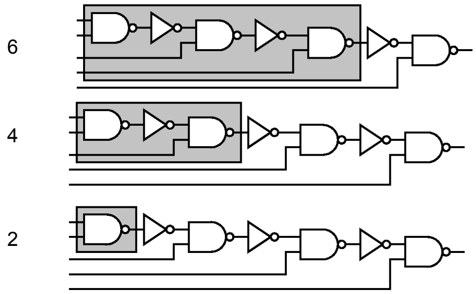
Approaches

Greedy work?

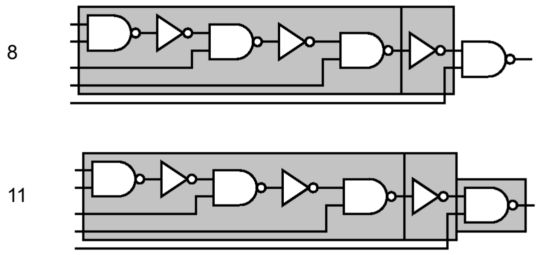
- Greedy = pick next locally "best" choice



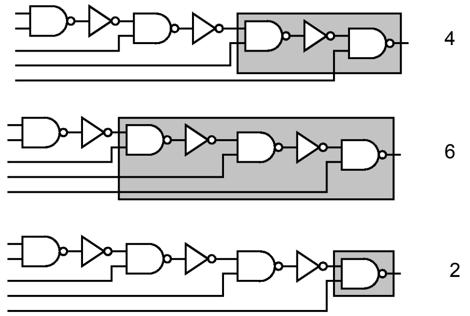
Greedy In→Out



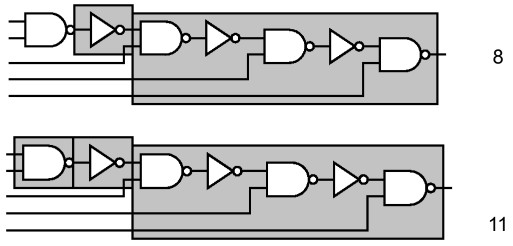
Greedy In→Out



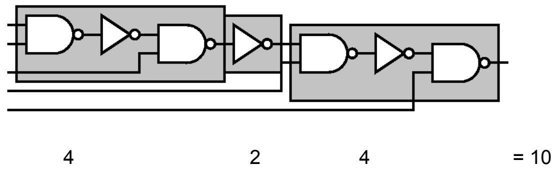
Greedy Out→In



Greedy Out→In



But…



Greedy Problem

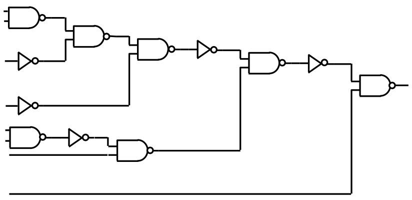
- What happens in the future (elsewhere in circuit) will determine what should be done at this point in the circuit.

- Can’t just pick best thing for now and be done.

Brute force?

- Pick a node (output)
- Consider
  - all possible gates which may cover that node
  - branch on all inputs after cover
  - pick least cost node

Pick a Node



Brute force?

- Pick a node (output)
- Consider
  - all possible gates which may cover that node
  - recurse on all inputs after cover
  - pick least cost node
- Explore all possible covers
  - can find optimum

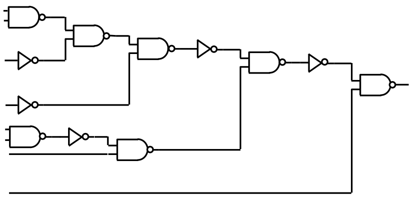
Analyze brute force?

- Time?

$$T_{brute}(node) = \sum_{i=0}^{\text{max pattern}} \left(T_{match}\,(P_i) + \sum_{j=0}^{\text{max in}} (T_{brute}\,(\text{in } j)) \right)$$

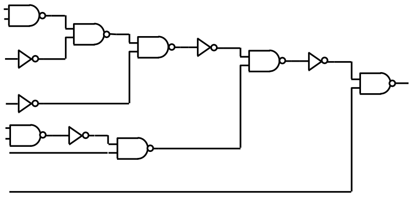
- Say P patterns, constant time to match each
  - (if patterns long could be > O(1))
- P-way branch at each node…
- …exponential
  - $O((P)^{depth})$

Structure inherent in problem to exploit?



Structure inherent in problem to exploit?

- There are only N unique nodes to cover!



