Logic Optimization Techniques

- Logic Optimization Techniques
  - K-maps (Graphical)
  - Quine-McCluskey (Exact Algorithm)
    - Tabular Minimization
    - Row/Column Dominance
    - Espresso (Heuristic) – we’ll see this one soon

- Other Generalized Algorithms
  - **Branch-and-bound**
  - Simulated Annealing
  - many more exists ...
    - Integer Linear Programming (ILP)
    - Dynamic Programming
    - Genetic Algorithms

**Very general algorithm – can be applied to a variety of problems**
- Based on the idea of a decision tree
- Varies in that it tries to visit only part of the tree

---

**Decision Trees**

- Decision tree
  - Enumeration approach in which we have $n$ decision variables, and list the $2^n$ possible values

Given a prime implicant chart and the corresponding essential prime implicants, how do we derive a minimum cover with the remaining prime implicants?

---

Remove the essential prime implicants, they are already in the cover.
Number the remaining prime implicants so it’s easier for us to read.

Let’s start our decision tree. What are the decision to make?
Should we include P1 in our cover?
Should we include P2 in our cover?
Should we include P3 in our cover?
Should we include P4 in our cover?
Decision Trees

The leaves are your possible solutions

Cover = P1, P2, P3, P4
Valid? yes, Cost = 4

Cover = P1
Valid? no

Cover = P2, P3, P4
Valid? yes, Cost = 3

Cover = P2, P3
Valid? yes, Cost = 2

Cover = P1, P4
Valid? yes, Cost = 2

Cover = P3, P4
Valid? yes, Cost = 2

Cover = none
Valid? no

Branch-and-bound Idea

- Several optimal solutions may exist, we only need to find one
- Idea is that maybe we only have to visit part of the decision tree
- If we can estimate the low bound to a subtree, and that low bound is higher than the current minimum, we don't need to look at that subtree

Branch-and-bound Pseudocode

```plaintext
BCP( F, U, currentSol ){ // Initial call to BCP
    ( F, currentSol ) = REDUCE( F, currentSol )
    if ( terminalCase( F ) ){ // currentSol set to empty
        if ( cost( currentSoln ) < U ){ // Upper bound (U) set to the number of decisions (prime implicants) + 1
            U = cost( currentSoln )
            return ( currentSoln )
        }
    }
    L = LOWER_BOUND( F, currentSoln ) // Guarantees that the first valid solution found will be accepted
    if ( L ≤ U ) return ( "no solution" )
    x_i = CHOOSE_VAR( F ) // F is the current constraint equation
    S^i = BCP( F_{x_i}, U, currentSoln ∪ { x_i } ) // Call to REDUCE(F)
    if ( cost( S^{i} ) = L ) return ( S^{i} ) // Try to simplify the matrix by recursively
    S^{i'} = BCP( F_{x_i}, U, currentSoln ) // Removing essential columns and adding it to currentSoln
    return BEST_SOLUTION( S^{i}, S^{i'} ) // Remove dominating rows
    // Remove dominated columns
    // Continue until matrix is empty, or problem is cyclic
    // Potential candidates?
    // Column that covers many rows is more likely to be part of optimal solution
    // Column that covers many short rows since short rows have a lower chance of being covered
```

Iteratively finds essential variables and applies row/column dominance to simplify matrix – updates current solution with these changes

Is currentSoln a valid solution?
If valid and better than existing solution, update solution and cost

Calculate the lower bound of the subtree to see if the subtree is worth looking at
Make a decision – which prime do we want to include/exclude?
- no effect on correctness, help with efficiency of runtime

Recursive call on subtree that includes the prime
If S^{i} subtree contained the low cost solution ignore S^{i'} subtree, otherwise recursive call on subtree excluding the prime

Return best solution

Iteratively finds essential variables and applies row/column dominance to simplify matrix – updates current solution with these changes
How do I calculate the lower bound of a subtree?
- Varies depending on your problem
- Minimum cover problem
  - lower bound = number of prime implicants (columns committed so far) + MIS

