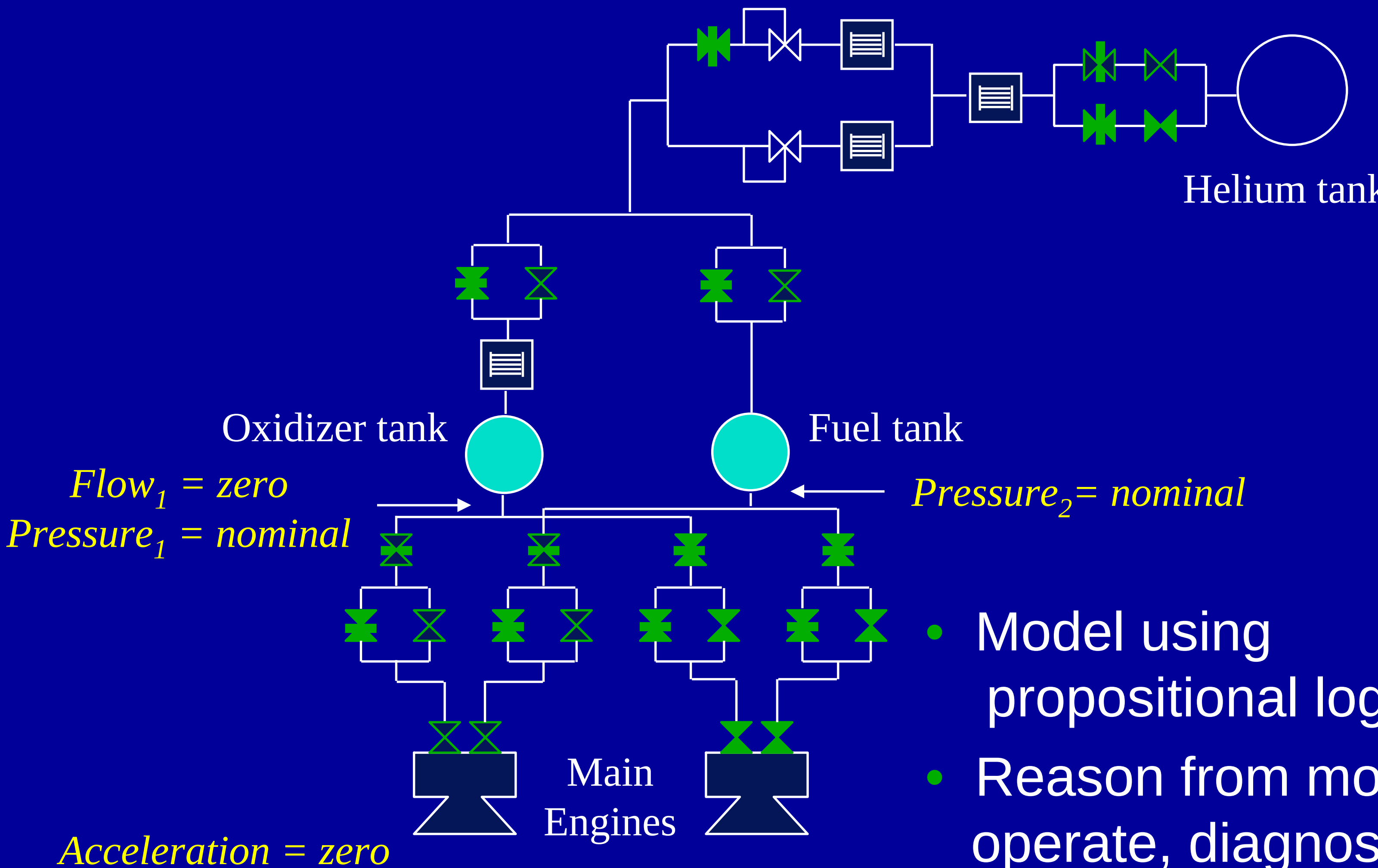
Propositional Logic and Satisfiability

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.

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)(IMPLIES S2 S1))

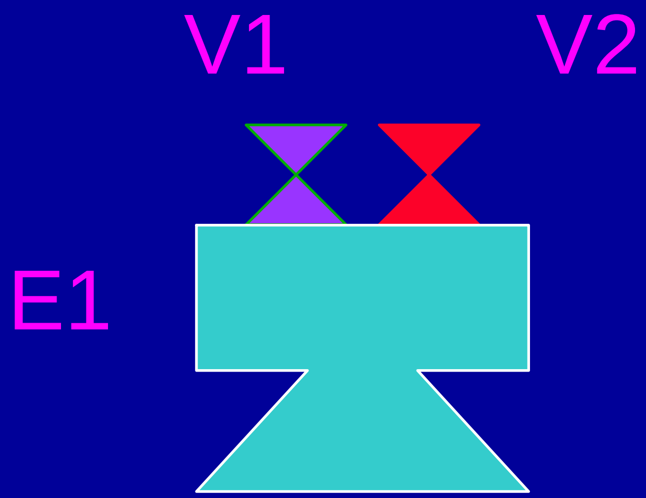
Propositional Sentences: Engine Example

(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 or B) and C
- I = {A=True, B=False, C=True}
- I = {A=False, B=True, C=False}

All Interpretations →

A	B	C
True	True	True
True	True	False
True	False	True
True	False	False
False	True	True
False	True	False
False	False	True
False	False	False

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

Not S	S
False	True
True	False

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

S1 and S2	S1	S2
True	True	True
False	True	False
False	False	True
False	False	False

S1 or S2	S1	S2
True	True	True
True	True	False
True	False	True
False	False	False

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
• "S_1" implies "S_2" is True	iff	"S_1" is False or "S_2" is True
• "S_1" iff S_2 is True	iff	"S_1 implies S_2" is True and "S_2 implies S_1" is True

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

(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

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

(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:

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 implies False) and (False implies 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

