Propositional Logic and Satisfiability

Slides draw upon material from:
Prof. Bart Selman
Cornell University

Brian C. Williams
16.410-13
October 13th, 2010

Assignments

• Assignment:
  • Problem Set #5: Activity Planning, due today Wednesday, October 13th, 2010.
  • Problem Set #6: Propositional Logic and Satisfiability, out today; due October 27th, 2010 (in 2 weeks).

• Reading:
  • Today: [AIMA] Ch. 7, 8
  • Monday: TBD

• Exam:
  • Mid-Term - October 20th.
Hidden Failures Require Reasoning from a Model:

**Symptoms:**
- Engine temp sensor high
- LOX level low
- GN&C detects low thrust
- H2 level possibly low

**Problem:** Liquid hydrogen leak

**Effect:**
- LH2 used to cool engine
- Engine runs hot
- Consumes more LOX

How Do We Reason About Complex Systems using Commonsense Models?

**Task:** Monitor engine operation
- You open the valves, and observe . . .
- Is the engine ok?
- Could the valve in red be stuck closed?

**Pressure**
- $\text{Pressure}_2 = \text{nominal}$
- $\text{Flow}_2 = \text{zero}$

**Oxidizer tank**
- $\text{Pressure}_1 = \text{nominal}$
- $\text{Flow}_1 = \text{zero}$

**Fuel tank**
- $\text{Pressure}_2 = \text{nominal}$

**Main Engines**

**Model-based Reasoning:**
- Reason from a single model to operate, diagnose, repair...
- **Model using Logic.**
- **Reason using Sat.**
Modeling an Engine in Propositional Logic

“An Engine E1 can either be okay, or broken in some unknown way. When E1 is okay, it will thrust when there is a flow through V1 and V2.”

\[
\text{mode}(E1) = \text{ok or mode}(E1) = \text{unknown} \quad \text{and}
\]

\[
\text{not (mode}(E1) = \text{ok and mode}(E1) = \text{unknown}) \quad \text{and}
\]

\[
\text{mode}(E1) = \text{ok implies}
\]

\[
\text{thrust}(E1) = \text{on if and only if flow}(V1) = \text{on and flow}(V2) = \text{on})
\]

Reasoning From the Model

**Monitoring:**
Are the observations O consistent with model M?

**Fault Diagnosis:**
What fault modes of M are consistent with O?

**Reconfiguration:**
What component modes of M produce behavior G?

\[\Rightarrow \quad \text{Propositional Satisfiability:}
\]

Find a truth assignment that satisfies some logical sentence S:

1. Reduce S to clausal form.
2. Perform search similar to MAC = (BT+CP)
   \[\text{[Davis, Logmann & Loveland, 1962]}\]

Brian Williams, Fall 10
Propositional Satisfiability

Find a truth assignment that satisfies logical sentence T:
• Reduce sentence T to clausal form.
• Perform search similar to MAC = (BT+CP)
  [Davis, Logmann & Loveland, 1962]

Propositional satisfiability testing
  1990: 100 variables / 200 clauses (constraints)
  1998: 10,000 - 100,000 vars / 10^6 clauses
  2010: millions

Novel applications:
  e.g. diagnosis, planning, software verification, circuit testing, machine learning, and protein folding

What Formal Languages Exist for Describing Constraints?

• Algebra values of variables
• Probability degree of belief
• Propositional logic truth of facts
• Temporal logic time, .....
• Modal logics knowledge, belief …
• First order logic facts,objects,relations
Outline

• Propositional Logic
  • Syntax
  • Semantics
    • Reduction to Clauses
• Propositional Satisfiability
• Empirical, Average Case Analysis
• Appendices

Logic in General

• Logic
  • A formal language for representing information that can be used to draw conclusions.
  • About the truth of statements and their consequences.
• Syntax
  • Defines the expressible sentences in the language.
• Semantics
  • Defines the “meaning” of these sentences
    $\models$ truth of a sentence in some world.
Logic Example: Arithmetic

• **Syntax** – legal sentences
  - “X + 2 > Y” is a legal sentence.
  - “X 2 + Y >” is not a legal sentence.

• **Semantics** - truth in world
  - “X + 2 > Y” is true iff the number X + 2 is not less than or equal to the number Y.
  - “X + 2 > Y” is true in a world where X = 7, Y = 1.
  - “X + 2 > Y” is false in a world where X = 0, Y = 6.