Maximally Independent Set (MIS)
- Equal to the number of independent rows in the table
-_rows are independent if no overlapping X's
- Indicates the lowest possible number of prime implicants required to cover the remaining minterms
- Better (higher value) low bound leads to more pruning
- Low bound = 1, you don't get much pruning
- Low bound = 5, you have better chance of pruning
- If no independent rows are found, the lower bound for a cyclic matrix is at least 2
  - If matrix cyclic no column covers all rows (which would have enabled reduction of matrix)
  - Thus, a minimum of two columns are required to cover all rows

Branch-and-bound – Lower Bound Calculation

Finding MIS
MIS_QUICK Heuristic
- Simple algorithm can be used to find MIS
  - || M || denotes rows left in M after deleting rows intersecting with row i

CHOOSE_SHORTEST_ROW subprocedure can be done in several ways
- Option 1 - Row i is row with the fewest nonzero columns, breaking ties in lexicographical order
- Option 2 - Row i is selected by column counts of its columns, breaking ties in lexicographical order
  - Does a better job finding larger MIS

\[
\text{MIS} = \{1, 2\} \\
\text{MIS} = \{3, 4\} \\
\text{MIS} = \{5, 6\} \\
\text{MIS} = 2
\]

MIS_QUICK Example
- Use MIS_QUICK (option 1) to find MIS
  - \[ \text{MIS} = \emptyset \]

- \[ \text{MIS} = \{1\} \]
  - Add row 1 to MIS
  - Delete intersecting rows (2, 7, 4)

- \[ \text{MIS} = \{1, 3\} \]
  - Add row 3 to MIS
  - Delete intersecting rows (5, 6)

- \[ \text{MIS} = \{1, 3\} \]
  - Low bound = 0 + 2 (no essentials previously added)

MIS_QUICK Example
- Use MIS_QUICK (option 2) to find MIS
  - \[ \text{MIS} = \emptyset \]

- \[ \text{MIS} = \{1\} \]
  - Add row 1 to MIS
  - Delete intersecting rows (2, 7, 4)

- \[ \text{MIS} = \{1\} \]
  - Add row 5 to MIS
  - Delete intersecting rows (3)

- \[ \text{MIS} = \{1, 5, 6\} \]
  - Add row 6 to MIS
  - Matrix empty - Done!

- \[ \text{MIS} = \{1, 5, 6\} \]
  - Low bound = 0 + 3 (no essentials previously added)

Option 2 found a larger MIS set which leads to higher lower bound (i.e. more pruning)
Branch-and-bound

Example 1

- Using Branch-and-bound find minimum cover

Call to BCP( F, U, (P1))

1. Initialize best solution (F) and current cost (U) variables
2. Reduce matrix
4. Calculate lower bound on subtree
   - MIS \text{QUICK} returns \{m1\}, but matrix is cyclic
   - so MIS is at least 2
   - Lower bound \( L = \# \text{prime implicants} + \text{MIS} = 0 + 2 = 2 \)
5. \( L \geq U? \) No.
6. \( x_i = P1 \)
7. \( S_1 = \text{BCP} (F_{x_i}, U, \text{currentSoln} \cup x_i) \)

Done! All options examined.

Branch-and-bound

Example 2

- Using Branch-and-bound find minimum cover

BCP( F, U, (P1))

1. Initialize best solution (F) and current cost (U) variables
2. Reduce matrix
4. Calculate lower bound on subtree
   - MIS \text{QUICK} returns \{m1, m3, m5, m7\}
   - lower bound \( L = \# \text{prime implicants} + \text{MIS} = 0 + 4 = 4 \)
5. \( L \geq U? \) No.
6. \( x_i = P1 \)
7. \( S_1 = \text{BCP} (F_{x_i}, U, \text{currentSoln} \cup x_i) \)

