Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher:	Robert Dick
Office:	L477 Tech
Email:	dickrp@northwestern.edu
Phone:	847–467–2298

NORTHWESTERN UNIVERSITY

Unix privacy hint

`chmod -R go-rwx ~`

Letter	Meaning	Letter	Meaning
u	user	r	read
g	group	w	write
o	other	x	execute

Two-level logic is necessary

$f(a,b)$	$b=0$	$b=1$
$a=0$	1	0
$a=1$	0	1

Some Boolean functions can not be represented with one logic level

$(\bar{a}\ \bar{b}) + (a\ b)$

Two-level logic is sufficient

$f(a,b)$	$b=0$	$b=1$
$a=0$	1	0
$a=1$	0	1

- All Boolean functions can be represented with two logic levels
- Given k variables, 2^K minterm functions exist
- Select arbitrary union of minterms

Two-level well-understood

- As we will see later, optimal minimization techniques known for two-level
- However, optimal two-level solution may not be optimal solution
  - Sometimes a suboptimal solution to the right problem is better than the optimal solution to the wrong problem

Two-level sometimes impractical

$f(a,b,c,d)$ ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Consider a 4-term XOR (parity) gate: $a \oplus b \oplus c \oplus d$

$(\bar{a}\ \bar{b}\ \bar{c}\ d) + (\bar{a}\ \bar{b}\ c\ \bar{d}) + (\bar{a}\ b\ \bar{c}\ \bar{d}) + (\bar{a}\ b\ c\ d) + (a\ b\ \bar{c}\ d) + (a\ b\ c\ \bar{d}) + (a\ \bar{b}\ \bar{c}\ \bar{d}) + (a\ \bar{b}\ c\ d)$

Two-level weakness

- Two-level representation is exponential
- However, it's a simple concept
  - Is $\sum_i^n x_i$ odd?
- Problem with representation, not function

Two-level weakness

Two-level representations also have other weaknesses

- Conversion from SOP to POS is difficult
  - Inverting functions is difficult
  - $\cdot$-ing two SOPs or $+$ing two POSs is difficult
- Neither general POS or SOP are canonical
  - Equivalence checking difficult
- POS satisfiability $\in \mathcal{NP}$-complete

Logic minimization motivation

- Want to reduce area, power consumption, delay of circuits
- Hard to exactly predict circuit area from equations
- Can approximate area with SOP cubes
- Minimize number of cubes and literals in each cube
- Algebraic simplification difficult
  - Hard to guarantee optimality

Logic minimization motivation

- K-maps work well for small problems
  - Too error-prone for large problems
  - Don't ensure optimal prime implicant selection
- Quine–McCluskey optimal and can be run by a computer
  - Too slow on large problems
- Espresso heuristic usually gets good results fast on large problems

Review: Algebraic simplification

Prove $XY + X\overline{Y} = X$

$$
\begin{aligned}
XY + X\overline{Y} &= X(Y + \overline{Y}) && \text{distributive law} \\
X(Y + \overline{Y}) &= X(1) && \text{complementary law} \\
X(1) &= X && \text{identity law}
\end{aligned}
$$

Boolean function minimization

- Algebraic simplification
  - Not systematic
  - How do you know when optimal solution has been reached?
- Optimal algorithm, e.g., Quine–McCluskey
  - Only fast enough for small problems
  - Understanding these is foundation for understanding more advanced methods
- Not necessarily optimal heuristics
  - Fast enough to handle large problems

Quine–McCluskey two-level logic minimization

- Compute prime implicants with a well-defined algorithm
  - Start from minterms
  - Merge adjacent implicants until further merging impossible
- Select minimal cover from prime implicants
  - Unate covering problem

Computing prime implicants

$\sum = 0$	0000	000X 00X0 X000	X00X X0X0
$\sum = 1$	0001 0010 1000	X001 X010 100X 10X0	
$\sum = 2$	1001 1010	1X01 1X10	
$\sum = 3$	1101 1110	111X 11X1	
$\sum = 4$	1111		

Definition: Unate covering

Given a matrix for which all entries are 0 or 1, find the minimum cardinality subset of columns such that, for every row, at least one column in the subset contains a 1.

