LECTURE 21 – Part 1

Dealing with NP-completeness

• Part 1: Solving SAT
  Slides from Prof. S. Malik of Princeton University

• Part 2: Solving a logic minimization problem
SAT in a Nutshell

• Given a Boolean formula (propositional logic formula), find a variable assignment such that the formula evaluates to 1, or prove that no such assignment exists.

\[ F = (a + b)(a' + b' + c) \]

• For \( n \) variables, there are \( 2^n \) possible truth assignments to be checked.

• First established NP-Complete problem.
Problem Representation

- Conjunctive Normal Form
  - \( F = (a + b)(a' + b' + c) \)

- Simple representation (more efficient data structures)

- Logic circuit representation
  - Circuits have structural and direction information

- Circuit – CNF conversion is straightforward

\[
\begin{align*}
d &\equiv (a + b) \\
(a + b + d') &\equiv (a' + d) \\
(b' + d) &\equiv (c' + d' + e) \\
(c &\cdot d) &\equiv (d + e') \\
(c + e')
\end{align*}
\]
Why Bother?

- Core computational engine for major applications
  - Electronic Design Automation (EDA)
    - Testing and Verification
    - Logic synthesis
    - Field Programmable Gate Array (FPGA) routing
    - Path delay analysis
    - And more…
  - AI
    - Knowledge base deduction
    - Automatic theorem proving
DLL Algorithm

- Davis, Logemann and Loveland


- Also known as DPLL for historical reasons

- Basic framework for many modern SAT solvers
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)
Basic DLL Procedure - DFS

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d' ) \\
(a + c' + d) \\
(a + c' + ) \\
(b' + c' + ) \\
(a' + b + ) \\
(a' + b' + ) \\
(c)
\end{align*}
\]
Basic DLL Procedure - DFS

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(a' + b + c')\]
\[(a' + b' + c)\]
Basic DLL Procedure - DFS

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + +)\]
\[(b' + c' +)\]
\[(a' + b +)\]
\[(a' + b' +)\]
\[(a' + b' + c)\]
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c' + d')
(a + c' + d)
(a + c' + 
(b' + c' +
(a' + b +
(a' + b' + c)

(⇐ Decision)
Basic DLL Procedure - DFS

(a' + b + c)  
(a + c + d)  
(a + c + d')  
(a + c' + d)  
(a + c' + )  
(b' + c' + )  
(a' + b + )  
(a' + b' + )

Implication Graph

Conflict!
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

Implication Graph

Conflict!
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

⇐ Backtrack

Design
Basic DLL Procedure - DFS

\[(a' + b + c)
(a + c + d)
(a + c + d')
(b' + c' + d)
(a' + b + c)
(a' + b' + c)\]

\[\begin{align*}
a &= 0 \\
b &= 0 \\
c &= 1 \\
d &= 1
\end{align*}\]

\[\begin{align*}
d &= 0 \\
c &= 1 \\
a &= 0
\end{align*}\]

Conflict!

\[\begin{align*}
a &= 0 \\
b &= 0 \\
c &= 1 \\
d &= 1
\end{align*}\]
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

⇐ Backtrack
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + c')
(a' + b + c)
(a' + b' + c)

Forced Decision
Basic DLL Procedure - DFS

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(b' + c' + (a' + b + (a' + b' + c))\]

```
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(b' + c' +
(a' + b +
(a' + b' + c)
```

Conflict!

Decision

\[\text{Conflict!}\]
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

\[ \begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' +) \\
(b' + c' +) \\
(a' + b +) \\
(a' + b' +) \\
c
\end{align*} \]

\[ \begin{align*}
a & \quad 0 \\
b & \quad 0 \quad 1 \\
c & \quad 0 \quad 1 \\
c & \quad 0
\end{align*} \]

⇐ Backtrack
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + )
(b' + c' + 
(a' + b + 
(a' + b' + 

(a' + b + c)
(a + c + d)
(a + c + d')

(d=1)
(c=1)
(a=0)
(d=1)
(c=1)
(d=0)

Conflict!