[(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

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:

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

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

- I satisfies S
- I is a *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

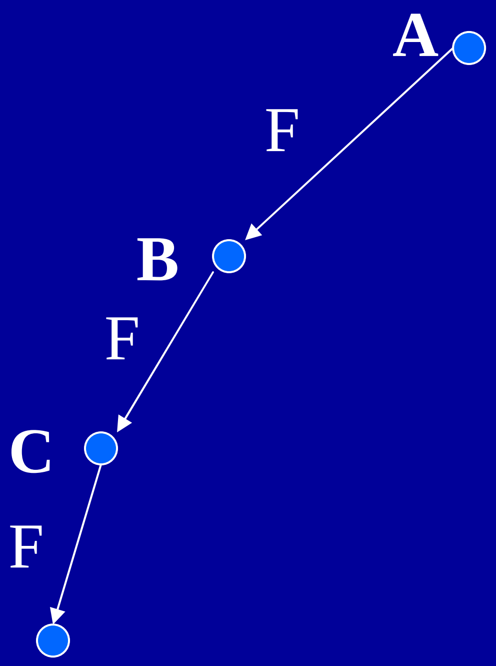
- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Appendices

Propositional Satisfiability using Backtrack Search

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

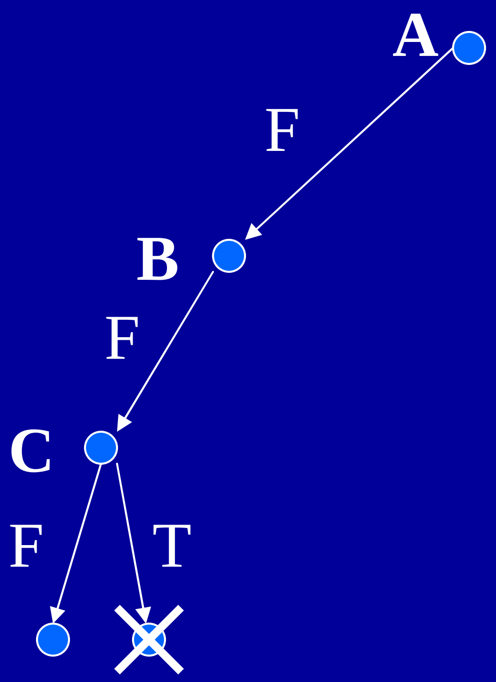
Example:

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



Propositional Satisfiability using Backtrack Search

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



Example:

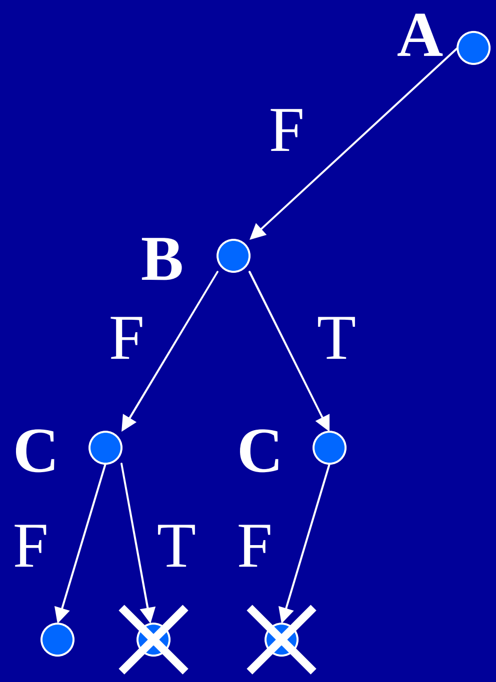
- C1: Not A or B   s
- C2: N~~ot~~ C or ~~A~~   u
- C3: Not B or C   s

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 S
- C2: Not C or ~~A~~ S
- C3: N~~ot~~ B or ~~C~~ V

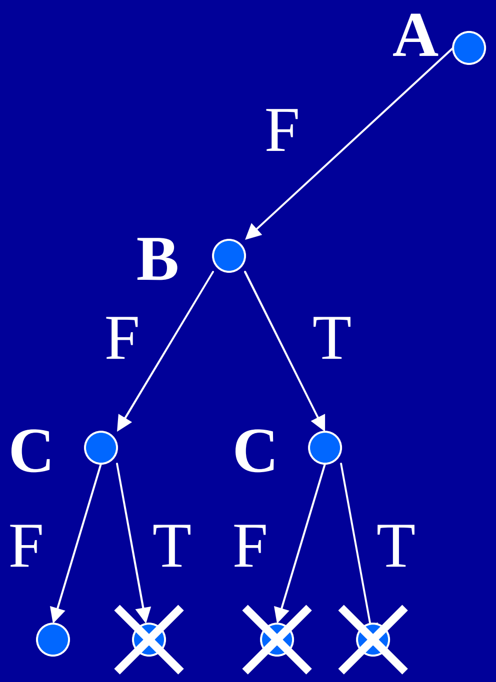


Propositional Satisfiability using Backtrack Search

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

Example:

- C1: Not A or B s
- C2: ~~Not~~ C or ~~A~~ v
- C3: ~~Not~~ B or C s

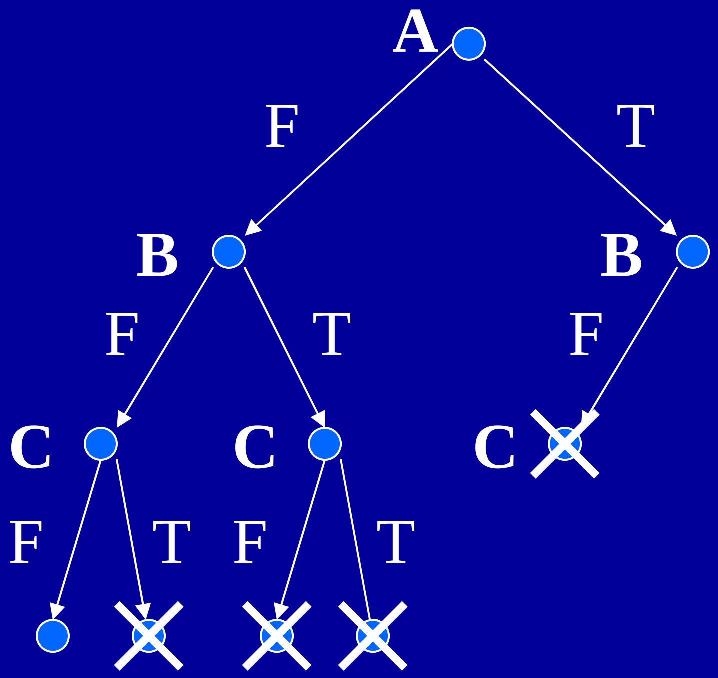


Propositional Satisfiability using Backtrack Search

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

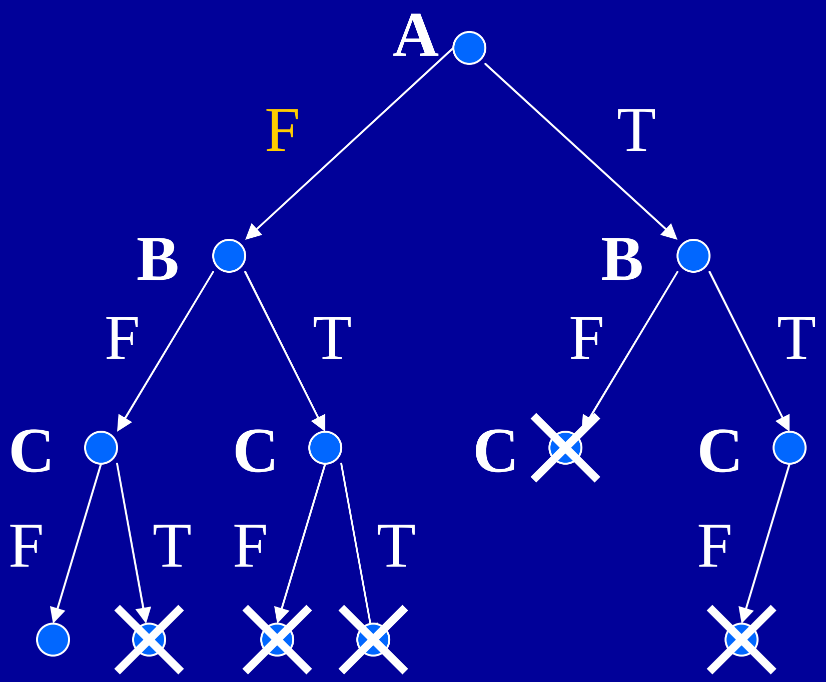
Example:

- C1: ~~Not~~ A or ~~B~~ v
- C2: Not C or A S
- C3: Not B or C S



Propositional Satisfiability using Backtrack Search

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



Example:

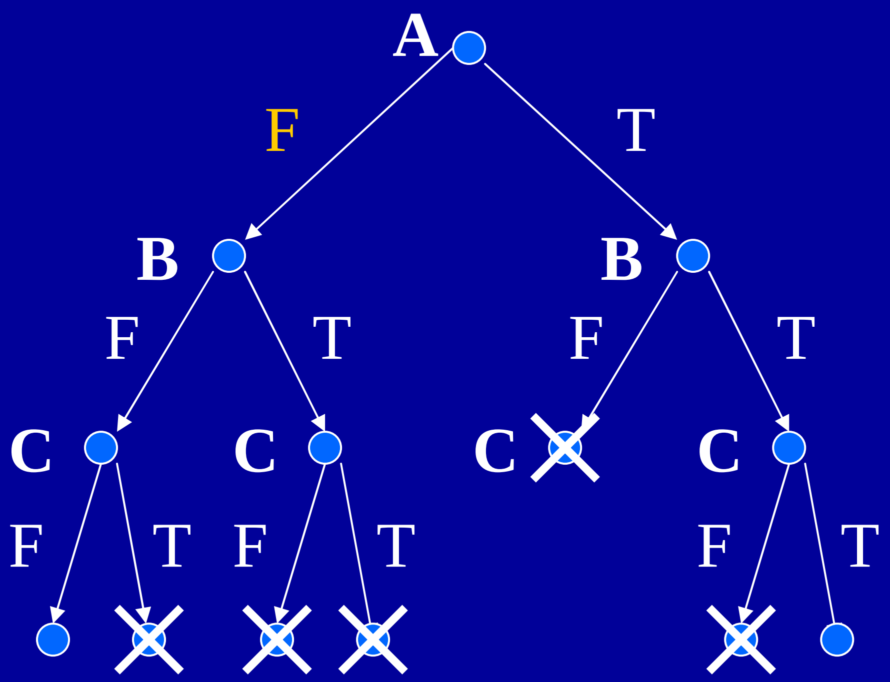
- C1: ~~Not~~ A or B S
- C2: Not C or A S
- C3: ~~Not~~ B or ~~C~~ v

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 s
- C2: Not C or A s
- C3: ~~Not~~ B or C s



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)

$\{F\} \longleftarrow \{T,F\}$?

$\{T\} \longleftarrow \{T,F\}$?

