Propositional Logic and Satisfiability

Brian C. Williams
16.410-13
November 22nd, 2004
How Do We Reason About Complex Systems at a Commonsense Level?

- Model using propositional logic.
- Reason from model to operate, diagnose and repair.

Flow$_1$ = zero
Pressure$_1$ = nominal
Acceleration = zero

Pressure$_2$ = nominal
Propositional Satisfiability

Find a truth assignment that satisfies logical sentence $T$:

- Reduce sentence $T$ to clausal form.
- Perform search similar to MAC = (BT+CP)

Propositional satisfiability testing:
- 1990: 100 variables / 200 clauses (constraints)
- 1998: 10,000 - 100,000 vars / $10^6$ clauses

Novel applications:
- e.g. diagnosis, planning, software / circuit testing, machine learning, and protein folding
Reading Assignment:
Propositional Logic & Satisfiability

- AIMA Ch. 6 – Propositional Logic
Outline

• Propositional Logic
  • Syntax
  • Semantics
  • Clausal Reduction
• Propositional Satisfiability
• Appendices
What formal languages exist for describing constraints?

Logic:
- **Propositional logic** truth of facts
- **First order logic** facts, objects, relations
- **Temporal logic** time, …
- **Modal logics** knowledge, belief …
- **Probability** degree of belief
- **Algebra** values of variables
Logic in General

- Logics
  - formal languages for representing information such that conclusions can be drawn.

- Syntax
  - defines the sentences in the language.

- Semantics
  - defines the “meaning” of sentences;
  - truth of a sentence in a world.
Propositional Logic: Syntax

Propositions
- A statement that is true or false
  - (valve v1)
  - (= voltage high)

Propositional Sentences (S)
- S ::= proposition |
- (NOT S) |
- (OR S1 ... Sn) |
- (AND S1 ... Sn)

Some Defined Constructs
- (implies S1 S2) => ((not S1) OR S2)
- (IFF S1 S2) => (AND (IMPLIES S1 S2)(IMPLES S2 S1))
Propositional Sentences: Engine Example

(mode(E1) = ok implies
  (thrust(E1) = on if and only if flow(V1) = on and flow(V2) = on)) and
  (mode(E1) = ok or mode(E1) = unknown) and
  not (mode(E1) = ok and mode(E1) = unknown)
Outline

• Propositional Logic
  • Syntax
  • Semantics
  • Clausal Reduction
• Propositional Satisfiability
• Appendices
Propositional Logic: Semantics

Interpretation $I$ of sentence $S$ assigns true or false to every proposition $P$ of $S$

- $S = (A \text{ or } B) \text{ and } C$
- $I = \{A=True, B=False, C=True\}$
- $I = \{A=False, B=True, C=False\}$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

All Interpretations
Propositional Logic: Semantics

The truth of sentence $S$ wrt interpretation $I$ is defined by a composition of boolean operators applied to $I$:

- “Not $S$” is True iff “$S$” is False

<table>
<thead>
<tr>
<th>Not $S$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
# Propositional Logic: Semantics

The truth of sentence $S_i$ wrt Interpretation $I$:

- "Not $S$" is True iff "$S$" is False
- "$S_1$ and $S_2$" is True iff "$S_1$" is True and "$S_2$" is True
- "$S_1$ or $S_2$" is True iff "$S_1$" is True or "$S_2$" is True

<table>
<thead>
<tr>
<th>S1 and S2</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1 or S2</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Propositional Logic: Semantics

The truth of sentence \( S_i \) wrt Interpretation I:

