CSE 473 Propositional Logic SAT Algorithms

Dan Weld

(With some slides from Mausam, Stuart Russell, Dieter Fox, Henry Kautz, Min-Yen Kan…)

Irrationally held truths may be more harmful than reasoned errors.

- Thomas Huxley (1825-1895)

Overview

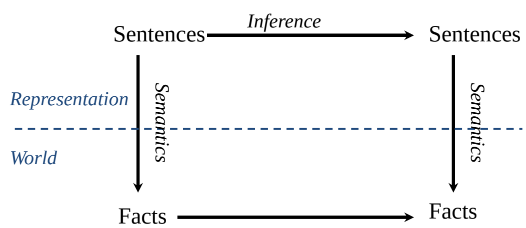
- Introduction & Agents
- Search, Heuristics & CSPs
- Adversarial Search
- **Logical Knowledge Representation**
- Planning & MDPs
- Reinforcement Learning
- Uncertainty & Bayesian Networks
- Machine Learning
- NLP & Special Topics

Propositional Logic

- Syntax
  - Atomic sentences: P, Q, …
  - Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$
- Semantics
  - Truth Tables
- Inference
  - Modus Ponens
  - Resolution
  - DPLL
  - GSAT
- Complexity

Semantics

- ***Syntax***: which arrangements of symbols are *legal*
  - (Def "sentences")
- ***Semantics***: what the symbols *mean* in the world
  - (Mapping between symbols and worlds)

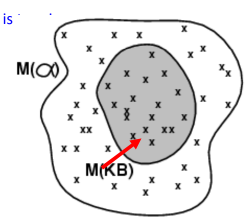


Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Models

- Logicians often think in terms of models, which are formally structured worlds with respect to which truth can be evaluated
  - In propositional case, each model = truth assignment
  - Set of models can be enumerated in a truth table
- We say *m* is a model ***of*** a sentence α if α is
- *M(α)* is the set of all models ***of*** α
- Then KB ⊨ α iff *M(KB)* ⊆ *M*(α)
-
  - E.g. *KB* = (P∨Q) ∧(¬P ∨ R)
    α = (P∨R)
- How to check?
  - One way is to enumerate all elements in the truth table – slow ☹



Satisfiability, Validity, & Entailment

- S is **satisfiable** if it is true in ***some*** world
- S is **unsatisfiable** if it is false ***all*** worlds
- S is **valid** if it is true in ***all*** worlds
- S1 **entails** S2 if ***wherever*** S1 is true S2 is also true

Types of Reasoning (Inference)

- Deduction (showing **entailment**, |=)
  - S = question
  - Prove that KB |= S
  - Two approaches:
    - Rules to derive new formulas from old (inference)
    - Show (KB $\wedge \neg$ S) is unsatisfiable
- Model Finding (showing **satisfiability**)
  - S = description of problem
  - Show S is satisfiable
  - A kind of constraint satisfaction

Propositional Logic: **Inference Algorithms**

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)

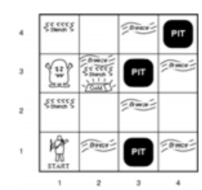
} **Deduction**

3. Exhaustive Enumeration
4. DPLL (Davis, Putnam Loveland & Logemann)
5. GSAT

} **Model Finding**

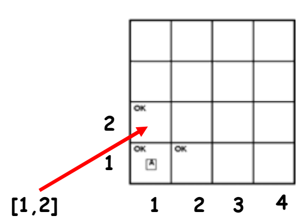
Wumpus World

- Performance measure
  - Gold: +1000, death: -1000
  - -1 per step, -10 for using the arrow
- Environment
  - Squares adjacent to wumpus are smelly
  - Squares adjacent to pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square

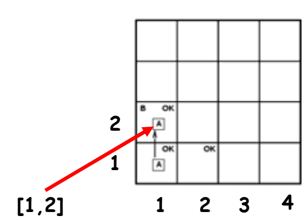


- Sensors: Stench, Breeze, Glitter, Bump, Scream
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot

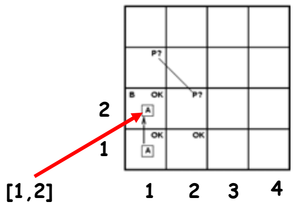
Exploring a wumpus world



Exploring a wumpus world

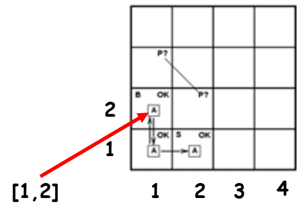


Exploring a wumpus world



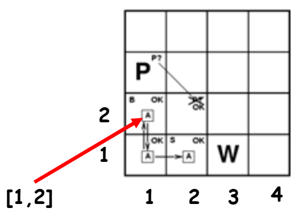
13

Exploring a wumpus world



14

Exploring a wumpus world



15

Wumpus world sentences: KB

Let $P_{i,j}$ be true if there is a pit in [i, j].
Let $B_{i,j}$ be true if there is a breeze in [i, j].

KB:

$\neg P_{1,1}$
$\neg B_{1,1}$

"Pits cause breezes in adjacent squares"

$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$
$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$

19

Full Encoding of Wumpus World

In propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
$B_{x,y} \Leftrightarrow (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y})$
$W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4}$
$\neg W_{1,1} \lor \neg W_{1,2}$
$\neg W_{1,1} \lor \neg W_{1,3}$
…

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

20

Model Finding

- Find an assignments to variables that makes a formula true

$\alpha = \neg P_{12}$ (ie "[1,2] is safe")

- Note: this is a kind of CSP, right?

State Estimation

- Maintaining a KB which records what you know about the (partially observed) world state
  - Prop logic
  - First order logic
  - Probabilistic encodings

---

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*, *breeze*, *glitter*]
  static: *KB*, initially containing the "physics" of the wumpus world
    x, *y*, *orientation*, the agent's position (init. [1,1]) and orient. (init. *right*)
    visited, an array indicating which squares have been visited, initially *false*
    action, the agent's most recent action, initially null
    plan, an action sequence, initially empty

  update *x*, *y*, *orientation*, *visited* based on *action*
  if *stench* **then** TELL($KB, S_{x,y}$) **else** TELL($KB, \neg S_{x,y}$)
  if *breeze* **then** TELL($KB, B_{x,y}$) **else** TELL($KB, \neg B_{x,y}$)
  if *glitter* **then** *action* ← *grab*
  else if *plan* is nonempty then *action* ← POP(*plan*)
  else if for some fringe square [*i*,*j*], ASK($KB, (\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**
    for some fringe square [*i*,*j*], ASK($KB, (P_{i,j} \vee W_{i,j})$) is *false* **then do**
   plan ← A*-GRAPH-SEARCH(ROUTE-PB([*x*,*y*], *orientation*, [*i*,*j*], *visited*))
   action ← POP(*plan*)
  else *action* ← a randomly chosen move
  return *action*

Propositional Logic: Inference Algorithms

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)

} Deduction

3. Exhaustive Enumeration
4. DPLL (Davis, Putnam Loveland & Logemann)
5. GSAT

} Model Finding

Inference 3: Model Enumeration

```
for (m in truth assignments){
   if  (m makes Φ true)
   then return "Sat!"
}
 return "Unsat!"
```

Model Checking: Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	false	false	false	false	false	false	false	true
false	false	false	false	false	false	true	false	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	false	true
false	true	false	false	false	false	true	<u>true</u>	<u>true</u>
false	true	false	false	false	true	false	<u>true</u>	<u>true</u>
false	true	false	false	false	true	true	<u>true</u>	<u>true</u>
false	true	false	false	true	false	false	false	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	true	true	true	true	true	true	false	false

alpha_1 = not P_{12} ("[1,2] is safe")

Inference 4: DPLL (Enumeration of *Partial* Models)