Unit resolution rule:

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

- $$\frac{(\text{not } A) \quad (\text{not } B) \quad (A \text{ or } B \text{ or } C)}{C}$$

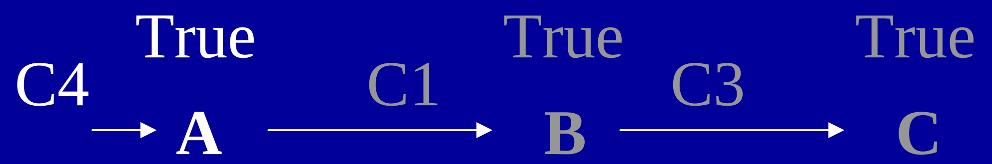
Unit Propagation Examples

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

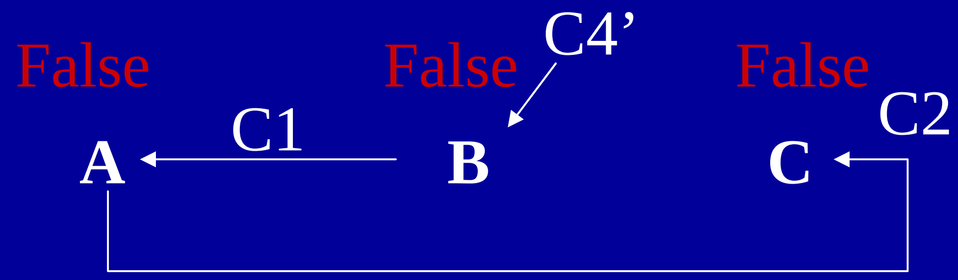
$$\xrightarrow{C4} \overset{\text{True}}{\mathbf{A}} \xrightarrow{\quad C1 \quad} \overset{\text{True}}{\mathbf{B}} \xrightarrow{\quad C3 \quad} \overset{\text{True}}{\mathbf{C}}$$

Unit Propagation Examples

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



- C4': Not B Satisfied



Unit Propagation

true *false*

$\boxed{r}$ $\boxed{q}$

$\boxed{t}$

$C_2: \neg p \vee \neg t$

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

$\boxed{p}$

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*

$\boxed{r}$ $\boxed{q}$

$\boxed{t}$

C_2: $\neg\, p \vee \neg\, t$

C_1 : $\neg r \vee q \vee p$ $\boxed{p}$

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* $\boxed{t}$

$\boxed{r}$ $\boxed{q}$

$C_2 : \neg p \vee \neg t$

true

$C_1 : \neg r \vee q \vee p$ $\boxed{p}$

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*

$\boxed{r}$ $\boxed{q}$ $\boxed{t}$

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

true

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

$\boxed{p}$

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

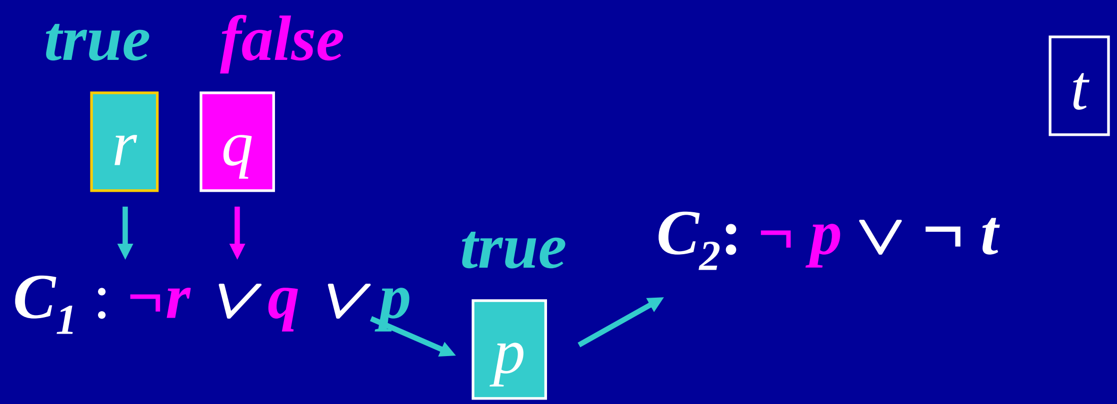
record **C** as a support for **L** and

for each clause ***C'*** mentioning **"**not **L"**,

propagate**(*C'*)**

end *propagate*

Unit Propagation



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

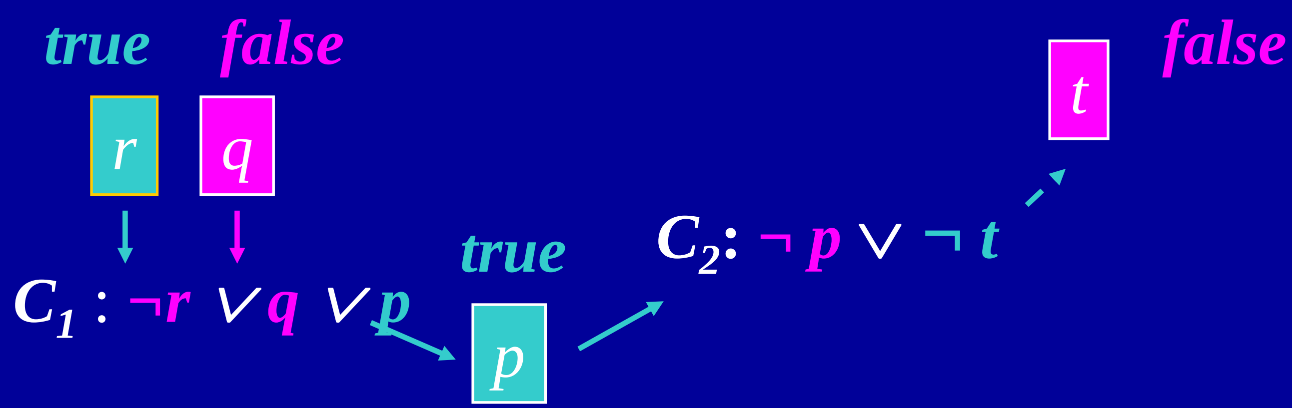
record **C** as a support for **L** and

for each clause ***C'*** mentioning **"**not **L"**,

propagate**(*C'***)