- “Not \( S \)” is True if and only if “\( S \)” is False
- “\( S_1 \) and \( S_2 \)” is True if and only if “\( S_1 \)” is True and “\( S_2 \)” is True
- “\( S_1 \) or \( S_2 \)” is True if and only if “\( S_1 \)” is True or “\( S_2 \)” is True
- “\( S_1 \)” implies “\( S_2 \)” is True if and only if “\( S_1 \)” is False or “\( S_2 \)” is True
- “\( S_1 \)” if \( S_2 \) is True if and only if “\( S_1 \)” implies “\( S_2 \)” is True and “\( S_2 \)” implies “\( S_1 \)” is True
Example: Determining the truth of a sentence

\[
\text{(mode(E1) = ok implies}
\]
\[
((\text{thrust(E1) = on if and only if (flow(V1) = on and flow(V2) = on)) and}
\]
\[
(\text{mode(E1) = ok or mode(E1) = unknown) and}
\]
\[
\text{not (mode(E1) = ok and mode(E1) = unknown))}
\]

Interpretation:

- \text{mode(E1) = ok} is True
- \text{thrust(E1) = on} is False
- \text{flow(V1) = on} is True
- \text{flow(V2) = on} is False
- \text{mode(E1) = unknown} is False
Example: Determining the truth of a sentence

(True implies
   [(False if and only if (True and False)) and
    (True or False) and
    not (True and False)])

Interpretation:

mode(E1) = ok is True
thrust(E1) = on is False
flow(V1) = on is True
flow(V2) = on is False
mode(E1) = unknown is False
Example: Determining the truth of a sentence

\[(\text{True implies} \quad [(\text{False if and only if (True and False)}) \quad \text{and} \quad \text{True or False}) \quad \text{and} \quad \text{not (True and False)}])\]

Interpretation:

- \(\text{mode}(E1) = \text{ok}\) is True
- \(\text{thrust}(E1) = \text{on}\) is False
- \(\text{flow}(V1) = \text{on}\) is True
- \(\text{flow}(V2) = \text{on}\) is False
- \(\text{mode}(E1) = \text{unknown}\) is False
Example: Determining the truth of a sentence

\[(\text{True implies} \quad [(\text{False if and only if } (\text{True and False})) \quad \text{and} \quad (\text{True or False}) \quad \text{and} \quad \text{not False}])\]

Interpretation:
- mode(E1) = ok is True
- thrust(E1) = on is False
- flow(V1) = on is True
- flow(V2) = on is False
- mode(E1) = unknown is False
Example: Determining the truth of a sentence

(True implies

[(False if and only if (True and False)) and
 (True or False) and
 True])

Interpretation:

mode(E1) = ok is True
thrust(E1) = on is False
flow(V1) = on is True
flow(V2) = on is False
mode(E1) = unknown is False
Example: Determining the truth of a sentence

\[(\text{True implies} \]
\[\text{[(False if and only if False) and True and True]}\]

Interpretation:

- \(\text{mode}(E1) = \text{ok}\) is True
- \(\text{thrust}(E1) = \text{on}\) is False
- \(\text{flow}(V1) = \text{on}\) is True
- \(\text{flow}(V2) = \text{on}\) is False
- \(\text{mode}(E1) = \text{unknown}\) is False
Example: Determining the truth of a sentence

(True implies
  [(False if and only if False) and
   True and
   True])

Interpretation:
  mode(E1) = ok is True
  thrust(E1) = on is False
  flow(V1) = on is True
  flow(V2) = on is False
  mode(E1) = unknown is False
Example: Determining the truth of a sentence

$$(\text{True implies } \left[ (\text{False implies False }) \land (\text{False implies False }) \right] \land \text{True} \land \text{True})$$

Interpretation:

- mode(E1) = ok is True
- thrust(E1) = on is False
- flow(V1) = on is True
- flow(V2) = on is False
- mode(E1) = unknown is False
Example: Determining the truth of a sentence

(True implies
   
   [(not False or False ) and (not False or False )) and
   True and
   True])

Interpretation:

   mode(E1) = ok is True
   thrust(E1) = on is False
   flow(V1) = on is True
   flow(V2) = on is False
   mode(E1) = unknown is False
Example: Determining the truth of a sentence

\[(\text{True implies } (\text{True or False} \land \text{True or False}) \land \text{True} \land \text{True})\]

