15-780: Graduate AI
Lecture 4. SAT, CSPs, and FOL

Geoff Gordon (this lecture)
Ziv Bar-Joseph
TAs Geoff Hollinger, Henry Lin

Admin

- *Questions on HW?*

Review

What you should know

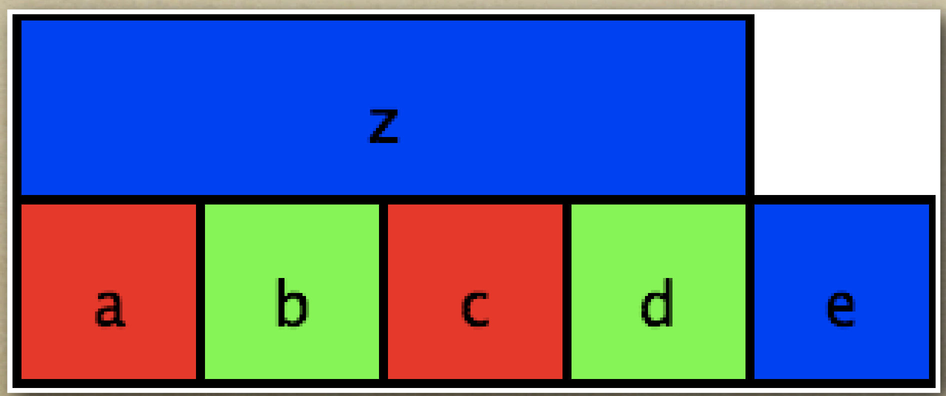
- *Propositional logic*
  - *syntax, truth tables*
  - *models, satisfiability, validity, entailment, etc.*
  - *equivalence rules (e.g., De Morgan)*
  - *inference rules (e.g., resolution)*

What you should know

- *Normal forms (e.g., CNF)*
- *Structure of a theorem prover*
  - *proof trees, knowledge bases*
- *SAT problem*
  - *its search graph(s)*
  - *reductions (e.g., 3-coloring to SAT)*