end *propagate*

Unit Propagation



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*

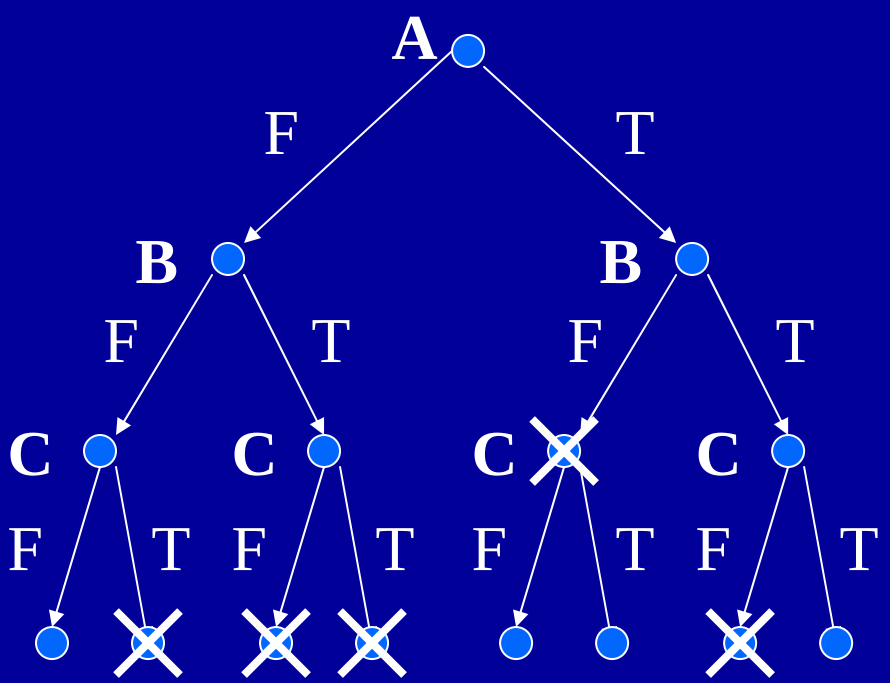
Outline

- 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.

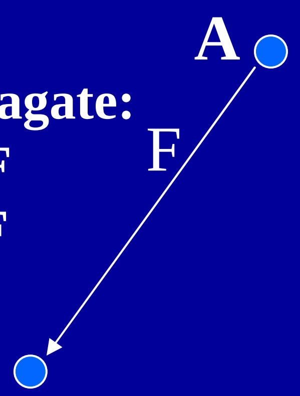
Example:

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

Propagate:

C = F

B = F



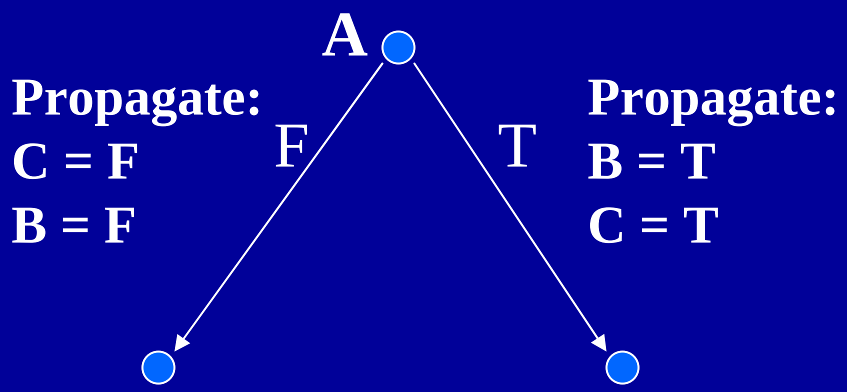
Propositional Satisfiability by DPLL

[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 <u>B</u> s
- C2: Not C or <u>A</u> s
- C3: ~~Not~~ B or <u>C</u> s



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’ = 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 = some unassigned proposition in Phi;
6. Return (DPLL(Phi, A’[Q = True]) or
7. DPLL(Phi, A’[Q = 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*

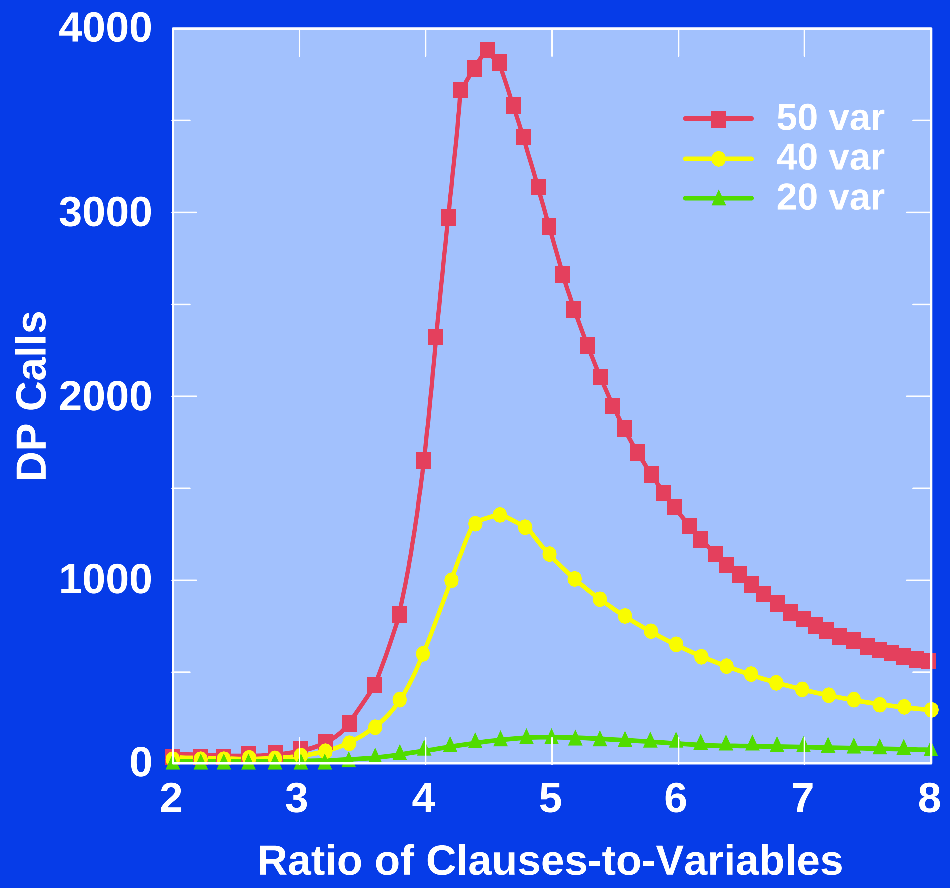
- Apply stochastic, incomplete procedures
  - GSAT (Selman et. al 1993)
  - Walksat (Selman & Kautz 1993)
    - greedy local search + noise to escape local minima