Forced Decision
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + 
(b' + c' +
(a' + b +
(a' + b' +
c)

⇐ Backtrack
Basic DLL Procedure - DFS

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(a' + b + c)\]
\[(b' + c' + 1)\]
\[(a' + b + c)\]
\[(a' + b' + c)\]

\[\Rightarrow\text{Forced Decision}\]
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + )
(b' + c' + 
(a' + b +
(a' + b' + c)

Diagram:

Decision

11/25/08 L21.22
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + )
( + c' +
(a' + b +
(a' + b' + c)

Conflict!
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c' + d)
(a + c' + )
(b' + c' +
(a' + b +
(a' + b' +
c)

\[\text{Backtrack}\]
Basic DLL Procedure - DFS

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' + ) \\
(b' + c' + ) \\
(a' + b + ) \\
(a' + b' + c) \\
\end{align*}
\]
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' +
(b' + c' +
(a' + b +
(a' + b' +
c)

a = 1 → (a' + b' + c)

b = 1

c = 1

Implication

 Forced Decision
Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ +
(b’ + c’ +
(a’ + b +
(a’ + b’ +
c)

(a’ + b’ + c)
(b’ + c’ + d)
(c’ +
(a’ + b’ +
c)

Design
11/25/08 L21.27
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + +
(b' + c' +
(a' + b +
(a' + b' + c

\( b = 1 \)
\( a = 1 \)
\( c = 1 \)
\( d = 1 \)

\( \text{SAT} \)
Implications and Boolean Constraint Propagation

- **Implication**
  - A variable is forced to be assigned to be True or False based on previous assignments.

- **Unit clause rule** (rule for elimination of one literal clauses)
  - An unsatisfied clause is a unit clause if it has exactly one unassigned literal.
  - The unassigned literal is implied because of the unit clause.

- **Boolean Constraint Propagation (BCP)**
  - Iteratively apply the unit clause rule until there is no unit clause available.
Boolean Constraint Propagation

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_{7'} + x_3' + x_9 \]
\[ x_{7'} + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
Boolean Constraint Propagation

\begin{align*}
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10'} \\
x_7 + x_{10} + x_{12'}
\end{align*}
Boolean Constraint Propagation

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

- \( x_1 = 0 \)
- \( x_4 = 1 \)
Boolean Constraint Propagation

\( x_1 + x_4 \)
\( x_1 + x_3' + x_8' \)
\( x_1 + x_8 + x_{12} \)
\( x_2 + x_{11} \)
\( x_7' + x_3' + x_9 \)
\( x_7' + x_8 + x_9' \)
\( x_7 + x_8 + x_{10'} \)
\( x_7 + x_{10} + x_{12'} \)

- \( x_4=1 \)
- \( x_1=0 \)
- \( x_3=1 \)

\( x_1=0, x_4=1 \)
\( x_3=1 \)
Boolean Constraint Propagation

\[ \begin{align*}
x_1 &+ x_4 \\
x_1 &+ x_3' + x_8' \\
x_1 &+ x_8 + x_{12} \\
x_2 &+ x_{11} \\
x_7' &+ x_3' + x_9 \\
x_7' &+ x_8 + x_9' \\
x_7 &+ x_8 + x_{10}' \\
x_7 &+ x_{10} + x_{12}' \\
x_4 &= 1 \\
x_1 &= 0, \quad x_4 = 1 \\
x_3 &= 1, \quad x_8 = 0 \\
x_8 &= 0
\end{align*} \]
Boolean Constraint Propagation

- $x_1 + x_4$
- $x_1 + x_3' + x_8'$
- $x_1 + x_8 + x_{12}$
- $x_2 + x_{11}$
- $x_7' + x_3' + x_9$
- $x_7' + x_8 + x_9'$
- $x_7 + x_8 + x_{10'}$
- $x_7 + x_{10} + x_{12'}$