CSPs

Constraint satisfaction



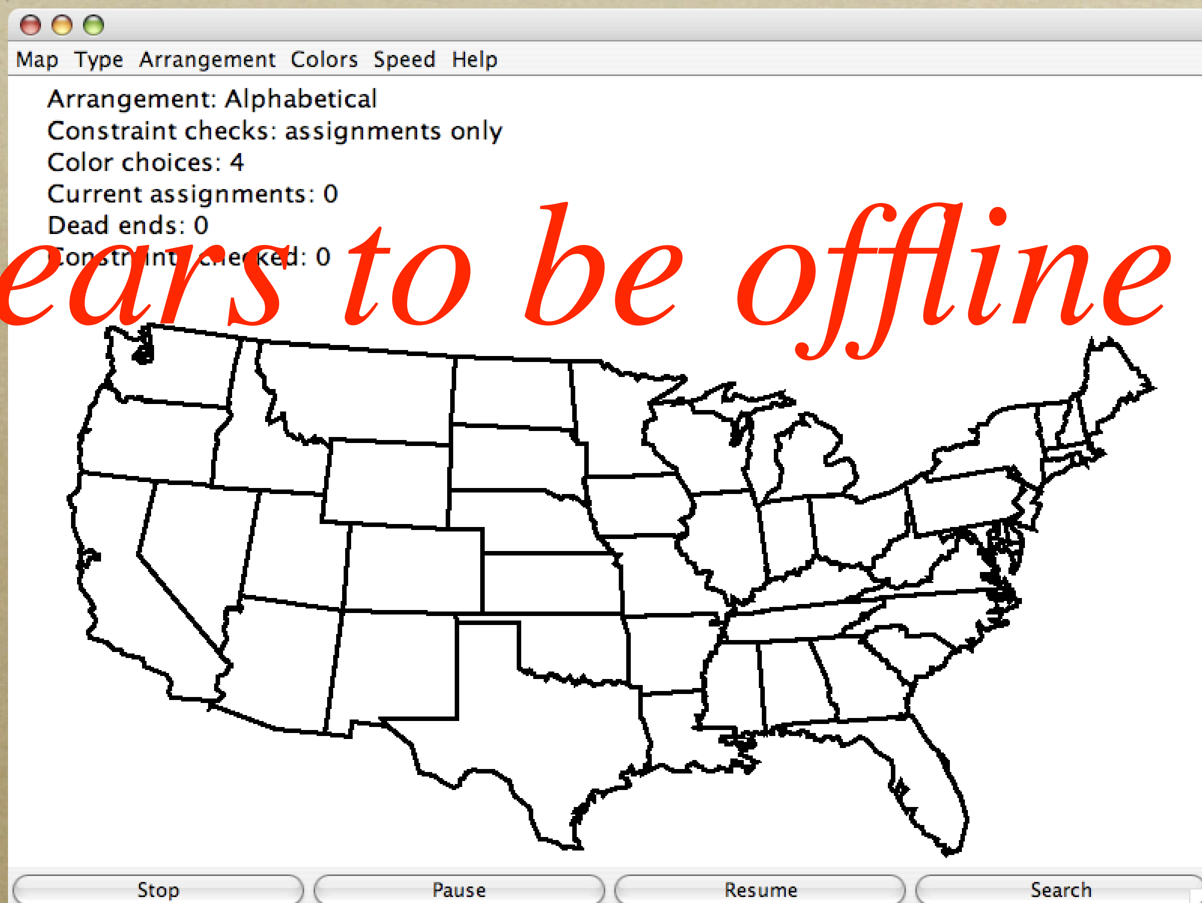
- *Recall 3-coloring*
- *Turned map into SAT problem (constant factor blowup)*
- *Did we have to do that?*

CSP definition

- *No: represent as CSP instead*
- *CSP = (variables, domains, constraints)*
- *Variable:* a
- *Domain: (R, G, B)*
- *Constraint:* a, b ∈ *(RG, RB, GR, GB, BR, BG)*
- *Constraints usually represented compactly*

Example

Appears to be offline now

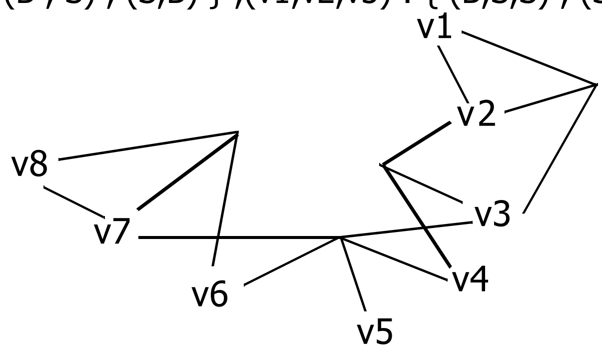


http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-034Artificial-IntelligenceFall2002/Tools/detail/mapresalloc.htm

