

EDA (CS286.5b)

Day 2
Covering

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
- first assignment based on exploiting ideas

Problem

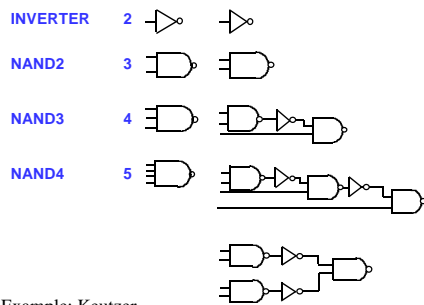
- Implement a “gate-level” netlist in terms of some library of primitives
- General
 - easy to change technology
 - easy to experiment with library requirements (benefits of new cells...)

Input

- netlist
- library
- represent both in normal form
 - nand gate
 - inverters

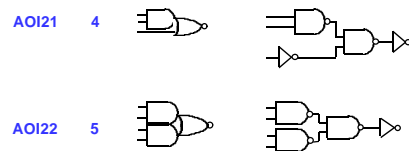
Elements of a library - 1

Element/Area Cost Tree Representation (normal form)

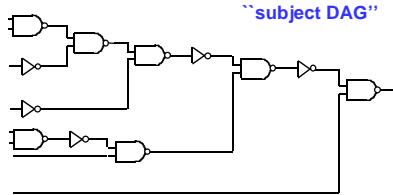


Elements of a library - 2

Element/Area Cost Tree Representation (normal form)

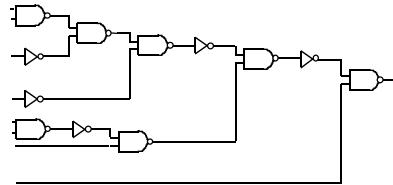


Input Circuit Netlist

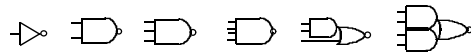


Problem statement

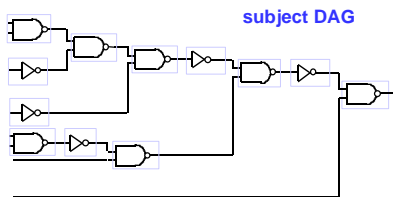
Find an "optimal" (in area, delay, power) mapping of this circuit (DAG)



into this library



What's the problem? Trivial Covering

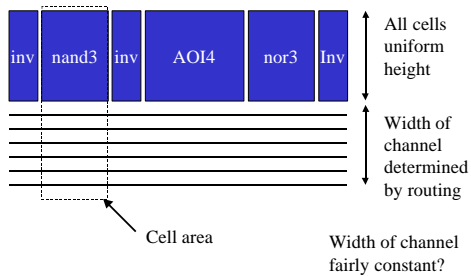


7 NAND2 (3) = 21
5 INV (2) = 10
Area cost 31

Cost Model: Area

- 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 Cell Area



Cost Model: Delay

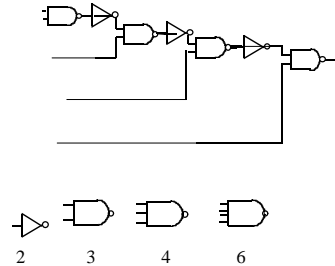
- Delay in gates
 - at least assignable to gates
 - $T_{wire} \ll T_{gate}$
 - $T_{wire} \approx \text{constant}$
 - delay exclusively/predominantly in gates
 - Gates have C_{out}, C_{in}
 - lump capacitance for output drive
 - $\text{delay} \sim T_{gate} + S C_{in}$
 - $C_{wire} \ll C_{in}$
 - or C_{wire} can lump with C_{out}/T_{gate}

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 make problems tractable
 - are surprisingly adequate
 - simple, at least, help bound solutions

Greedy work?