Returns from here with updated F and U

Done! All options examined.
Example 2

**Branch-and-bound**

BCP($F_p$, $U$, ($P_1$))

1. Calculate lower bound on subtree
2. Reduce matrix

Example 2

**Branch-and-bound**

BCP($F_p$, $U$, ($P_1, P_5$))

1. Calculate lower bound on subtree
2. Reduce matrix

Example 2

**Branch-and-bound**

BCP($F_p$, $U$, ($P_1$, $P_5$, $P_6$))

1. Calculate lower bound on subtree
2. Reduce matrix

Example 2

**Branch-and-bound**

BCP($F_p$, $U$, ($P_1$, $P_5$, $P_6$, $P_8$))

1. Calculate lower bound on subtree
2. Reduce matrix
Branch-and-bound

Example 2

BCP( F_{P_{13}}, U, (P_1, P_5))

1. Reduce matrix
2. Solution found? No.
3. Calculate lower bound on subtree
   MIS_QUICK returns 2 (No independent sets)
   lower bound \((L) = 3 + 2 = 5\)
4. \(L \geq 7\) No.
5. \(x_1 = P_6\)
6. \(5' = BCP( F_{P_{13}}, U, \text{currentSoln} \cup x_1)\)
7. Cost \((5') = 1\) Yes.
   \(5'\) subtree.
8. Cost \((5') = 5\) No.
9. Return BEST_SOLUTION(5', 5')

Solution = \(\{P_1, P_3, P_5, P_6, P_8\}\)
Cost = 5

Example 2

BCP( F_{P_{13}}, U, (P_1))

1. Reduce matrix
2. Solution found? No.
3. Calculate lower bound on subtree
   MIS_QUICK returns 2 (No independent sets)
   lower bound \((L) = 3 + 2 = 5\)
4. \(L \geq 7\) No.
5. \(x_1 = P_5\)
6. \(5' = BCP( F_{P_{13}}, U, \text{currentSoln} \cup x_1)\)
7. Cost \((5') = 1\) Yes.
   \(5'\) subtree.
8. Cost \((5') = 5\) No.
9. Return BEST_SOLUTION(5', 5')

Solution = \(\{P_1, P_3, P_5, P_6, P_8\}\)
Cost = 5

Example 2

BCP( F_{P_{13}}, U, (P_1, P_5))

1. Reduce matrix
2. Solution found? No.
3. Calculate lower bound on subtree
   MIS_QUICK returns 2 (No independent sets)
   lower bound \((L) = 3 + 2 = 5\)
4. \(L \geq 7\) No.
5. \(x_1 = P_6\)
6. \(5' = BCP( F_{P_{13}}, U, \text{currentSoln} \cup x_1)\)
7. Cost \((5') = 1\) Yes.
   \(5'\) subtree.
8. Cost \((5') = 5\) No.
9. Return BEST_SOLUTION(5', 5')

Solution = \(\{P_1, P_3, P_5, P_6, P_8\}\)
Cost = 5

Example 2

BCP( F_{P_{13}}, U, (P_1))

1. Reduce matrix
2. Solution found? No.
3. Calculate lower bound on subtree
   MIS_QUICK returns 2 (No independent sets)
   lower bound \((L) = 3 + 2 = 5\)
4. \(L \geq 7\) No.
5. \(x_1 = P_5\)
6. \(5' = BCP( F_{P_{13}}, U, \text{currentSoln} \cup x_1)\)
7. Cost \((5') = 1\) Yes.
   \(5'\) subtree.
8. Cost \((5') = 5\) No.
9. Return BEST_SOLUTION(5', 5')

Solution = \(\{P_1, P_3, P_5, P_6, P_8\}\)
Cost = 5

Example 2

BCP( F_{P_{13}}, U, (P_1, P_5))