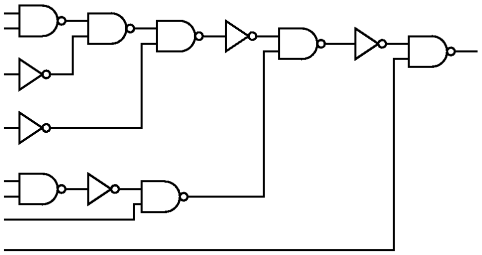
Structure

- **If** subtree solutions do not depend on what happens outside of its subtree
  - separate tree
  - farther up tree
- Should only have to look at N nodes.
- Time(N) = N*P*T(match)
  - w/ P fixed/bounded ➔ linear in N
  - w/ cleverness work isn't P*T(match) at every node

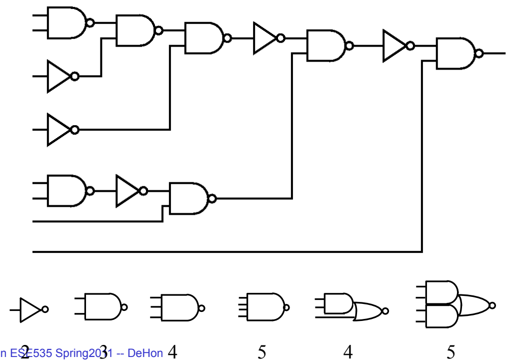
Idea Re-iterated

- Work from inputs
- Optimal solution to subproblem is contained in optimal, global solution
- Find optimal cover for each node
- Optimal cover:
  - examine all gates at this node
  - look at cost of gate and its inputs
  - pick least

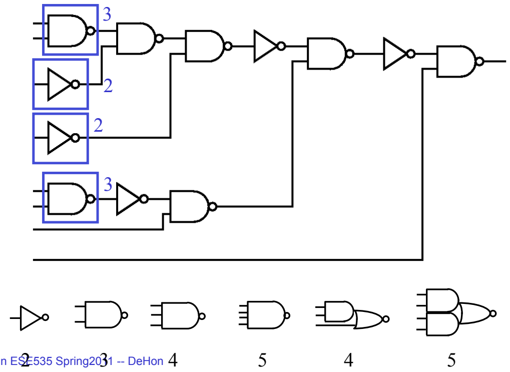
Work front-to-back



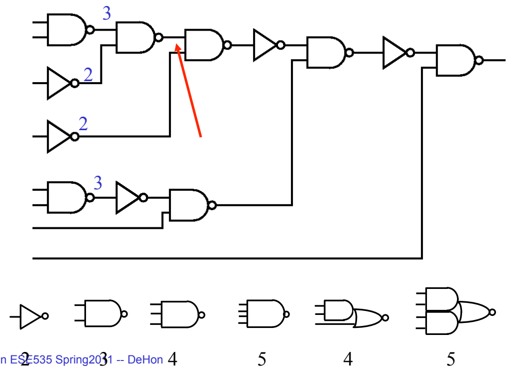
Work Example (area)



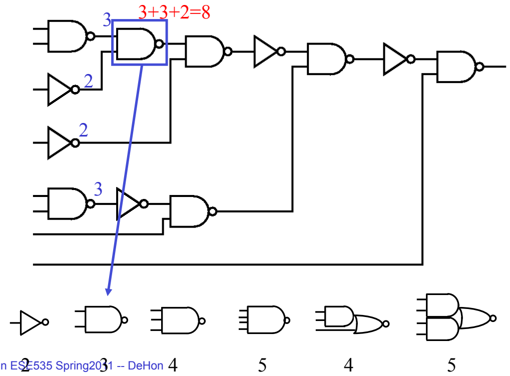
Work Example (area)

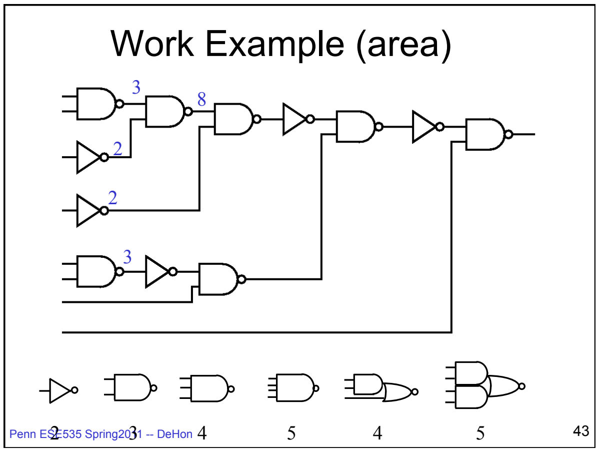


Work Example (area)

