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

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

<table>
<thead>
<tr>
<th>Element/Area Cost</th>
<th>Tree Representation (normal form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVERTER</td>
<td><img src="image1" alt="INVERTER Diagram" /></td>
</tr>
<tr>
<td>NAND2</td>
<td><img src="image2" alt="NAND2 Diagram" /></td>
</tr>
<tr>
<td>NAND3</td>
<td><img src="image3" alt="NAND3 Diagram" /></td>
</tr>
<tr>
<td>NAND4</td>
<td><img src="image4" alt="NAND4 Diagram" /></td>
</tr>
</tbody>
</table>

Elements of a library - 2

<table>
<thead>
<tr>
<th>Element/Area Cost</th>
<th>Tree Representation (normal form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOI21</td>
<td><img src="image5" alt="AOI21 Diagram" /></td>
</tr>
<tr>
<td>AOI22</td>
<td><img src="image6" alt="AOI22 Diagram" /></td>
</tr>
</tbody>
</table>
Input Circuit Netlist

``subject DAG``

Problem statement

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

What's the problem? Trivial Covering

subject DAG

<table>
<thead>
<tr>
<th>Gate</th>
<th>Count</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAND2</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>INV</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Area cost</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

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 Cell Area

<table>
<thead>
<tr>
<th>Gate</th>
<th>Width of channel</th>
<th>All cells uniform height</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nand3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOI4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nor3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inv</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell area

Width of channel fairly constant?
Cost Model: Delay

- Delay in gates
  - at least assignable to gates
    - Twire << Tgate
    - Twire = constant
  - delay exclusively/predominantly in gates
    - Gates have Cout, Cin
    - lump capacitance for output drive
    - delay ~ Tgate + fanout*Cin
    - Cwire << Cin
    - or Cwire can lump with Cout/Tgate

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

Approaches

Greedy work?

- Greedy = pick next locally “best” choice

Greedy In→Out
Greedy Out → In

But...

Greedy Problem

Brute force?

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?
  \[ T_{\text{brute}}(\text{node}) = \max_{i \in \text{patterns}} \left( T_{\text{match}}(P_i) + \max_{j \in \text{inputs}} \left( T_{\text{brute}}(M_{i,j}) \right) \right) \]
• Say \( P \) patterns, linear time to match each
  – (can do better…)
• \( P \)-way branch at each node…
• …exponential
  – \( O(\text{depth}^P) \)

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(\( N \)) = \( N^P \ast T(\text{match}) \)
  – w/ \( P \) fixed/bounded, technically linear in \( N \)
  – w/ cleverness work isn’t \( P^*T(\text{match}) \) 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)

$8 + 2 + 3 = 13$

$13 + 2 = 15$

$3 + 2 + 4 = 9$

$9 + 4 + 3 = 16$
Work Example (area)

\[
8 + 2 + 4 + 4 = 18
\]

Work Example (area)

\[
16 + 2 = 18
\]

Work Example (area)

\[
18 + 3 = 21
\]
Work Example (area)

Work Example (area)

Work Example (area)

Optimal Cover

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

Trees vs. DAGs

- Optimal for trees
  - why?
    - Area
    - Delay

Not optimal for DAGs

- Why?
Not optimal for DAGs

- Why?

1+1+1=3

1+1+1=3

Not Optimal for DAGs (area)

- Cost(N) = Cost(gate) + \( \Sigma \) 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...

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
  - (touch on some later)

- …see literature
  - (put some refs on web)

Big Ideas

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