- Greedy = pick next locally "best" choice



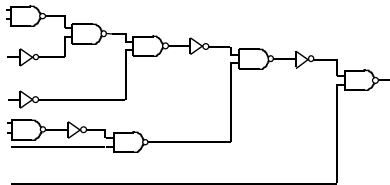
Greedy Problem

- 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



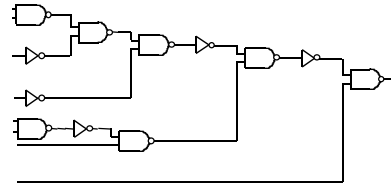
Brute force?

- 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?

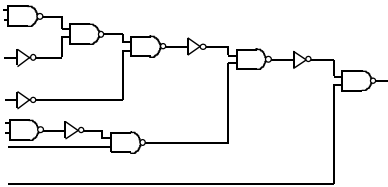
- Time?
- $T(N) = \text{Sigma}(T(\text{match})) * \text{Sigma}(T(\text{inputs}))$
- Say P patterns, linear time to match each
 - (can do better...)
- P-way branch at each node...
- ...exponential

Structure inherent in problem to exploit?



Structure inherent in problem to exploit?

- There are only N unique nodes to cover!



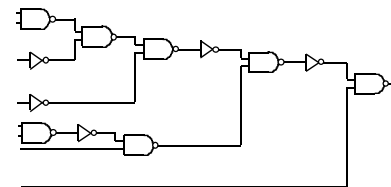
Structure

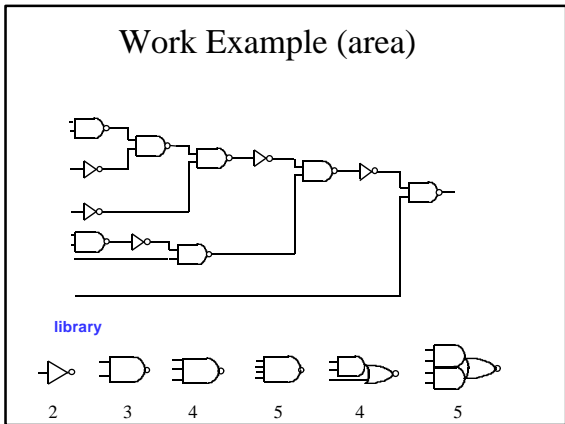
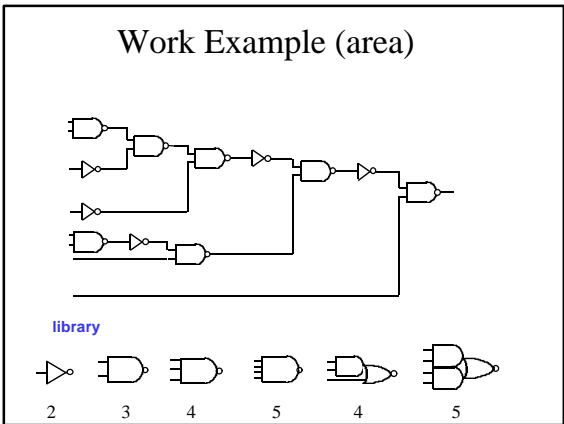
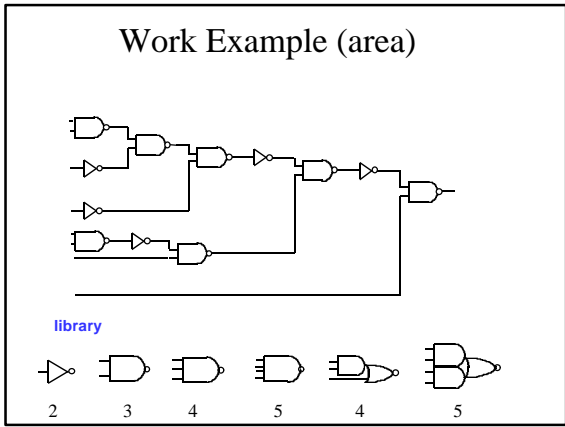
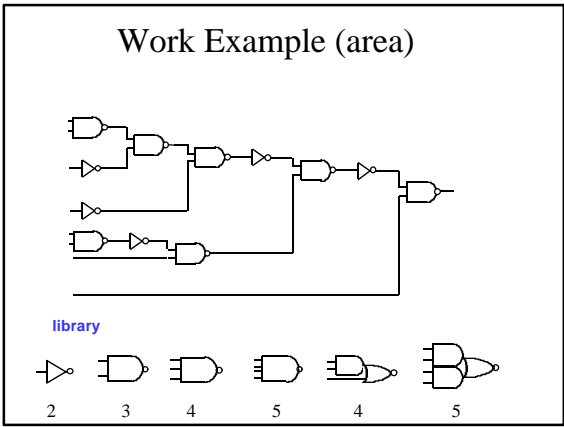
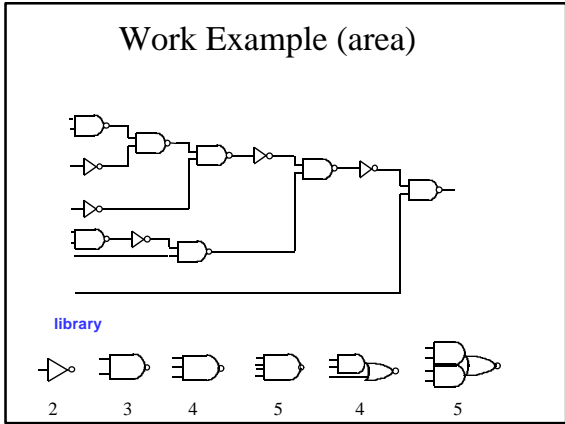
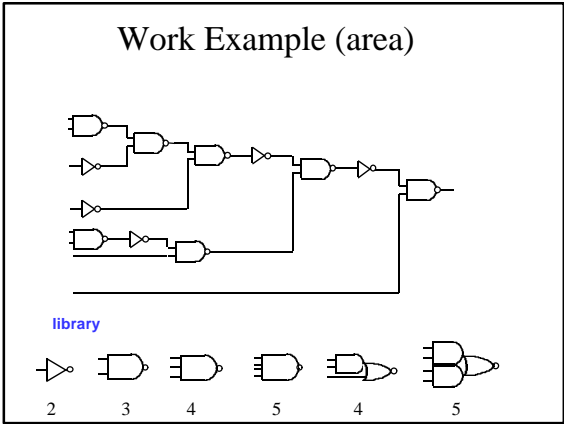
- **If** subtree solutions do not depend on what happens outside of its subtree
 - separate tree
 - farther up tree
- Should only have to look at N nodes.
- $\text{Time}(N) = N * P * T(\text{match})$
 - w/ P fixed/bounded, technically linear in N
 - w/ cleverness work isn't $P * T(\text{match})$ at ever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