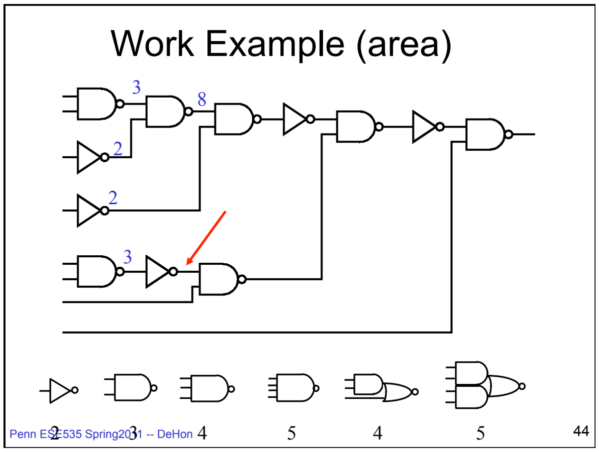


Work Example (area)

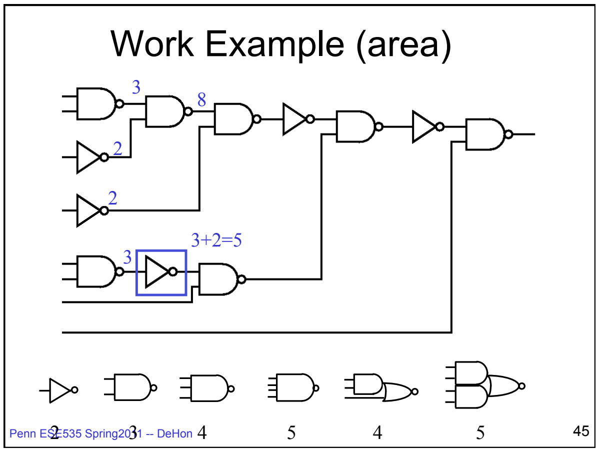




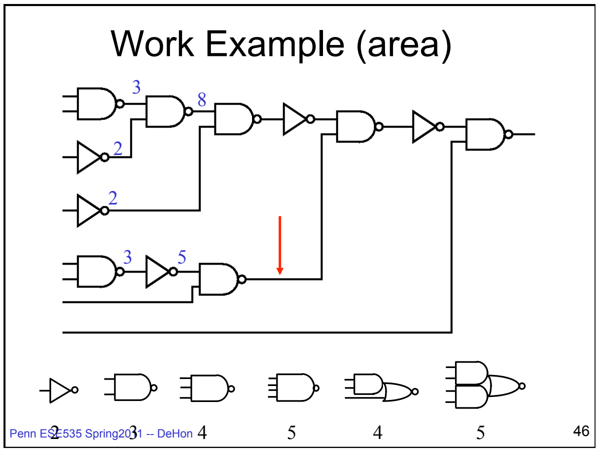
Work Example (area)



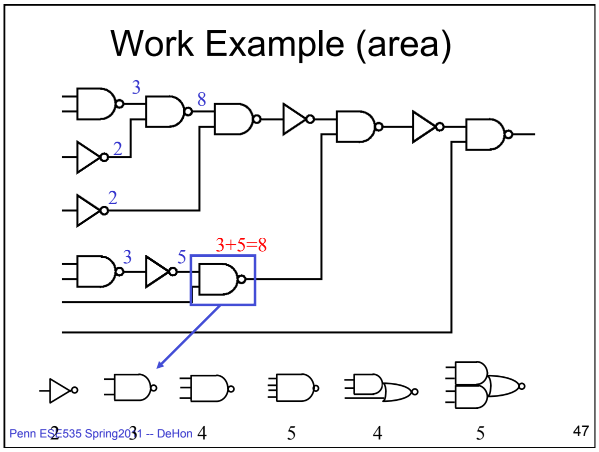
Work Example (area)



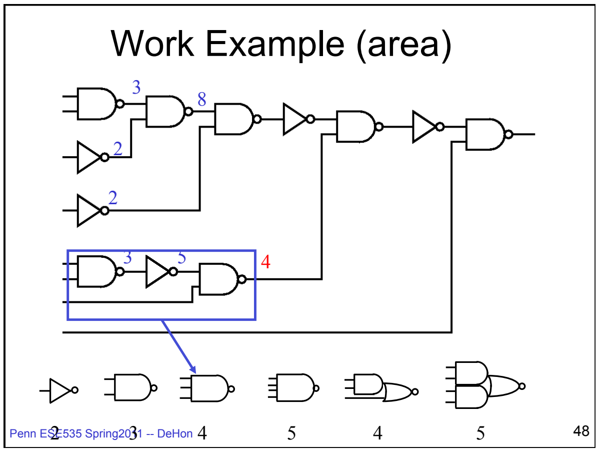
Work Example (area)