Interpretation:

- \text{mode}(E1) = \text{ok} is True
- \text{thrust}(E1) = \text{on} is False
- \text{flow}(V1) = \text{on} is True
- \text{flow}(V2) = \text{on} is False
- \text{mode}(E1) = \text{unknown} is False
Example: Determining the truth of a sentence

(True implies
   [(True and True) and
    True and
    True])

Interpretation:

mode(E1) = ok    is True
thrust(E1) = on  is False
flow(V1) = on    is True
flow(V2) = on    is False
mode(E1) = unknown is False
Example: Determining the truth of a sentence

(True implies
   [True and
    True and
    True])

Interpretation:

- mode(E1) = ok is True
- thrust(E1) = on is False
- flow(V1) = on is True
- flow(V2) = on is False
- mode(E1) = unknown is False
Example: Determining the truth of a sentence

(True implies True)

Interpretation:

- mode(E1) = ok is True
- thrust(E1) = on is False
- flow(V1) = on is True
- flow(V2) = on is False
- mode(E1) = unknown is False
Example: Determining the truth of a sentence

(not True or True)

Interpretation:

mode(E1) = ok is True
thrust(E1) = on is False
flow(V1) = on is True
flow(V2) = on is False
mode(E1) = unknown is False
Example: Determining the truth of a sentence

(False or True)

Interpretation:

- mode(E1) = ok is True
- thrust(E1) = on is False
- flow(V1) = on is True
- flow(V2) = on is False
- mode(E1) = unknown is False
Example: Determining the truth of a sentence

True!

Interpretation:

- \text{mode}(E1) = \text{ok} \quad \text{is True}
- \text{thrust}(E1) = \text{on} \quad \text{is False}
- \text{flow}(V1) = \text{on} \quad \text{is True}
- \text{flow}(V2) = \text{on} \quad \text{is False}
- \text{mode}(E1) = \text{unknown} \quad \text{is False}

If a sentence S evaluates to True in interpretation I, then:

- I satisfies S
- I is a \textit{Model} of S
Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Clausal Reduction
- Propositional Satisfiability
- Appendices
Propositional Clauses: A Simpler Form

- Literal: proposition or its negation
  - B, Not A
- Clause: disjunction of literals
  - (not A or B or E)
- Conjunctive Normal Form
  - Phi = (A or B or C) and
    (not A or B or E) and
    (not B or C or D)
  - Viewed as a set of clauses
Reduction to Clausal Form: Engine Example

(mode(E1) = ok implies
    (thrust(E1) = on iff (flow(V1) = on and flow(V2) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V1) = on;
not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V2) = on;
not (mode(E1) = ok) or not (flow(V1) = on) or not (flow(V2) = on)
    or thrust(E1) = on;
mode(E1) = ok or mode(E1) = unknown;
not (mode(E1) = ok) or not (mode(E1) = unknown);
Reducing Propositional Formula to Clauses (CNF)

See Appendix for Detailed Example:

1) Eliminate IFF and Implies
   - E1 iff E2 => (E1 implies E2) and (E2 implies E1)
   - E1 implies E2 => not E1 or E2

2) Move negations in towards propositions using De Morgan’s Theorem:
   - Not (E1 and E2) => (not E1) or (not E2)
   - Not (E1 or E2) => (not E1) and (not E2)
   - Not (not E1) => E1

3) Move conjunctions out using Distributivity
   - E1 or (E2 and E3) => (E1 or E2) and (E1 or E3)
Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Clausal Reduction
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Appendices
Propositional Clauses form a Constraint Satisfaction Problem

- Variables: Propositions
- Domain: \{True, False\}
- Constraints: Clauses that must be true

- Clause: (not A or B or E)
  - A disjunction of Literals

- Literal: Proposition or its negation
  - Positive Literal  B
  - Negative Literal  Not A
Propositional Satisfiability

- An interpretation (truth assignment to all propositions) such that all clauses are satisfied:

- A clause is **satisfied** if and only if at least one literal is true.