Required Appendices

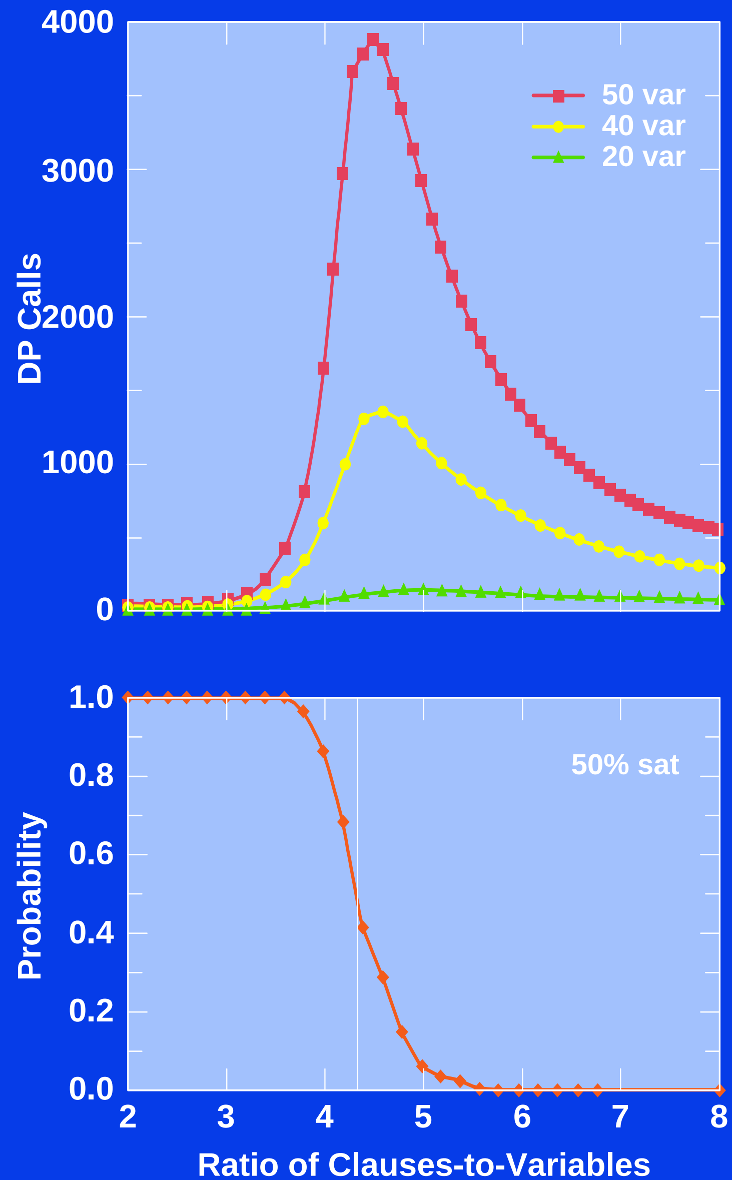
You are responsible for reading and knowing this material:

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

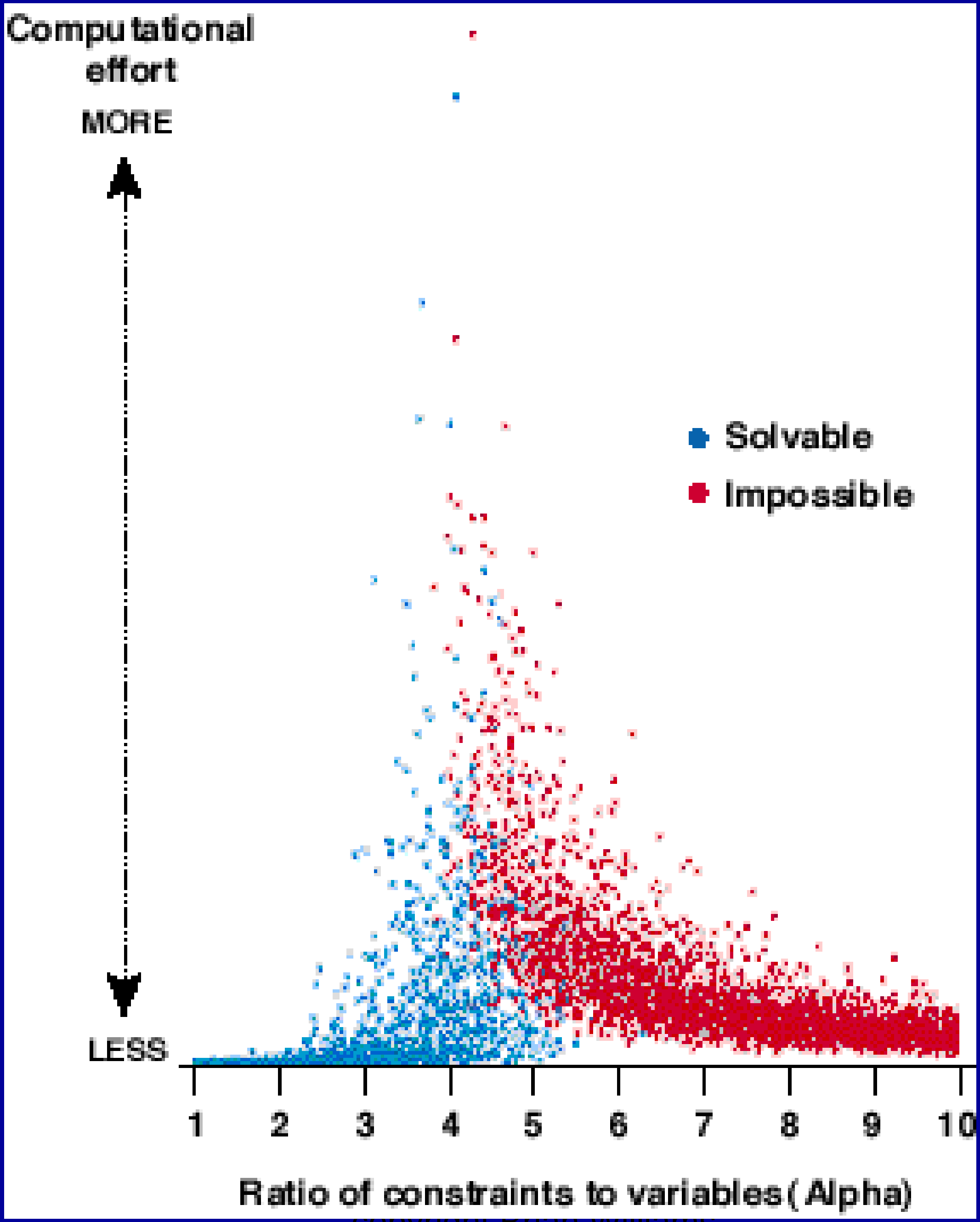
Hardness of 3SAT



The 4.3 Point



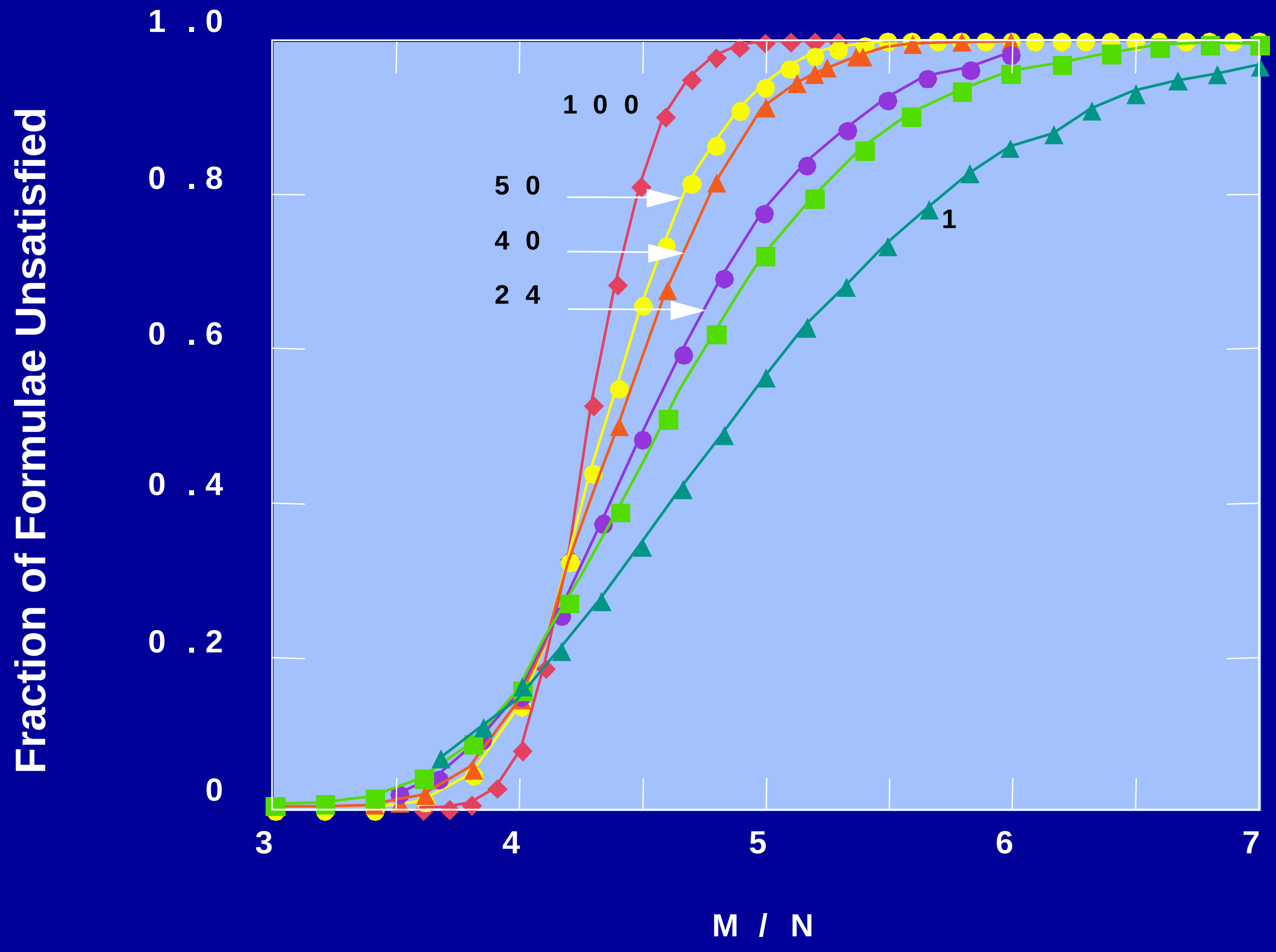
Mitchell, Selman, and Levesque 1991


Computational effort
MORE
LESS
Solvable
Impossible
1
2
3
4
5
6
7
8
9
10
Ratio of constraints to variables (Alpha)