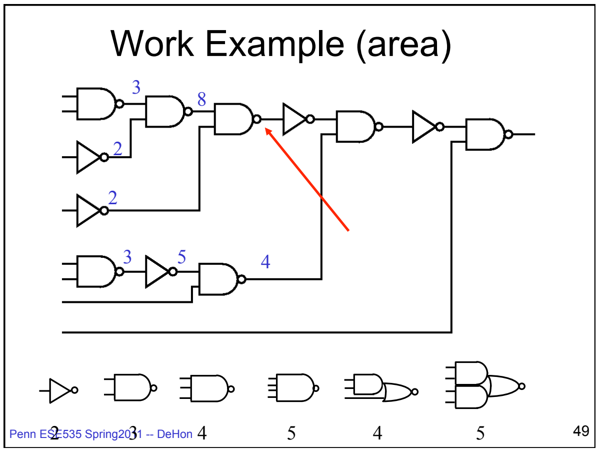
Work Example (area)



Work Example (area)

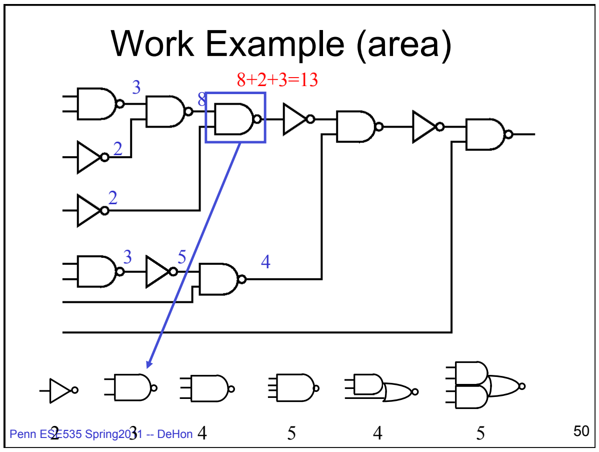


Work Example (area)



Work Example (area)

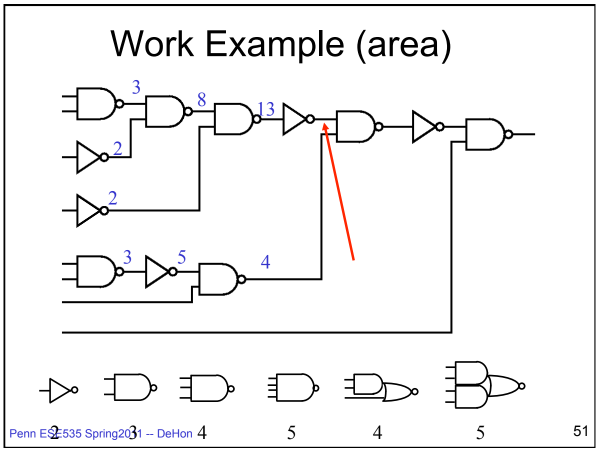
Penn ESE535 Spring2011 -- DeHon 49



Work Example (area)

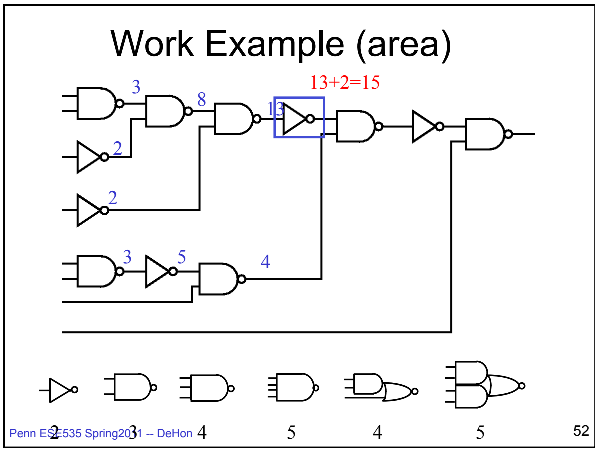
8+2+3=13

Penn ESE535 Spring2011 -- DeHon 50



Work Example (area)

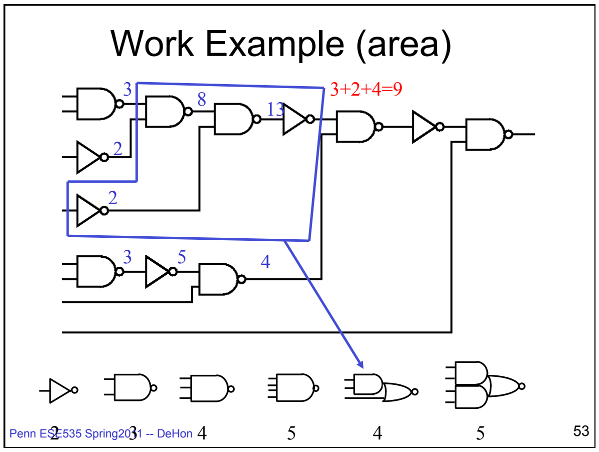
Penn ESE535 Spring2011 -- DeHon 51



Work Example (area)