Propositional Logic: Syntax

Propositions
- A statement that is true or false
  - (valve v1)
- Assignments to finite domain variables - State Logic
  - (= voltage high)

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

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

$(\text{mode}(E1) = \text{ok} \text{ or } \text{mode}(E1) = \text{unknown}) \text{ and } \neg (\text{mode}(E1) = \text{ok} \text{ and } \text{mode}(E1) = \text{unknown}) \text{ and }$

$(\text{mode}(E1) = \text{ok} \text{ implies }\neg (\text{thrust}(E1) = \text{on} \text{ if and only if }\neg (\text{flow}(V1) = \text{on} \text{ and } \text{flow}(V2) = \text{on})))$

Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Reduction to Clauses
- Propositional Satisfiability
- Empirical, Average Case Analysis
- Appendices
Propositional Logic: Semantics

Interpretation I of sentence S assigns true or false to every proposition P in 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></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td></td>
<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>

Brian Williams, Fall 10
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>$S_1$ and $S_2$</th>
<th>$S_1$</th>
<th>$S_2$</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>$S_1$ or $S_2$</th>
<th>$S_1$</th>
<th>$S_2$</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>

Brian Williams, Fall 10
Example: Determining the Truth of a Sentence

(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)])

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

Interpretation:

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

Brian Williams, Fall 10
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

(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

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

Interpretation:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode(E1) = ok</td>
<td>is True</td>
</tr>
<tr>
<td>thrust(E1) = on</td>
<td>is False</td>
</tr>
<tr>
<td>flow(V1) = on</td>
<td>is True</td>
</tr>
<tr>
<td>flow(V2) = on</td>
<td>is False</td>
</tr>
<tr>
<td>mode(E1) = unknown</td>
<td>is False</td>
</tr>
</tbody>
</table>

Example: Determining the Truth of a Sentence

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

Interpretation:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode(E1) = ok</td>
<td>is True</td>
</tr>
<tr>
<td>thrust(E1) = on</td>
<td>is False</td>
</tr>
<tr>
<td>flow(V1) = on</td>
<td>is True</td>
</tr>
<tr>
<td>flow(V2) = on</td>
<td>is False</td>
</tr>
<tr>
<td>mode(E1) = unknown</td>
<td>is False</td>
</tr>
</tbody>
</table>
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

(True implies
  [(True or False ) and (True 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**

(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

Brian Williams, Fall 10

---

**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

Brian Williams, Fall 10
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

Brian Williams, Fall 10

---

**Example: Determining the Truth of a Sentence**

True!

If a sentence S evaluates to True in interpretation I, then:
- “I satisfies S”
- “I is a Model of S”

**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

Brian Williams, Fall 10
Satisfiability versus Validity

Satisfiable

A sentence is satisfiable if there is an interpretation (a truth assignment) that makes the clause true.

- (not A or B) is satisfiable.
- (A implies not B) and (A implies B) is unsatisfiable.

Valid

A sentence is valid if it is true for all interpretations.

- Is (not A or A or B) valid?
  Yes, it is valid over all possible interpretations.
- Is (A or B) valid with respect to the interpretations {A=true, B=false} and {A=false, B=false}?

Outline

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

- Literal: A proposition or its negation.
  - B, Not A
- Clause: A disjunction (“or”) of literals.
  - (not A or B or E)
- Conjunctive Normal Form:
  A conjunction (“and”) of clauses.
  - \( \Phi = (A \lor B \lor C) \land (\neg A \lor B \lor E) \land (\neg B \lor C \lor D) \)
  - Represented by a set of clauses.

Reduction to Clausal Form: Engine Example

\[(\text{mode}(E1) = \text{ok implies } \text{thrust}(E1) = \text{on iff } (\text{flow}(V1) = \text{on and flow}(V2) = \text{on})) \land \]
\[(\text{mode}(E1) = \text{ok or mode}(E1) = \text{unknown}) \land \]
\[\neg (\text{mode}(E1) = \text{ok and mode}(E1) = \text{unknown}) \]

- 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
  - Reduction to Clauses
- Propositional Satisfiability
- Empirical, Average Case Analysis
- Appendices
Propositional Satisfiability

Input: A Propositional Satisfiability Problem is a pair \(<P, \Phi>\), where:
- \(P\) is a finite set of propositions.
- \(\Phi\) is a propositional sentence over \(P\)
  - We assume it is reduced to a set of clauses.

Output: True iff there exists a model of \(\Phi\).