I'll give an example

Prime implicant selection

	01X	0X0	X00	X11
~~000~~		1	1	
~~010~~	1	1		
~~011~~	1			1
~~111~~				1
~~100~~			1	

Cyclic core

a \ bc	00	01	11	10
0	0	1	1	1
1	1	1	0	1

	0X1	01X	X01	X10	10X	1X0
001	1		1			
011	1	1				
010		1		1		
100					1	1
101			1		1	
110				1		1

Implicant selection reduction

- Eliminate rows covered by essential columns
- Eliminate rows dominated by other rows
- Eliminate columns dominated by other columns

Eliminate rows covered by essential columns

	A	B	C
H		1	
I	1		1
J	1	1	
K		1	1

Eliminate rows dominating other rows

	A	B	C
H	1		
I	1	1	
J	1		1

Eliminate columns dominated by other columns

	A	B	C
H	1		
I	1	1	
J	1		1
K		1	

Backtracking

- Will proceed to complete solution unless cyclic
- If cyclic, can bound cover size
  - Compute independent sets

Find lower bound

a \ bc	00	01	11	10
0	0	1	1	1
1	1	1	0	1

	0X1	01X	X01	X10	10X	1X0
001	1		1			
011	1	1				
010		1		1		
100					1	1
101			1		1	
110				1		1

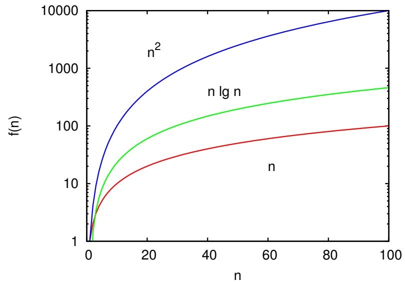
3 disjoint rows → 3 columns required

Use bound to constrain search space

- Eliminate rows covered by essential columns
- Eliminate rows dominated by other rows
- Eliminate columns dominated by other columns
- Branch-and-bound on cyclic problems
  - Use independent sets to bound
- Speed improved, still $\in \mathcal{NP}$-complete

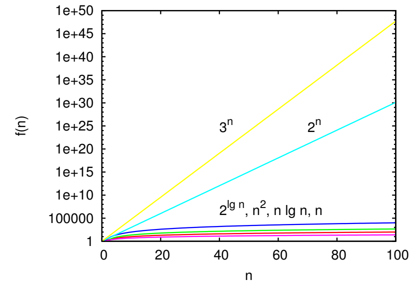
Extremely brief introduction to $\mathcal{NP}$-completeness

Polynomial-time algorithms: $\mathcal{O}(n)$, $\mathcal{O}(n \lg n)$, $\mathcal{O}(n^2)$



Extremely brief introduction to $\mathcal{NP}$-completeness

There also exist exponential-time algorithms: $\mathcal{O}(2^{\lg n})$, $\mathcal{O}(2^n)$, $\mathcal{O}(3^n)$

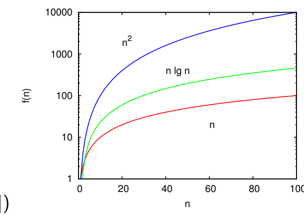


Extremely brief introduction to $\mathcal{NP}$-completeness

- Any $\mathcal{NP}$-complete problem instance can be converted to any other $\mathcal{NP}$-complete problem instance in polynomial time (quickly)
- Nobody has ever developed a polynomial time (fast) algorithm that optimally solves an $\mathcal{NP}$-complete problem
- It is generally believed (but not proven) that it is not possible to devise a polynomial time (fast) algorithm that optimally solves an $\mathcal{NP}$-complete problem
- Can use heuristics
  - Fast algorithms that often produce good solutions

$\mathcal{NP}$-completeness

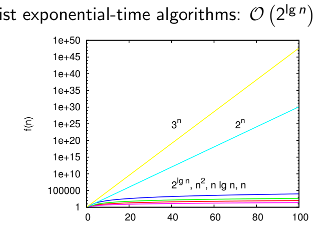
Recall that sorting may be done in $\mathcal{O}(n \lg n)$ time
DFS $\in \mathcal{O}(|V| + |E|)$, BFS $\in \mathcal{O}(|V|)$, Topological sort $\in$



$\mathcal{O}(|V| + |E|)$

$\mathcal{NP}$-completeness

There also exist exponential-time algorithms: $\mathcal{O}(2^{\lg n})$, $\mathcal{O}(2^n)$, $\mathcal{O}(3^n)$