13+2=15

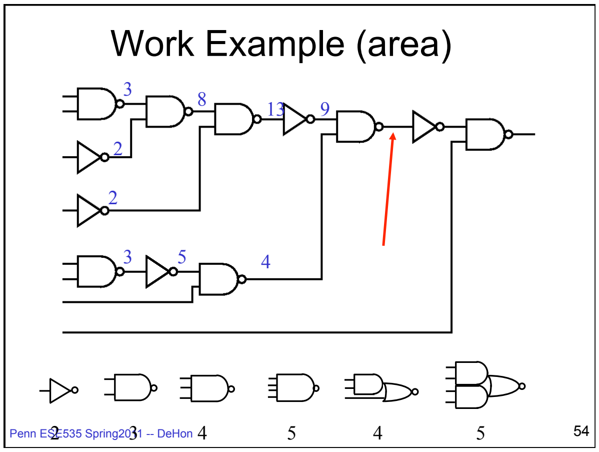
Penn ESE535 Spring2011 -- DeHon 52



Work Example (area)

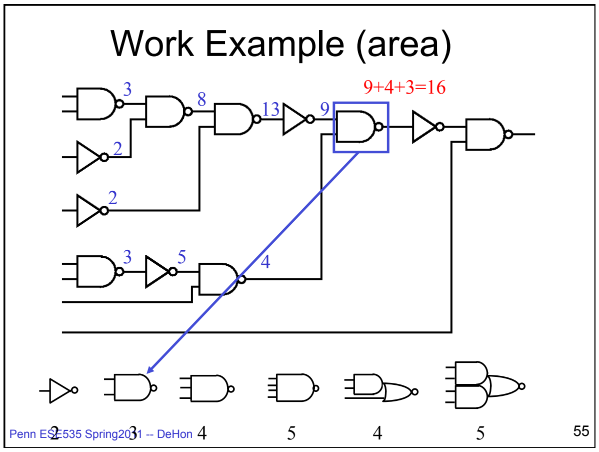
3+2+4=9

Penn ESE535 Spring2011 -- DeHon 53



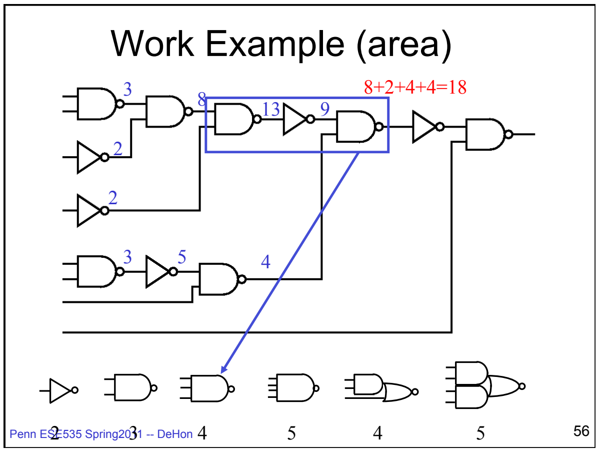
Work Example (area)

Penn ESE535 Spring2011 -- DeHon 54



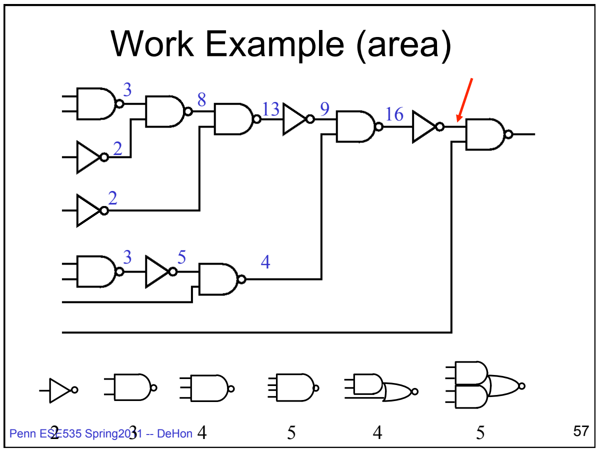
Work Example (area)

