Local Search

- Local search algorithms begin with initial solution and examine its neighborhood to find a feasible point with lower cost.
Convex versus Non-Convex

Convex

Non-convex

Input Expansion

<table>
<thead>
<tr>
<th>xyz</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td>01−</td>
<td>1</td>
</tr>
<tr>
<td>−11</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xyz</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0−0</td>
<td>1</td>
</tr>
<tr>
<td>01−</td>
<td>1</td>
</tr>
<tr>
<td>−11</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xyz</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0−0</td>
<td>1</td>
</tr>
<tr>
<td>−11</td>
<td>1</td>
</tr>
</tbody>
</table>
**Output Expansion**

```
xyz | fg
---|---
-1  0  1  | 10
 1  0  1  | 10
 0  0  1  | 10
-1  0  1  | 01
 1  0  1  | 01
-1  1  0  | 11
-1  1  1  | 11
```

**Input Reduction**

```
xyz | fg
---|---
-1  0  1  | 10
 1  0  1  | 10
 0  0  1  | 10
-1  0  1  | 01
 1  0  1  | 01
-1  1  0  | 11
-1  1  1  | 11
```

Simple Minimization Loop

\[ F = \text{Expand}(F, D); \]
\[ F = \text{Irredundant}(F, D); \]
do {
    cost = \|F\|;
    F = \text{Reduce}(F, D);
    F = \text{Expand}(F, D);
    F = \text{Irredundant}(F, D);
} while (\|F\| < \text{cost});
\[ F = \text{Make}\_\text{Sparse}(F, D); \]
Checking for Equivalence

- When a cube is expanded the new minterms added must not be in the OFF-set.
- When a cube is reduced the removed minterms must be covered by other cubes or belong to the DC-set.
- If $F$ is the cover, $D$ the don’t cares, $c_i$ the cube being expanded or reduced, and $c_i'$ the minterms added or removed:

$$c_i' \leq (F - \{c_i\}) \cup D.$$ 

<table>
<thead>
<tr>
<th>xyz</th>
<th>f</th>
<th></th>
<th>xyz</th>
<th>f</th>
<th></th>
<th>xyz</th>
<th>(F- {c_i})</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td></td>
<td>c_i : 000</td>
<td>1</td>
<td>01-</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01-</td>
<td>1</td>
<td></td>
<td>0-0</td>
<td>1</td>
<td>-11</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>1</td>
<td></td>
<td>c_i' : 010</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In general, a term may be contained in a set of terms: $xz \quad xy + y'z$
Containment and Tautologies

- Recall: \( f = x f_x + x' f_x \).
- Cofactoring is commutative: \((f_{x1})_{x2} = (f_{x2})_{x1}\).
- The cofactor of a function \( f \) with respect to a cube \( c \) is the successive cofactoring of \( f \) with respect to all the literals in \( c \).
- A function identically 1 is a *tautology*.

---

**Theorem 5.2.1:** for a function \( f \) and cube \( c \),

\[
c f \iff f_c 1.
\]

- Observation: \( g f \Rightarrow g_c f_c \).
- Proof:

\[
c f \Rightarrow c_c f_c \iff 1 f_c \iff f_c = 1.
\]

- Cofactoring simplifies the function.
- We deal with a uniform problem--tautology.
Tautology Examples

• Is $xz \ xy + y'z$?
• $c = xz$ and $f = xy + y'z$.
• $f_c = y + y' = 1$.

\[
\begin{array}{c|cc}
xyz & fg \\
\hline
1-- & 10 \\
--1 & 10 \\
0-1 & 01 \\
-10 & 01 \\
\end{array}
\]

$c_i = --1 | 10$

\[
\begin{array}{c|cc}
xyz & fg \\
\hline
1-- & 10 \\
--1 & 10 \\
0-1 & 01 \\
-10 & 01 \\
\end{array}
\]

$c_i' = 1-1 | 10$

\[
\begin{array}{c|c}
y & f \\
\hline
- & 1 \\
\end{array}
\]

Multiple Output Cofactors

• Eliminate the rows that disagree with the input part of $c_i$.
• A disagreement means the row has a 1 in an input column where $c_i$ has a 0 or vice versa.
• Eliminate the rows that do not have at least a 1 in an output column where $c_i$ has a 1.
Unate Functions

• A function \( f(x_1, x_2, ..., x_n) \) is *monotonically increasing* in variable \( x_j \) iff:
  \[
  f(0, x_2, ..., x_n) \leq f(1, x_2, ..., x_n), \quad \forall (x_2, ..., x_n).
  \]
• It is *monotonically decreasing* in \( x_j \) iff:
  \[
  f(0, x_2, ..., x_n) \geq f(1, x_2, ..., x_n), \quad \forall (x_2, ..., x_n).
  \]
• If neither is true, \( f \) is *non-monotonic* in \( x_j \).
• \( f(x_1, x_2, ..., x_n) \) is *unate* iff it is monotonic increasing or decreasing in all variables.

Unate Covers

• A cover \( F \) is *monotonic increasing* in \( x_j \) iff \( x_j \) never appears complemented in \( F \).
• A cover \( F \) is *monotonic decreasing* in \( x_j \) iff \( x_j \) never appears uncomplemented in \( F \).
• Theorem 5.2.2:
  – If function \( f \) is unate in \( x_j \) then there exists a cover of \( f \) unate in \( x_j \).
  – If a cover \( F \) is unate in \( x_j \) then the function \( F \) represents is also unate in \( x_j \).
Unate Covers and Tautologies

• Theorem 5.2.3: A unate cover $F$ is a tautology iff it contains the constant term 1.
• Observation: $f = 1 \iff f_x = f_{x'} = 1$.
• To check if a cover $F$ is a tautology,
  – Apply recursive cofactors until they are unate.
  – If a cofactor is unate, it must contain a term of all ‘-’ (don’t cares) in the input part.
  – If all cofactors are tautologies, then $F$ is also.
  – If one cofactor is not a tautology, then $F$ is not.

