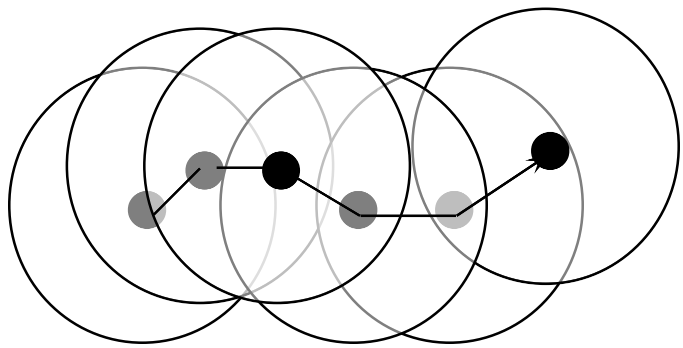
Two Level Logic Optimization: Heuristic Minimization

Two-Level Logic Minimization

- Exact minimization
  - problem : very large number of prime ($3^n / n$) and very large number of minterms
- Heuristic minimization
  - avoid computing all primes
  - successively modify a given initial cover of the function until a suitable stopping (local search) criterion is met

Local Search



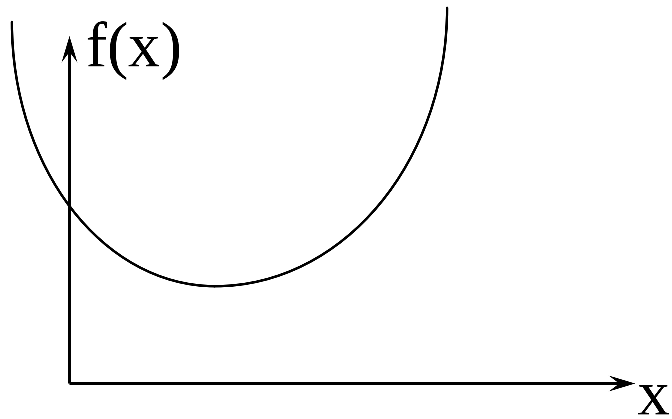
A Pictorial Representation of Local Search

- Distance : the difference of two solutions

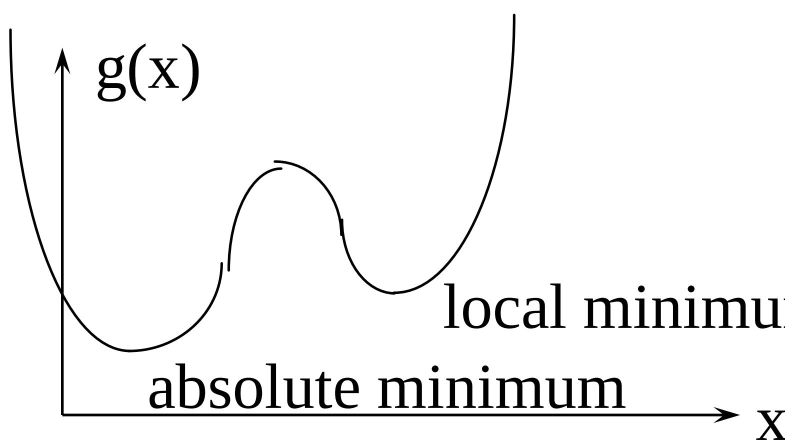
  Ex: For two sets of columns in a covering problem, the distance is the number of columns that appears in the first set but not in the second set.

- Neighborhood of a point, s, of radius, r : The set of points in the search space whose distance from s is less than r