Other important CSPs

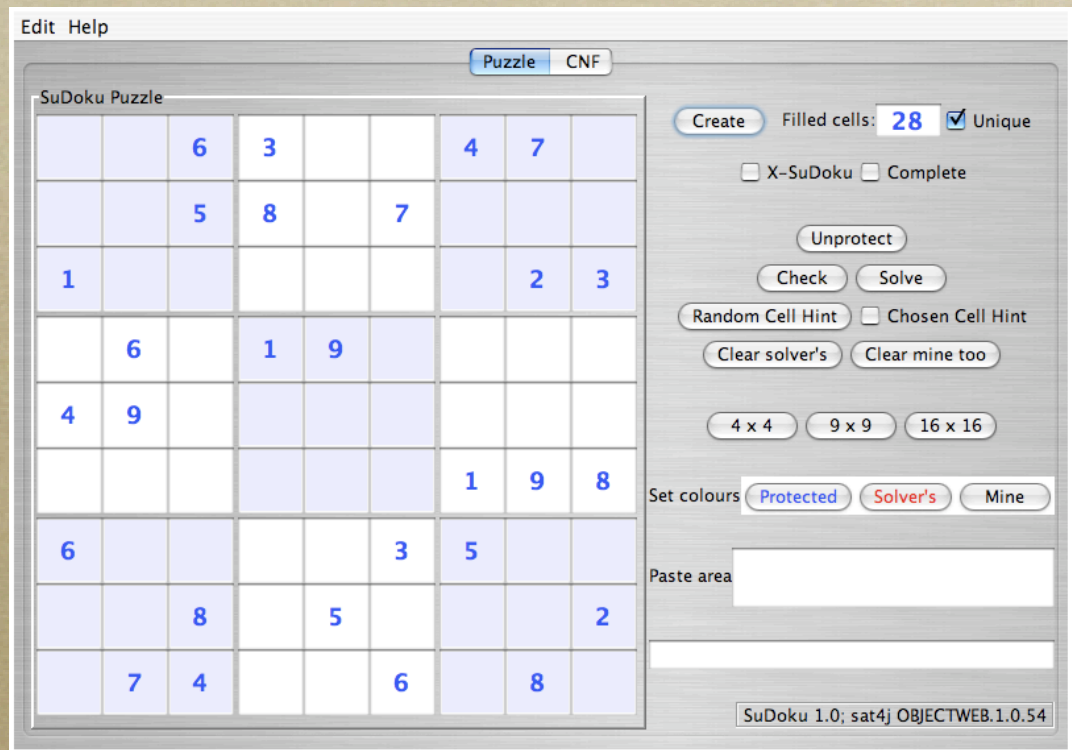
0	0	1	v1		
0	0	1	v2		
0	0	1	v3		
1	1	2	v4		
v8	v7	v6	v5		

V = { v1 , v2 , v3 , v4 , v5 , v6 , v7 , v8 }, D = { B (bomb) , S (space) }
C = { (v1,v2) : { (B , S) , (S,B) } ,(v1,v2,v3) : { (B,S,S) , (S,B,S) , (S,S,B)},...}



- *Minesweeper (courtesy Andrew Moore)*

Other important CSPs



- *Sudoku*

  http://www.cs.qub.ac.uk/~I.Spence/SuDoku/SuDoku.html

Other important CSPs

- *Job-shop scheduling*
- *A bunch of jobs*
  - *each job is a sequence of operations*
  - *drill, polish, paint*
- *A bunch of resources*
  - *each operation needs several resources*
- *Is there a schedule of length* $\leq k$*?*

Solving SAT&CSP

SAT & CSP solvers

- *Search algorithms routinely handle SAT or CSP problems with 1,000,000 variables*
- *Such a solver is a subroutine in one of the planning algorithms we'll discuss soon*

Hard instances

- *SAT, CSP are NP-complete! How can we do problems with 1,000,000 variables?!?*
- *NP-complete doesn't mean runtime has to be exponential for all examples*
  - *e.g.,* $(a \vee b) \wedge (c \vee d) \wedge (e \vee f \vee g)$
- *Many practical examples are apparently not all that hard*