$\mathcal{NP}$-completeness

For $t(n) = 2^n$ seconds

$$\begin{aligned}
t(1) &= 2 \text{ seconds} \\
t(10) &= 17 \text{ minutes} \\
t(20) &= 12 \text{ days} \\
t(50) &= 35,702,052 \text{ years} \\
t(100) &= 40,196,936,841,331,500,000,000 \text{ years}
\end{aligned}$$

$\mathcal{NP}$-completeness

- Digital design and synthesis is full of NP-complete problems
- Graph coloring
- Scheduling
- Graph partitioning
- Satisfiability (and 3SAT)
- *Covering*
- . . . and many more

$\mathcal{NP}$-completeness

- There is a class of problems, $\mathcal{NP}$-complete, for which nobody has found polynomial time solutions
- It is possible to convert between these problems in polynomial time
- Thus, if it is possible to solve any problem in $\mathcal{NP}$-complete in polynomial time, all can be solved in polynomial time
- Unproven conjecture: $\mathcal{NP} \neq \mathcal{P}$

$\mathcal{NP}$-completeness

- What is $\mathcal{NP}$? Nondeterministic polynomial time.
- A computer that can simultaneously follow multiple paths in a solution space exploration tree is nondeterministic. Such a computer can solve $\mathcal{NP}$ problems in polynomial time.
- I.e., a computer that can simultaneously be in multiple states.
- Nobody has been able to prove either

$$\mathcal{P} \neq \mathcal{NP}$$

or

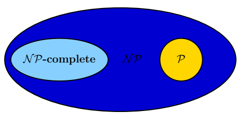
$$\mathcal{P} = \mathcal{NP}$$

$\mathcal{NP}$-completeness

If we define $\mathcal{NP}$-complete to be a set of problems in $\mathcal{NP}$ for which any problem's instance may be converted to an instance of another problem in $\mathcal{NP}$-complete in polynomial time, then

$$\mathcal{P} \subsetneq \mathcal{NP} \Rightarrow \mathcal{NP}\text{-complete} \cap \mathcal{P} = \varnothing$$

Basic complexity classes



- $\mathcal{P}$ solvable in polynomial time by a computer (Turing Machine)
- $\mathcal{NP}$ solvable in polynomial time by a nondeterministic computer
- $\mathcal{NP}$-complete converted to other $\mathcal{NP}$-complete problems in polynomial time

How to deal with hard problems

- What should you do when you encounter an apparently hard problem?
- Is it in $\mathcal{NP}$-complete?
- If not, solve it
- If so, then what?

Despair. Solve it! Resort to a suboptimal heuristic.
Bad, but sometimes the only choice. Develop an approximation algorithm.
Better. Determine whether all encountered problem instances are constrained.
Wonderful when it works.

One example

O. Coudert. Exact coloring of real-life graphs is easy. *Design Automation*, pages 121–126, June 1997.

Heuristic logic minimization

Optimal two-level logic synthesis is $\mathcal{NP}$-complete

- Upper bound on number of prime implicants grows $3^n/n$ where n is the number of inputs
- Given > 16 inputs, can be intractable
- However, there have been advances in complete solvers for many functions
  - Optimal solutions are possible for some large functions

Heuristic logic minimization

- For difficult and large functions, solve by heuristic search
- Multi-level logic minimization is also best solved by search
- The general search problem can be introduced via two-level minimization
  - Examine simplified version of the algorithms in Espresso

Espresso two-level logic minimization heuristic

- Generate only a subset of prime implicants
- Carefully selects subset of prime implicants covering on-set
- Guaranteed to be correct
  - May not be optimal

Espresso

Can be viewed in the following

- Start with a potentially optimal algorithm
- Add numerous techniques for constraining the search space
- Uses efficient move order to allow pruning
- Disable backtracking to arrive at a heuristic solver
- Widely used in industry
- Still has room for improvement
  - E.g., early recursion termination

Summary

- Properties of two-level logic
- The Quine-McCluskey (tabular) method
- $\mathcal{NP}$-complete: Why use heuristics?
- Espresso

Homework assignment one

- Algebraic manipulation (Review)
- K-Maps (Review)
- Quine-McCluskey
- Espresso

Next lecture

- More on Espresso algorithm
- Technologies and implementation methods

Reading assignment

- http://www.writphotec.com/mano/reading_supplements.html
- *More Optimization* for Quine–McCluskey