- A clause is **violated** if and only if all literals are false.

C1: Not A or B
C2: Not C or A
C3: Not B or C
Satisfiability Testing Procedures

Reduce to CNF (Clausal Form) then:

1. Apply systematic, complete procedure
   - Depth-first backtrack search
     (Davis, Putnam, & Loveland 1961)
     - unit propagation, shortest clause heuristic

2. Apply stochastic, incomplete procedure
   - GSAT (Selman et. al 1993) – see Appendix
Outline

• Propositional Logic
• Propositional Satisfiability
  • Backtrack Search
  • Unit Propagation
  • DPLL: Unit Propagation + Backtrack Search
• Appendices
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C

Diagram:

A ➔ F
B ➔ F
C ➔ F

S

3/19/2003
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C

Diagram:

```
A
  /  
F    T
   /  
B
 F   T
 /  /  
C C
 F T F
 / / /  
C C C
 F T F
 / / / /  
C C C
 F T F
```

S S S
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C
Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C
Clausal Backtrack Search: Recursive Definition

BT(Phi, A)

Input: A cnf theory Phi,
       An assignment A to propositions in Phi

Output: A decision of whether Phi is satisfiable.

1. If a clause is violated, Return false;
2. Else If all propositions are assigned, Return true;
3. Else Q = some unassigned proposition in Phi;
4. Return (BT(Phi, A[Q = True]) or
5. BT(Phi, A[Q = False])
Outline

• Propositional Logic
• Propositional Satisfiability
  • Backtrack Search
  • Unit Propagation
  • DPLL: Unit Propagation + Backtrack Search
• Appendices
Unit Propagation

Idea: Arc consistency (AC-3) on binary clauses

(not A or B)

{F} ← {T,F} ?

{T} ← {T,F} ?

Unit resolution rule:

If all literals are false save L, then assign true to L:

- (not A) (not B) (A or B or C)
  C
Unit Propagation Examples

- C1: Not A or B
- C2: Not C or A
- C3: Not B or C
- C4: A

Satisfied

C4 → A → C1 → B → C3 → C

True

Copyright Brian Williams 2003
Unit Propagation Examples

- C1: Not A or B  Satisfied
- C2: Not C or A  Satisfied
- C3: Not B or C  Satisfied
- C4: A

C4  →  A  →  C1  →  B  →  C  

C4'  →  B  →  C1  →  A  →  C2  →  C

C4: A  True
C1: True
C3: True

C4': Not B  Satisfied
C1: False
C3: False
C2: False

Satisfied
Unit Propagation

true  false
$\bar{r}$  $q$

$C_1: \neg r \lor q \lor p$

$C_2: \neg p \lor \neg t$

procedure propagate($C$)  // $C$ is a clause
if all literals in $C$ are false except $L$, and $L$ is unassigned
then assign true to $L$ and
record $C$ as a support for $L$ and
for each clause $C'$ mentioning “not $L$”,
propagate($C'$)
end propagate
Unit Propagation

true  false

\[ C_1 : \neg r \lor q \lor p \]

\[ C_2 : \neg p \lor \neg t \]

\[ p \]

\[ t \]

procedure propagate(C)  // C is a clause

\[ \text{if all literals in } C \text{ are false except } L, \text{ and } L \text{ is unassigned} \]

\[ \text{then assign true to } L \text{ and} \]

\[ \text{record } C \text{ as a support for } L \text{ and} \]

\[ \text{for each clause } C' \text{ mentioning “not } L”, \]

\[ \text{propagate}(C') \]

end propagate
Unit Propagation

\[ \begin{align*} C_1 : & \neg r \lor q \lor p \\ C_2 : & \neg p \lor \neg t \end{align*} \]

procedure \textit{propagate}(C) \hspace{1cm} // C is a clause

if all literals in C are false except L, and L is unassigned

\[ \Rightarrow \] then assign true to L and

record C as a support for L and

for each clause C’ mentioning “not L”,

\textit{propagate}(C’)