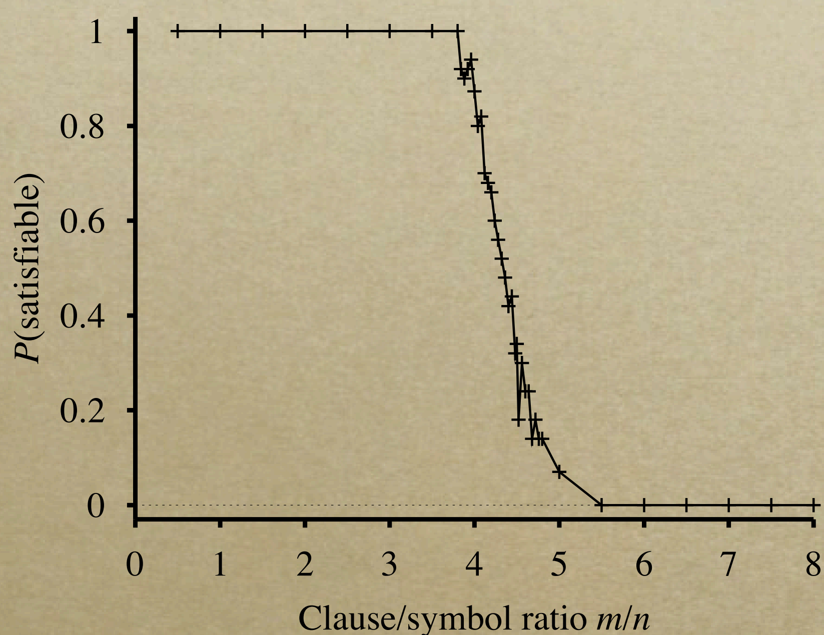
So where are the hard examples?

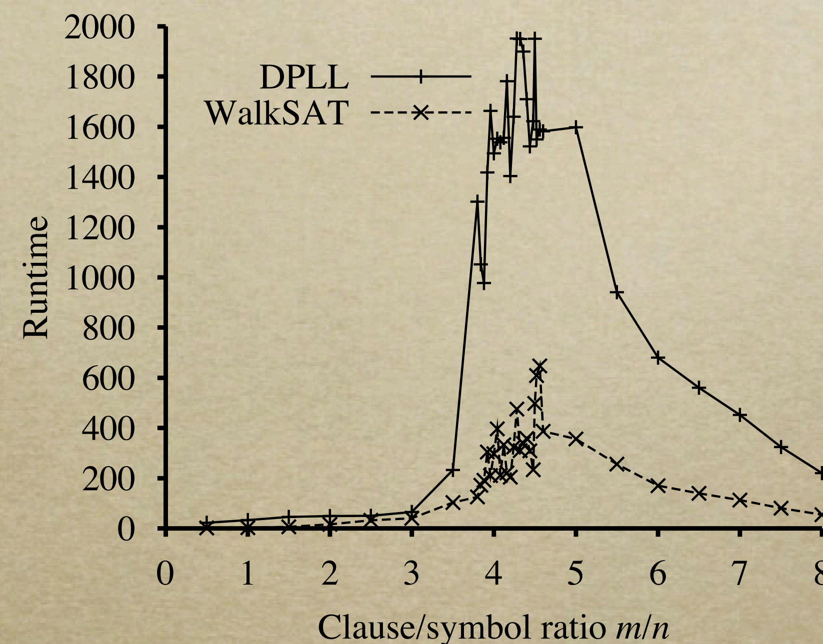
- *Why are some practical examples easy?*
- *They are over- or under-constrained*
  - *under-constrained* $\Rightarrow$ *succeed quickly*
  - *over-constrained* $\Rightarrow$ *fail quickly*
- *Where are the hard examples?*
  - *"critically constrained"*

Aside: random 3CNF formulas

- *It turns out that **random** formulas can be quite hard to solve*
- *Randomly select variables to be in each clause, randomize +ve vs. -ve*
- *If we generate too few clauses, formula is under-constrained*
- *Too many: over-constrained*

Just right





- *Random formulas w/ n=50 vars, m clauses*
- *Clauses have 3 distinct literals, 50% negated*

4.3

- *It turns out $m/n = 4.3$ (and change) is the hard area, for any sufficiently large n*
- *What's special about 4.3? I don't know.*
- *Unfortunately real formulas don't look like random ones, so it's not so easy to check whether they are critically constrained*