[Davis, Putnam, Loveland & Logemann 1962]

Version 1

```
dpll_1(pa){
  if (pa makes F false) return false;
  if (pa makes F true) return true;
  choose P in F;
  if (dpll_1(pa ∪ {P=0})) return true;
  return dpll_1(pa ∪ {P=1});
}
```

Returns true if F is satisfiable, false otherwise

DPLL Version 1

($a \lor b \lor c$)

($a \lor \neg b$)

($a \lor \neg c$)

($\neg a \lor c$)

DPLL Version 1

($a \lor b \lor c$)

($a \lor \neg b$)

($a \lor \neg c$)

($\neg a \lor c$)

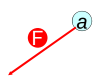
DPLL Version 1

($F \lor b \lor c$)

($F \lor \neg b$)

($F \lor \neg c$)

($T \lor c$)



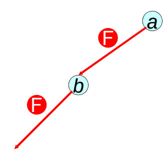
DPLL Version 1

($F \lor F \lor c$)

($F \lor T$)

($F \lor \neg c$)

($T \lor c$)



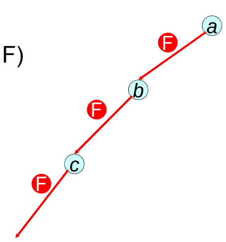
DPLL Version 1

($F \lor F \lor F$)

($F \lor T$)

($F \lor T$)

($T \lor F$)



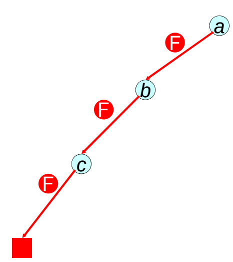
DPLL Version 1

F

T

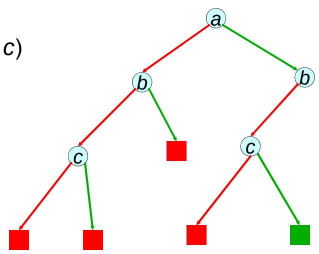
T

T



DPLL Version 1

$(a \vee b \vee c)$
$(a \vee \neg b)$
$(a \vee \neg c)$
$(\neg a \vee c)$



DPLL as Search

- Search Space?
- Algorithm?

Improving DPLL

If literal L_1 is true, then clause $(L_1 \vee L_2 \vee ...)$ is true
If clause C_1 is true, then $C_1 \wedge C_2 \wedge C_3 \wedge ...$ has the same value as $C_2 \wedge C_3 \wedge ...$
Therefore: Okay to delete clauses containing true literals!

Improving DPLL

If literal L_1 is true, then clause $(L_1 \vee L_2 \vee ...)$ is true
If clause C_1 is true, then $C_1 \wedge C_2 \wedge C_3 \wedge ...$ has the same value as $C_2 \wedge C_3 \wedge ...$
Therefore: Okay to delete clauses containing true literals!
If literal L_1 is false, then clause $(L_1 \vee L_2 \vee L_3 \vee ...)$ has the same value as $(L_2 \vee L_3 \vee ...)$
Therefore: Okay to delete shorten containing false literals!

Improving DPLL

If literal L_1 is true, then clause $(L_1 \vee L_2 \vee ...)$ is true
If clause C_1 is true, then $C_1 \wedge C_2 \wedge C_3 \wedge ...$ has the same value as $C_2 \wedge C_3 \wedge ...$
Therefore: Okay to delete clauses containing true literals!
If literal L_1 is false, then clause $(L_1 \vee L_2 \vee L_3 \vee ...)$ has the same value as $(L_2 \vee L_3 \vee ...)$
Therefore: Okay to delete shorten containing false literals!
If literal L_1 is false, then clause (L_1) is false
Therefore: the empty clause means false!