Is an instance of a CSP:
- Variables: Propositions
- Domain: \{True, False\}
- Constraints: Clauses

Models of \(<P, \Phi>\)

- An interpretation is a truth assignment to all propositions \(P\).
- A model is an interpretation such that all clauses are satisfied:
  - A clause is satisfied iff at least one literal is true.
  - A clause is violated iff all literals are false.

Example:  
C1: Not A or B  
C2: Not C or A  
C3: Not B or C
Testing Satisfiability of \(<P, \Phi >\)

1. Apply systematic, complete procedure
   • BT + unit propagation, shortest clause heuristic
     - [Davis, Logmann, & Loveland 1962; Crawford & Auton 1997; Nayak & Williams, 1997]

2. Apply stochastic, incomplete procedure
   • [Minton et al. 90; Selman et al. 1993] – see Appendix

3. Apply exhaustive clausal resolution
   • [Davis, Putnam 1960; Dechter Rish 1994]
Outline

- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- 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  \[\text{satisfied}\]
- C2: Not C or A  \[\text{satisfied}\]
- C3: Not B or C  \[\text{satisfied}\]
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

Brian Williams, Fall 10
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

Brian Williams, Fall 10
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
  - F
  - T
• C2: Not C or A
  - F
  - T
• C3: Not B or C
  - F
  - T

Brian Williams, Fall 10
Clausal Backtrack Search: Recursive Formulation

**Procedure**: $BT(\Phi, A)$

**Input**: A *cnf* theory $\Phi$, An assignment $A$ to some propositions in $\Phi$.

**Output**: true if $\Phi$ is satisfiable; false otherwise.

If a clause in $\Phi$ is violated, Return false;
Else If all propositions in $\Phi$ are assigned by $A$, Return true;
Else $Q =$ some proposition in $\Phi$ unassigned by $A$;
    Return ($BT(\Phi, A[Q = True])$ or $BT(\Phi, A[Q = False])$)

Outline

- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- Appendices
Unit Clause Resolution

Idea: Apply arc consistency (AC-3) to binary clauses

Clause: (not A or B)

\[ \begin{array}{cc}
A & \text{not} \ A \\
T & F \\
B & \text{not} \ B \\
T & F \\
\end{array} \]

Unit clause resolution (aka unit propagation 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 = repeated application of rule.

Brian Williams, Fall 10

Unit Propagation Examples

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

Support

C4 \rightarrow A \rightarrow True \rightarrow C1 \rightarrow True \rightarrow B \rightarrow True \rightarrow C3 \rightarrow True \rightarrow C

Brian Williams, Fall 10
Unit Propagation Examples

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

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  
record C as a support for L and  
for each clause C’ mentioning “not L”,  
propagate(C )  
end propagate
Unit Propagation

\[ C_1: \neg r \lor q \lor p \]
\[ C_2: \neg p \lor \neg t \]

Procedure: \texttt{propagate}(C)  
\begin{align*}
&\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') \\
&\text{end propagate}
\end{align*}
Unit Propagation

**Procedure:** \( \text{propagate}(C) \)  
// \( C \) 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 “\( \neg L \)”,
- \( \text{propagate}(C') \)

**Unit Propagation**

**Procedure:** \( \text{propagate}(C) \)  
// \( C \) 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 “\( \neg L \)”,
- \( \text{propagate}(C') \)

Brian Williams, Fall 10
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

Outline

• Propositional Logic
• Propositional Satisfiability
  • Backtrack Search
  • Unit Propagation
  • DPLL: Unit Propagation + Backtrack Search
• Empirical, Average Case Analysis
• Appendices
Propositional Satisfiability using DPLL
[Davis, 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 satisfied
• C2: Not C or A satisfied
• C3: Not B or C satisfied

Brian Williams, Fall 10
How Do We Fold Unit Propagation into Backtracking?

**Procedure**: BT(Φ, A)

**Input**: A *cnf* theory Φ,
An assignment A to some propositions in Φ

**Output**: A decision of whether Φ is satisfiable.

If a clause in Φ is violated, Return false;
Else If all propositions of Φ are assigned in A, Return true;
Else Q = some unassigned proposition in Φ;
Return (BT(Φ, A[Q = True]) or
BT(Φ, A[Q = False]))

Hint: Like MAC and Forward Checking:
- limited inference
- apply inference after assigning each variable.