Local Search



A Convex Optimization Problem

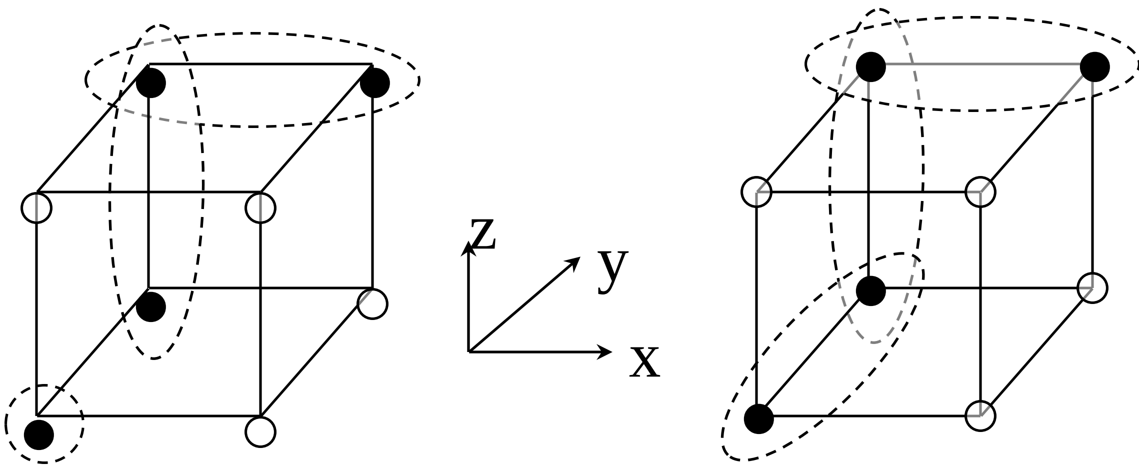


A Non-Convex Optimization Problem

Heuristic Minimization of Two-Level Logic

- Espresso
  - near optimal solution
  - fast

Expand Input, Reduce Output (Single Output)

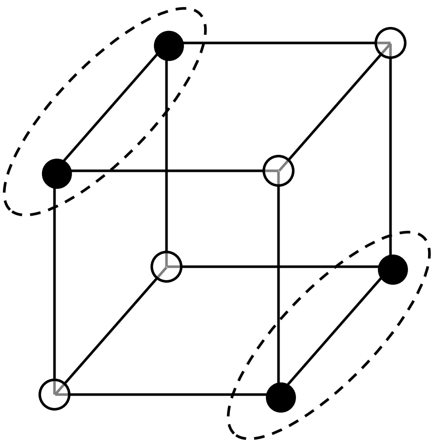


(a) (b)

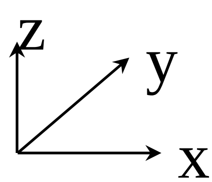
xyz	f
000	1
01-	1
-11	1

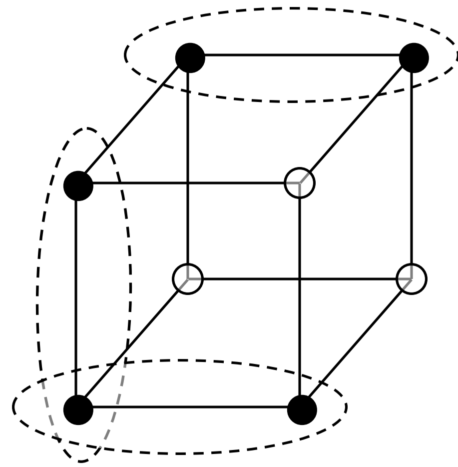
xyz	f
0-0	1
-11	1

Expand Output, Reduce Input (Multiple Output)