Penn ESE535 Spring2011 -- DeHon 55



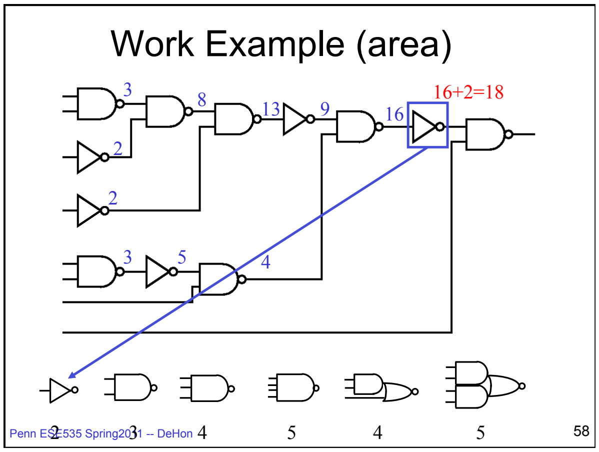
Work Example (area)

Penn ESE535 Spring2011 -- DeHon 56



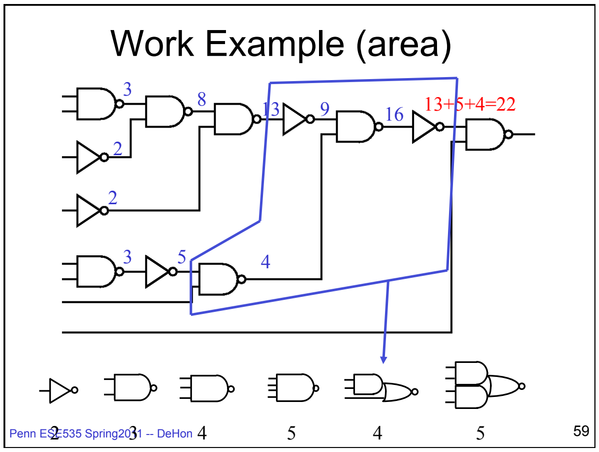
Work Example (area)

Penn ESE535 Spring2011 -- DeHon 57



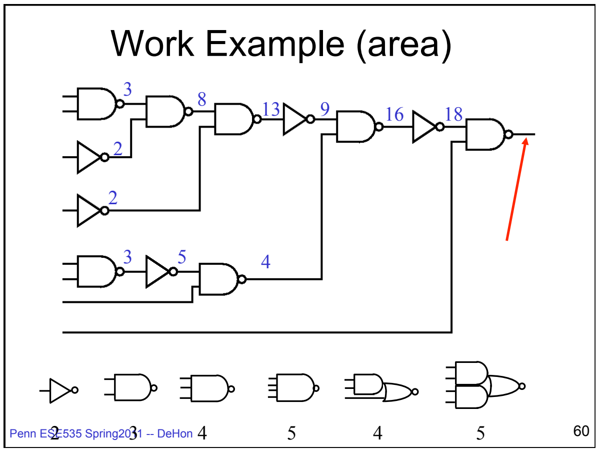
Work Example (area)

Penn ESE535 Spring2011 -- DeHon 58



Work Example (area)

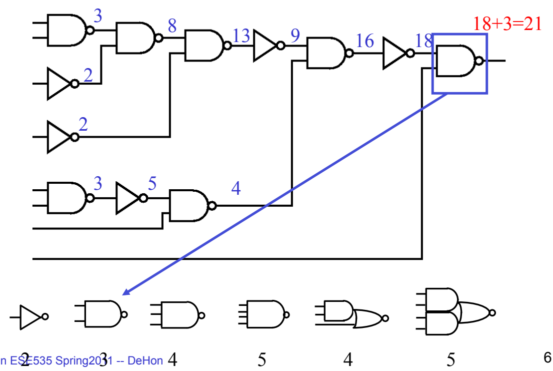
Penn ESE535 Spring2011 -- DeHon 59



Work Example (area)

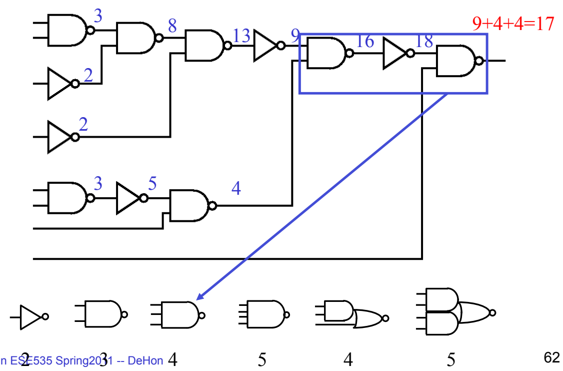
Penn ESE535 Spring2011 -- DeHon 60

Work Example (area)



