6.825 Techniques in Artificial Intelligence

Satisfiability and Validity

Satisfiable sentence: there exists a truth value assignment for the variables that makes the sentence true (truth value = **t**).

- Algorithm?
- Try all the possible assignments to see if one works.

Valid sentence: all truth value assignments for the variables make the sentence true.

- Algorithm?
- Try all possible assignments and check that they all work.

Are there better algorithms than these?

Satisfiability Problems

Many problems can be expressed as a list of constraints. Answer is assignment to variables that satisfy all the constraints.

Examples:

- Scheduling people to work in shifts at a hospital
  - Some people don't work at night
  - No one can work more than x hours a week
  - Some pairs of people can't be on the same shift
  - Is there assignment of people to shifts that satisfy all constraints?
- Finding bugs in programs [Daniel Jackson, MIT]
  - Write logical specification of, e.g. air traffic controller
  - Write assertion "two airplanes on same runway at same time"
  - Can these be satisfied simultaneously?

Conjunctive Normal Form

Satisfiability problems are written as conjunctive normal form (CNF) formulas:

$$(A \quad B \quad \neg C) \wedge (B \quad D) \wedge (\neg A) \wedge (B \quad C)$$

- $(A \quad B \quad \neg C)$ is a clause, which is a disjunction of literals
- A, B, and : C are literals, each of which is a variable or the negation of a variable.
- Each clause is a requirement which must be satisfied and it has different ways of being satisfied.
- Every sentence in propositional logic can be written in CNF

Converting to CNF

1. Eliminate arrows using definitions
2. Drive in negations using De Morgan's Laws

$$\neg(\quad) \equiv \quad \neg\neg$$
$$\neg(\quad) \equiv \quad \neg\neg$$

3. Distribute **or** over **and**

$$A \quad (B \wedge C) \equiv (A \quad B) \wedge (A \quad C)$$

4. Every sentence can be converted to CNF, but it may grow exponentially in size

CNF Conversion Example

$$(A \quad B) \rightarrow (C \rightarrow D)$$

1. Eliminate arrows

$$\neg(A \quad B) \quad (\neg C \quad D)$$

2. Drive in negations

$$(\neg A \wedge \neg B) \quad (\neg C \quad D)$$

3. Distribute

$$(\neg A \quad \neg C \quad D) \wedge (\neg B \quad \neg C \quad D)$$

Simplifying CNF

- An empty clause is false (no options to satisfy)
- A sentence with no clauses is true (no requirements)
- A sentence containing an empty clause is false (there is an impossible requirement)

Recitation Problems - I

Convert to CNF

1. $(A \Leftrightarrow B) \Leftrightarrow C$
2. $A \Leftrightarrow (B \Leftrightarrow C)$
3. $(A \Rightarrow B) \wedge (B \Rightarrow A)$
4. $\neg(\neg P \Rightarrow (P \wedge Q))$
5. $(P \vee (Q \Rightarrow R)) \Rightarrow (P \vee (R \Rightarrow Q))$
6. $(P \wedge Q) \vee ((Q \Rightarrow R) \Rightarrow (P \wedge R))$

Algorithms for Satisfiability

Given a sentence in CNF, how can we prove it is satisfiable?

Enumerate all possible assignments and see if sentence is true for any of them. But, the number of possible assignments grows exponentially in the number of variables.

Consider a search tree where at each level we consider the possible assignments to one variable, say P. On one branch, we assume P is **f** and on the other that it is **t**.

Given an assignment for a variable, we can simplify the sentence and then repeat the process for another variable.

Assign and Simplify Example

$$(P \vee Q) \wedge (P \vee \neg Q \vee R) \wedge (T \vee \neg R) \wedge (\neg P \vee \neg T) \wedge \neg(P \vee S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$$

If we assign P=**f**, we get simpler set of constraints

- $P \vee Q$ simplifies to Q
- $P \vee \neg Q \vee R$ simplifies to $\neg Q \vee R$
- $\neg P \vee \neg T$ is satisfied and can be removed
- $P \vee S$ simplifies to S

Result is

$$(Q) \wedge (\neg Q \vee R) \wedge (T \vee \neg R) \wedge (S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$$

Assign and Simplify

Given a CNF sentence Δ and a literal U

- Delete all clauses containing U (they're satisfied)
- Delete :U from all remaining clauses (because U is not an option)

We denote the simplified sentence by Δ(U)

Works for positive and negative literals U

Search Example

$$(P \vee Q) \wedge (P \vee \neg Q \vee R) \wedge (T \vee \neg R) \wedge (\neg P \vee \neg T) \wedge \neg(P \vee S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$$

Δ(:P) Δ(P)

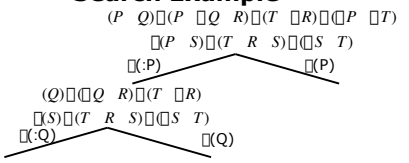
Search Example

$$(P \vee Q) \wedge (P \vee \neg Q \vee R) \wedge (T \vee \neg R) \wedge (\neg P \vee \neg T) \wedge \neg(P \vee S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$$