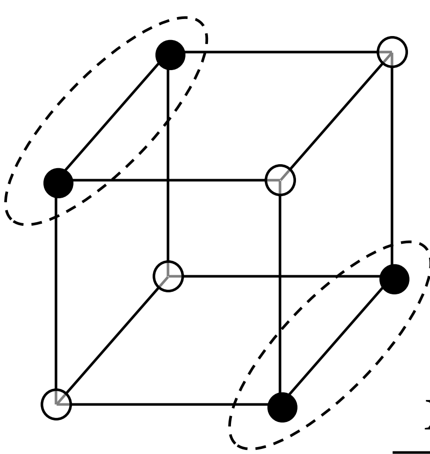
f_1



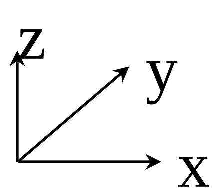


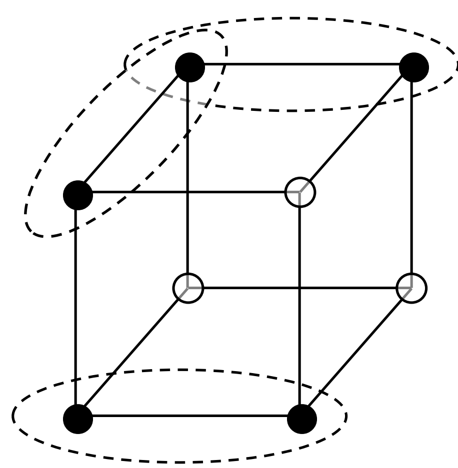
f_2

xyz	f_1f_2
0-1	10
1-0	10
00-	01
-00	01
-11	01



f_1

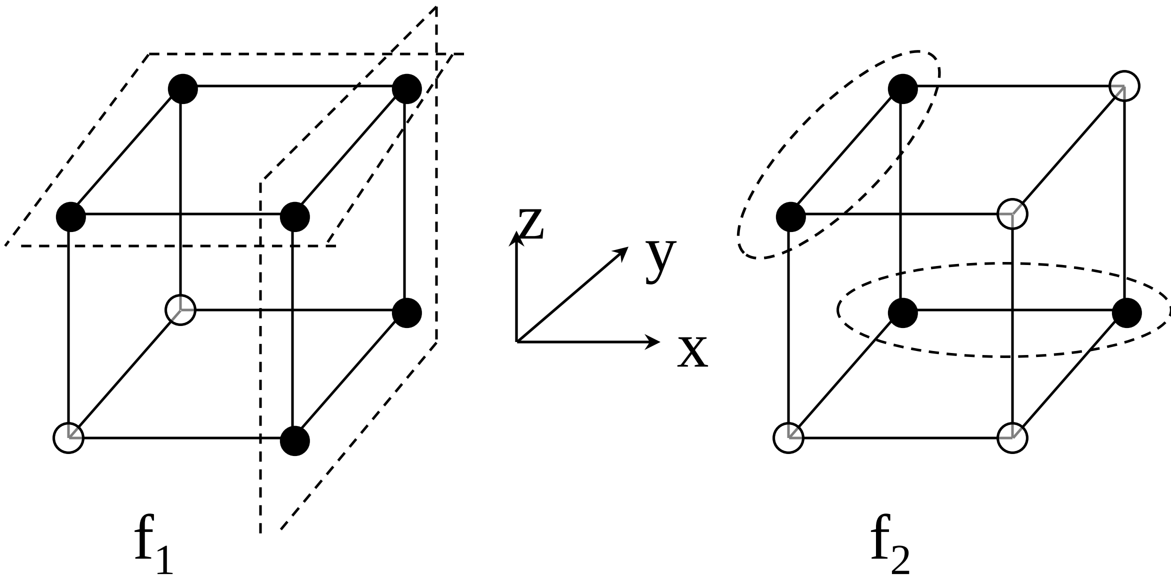




f_2

xyz	f_1f_2
0-1	11
1-0	10
-00	01
-11	01

Reduce Input, Expand Output (Multiple Output)



xyz	$f_1 f_2$
1--	10
--1	10
0-1	01
-10	01

xyz	$f_1 f_2$
1--	10
0-1	11
-10	01

Simple Minization Loop

```
F = EXPAND(F,D);
F = IRREDUNDANT(F,D);
do {
        cost = |F|;
        F = REDUCE(F,D);
        F = EXPAND(F,D);
        F = IRREDUNDANT(F,D);
} while (|F| < cost );
F = MAKE_SPARSE(F,D);
```

Expand

- *Expand*
  - Carry out one cube at a time
  - *Expand* cubes to prime and delete those cubes of F contained in the prime