Representing Formulae

- CNF = Conjunctive Normal Form
  - Conjunction ($\wedge$) of Disjunctions ($\vee$)
- Represent as set of sets
  - ((A, B), (¬A, C), (¬C))
  - ((¬A), (A))
  - (())
  - ((A))
  - ()

DPLL version 2

```
dpll_2(F, literal){
  remove clauses containing literal
  if (F contains no clauses)return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;
  choose V in F;
  if (dpll_2(F, ¬V))return true;
  return dpll_2(F, V);
}
```

Partial assignment corresponding to a node is the set of chosen literals on the path from the root to the node

DPLL Version 2

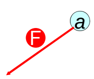
$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$

DPLL Version 2



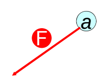
$(F \vee b \vee c)$

$(F \vee \neg b)$

$(F \vee \neg c)$

$(T \vee c)$

DPLL Version 2

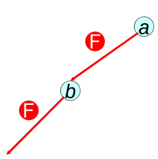


$(b \vee c)$

$(\neg b)$

$(\neg c)$

DPLL Version 2

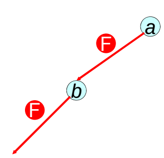


$(F \vee c)$

(T)

$(\neg c)$

DPLL Version 2



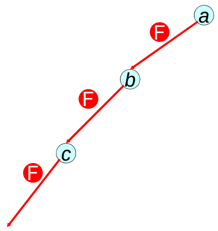
(c)

$(\neg c)$

DPLL Version 2

(F)

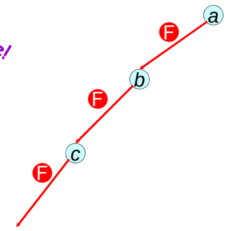
(T)



DPLL Version 2

Empty clause!

()



Benefit

- Can backtrack before getting to leaf

Structure in Clauses

- Unit Literals

  A literal that appears in a singleton clause

  {{¬b c}**{¬c}**{a ¬b e}{d b}{e a ¬c}}

  Might as well set it true! *And simplify*

  {{¬b} {a ¬b e}{d b}}

  {{d}}

- Pure Literals
  - A symbol that always appears with same sign
  - {{a ¬b c} {¬c **d** ¬e} {¬a ¬b e}{**d** b} {e a ¬c}}

  Might as well set it true! *And simplify*

  {{a ¬b c} {¬a ¬b e} {e a ¬c}}

In Other Words

Formula $(L) \wedge C_2 \wedge C_3 \wedge ...$ is only true when literal L is true

Therefore: Branch immediately on unit literals!

May view this as adding constraint propagation techniques into play

In Other Words

Formula $(L) \wedge C_2 \wedge C_3 \wedge ...$ is only true when literal L is true

Therefore: Branch immediately on unit literals!

If literal L does not appear negated in formula F, then setting L true preserves satisfiability of F

Therefore: Branch immediately on pure literals!

May view this as adding constraint propagation techniques into play

DPLL (previous version)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;


  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```


DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;
  if (F contains a unit or pure L)
     return dpll(F, L);
  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```

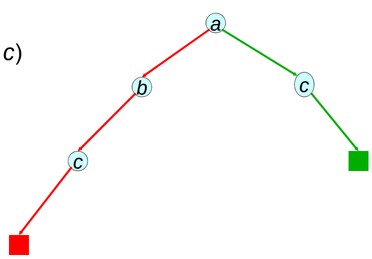

DPLL (for real)

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



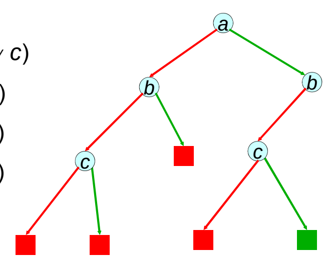
Compare with DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;
  if (F contains a unit or pure L)
     return dpll(F, L);
  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```

Where could we use a heuristic to further improve performance?