1. Reduce matrix
2. Solution found? No.
3. Calculate lower bound on subtree
   MIS_QUICK returns 2 (No independent sets)
   lower bound \((L) = 3 + 2 = 5\)
4. \(L \geq 7\) No.
5. \(x_1 = P_6\)
6. \(5' = BCP( F_{P_{13}}, U, \text{currentSoln} \cup x_1)\)
7. Cost \((5') = 1\) Yes.
   \(5'\) subtree.
8. Cost \((5') = 5\) No.
9. Return BEST_SOLUTION(5', 5')

Solution = \(\{P_1, P_3, P_5, P_6, P_8\}\)
Cost = 5
Branch-and-bound

Example 2

BCP( F, U, ⟨⟩)

1. Call to BCP( F, U, ⟨⟩)
   - F = {P1, P3, P5, P6, P8}
   - U = 5
     - L = 4
     - m3
     - m4
     - m5
     - m6
     - m7
     - m8
     - m9
     - m10
     - m11
     - m12
     - m13
   - Solution found? No.
   - Cost: L = 4

2. Reduce matrix


4. Calculate lower bound on subtree
   - MIS_QICK returns (m7, m5)
   - lower bound (L) = # prime implicants + MIS
     - L = 5
   - Solution found? No.
   - Cost: L = 5

5. L ≥ U? Yes.

6. xi = P1

7. S' = BCP( F, U, currentSoln ∪ x)
   - S' = BCP( F, U, {m5, m7})

8. Cost( S') = L? No.

9. S'' = BCP( F, U, currentSoln ∪ x)
   - S'' = BCP( F, U, {P1, P3, P5, P6, P8, P10})
   - Returns from here with updated F and U

Done! All options examined.

Branch-and-bound

Example 2

Call to BCP( F, U, ⟨P1'⟩)

1. Call to BCP( F, U, ⟨P1'⟩)
   - F = {P1, P3, P5, P6, P8}
   - U = 5
     - L = 4
     - m3
     - m4
     - m5
     - m6
     - m7
     - m8
     - m9
     - m10
     - m11
     - m12
     - m13
   - Solution found? No.
   - Cost: L = 5

2. Reduce matrix


4. Calculate lower bound on subtree
   - MIS_QICK returns (m6, m7)
   - lower bound (L) = # prime implicants + MIS
     - L = 6
   - Solution found? No.
   - Cost: L = 6

5. L ≥ U? Yes.

6. xi = P1

7. S' = BCP( F, U, currentSoln ∪ x)
   - S' = BCP( F, U, {m5, m7})

8. Cost( S') = L? No.

9. S'' = BCP( F, U, currentSoln ∪ x)
   - S'' = BCP( F, U, {P1, P3, P5, P6, P8, P10})
   - Returns from here with updated F and U

Done! All options examined.

Branch-and-Bound Summary

- **Branch-and-Bound** algorithm used to help determine a minimal cover
  - We have a set of possible prime implicants to choose from (i.e. P1, P2, P3, P4)
  - Methods to choose splitting variable – we skipped
  - Solution still optimal, maybe just slower

- Which one should we choose first?
  - Methods to choose splitting variable – we skipped
  - Solution still optimal, maybe just slower

- Determining the lower bound is very important
  - We want to be accurate so we don't waste our time
  - However, this step should still be fast

- Additionally, as prime implicants are added, we can use row/column dominance to try and simplify remaining matrix
  - Helps to speed up algorithm

- Solution is exact (optimal), running time varies on selection process and bounding calculation
Logic Optimization Techniques

- Logic Optimization Techniques
  - K-maps (Graphical)
  - Quine-McCluskey (Exact Algorithm)
  - Espresso (Heuristic)

- Other Generalized Algorithms
  - Branch-and-bound
  - Simulated Annealing
  - many more exists ...
    - Integer Linear Programming (ILP)
    - Dynamic Programming
    - Genetic Algorithms

Simulated Annealing - Background

- Simulated Annealing
  - Name and inspiration come from annealing in metallurgy
  - Heating and controlled cooling of a material to reduce defects/increase strength