Irredundant Cover

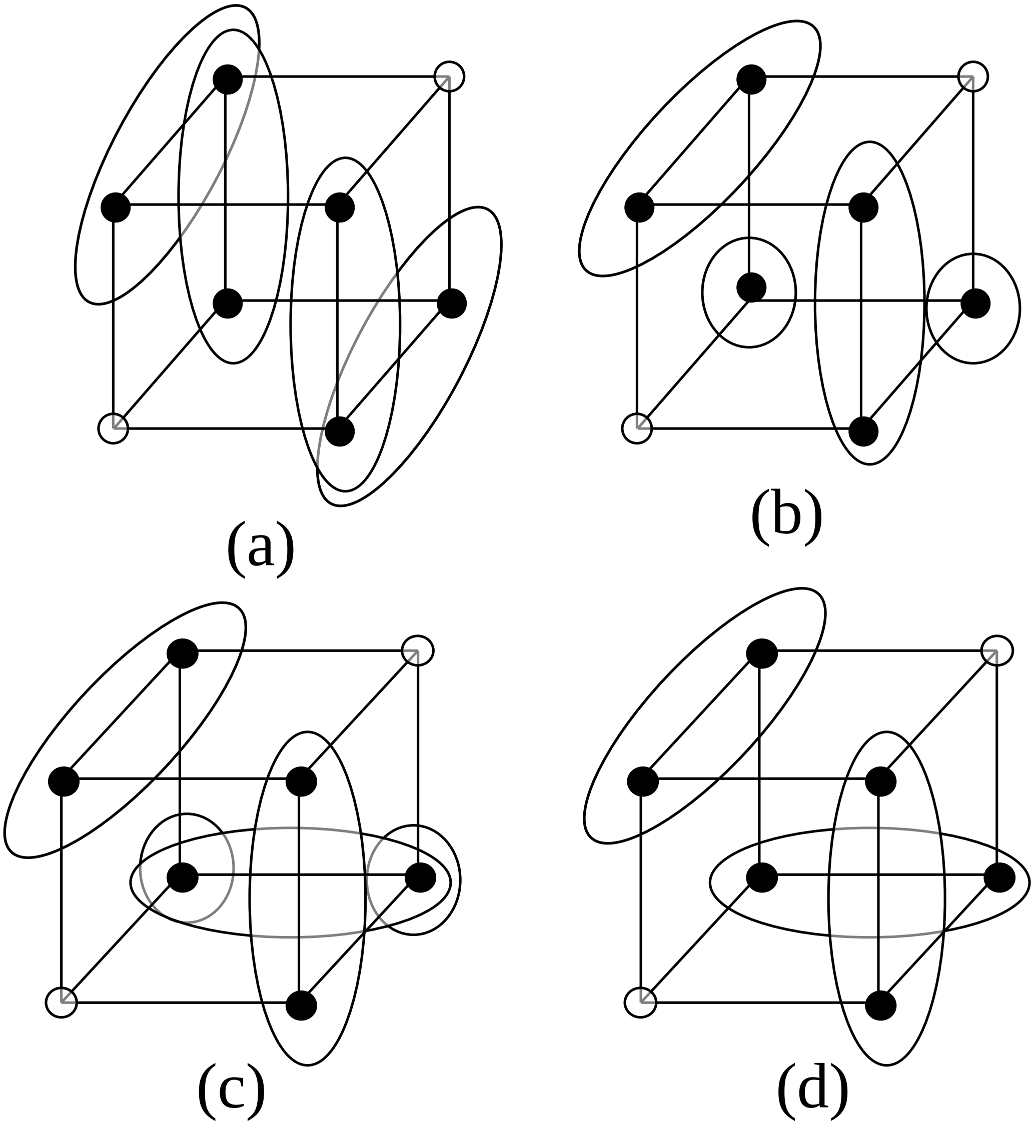
- After performing *Expand*, we have a prime cover without single cube containment now
- Find a proper subset which is also a cover (irredundant)

Reduce

- *Reduce*:

  Replace each prime by a smaller cube contained in it.
- $|\underline{F}| = |F|$ after reduce
- Since some of cubes of $\underline{F}$ are not prime, *Expand* can be applied to $\underline{F}$ to yield a different cover that may have fewer cubes
- $|\underline{F}| \leqslant |F|$ after *Expand*
- Move from locally optimal solution to a better one

Heuristic Minimization of Two-level Logic



(a) Initial cover
(b) After Reduction
(c) After Expansion in the right direction
(d) Irredundant cover

Two Principles for the Espresso

1. Decomposition
   - recursive divide and conquer by cofactor operation
   - the shannon expansion

$$f = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$$

2. Unate function
   - tautology checking
   - covering
   - essential prime

Example of Decomposition by Cofactor

Ex:

$$G = x_1x_2x_3' + x_1'x_2x_4' + x_1x_2x_3x_4$$

$$G = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$Gx_4' = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 0 & 1 & 2 & 2 \end{bmatrix}$$

$$Gx_4 = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

$$x_4'Gx_4' + x_4Gx_4 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Divide & Conquer by Cofactor Op.

- Tautology

$$f = x_i \cdot f_{x_i} + \overline{x}_i \cdot f_{\overline{x}_i}$$

$$f \equiv 1 \Leftrightarrow f_{\overline{x}_i} \equiv 1 \ \ and \ \ f_{x_i} \equiv 1$$

Unate Function

- A logic function is monotone increasing (decreasing) in a variable x_j if changing x_j from 0 to 1 causes all the outputs that change, to increase from 0 to 1 (from 1 to 0).