Tautology Check Example

<table>
<thead>
<tr>
<th>$xyz$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$yz$</th>
<th>$f_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$yz$</th>
<th>$f_{x'}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>
A Further Simplification

• If $F$ is monotonic increasing in $x_j$:

$$F(x_1,\ldots,x_n) = x_1 A(x_2,\ldots,x_n) + B(x_2,\ldots,x_n)$$

$$F_{x_j} = A + B \quad F_{x_j'} = B.$$  

$$
\begin{array}{ccc}
F_{x_j'} & F_{x_j} \\
F_{x_j'} & 1 \Rightarrow F_{x_j} & 1 \\
F_{x_j'} & 1 \Rightarrow F & 1 \\
F_{x_j'} & 1 \Rightarrow F & 1 \\
F_{x_j'} & 1 \Leftrightarrow F & 1
\end{array}
$$

Additional Speed-Up Techniques

• A row of ‘-’ implies that the function is tautologous in all outputs with a ‘1’.

• An input column of all 1s or 0s implies the function is not a tautology.

• If there are less than 8 inputs, then generate a truth table.

• If the vertex count is insufficient, then it is not a tautology (i.e., $2^d$ for each cube).
Partitioning

• If a cover $F$ can be written as:
  $$F = G + H$$
where $G$ and $H$ have disjoint support, then $F$ is a tautology iff either $G$ or $H$ is one.

• To construct a bipartition $(C_1, C_2)$:
  – Pick a cube and put all columns w/0 or 1 in $C_1$.
  – Select all cubes that intersect $C_1$, and add their columns with 0 or 1 to $C_1$.
  – Repeat until $C_1$ does not change.

Partitioning Example

1 2 3 4 5
1 1-1--
2 -1--0
3 0--0-
4 --01-
5 -----1

1 11-
1 1
3 0-0
4 -01

25
2 10
5 -1
Choosing the Right Direction

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Recursive Complementation

- **Theorem 5.3.1**: for a boolean function $f$,
  \[ f' = x f'_x + x' f'_x. \]
- To compute the complement of a cover $F$,
  - Recursively cofactor until a single cube.
  - Apply DeMorgan’s laws to get a complement.
  - Merge cubes that occur in both cofactors (i.e., $xc + x’c = c$).
Using the OFF-set in Expansion

- A cube can be expanded if it does not intersect any cube of the OFF-set.
- The expanded cube must conflict in at least one position with each cube in OFF-set.
- The blocking matrix has a row for each cube in the OFF-set, a column for each variable, and ones where there is a conflict.
- Maximum expansion requires finding the minimum set of columns to cover all rows.
Irredundant Step

- Can be achieved with a covering problem of manageable size.
- Find the minimum subset of the cubes in the current cover that covers all minterms in the ON-set.

Essential Primes

- Should identify all essential primes because they will be part of every optimal solution.
- After initial expansion, all cubes in cover are primes and all essentials are present.
- Should put essentials aside to avoid silly things like reducing them.
Essential Primes

• Theorem 5.4.1: $F$ is a cover of primes, $e$ is one of the primes, and $G$ is remaining primes. Then, $e$ is an essential prime iff it is not covered by the union of:
  – The consensus terms of $e$ and each term of $G$.
  – The intersections of $e$ and each term of $G$.

Essential Primes Example

\[ y'z' + xy' + xz \]

• Test $y'z'$:
  – Intersects $xy'$ (intersection is $xy'z'$).
  – Consensus with $xz$ ($xy'$).
  – $y'z' \ xy' + xy'z' = y'z' \ xy'$? Essential.

• Test $xy'$:
  – Intersects $xz$ (intersection is $xy'z$).
  – Consensus with $y'x'$ ($xy'z'$).
  – $xy' \ xy'z + xy'z' = xy'$? Not essential.
Essential Primes Example

\[ x'z' + x'y + xz \]

- Test \( x'y \):
  - Intersects with \( x'z' \) (\( x'yz' \)).
  - Consensus with \( xz \) (\( yz \)).
  - \( x'y \quad x'yz' + yz \).
  - Cofactor right-side: \( z' + z \) (tautology).
  - Hence, \( x'y \) is essential.

---

Multiple-Valued Logics

- Let \( P_i = \{0, \ldots, p_i -1\} \) and \( B = \{0, 1\} \).

- A multiple-valued input, binary-valued output function \( f \) is:
  \[ f: P_1 \quad \ldots \quad P_n \rightarrow B. \]

- Let \( X_i \) be a variable over \( P_i \) and let \( S_i \) be a subset of \( P_i \), then \( X_i^{S_i} \) is a mapping:
  \[ 
  \begin{array}{c|c}
  0 & \text{if } X_i \in S_i \\
  1 & \text{if } X_i \not\in S_i \\
  \end{array} 
  \]
Multiple-Valued Logics

• $X_i^S_i$ is a literal of variable $X_i$.
• We build SOP formulaes in same way.
• Can define implicants, prime implicant, etc.
• Many laws of Boolean algebras still hold:
  \[ f = X_i^S_i f_{X_iS_i} + X_i^{S_i'} f_{X_iS_i'} \]
• **Espresso-mv** uses multi-valued functions.