SAT & CSP as search problems

- *Search space: models or partial models*
- *Neighbors: change assignment to one variable*
- *Search space may also include changes to constraints / clauses*
    - *add a new constraint / clause*
    - *simplify existing ones*

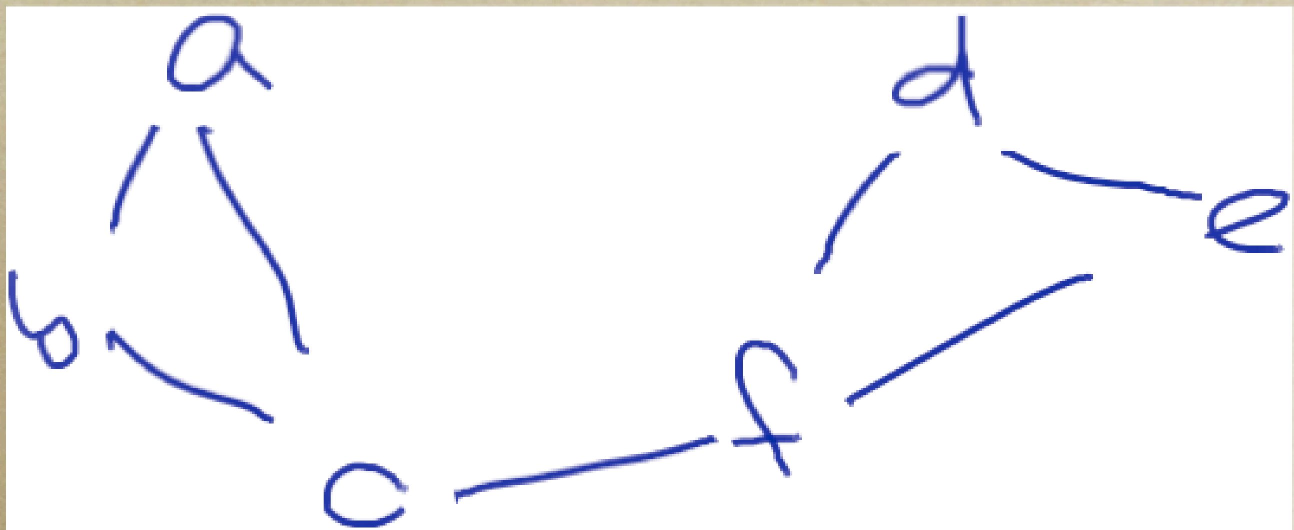
Search in a CSP

- *Let's try DFS using partial assignments*
- *top to bottom, RGB*

DFS looks stupid

- *OK, that wasn't the right way*
- *Blindingly obvious: consistency checking*
- *Don't assign a variable to a value that conflicts with a neighbor*