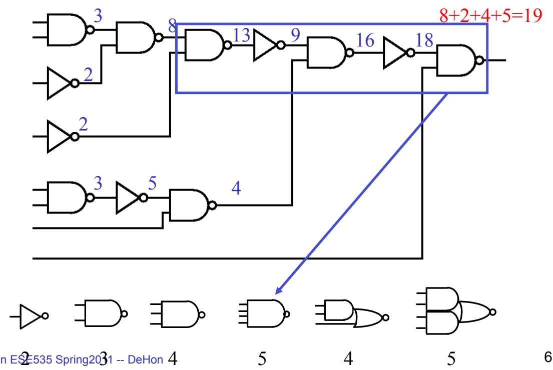
18+3=21

Work Example (area)



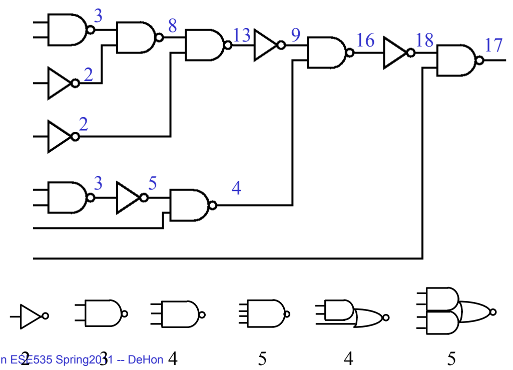
9+4+4=17

Work Example (area)

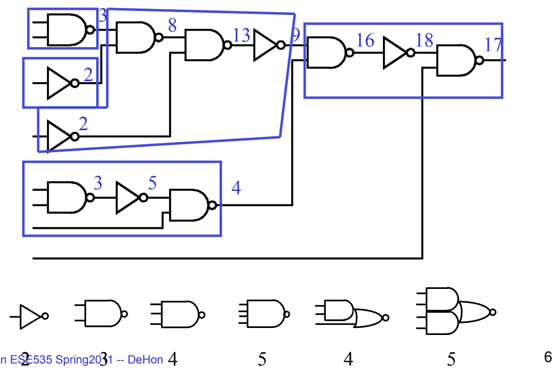


8+2+4+5=19

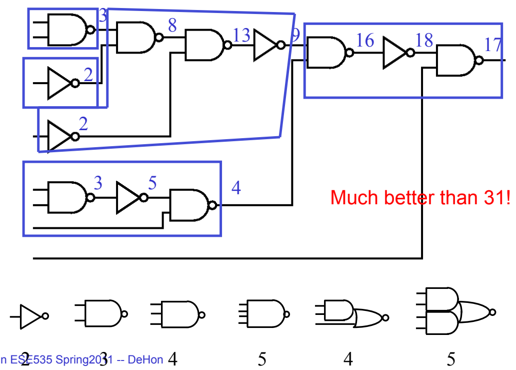
Work Example (area)



Optimal Cover



Optimal Cover

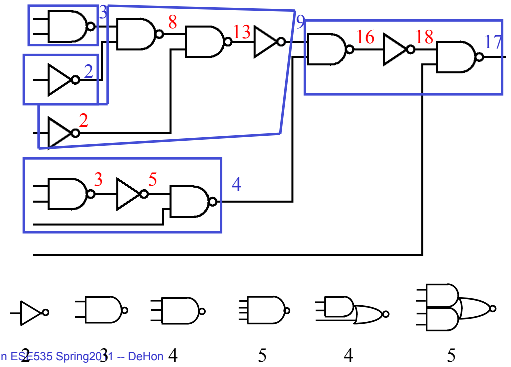


Much better than 31!

Note

- There are nodes we cover which will **not** appear in final solution.

“Unused” Nodes

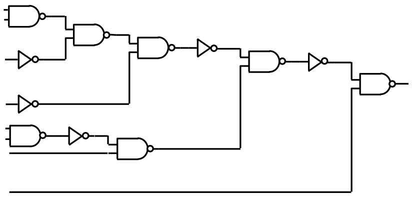


Dynamic Programming Solution

- Solution described is general instance of dynamic programming
- Require:
  - optimal solution to subproblems is optimal solution to whole problem
  - (all optimal solutions equally good)
  - divide-and-conquer gets same (finite/small) number of subproblems
- Same technique used for instruction selection in code generation for processors

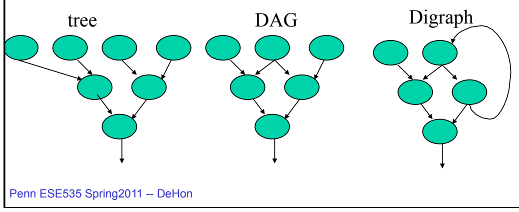
Delay

- Similar
  - Delay(node) = Delay(gate)+Max(Delay(input))