- Applied to local search methodology to avoid getting stuck at the local minimum

Simulated Annealing Pseudocode

```plaintext
Simulated_Annealing{
    S = initial solution
    T = initial temperature (>0)
    while( T > 0 ){
        S' = pick a random neighbor to S
        C = cost of S – cost of S'
        if( C > 0 ){ 
            S = S' 
        } else{ 
            r = random number in range [0…1]
            m = 1/\text{e}^{C/T}
            if( r < m ){
                S = S'
            }
        }
        T = reduced T;
    }
}
```

General Simulated Annealing Pseudocode

- Derive a new solution $S'$, by randomly making a change to the current solution
- Decrease the temperature
- This is the cooling schedule – how fast does the temperature decrease?

Simulated Annealing – Cooling Schedules

- Choosing initial temperature and cooling schedule has great impact on the algorithm
  - Make sure we run long enough to find a good solution
  - Make sure we get out of local optimum (take chances on worse solutions)
  - Many options available, no definitive way to choose these

Simulated Annealing Cooling Schedules

- Brian T. Luke, Ph.D.
Simulate Annealing – Example

- How do we apply to the minimum cover problem?

1. Choose an initial solution, set an initial temperature

2. Is temperature \( T > 0 \)? Yes

3. Make a random change to \( S \)
   - What can we change?
     - Adding another prime implicant to our cover \( \checkmark \) Remove P2
     - Removing a prime implicant from the current cover

4. Determine cost difference
   \[ C = \text{cost of } S - \text{cost of } S' \]
   \( C = 4 - 3 = 1 \)
   We should also consider if this solution is correct. (Yes)
   
   a) Is the solution better? Yes.
      Keep new solution

5. Decrease Temperature
   \[ T = T - 25 = 75 \]

6. Is temperature \( T > 0 \)? Yes

3. Make a random change to \( S \)
   - What can we change?
     - Adding another prime implicant to our cover
     - Removing a prime implicant from the current cover

4. Determine cost difference
   \[ C = \text{cost of } S - \text{cost of } S' \]
   \( C = 4 - 3 = 1 \)
   We should also consider if this solution is correct. (Yes)

   a) Is the solution better? Yes.
      Keep new solution

5. Decrease Temperature
   \[ T = T - 25 = 25 \]

6. Is temperature \( T > 0 \)? Yes

3. Make a random change to \( S \)
   - What can we change?
     - Adding another prime implicant to our cover
     - Removing a prime implicant from the current cover

4. Determine cost difference
   \[ C = \text{cost of } S - \text{cost of } S' \]
   \( C = 4 - 3 = 1 \)
   We should also consider if this solution is correct. (Yes)

   a) Is the solution better? Yes.
      Keep new solution

5. Decrease Temperature
   \[ T = T - 25 = 0 \]
Simulate Annealing – Example

1. Is temperature $T > 0$? No

   Done!
   Solution: P4, P5, P6

   $S = \begin{bmatrix}
   P1 & P2 & P3 & P4 & P5 & P6 \\
   \hline
   s1 & & & & & \\
   s2 & & & & & \\
   s3 & & & & & \\
   s4 & & & & & \\
   \end{bmatrix}$

   $T = \begin{bmatrix}
   P1 & P2 & P3 & P4 & P5 & P6 \\
   \hline
   0 & & & & & \\
   \end{bmatrix}$

   - Is this solution optimal?
     - No
   - Ideally, this algorithm would run longer so we can explore more of the solution space and possibly find a better solution

Conclusion

- Considered several logic optimization techniques
  - K-maps
  - Quine-McCluskey
  - Espresso

- Considered several other generalized algorithms useful for logic optimization as well as other applications
  - Branch-and-bound
  - Simulated Annealing
  - Many more exist...
    - Integer Linear Programming (ILP)
    - Dynamic Programming
    - Genetic Algorithms