Search in a CSP

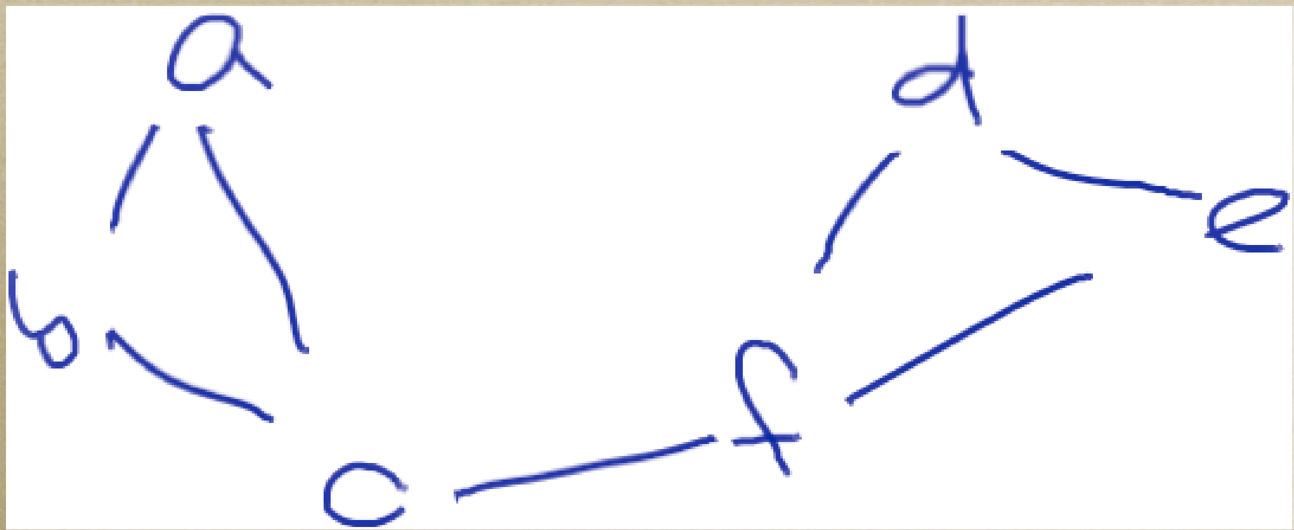


- *DFS with consistency checking*

Well, that's better

- *But it still doesn't notice the problem as soon as it could*
- *Forward checking: delete conflicting values from neighbors' domains*
  - *remember to put them back if we backtrack*
  - *can do this with reference counts*

Search in a CSP



- *Try again with forward checking*

Can we do even better?

- *Constraint propagation*
- *If we notice a variable has just one consistent value, assign it immediately*
- *And delete from neighbors' domains, and recurse*

Search in a CSP

- *Constraint propagation solves it without backtracking!*

Search in a CSP

- *Now let's make it harder*

Constraint learning

- *When we reach a dead end, can spend time analyzing why it is dead*
- *If there's a simple reason, distill it into a constraint and add it to problem*
- *Saves backtracking later*

Intuition

- *Suppose we can learn [subset of previous decisions] $\Rightarrow$ [setting for x]*
- *Didn't know how to set x on this branch, so might not know on future branches*
- *Any time this same subset of decisions appears on a future branch, won't have to search both values of x*

Constraint learning

- *In reaching a dead end model*
  - *we set some variables by propagation*
  - *others we picked arbitrarily (**decision variables**)*
- *Goal: find a set of decision variables that are responsible for failure, guarantee we won't look at their current setting again*
- *Might leave in some non-decision vars*

Finding a new constraint

- *<u>a:R</u>, <u>d:R</u>, <u>g:R</u>, <u>h:G</u>, <u>e:G</u>, f:B, c:G, b:B*
- *Conflict set: f:B, b:B*
- *We set b:B because of <u>a:R</u>, c:G*
- *So, f:B, <u>a:R</u>, c:G is a conflict set too*

Finding a new constraint

- *a:R, d:R, g:R, h:G, e:G, f:B, c:G, b:B*
- *Conflict set: f:B, a:R, c:G*
- *Setting c:G was from a:R, f:B*
- *And f:B was from d:R, e:G*
- *So, a:R, d:R, e:G is an impossible setting*

Search in a CSP

- *Rule out a:R, d:R, e:G*

Finding a new constraint

- *In general: start from the variables of a violated constraint*
- *Pick a non-decision variable, replace it with the variables that caused it to be set*
- *Terminate at some point; get a conflict set*
- *Add a new constraint forbidding current setting of conflict set*

When should we stop?

- *Process variables in reverse chronological order*
- *Eventually, will hit a decision variable x*
- *Could skip x, continue with next variable*
- *But literature recommends stopping at x*

Why is this a good idea?