Note

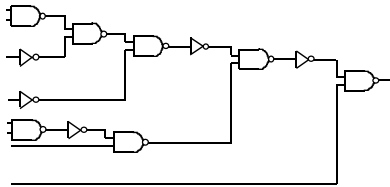
- There are nodes we cover which will **not** appear in final solution.

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

Delay

- Similar
 - $\text{Cost}(\text{node}) = \text{Delay}(\text{gate}) + \text{Max}(\text{Delay}(\text{input}))$

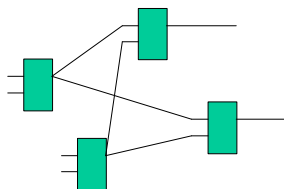


Trees vs. DAGs

- Optimal for trees
 - why?
 - Area
 - Delay

Not optimal for DAGs

- Why?

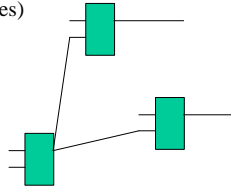


Not Optimal for DAGs (area)

- $\text{Cost}(N) \neq \text{Cost}(\text{gate}) + \text{SCost}(\text{input nodes})$
- 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...

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

Power Objective/Model?

- Optimality?

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
 - (touch on some later)
- ...see literature
 - (put some refs on web)

Today's Big Ideas:

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