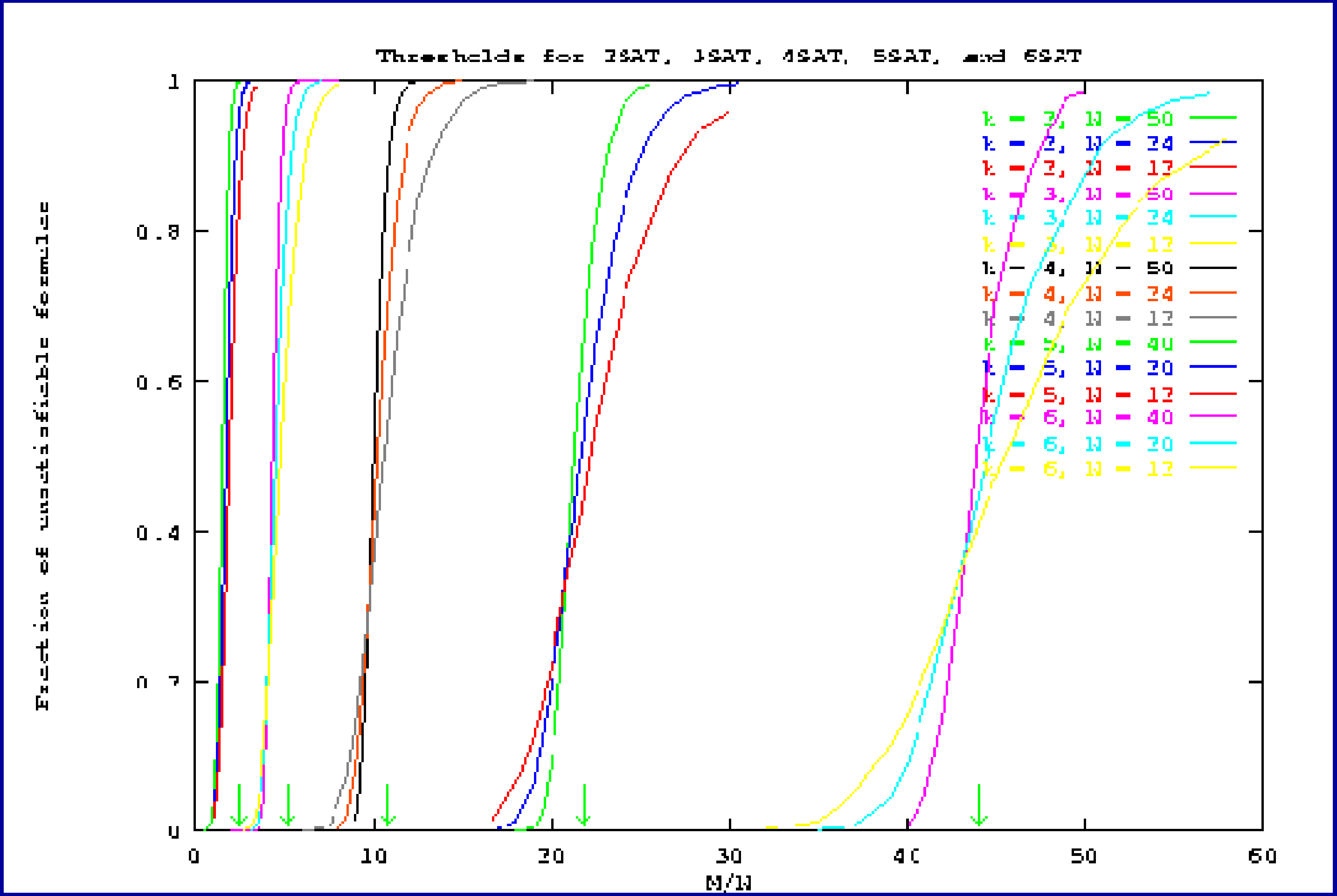
Intuition

- 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



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)

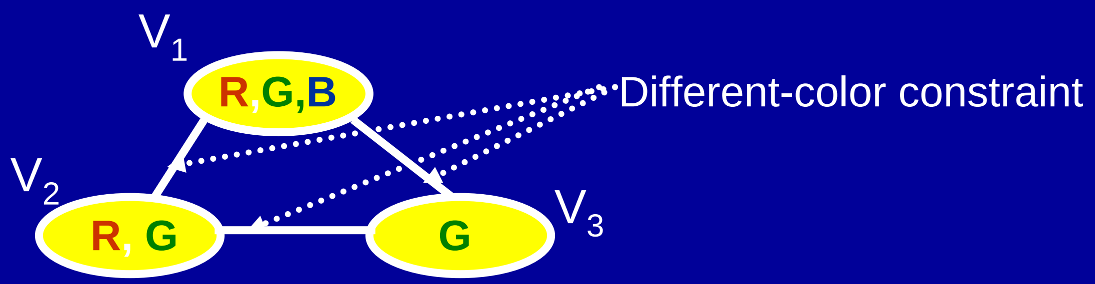
Spike Hubble Telescope Scheduler [Minton et al.]

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).

Graph Coloring

Initial Domains



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	B	C
	True	False	True
C1, C2, C3 violated	**A**	**B**	**C**
	False	True	False
	C3 violated	C2 violated	C1 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

	True	False	False
C1 violated	**A**	**B**	**C**
	False	True	True
	Satisfied	Satisfied	C1,C2,C3 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

	True	True	False
Satisfied	**A**	**B**	**C**

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 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

(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)

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)**))

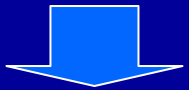
Reducing Propositional Formula to Clauses (CNF)

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)

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)