Ex: $f = x_1x_2' + x_2'x_3$

x_1	x_2	x_3	f	
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	0	**x_1: positive unate**
1	0	0	1	x_2: negative unate
1	0	1	1	x_3: positive unate
1	1	0	0	
1	1	1	0	

- A function is unate if it is unate in all its variables.

Unate Function

- A logic function is monotone increasing (decreasing) in a variable x_j if changing x_j from 0 to 1 causes all the outputs that change, to increase from 0 to 1 (from 1 to 0).

Ex: $f = x_1x_2' + x_2'x_3$

x_1	x_2	x_3	f	
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	0	x_1: positive unate
1	0	0	1	**x_2: negative unate**
1	0	1	1	x_3: positive unate
1	1	0	0	
1	1	1	0	

- A function is unate if it is unate in all its variables.

Unate Function

- A logic function is monotone increasing (decreasing) in a variable x_j if changing x_j from 0 to 1 causes all the outputs that change, to increase from 0 to 1 (from 1 to 0).

Ex: $f = x_1x_2' + x_2'x_3$

x_1	x_2	x_3	f	
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	0	x_1: positive unate
1	0	0	1	x_2: negative unate
1	0	1	1	**x_3: positive unate**
1	1	0	0	
1	1	1	0	

- A function is unate if it is unate in all its variables.

Unate Cover

- A cover C is said to be monotone increasing (decreasing) in the variable x_j if all the cubes in C have either 1(0) or 2 in variable x_j.
- Proposition :

  A cover is unate if it is monotone in all its variables.
- $\left|\begin{matrix} 1 & 0 & 1 \\ 2 & 0 & 1 \\ 2 & 2 & 1 \\ 1 & 0 & 2 \end{matrix}\right|$ => unate cover => unate function
- not unate cover $\overset{?}{=>}$ not unate function

$$F = \left|\begin{matrix} 1 & 1 & 0 \\ 2 & 0 & 2 \\ 1 & 0 & 0 \end{matrix}\right| \quad => \quad \left|\begin{matrix} 1 & 2 & 0 \\ 2 & 0 & 2 \end{matrix}\right|$$

- Proposition:

  A logic function f is monotone increasing (decreasing) in x_j if and only if no prime implicant of f has a 0(1) in the jth position.

Property of Unate Function

- Proposition:

  A unate cover is a tautology if and only if it contains a row of 2’s.

pf:

(<=) trivial

(=>) Assume the cover represent a monotone increasing function. Then, the function contains 1’s and 2’s of the components of all cubes.

The minterm (0,0,....0) must be covered. Unless the cover contains (2,2,....2) , the minterm (0,0,...0) will not be covered.

Other Property of Unate Function

- Proposition:

  If a logic function f is monotone increasing in x_j, then the complement of f (f ') is monotone decreasing in x_j.

- Proposition:

  The complement of a unate function is unate.

- Proposition:

  The cofactor of a unate function f is unate.

The Paradigm of Espresso

1. Apply the operation to cofactor
2. Merge the result

$$\text{operate}(f,g) = \text{merge}(\, x_i\, \text{operate}(f_{xi},g_{xi}),\; x_i'\text{operate}(f_{xi'},g_{xi'})\,)$$

The choice of the splitting variable?

=> Select splitting variable such that the cofactors made are as close as possible to a unate function.

Choice of Spitting Variable

- The choice of splitting variable when performing decomposition is guided by

  => making Cx and Cx' unate, given cover C

  - Select a binate variable to split
  - Select the "most" binate variable to split to keep Cx and Cx' size small

Ex:

$$C = \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \left| \begin{array}{cccc} 2 & 1 & 2 & 0 \\ 2 & 2 & 1 & 0 \\ 1 & 2 & 1 & 1 \\ 0 & 2 & 2 & 2 \end{array} \right| \end{array}$$

x_4 is selected as splitting variable

$$C_{x4} = \left| \begin{array}{cccc} 1 & 2 & 1 & 2 \\ 0 & 2 & 2 & 2 \end{array} \right|$$

$$C_{x4'} = \left| \begin{array}{cccc} 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 2 \\ 0 & 2 & 2 & 2 \end{array} \right|$$