Diagrams:
- $x_1 = 0, x_4 = 1$
- $x_3 = 1, x_8 = 0, x_{12} = 1$
- $x_4 = 1$
- $x_1 = 0, x_3 = 1$
- $x_8 = 0$
- $x_{12} = 1$
Boolean Constraint Propagation

\begin{align*}
  x_1 + x_4 \\
  x_1 + x_3' + x_8' \\
  x_1 + x_8 + x_12 \\
  x_2 + x_11 \\
  x_7' + x_3' + x_9 \\
  x_7' + x_8 + x_9' \\
  x_7 + x_8 + x_10' \\
  x_7 + x_10 + x_12' \\
 \end{align*}

- **x4=1**
- **x1=0**
- **x3=1**
- **x8=0**
- **x2=0**
- **x12=1**

Values:
- **x1=0, x4=1**
- **x3=1, x8=0, x12=1**
- **x2=0**
Boolean Constraint Propagation

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]

\[ x_4 = 1 \]
\[ x_{11} = 1 \]
\[ x_{12} = 1 \]
\[ x_2 = 0 \]
\[ x_3 = 1 \]
\[ x_8 = 0 \]
\[ x_{1} = 0, x_4 = 1 \]
\[ x_{2} = 0, x_{11} = 1 \]
\[ x_{3} = 1, x_8 = 0, x_{12} = 1 \]
Boolean Constraint Propagation

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

\[ x_4 = 1 \]
\[ x_{11} = 1 \]
\[ x_2 = 0 \]
\[ x_{12} = 1 \]
\[ x_7 = 1 \]
\[ x_3 = 1 \]
\[ x_8 = 0 \]
\[ x_{12} = 1 \]
\[ x_{11} = 1 \]
\[ x_7 = 1 \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_{7'} + x_3' + x_9 \]
\[ x_{7'} + x_8 + x_{9'} \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

\[ x_4 = 1 \]
\[ x_1 = 0 \]
\[ x_{11} = 1 \]
\[ x_2 = 0 \]
\[ x_{12} = 1 \]

\[ x_3 = 1, x_8 = 0, x_{12} = 1 \]
\[ x_2 = 0, x_{11} = 1 \]
\[ x_7 = 1, x_9 = 0, 1 \]
Conflict Driven Learning

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
Conflict Driven Learning

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_12 \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

Add conflict clause: \( x_3' + x_7' + x_8 \)

\[ x_3=0, x_4=1 \]
\[ x_3=1, x_8=0, x_{12}=1 \]
\[ x_2=0, x_{11}=1 \]
\[ x_7=1, x_9=1 \]
\[ x_4=1 \]
\[ x_9=1 \]
\[ x_7=1 \]
\[ x_8=0 \]
\[ x_{12}=1 \]

Design
Conflict Driven Learning

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]

Add conflict clause: \[ x_3' + x_7' + x_8 \]

\[ x_3 = 1 \land x_7 = 1 \land x_8 = 0 \rightarrow \text{conflict} \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
\[ x_3' + x_8 + x_7' \]

Backtrack to the decision level of \( x_3 = 1 \)
With implication \( x_7 = 0 \)

\( x_1 = 0, \ x_4 = 1 \)
\( x_3 = 1, \ x_8 = 0, \ x_{12} = 1 \)
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
\[ x_3' + x_8 + x_7' \]

This is exactly what would happen if the new clause existed from the beginning
What’s the big deal?

Conflict clause: $x_1' + x_3 + x_5'$

Significantly prune the search space – *learned clause is useful forever!*

Useful in generating future conflict clauses.
SAT becomes “practical!”

- Conflict driven learning greatly increases the capacity of SAT solvers (several thousand variables) for structured problems
- Realistic applications became plausible
  - Usually thousands and even millions of variables
  - Typical EDA applications that can make use of SAT
    - circuit verification
    - FPGA routing
    - many other applications…
- Research direction changes towards more efficient implementations