end \textit{propagate}
Unit Propagation

$C_1: \neg r \lor q \lor p$

$C_2: \neg p \lor \neg t$

procedure propagate(C)  // C is a clause
    if all literals in C are false except L, and L is unassigned
    then assign true to L and
    $\Rightarrow$ record C as a support for L and
    for each clause C’ mentioning “not L”,
    propagate(C’)
end propagate
Unit Propagation

procedure propagate(C)  // C is a clause
    if all literals in C are false except L, and L is unassigned
    then assign true to L and
        record C as a support for L and
        for each clause C’ mentioning “not L”,
        propagate(C’)
end propagate
procedure propagate(C) // C is a clause
    if all literals in C are false except L, and L is unassigned
    then assign true to L and
       record C as a support for L and
       for each clause C’ mentioning “not L”,
       propagate(C’)
end propagate
Outline

• Propositional Logic
• Propositional Satisfiability
  • Backtrack Search
  • Unit Propagation
  • DPLL: Unit Propagation + Backtrack Search
• Appendices
How Do We Combine Unit Resolution and Back Track Search?

Backtrack Search

• Assign true or false to an unassigned proposition.
• Backtrack as soon as a clause is violated.
• Theory is satisfiable if assignment is complete.

Example:
• C1: Not A or B
• C2: Not C or A
• C3: Not B or C

- Similar to MAC and Forward Checking:
  - Perform limited form of inference
  - apply inference rule after assigning each variable.
Propositional Satisfiability by DPLL

[Davis, Putnam, Logmann, Loveland, 1962]

Initially:
- Unit propagate.

Repeat:
1. Assign true or false to an unassigned proposition.
2. Unit propagate.
3. Backtrack as soon as a clause is violated.
4. Satisfiable if assignment is complete.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C

Initially:
A

Propagate:
C = F
B = F
Propositional Satisfiability by DPLL

[Davis, Putnam, Logmann, Loveland, 1962]

Initially:
- Unit propagate.

Example:
- C1: Not A or B
- C2: Not C or A
- C3: Not B or C

Initially:
- Unit propagate.

Repeat:
1. Assign true or false to an unassigned proposition.
2. Unit propagate.
3. Backtrack as soon as a clause is violated.
4. Satisfiable if assignment is complete.
**DPLL Procedure**

[Davis, Putnam Logmann, Loveland, 1962]

DPLL(\(\Phi, A\))

**Input:** A *cnf* theory \(\Phi\),
An assignment \(A\) to propositions in \(\Phi\)

**Output:** A decision of whether \(\Phi\) is satisfiable.

1. \(A' = \text{propagate}(\Phi)\);
2. If a clause is violated given \(A'\) return(false);
3. Else if all propositions in \(A'\) are assigned, return(true);
4. Else \(Q = \text{some unassigned proposition in } \Phi\);
5. Return \((\text{DPLL}(\Phi, A'[Q = \text{True}]) \text{ or } \text{DPLL}(\Phi, A'[Q = \text{False}])\)
Satisfiability Testing Procedures

• Reduce to CNF (Clausal Form) then:

• Apply systematic, complete procedure
  • Depth-first backtrack search (Davis, Putnam, & Loveland 1961)
    • unit propagation, shortest clause heuristic
  • State-of-the-art implementations:
    • ntab (Crawford & Auton 1997)
    • itms (Nayak & Williams 1997)
    • many others! See SATLIB 1998 / Hoos & Stutzle

• Apply stochastic, incomplete procedures
  • GSAT (Selman et. al 1993)
  • Walksat (Selman & Kautz 1993)
    • greedy local search + noise to escape local minima
Required Appendices

You are responsible for reading and knowing this material:

1. Characteristics of DPLL
2. Local Search using GSAT
3. Reduction to Clausal Form
Hardness of 3SAT

Ratio of Clauses-to-Variables

DP Calls

- 50 var
- 40 var
- 20 var
The 4.3 Point

![Graph showing DP Calls vs. Ratio of Clauses-to-Variables]

- 50 var
- 40 var
- 20 var

Mitchell, Selman, and Levesque 1991
Intuition

• At low ratios:
  • few clauses (constraints)
  • many assignments
  • easily found

• At high ratios:
  • many clauses
  • inconsistencies easily detected
Phase Transitions for Different Numbers of Variables
Phase transition 2-, 3-, 4-, 5-, and 6-SAT

Thresholds for 2SAT, 3SAT, 4SAT, 5SAT, and 6SAT

Fraction of unsatisfiable formulas

M/11

0 10 20 30 40 50 60
Required Appendices

You are responsible for reading and knowing this material:

1. Characteristics of DPLL
2. Local Search using GSAT
3. Reduction to Clausal Form
Incremental Repair
(min-conflict heuristic)

Spike Hubble Telescope Scheduler [Minton et al.]

1. Initialize a candidate solution using “greedy” heuristic – get solution “near” correct one.

2. Select a variable in conflict and assign it a value that minimizes the number of conflicts (break ties randomly).
1. Init: Pick random assignment
2. Check effect of flipping each assignment, counting violated clauses.
3. Pick assignment with fewest violations,
4. End if consistent, Else goto 2

GSAT

- C1: Not A or B
- C2: Not C or Not A
- C3: or B or Not C

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

C3 violated

C1, C2, C3 violated

C2 violated

C1 violated

C3 violated
**GSAT**

1. Init: Pick random assignment
2. Check effect of flipping each assignment, counting violated clauses.
3. Pick assignment with fewest violations,
4. End if consistent, Else goto 2

- C1: Not A or B
- C2: Not C or Not A
- C3: or B or Not C

A: True
B: False
C: False

C1 violated

True False False
A B C

False True True

Satisfied Satisfied C1,C2,C3 violated
**GSAT**

1. Init: Pick random assignment
2. Check effect of flipping each assignment, counting violated clauses.
3. Pick assignment with fewest violations,
4. End if consistent, Else goto 2

- C1: Not A or B
- C2: Not C or Not A
- C3: or B or Not C

<table>
<thead>
<tr>
<th>Satisfied</th>
<th>True</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

**Problem:** Pure hill climbers get stuck in local minima.

**Solution:** Add random moves to get out of minima (**WalkSAT**)
Required Appendices

You are responsible for reading and knowing this material:

1. Local Search using GSAT
2. Characteristics of DPLL
3. Reduction to Clausal Form
Reduction to Clausal Form: Engine Example

(mode(E1) = ok implies
  (thrust(E1) = on iff flow(V1) = on and flow(V2) = on)) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V1) = on;
not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V2) = on;
not (mode(E1) = ok) or not (flow(V1) = on) or not (flow(V2) = on) or
  thrust(E1) = on;
mode(E1) = ok or mode(E1) = unknown;
not (mode(E1) = ok) or not (mode(E1) = unknown);
Reducing Propositional Formula to Clauses (CNF)

1) Eliminate IFF and Imply
   • $E_1$ iff $E_2 => (E_1$ implies $E_2)$ and $(E_2$ implies $E_1)$
   • $E_1$ implies $E_2 => not E_1 or E_2$
Eliminate IFF: Engine Example

(mode(E1) = ok implies
  (thrust(E1) = on iff (flow(V1) = on and flow(V2) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