Simple Minization Loop

```
F = EXPAND(F,D);
F = IRREDUNDANT(F,D);
do {
        cost = |F|;
        F = REDUCE(F,D);
        F = EXPAND(F,D);
        F = IRREDUNDANT(F,D);
} while (|F| < cost );
F = MAKE_SPARSE(F,D);
```

Irredundant Cover
(Simplified Version)

Irredundant Cover

- After performing *Expand*, we have a prime cover without single cube containment now
- Find a proper subset which is also a cover (irredundant)

Irredundant Cover

- Proposition:

  A set of cubes C covers a cube p if and only if C_p is a tautology.

  proof:

$$\Rightarrow \quad C \cap p = p$$

$$\Rightarrow \quad (C \cap p)_p = p_p$$

$$\Rightarrow \quad C_p \cap p_p = 1$$

$$\Rightarrow \quad C_p = 1$$

$$\Leftarrow \quad C_p = 1$$

$$\Rightarrow \quad C_P \cap P = P$$

$$\Rightarrow \quad C \cap P = P$$

Example

Ex: $p = 1\ 1\ 2$

$C = 2\ 1\ 0$

$\quad\ \ 1\ 2\ 1$

$C_p = 2\ 2\ 0$

$\quad\ \ 2\ 2\ 1$

Check tautology(C_p)

If (C_p) is a tautology, C covers p.

Divide & Conquer by Cofactor Op.

- Tautology

$$f = x_i \cdot f_{x_i} + \overline{x}_i \cdot f_{\overline{x}_i}$$

$$f \equiv 1 \;\Leftrightarrow f_{\overline{x}_i} \equiv 1 \;\; and \;\; f_{x_i} \equiv 1$$

- Tautology checking at terminal nodes

Tautology Check at Terminal Nodes

1. Speed-up by unate variables

Let $F(x_1,x_2,...x_n)$, x_1 : positive unate

- $F(x_1,x_2,...x_n) = x_1A(x_2,...x_n) + B(x_2,...x_n)$

  A : terms with x_1

  B : terms without x_1

  $F_{x1} = A + B \quad F_{x1}' = B$

- If ($F_{x1'} = B$) is tautology, $F_{x1} = A + B$ is a tautology.
- If ($F_{x1'} = B$) is not a tautology, F is not a tautology.

If x is positive unate, test if F is a tautology.

<=> $F_{x'}$ is a tautology

Tautology Check at Terminal Nodes

2. Other techniques
   - a row of 2's, answer 'Yes'
   - a column of all 1's or all 0's, answer 'No'
   - compute an upper bound on the no. of minterms of on-set

     Ex:

	number of minterms
0 2 0 1	2
1 1 2 2	4
1 2 1 1	2

     $2^n = 16 \quad > \quad 8$ answer 'No'

   - $n \leq 7$, test by truth table

A Possible Irredundant Cover Algorithm

- For each C_i in F
  compute t = tautology($(F \setminus C_i)_{C_i}$)
  if t then $F = F \setminus C_i$

An Example of Irredundant Cover

	a	b	c	d	e	
F=	2	0	0	1	2	C1
	2	1	1	2	0	C2
	1	0	1	2	2	C3
	1	2	1	0	2	C4
	0	1	2	2	1	C5
	1	1	0	1	2	C6
	1	2	2	1	0	C7

Checking Irredundant Cover

- $F \setminus C_3$ covers $C_3 = 1\ 0\ 1\ 2\ 2$?
  - Check tautology $(F \setminus C_3)_{C3}$

	a	b	c	d	e	
F=	2	2	2	0	2	from C4 => on-set count = 16
	2	2	2	1	0	from C7 => on-set count = 8

$16 + 8 < 32$

  - C_3 can not be removed from the cover

Checking Irredundant Cover

- F \ C_7 covers C_7 = 1 2 2 1 0 ?
  - Check tautology $(F \setminus C_7)_{C7}$

	a	b	c	d	e	
F=	2	0	0	2	2	from C1
	2	1	1	2	2	from C2
	2	0	1	2	2	from C3
	2	1	0	2	2	from C6

b	b'
2 2 1 2 2	2 2 0 2 2
2 2 0 2 2	2 2 1 2 2
tautology	tautology

  - C_7 can be removed from the cover

Local Minimal

- Order dependent
- Local optimal, not a minimum subset