DAG

- DAG = Directed Acyclic Graph
  - Distinguish from tree (tree $\subset$ DAG)
  - Distinguish from cyclic Graph
  - DAG $\subset$ Directed Graph (digraph)

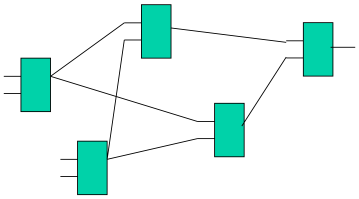


Trees vs. DAGs

- Optimal for trees
  - why?
    - Delay
    - Area

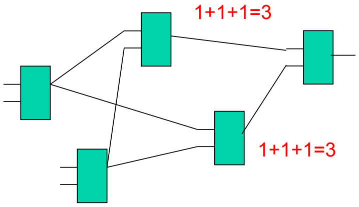
Not optimal for DAGs

- Why?



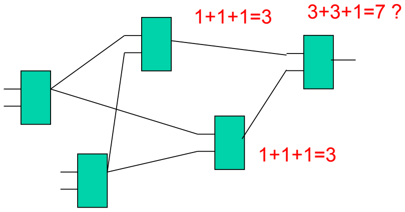
Not optimal for DAGs

- Why?



Not optimal for DAGs

- Why?



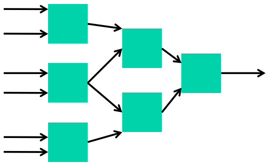
Not Optimal for DAGs (area)

- Cost(N) = Cost(gate) + Σ Cost(input nodes)

- think of sets
- cost is magnitude of set union
- **Problem**: minimum cost (magnitude) solution isn't necessarily the best pick
  - get interaction between subproblems
  - subproblem optimum not global...

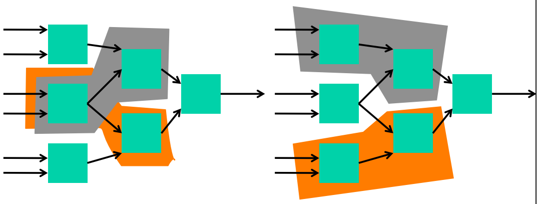
DAG Example

- Cover with 3 input gates



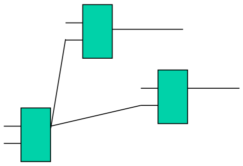
DAG Example

- Cover with 3 input gates



Not Optimal for DAGs

- Delay:
  - in fanout model, depends on problem you haven't already solved (delay of node depends on number of uses)



What do people do?

- Cut DAGs at fanout nodes
- optimally solve resulting trees

- Area
  - guarantees covered once
    - get accurate costs in covering trees, made "premature" assignment of nodes to trees
- Delay
  - know where fanout is

Bounding

- Tree solution give bounds (esp. for delay)
  - single path, optimal covering for delay
  - (also make tree by replicating nodes at fanout points)
- no fanout cost give bounds
  - know you can't do better
- delay bounds useful, too
  - know what you're giving up for area
  - when delay matters

(Multiple Objectives?)

- Like to say, get delay, then area
  - won't get minimum area for that delay
  - algorithm only keep best delay
  - …but best delay on off critical path piece not matter
    - …could have accepted more delay there
  - don't know if on critical path while building subtree
  - (iterate, keep multiple solutions)

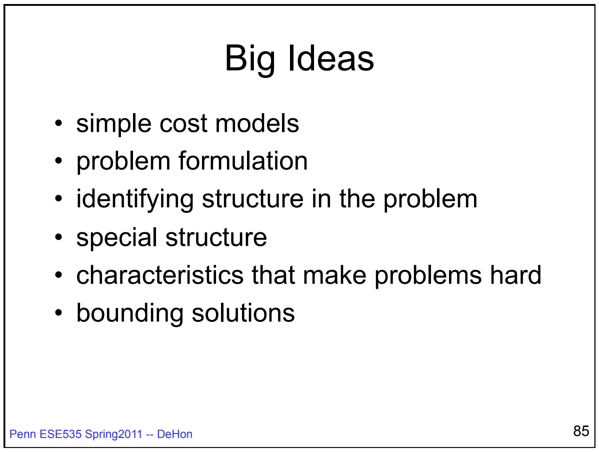
Many more details...

- Implement well

- Combine criteria

Admin

- Reading for today: blackboard
- Reading for Monday: online/Xplorer
- Office Hour: T4:30pm
  - Or make an appointment
- Project: C is common language
  - What will support



Big Ideas

- simple cost models
- problem formulation
- identifying structure in the problem
- special structure
- characteristics that make problems hard
- bounding solutions