Heuristic Search in DPLL

- Heuristics are used in DPLL to select a (non-unit, non-pure) proposition for branching
- Idea: identify a most constrained variable
  - Likely to create many unit clauses
- MOM's heuristic:
  - **M**ost **o**ccurrences in clauses of **m**inimum length

Success of DPLL

- 1962 – DPLL invented
- 1992 – 300 propositions
- 1997 – 600 propositions (satz)
- Additional techniques:
  - Learning conflict clauses at backtrack points
  - Randomized restarts
  - 2002 (zChaff) 1,000,000 propositions – encodings of hardware verification problems

Other Ideas?

- How else could we solve SAT problems?

WalkSat (Take 1)

- ***Local*** search (Hill Climbing + Random Walk) over space of ***complete*** truth assignments
  - With prob p: flip any variable in any unsatisfied clause
  - With prob (1-p): flip best variable in any unsat clause
    - best = one which minimizes #unsatisfied clauses

Refining Greedy Random Walk

- Each flip
  - makes some false clauses become true
  - breaks some true clauses, that become false
- Suppose s1→s2 by flipping x. Then:

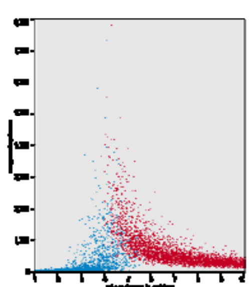
  #unsat(s2) = #unsat(s1) – make(s1,x) + break(s1,x)
- Idea 1: if a choice breaks nothing, it's likely good!
- Idea 2: near the solution, only the break count matters
  - the make count is usually 1

Walksat (Take 2)

```
state = random truth assignment;
while ! GoalTest(state) do
    clause := random member { C | C is false in state };
    for each x in clause do compute break[x];
    if exists x with break[x]=0 then var := x;
    else
        with probability p do
            var := random member { x | x is in clause };
        else
            var := arg x min { break[x] | x is in clause };
    endif
    state[var] := 1 – state[var];
end
return state;
```

Put everything inside of a restart loop.
Parameters: p, max_flips, max_runs

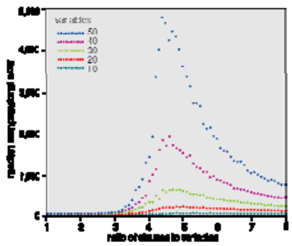
Random 3-SAT



- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - *n* variables, *l* clauses
- Which are the hard instances?
  - around $l/n = 4.3$

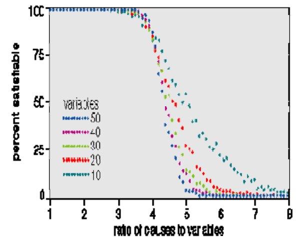
Random 3-SAT

- Varying problem size, *n*
- Complexity peak appears to be largely invariant of algorithm
  - backtracking algorithms like Davis-Putnam
  - local search procedures like GSAT
- *What's so special about 4.3?*



Random 3-SAT

- Complexity peak coincides with solubility transition
  - l/n < 4.3 problems under-constrained and SAT
  - l/n > 4.3 problems over-constrained and UNSAT
  - l/n=4.3, problems on "knife-edge" between SAT and UNSAT



Special Syntactic Forms

- General Form:
  - ((q∧¬ r) → s)) ∧ ¬ (s ∧ t)
- Conjunction Normal Form (CNF)
  - (¬ q ∨ r ∨ s) ∧ (¬ s ∨ ¬ t)
  - Set notation: { (¬ q, r, s), (¬ s, ¬ t) }
  - empty clause () = *false*
- Binary clauses: 1 or 2 literals per clause
  - (¬ q ∨ r) (¬ s ∨ ¬ t)
- Horn clauses: 0 or 1 positive literal per clause
  - (¬ q ∨ ¬ r ∨ s) (¬ s ∨ ¬ t)
  - (q∧r) → s (s∧t) → *false*

Prop. Logic Themes

- Expressiveness
  - Expressive but awkward
  - No notion of objects, properties, or relations
  - Number of propositions is fixed
- Tractability
  - NP in general
  - Completeness / speed tradeoff
  - Horn clauses, binary clauses