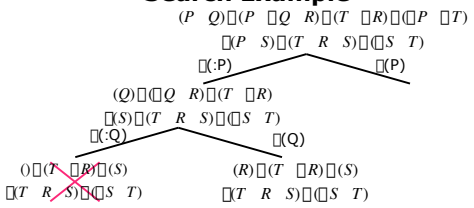
Δ(:P) Δ(P)

$$(Q) \wedge (\neg Q \vee R) \wedge (T \vee \neg R)$$
$$\neg(S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$$

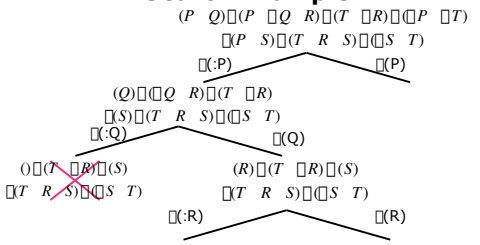
Search Example



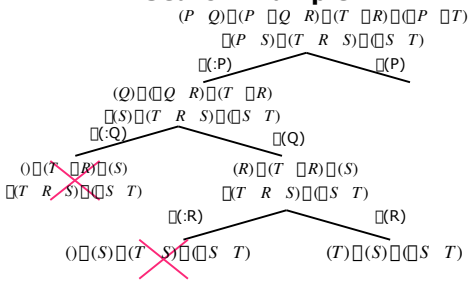
Search Example



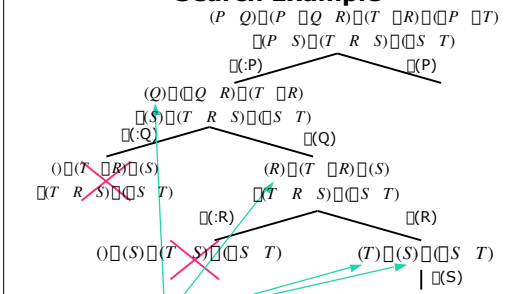
Search Example



Search Example



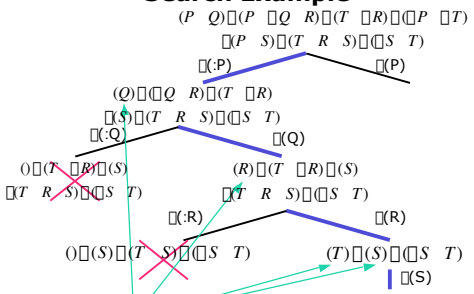
Search Example



The assignment for a literal appearing by itself in a clause is forced: true for a positive literal, false for a negative literal.

True

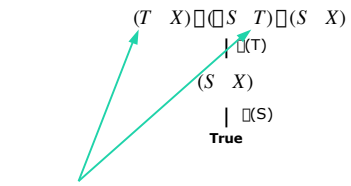
Search Example



The assignment for a literal appearing by itself in a clause is forced: true for a positive literal, false for a negative literal.

True

Another Example



Since T occurs only positively, it might as well be assigned to true

DPLL(Δ)

- If Δ is empty, return true
  (embrace truth)
- If there is an empty clause in Δ, return false
  (reject falsity)
- If there is a unit clause U in Δ, return DPLL(Δ(U))
  (accept the inevitable)
- If there is a pure literal U in Δ, return DPLL(Δ(U))
  (go with the flow)
- For some variable v
  (take a guess)
  – If DPLL(Δ(v)) then return true
  – Else return DPLL(Δ(: v))

Unit clause has only one literal
Pure literal only occurs positively or negatively

Recitation Problems - II

How would you modify DPLL so it:

- returns a satisfying assignment if there is one, and false otherwise
- returns *all* satisfying assignments

Would using DPLL to return all satisfying assignments be any more efficient than simply listing all the assignments and checking to see whether they're satisfying? Why or why not?

Making good guesses

MOMS heuristic for choosing variable v:

Maximum number of Occurrences,
Minimum Sized clauses

- Choose highly constrained variables
- If you're going to fail, fail early

Soundness and Completeness

Properties of satisfiability algorithms:

- Sound – if it gives you an answer, it's correct
- Complete – it always gives you an answer

DPLL is sound and complete

We will now consider some algorithms for satisfiability that are sound but not complete.

- If they give an answer, it is correct
- But, they may not give an answer
- They may be faster than any complete algorithm

GSAT

Hill climbing in the space of total assignments

- Starts with random assignment for all variables
- Moves to "neighboring" assignment with least cost (flip a single bit)

Cost(assignment) = number of unsatisfied clauses

Loop n times

- Randomly choose assignment A
- Loop m times
  - Flip the variable that results in lowest cost
  - Exit if cost is zero

GSAT vs DPLL

- GSAT is sound
- It's not complete
- You couldn't use it effectively to generate all satisfying assignments
- For a while, it was beating DPLL in SAT contests, but now the DPLL people are tuning up their heuristics and doing better
- Weakly constrained problems are easy for both DPLL and GSAT
- Highly constrained problems are easy for DPLL but hard for GSAT
- Problems in the middle are hard for everyone

WALKSAT

Like GSAT with additional "noise"

Loop n times
- Randomly choose assignment A
- Loop m times
  - Randomly select unsatisfied clause C
  - With p = 0.5 either
    - Flip the variable in C that results in lowest cost, or
    - Flip a randomly chosen variable in C
  - Exit if cost is zero