D(P)LL Procedure
[Davis, Logmann, Loveland, 1961]

**Procedure**: DPLL(Φ, A)

**Input**: A *cnf* theory Φ,
An assignment A to propositions in Φ

**Output**: A decision of whether Φ is satisfiable.

**A’ = propagate(Φ)**;
If a clause in Φ is violated, given A’ Return false;
Else If all propositions of Φ are assigned in A’, Return true;
Else Q = some unassigned proposition in Φ;
Return (DPLL(Φ, A [Q = True]) or
DPLL(Φ, A [Q = False]))
Outline

- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- Appendices

Hardness of 3SAT

Courtesy of Bart Selman. Used with permission.
The 4.3 Point

![Graph showing the relationship between the ratio of constraints to variables (Alpha) and computational effort. The graph displays two axes: the x-axis represents the ratio of constraints to variables, while the y-axis represents the computational effort. The graph includes two sets of data points: one for solvable cases (green) and another for impossible cases (orange). The presence of a peak indicates a critical point where computational effort significantly increases as the ratio of constraints to variables increases.]

Courtesy of Bart Selman. Used with permission.

Image by MIT OpenCourseWare.
Intuition

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

• At high ratios:
  • many clauses
  • inconsistencies easily detected

Courtesy of Bart Selman. Used with permission.
Phase Transitions: 2, 3 4, 5 and 6-SAT

![Graph showing phase transitions for 2, 3, 4, 5, and 6-SAT]

Courtesy of Bart Selman. Used with permission.

Required Appendices

You are responsible for reading and knowing this material:

1. Local Search using Min_Conflict and GSAT
2. 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. Repeat until consistent:
   1. Select a variable in a conflict (violated constraint)
   2. Assign it a value that minimizes the number of conflicts (break ties randomly).

Graph Coloring
Initial Domains

GSAT

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

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

C1, C2, C3 violated

True     False     True
A        B         C

C3 violated    C2 violated    C1 violated
**GSAT**

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

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

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

C1 violated

<table>
<thead>
<tr>
<th>False</th>
<th>True</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfied</td>
<td>Satisfied</td>
<td>C1,C2,C3 violated</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 Min_Conflict and GSAT
2. 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 Implies:
   • E1 iff E2  =>  (E1 implies E2) and (E2 implies E1)
   • E1 implies E2  =>  not E1 or E2

Eliminate IFF:
Engine Example

(mode(E1) = ok implies
   (thrust(E1) = on iff (flow(V1) = on and flow(V2) = on))) and
(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)

(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

(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)

(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)

Brian Williams, Fall 10

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))

Move Conjunctions Out: Engine Example

(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))

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

4) “Simplify by Equivalence”
   remove double negations
   - \((\neg \neg E_1) \implies E_1\)
   apply commutativity and associativity
   - \((E_1 \text{ or } (E_3 \text{ or } (\neg E_1))) \implies (E_1 \text{ or } (\neg E_1) \text{ or } E_3)\)
   remove duplicate literals
   - \((E_1 \text{ or } E_1) \implies E_1\)
   remove duplicate clauses
   - \((E_1 \text{ or } (\neg E_2)) \text{ and } (E_1 \text{ or } (\neg E_2)) \implies (E_1 \text{ or } (\neg E_2))\)
   reduce by tautology
   - \((E_1 \text{ or } \ldots \text{ or } (\neg E_1)) \implies \text{true}\)
   definition of and/or
   - \(\text{true and } E_1 \implies E_1\)
   - \(\text{false and } E_1 \implies \text{false}\)
   - \((\text{false or } E_1) \implies E_1\)

Brian Williams, Fall 10

Reducing Propositional Formula to Clauses (CNF)

1) Eliminate IFF and Implies
   - \(E_1 \text{ iff } E_2 \implies (E_1 \text{ implies } E_2) \text{ and } (E_2 \text{ implies } E_1)\)
   - \(E_1 \text{ implies } E_2 \implies \neg E_1 \text{ or } E_2\)

2) Move negations in towards propositions using
   De Morgan’s Theorem:
   - \(\neg (E_1 \text{ and } E_2) \implies (\neg E_1) \text{ or } (\neg E_2)\)
   - \(\neg (E_1 \text{ or } E_2) \implies (\neg E_1) \text{ and } (\neg E_2)\)
   - \(\neg (\neg E_1) \implies E_1\)

3) Move conjunctions out using Distributivity
   - \(E_1 \text{ or } (E_2 \text{ and } E_3) \implies (E_1 \text{ or } E_2) \text{ and } (E_1 \text{ or } E_3)\)

4) Simplify by Equivalence

Brian Williams, Fall 10