(mode(E1) = ok implies
  ((thrust(E1) = on implies (flow(V1) = on and flow(V2) = on)) and
  ((flow(V1) = on and flow(V2) = on) implies thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)
Eliminate Implies: Engine Example

\[
\begin{align*}
\text{(mode}(E_1) = \text{ok} & \implies \left( (\text{thrust}(E_1) = \text{on} \implies (\text{flow}(V_1) = \text{on} \text{ and } \text{flow}(V_2) = \text{on})) \text{ and } \\
& \quad \left( (\text{flow}(V_1) = \text{on} \text{ and } \text{flow}(V_2) = \text{on}) \implies \text{thrust}(E_1) = \text{on} \right) \right) \right) \text{ and } \\
& \quad \left( \text{mode}(E_1) = \text{ok or mode}(E_1) = \text{unknown} \right) \text{ and } \\
& \quad \neg (\text{mode}(E_1) = \text{ok and mode}(E_1) = \text{unknown}) \\
& \text{or} \\
\text{(not (mode}(E_1) = \text{ok}) & \text{ or } \\
\left( (\neg (\text{thrust}(E_1) = \text{on}) \text{ or } (\text{flow}(V_1) = \text{on} \text{ and } \text{flow}(V_2) = \text{on})) \text{ and } \\
& \quad \left( \neg (\text{flow}(V_1) = \text{on} \text{ and } \text{flow}(V_2) = \text{on}) \text{ or } \text{thrust}(E_1) = \text{on} \right) \right) \right) \text{ and } \\
& \quad \left( \text{mode}(E_1) = \text{ok or mode}(E_1) = \text{unknown} \right) \text{ and } \\
& \quad \neg (\text{mode}(E_1) = \text{ok and mode}(E_1) = \text{unknown}) 
\end{align*}
\]
Reducing Propositional Formula to Clauses (CNF)

2) Move negations in towards propositions using De Morgan’s Theorem:
   • Not (E1 and E2) => (not E1) or (not E2)
   • Not (E1 or E2) => (not E1) and (not E2)
   • Not (not E1) => E1
Move Negations In:
Engine Example

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
    (not (flow(V1) = on and flow(V2) = on)) or thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
    (not (flow(V1) = on) or not (flow(V2) = on)) or thrust(E1) = on) ) and
(mode(E1) = ok or mode(E1) = unknown) and
(not (mode(E1) = ok) or not (mode(E1) = unknown)))
Reducing Propositional Formula to Clauses (CNF)

3) Move conjunctions out using distributivity
   • E1 or (E2 and E3) => (E1 or E2) and (E1 or E3)
Move Conjunctions Out: Engine Example

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
   (not (flow(V1) = on) or not (flow(V2) = on) or thrust(E1) = on))) and
  (mode(E1) = ok or mode(E1) = unknown) and
  (not (mode(E1) = ok) or not (mode(E1) = unknown))

  (not (mode(E1) = ok) or
    (((not (thrust(E1) = on) or flow(V1) = on) and
      (not (thrust(E1) = on) or flow(V2) = on)) and
      (not (flow(V1) = on) or not (flow(V2) = on) or thrust(E1) = on))) and
    (mode(E1) = ok or mode(E1) = unknown) and
    (not (mode(E1) = ok) or not (mode(E1) = unknown))

3/19/2003
Move Conjunctions Out: Engine Example

\[
\text{(not (mode(E1) = ok) or (not (thrust(E1) = on) or flow(V1) = on)} \text{ and }
\text{(not (thrust(E1) = on) or flow(V2) = on)) and (not (flow(V1) = on) or not (flow(V2) = on) or thrust(E1) = on)) and (mode(E1) = ok or mode(E1) = unknown) and (not (mode(E1) = ok) or not (mode(E1) = unknown))}
\]
Reducing Propositional Formula to Clauses (CNF)

1) Eliminate IFF and Implies
   - E1 iff E2 => (E1 implies E2) and (E2 implies E1)
   - E1 implies E2 => not E1 or E2

2) Move negations in towards propositions using
   De Morgan’s Theorem:
   - Not (E1 and E2) => (not E1) or (not E2)
   - Not (E1 or E2) => (not E1) and (not E2)
   - Not (not E1) => E1

3) Move conjunctions out using Distributivity
   - E1 or (E2 and E3) => (E1 or E2) and (E1 or E3)