- *Next backtrack will unset x*
- *Learned clause will have x as its only unsatisfied literal*
- *Will immediately set x via a unit resolution*

Basic CSP or SAT search

```
global problem
function search(model)
    model ← propagate(model)
    if is_solution(model) then return T
    if is_failed(model) then
        learn_constraints(model)
        return F
    var ← pick_var(model)
    vals ← sort_vals(var, model)
    for val in vals
        if search(model / var: val) then return T
```

Choices

Main choices

- *Fancier propagate()*
- *Ordering heuristics*
- *Deleting learned constraints*

Fancier propagate()

- *Pure literal rule*
  - *If setting variable x to value Y doesn't reduce range of any group of vars*
  - *Then go ahead and set x*
  - *E.g., if all neighbors already can't be blue but I can, set me blue*

Fancier propagate()

- *In general, could put any inference rule in propagate()—usually search-free, though*
- *But must be fast, so we will always have to miss some inferences*
- *E.g, Sudoku requires no search, but most propagate() implementations won’t solve it*

Variable ordering

- *Most constrained variable first*
  - *natural generalization of propagation*
  - *tends to find inconsistencies quickly*
  - *cheap to do, often a big win*

Variable ordering

- *Activity rules*
- *Each time a literal seems important, increment its score; decay all scores at a constant rate over time*
- *"Important" literals are*
  - *ones in learned constraints*
  - *ones in conflict sets*

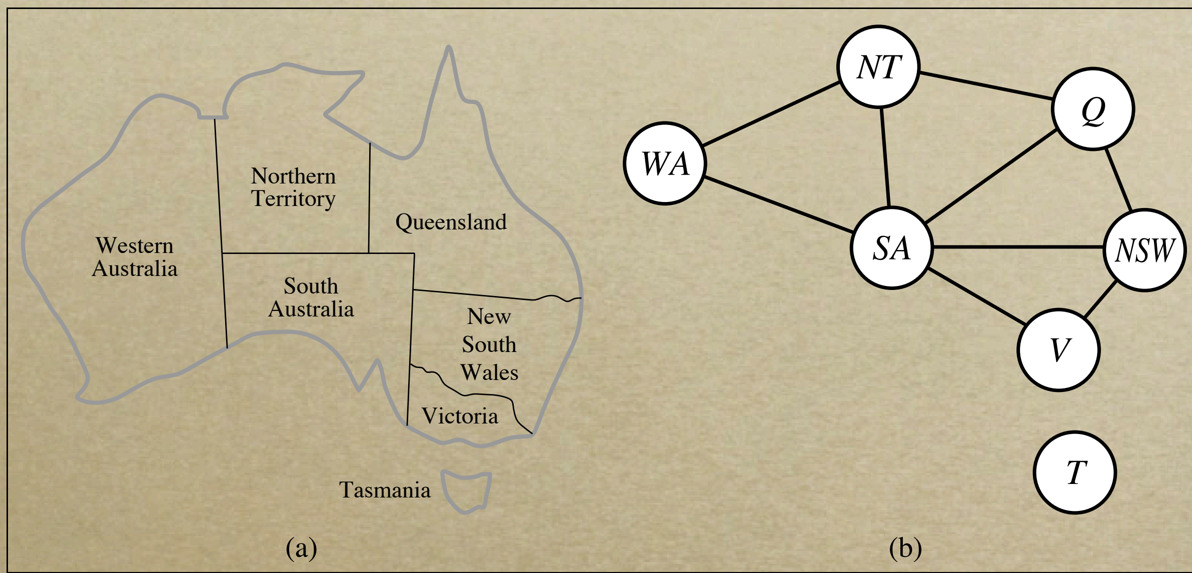
Value ordering

- *Least-constraining value first*
  - *Natural generalization of pure literal*
  - *Give ourselves more flexibility later on*
  - *Delay decisions*
- *Less important, but sometimes helpful*

Deleting learned constraints

- *Learned constraints make problem bigger*
- *So, if they fail to reduce backtracking, we want to get rid of them*
- *Increment constraint's activity level when we use it, decay all activities over time*
- *Delete low-activity constraints*

Example from book



- *Be able to simulate variants of basic search*

SAT Solvers

DPLL

- *Basic search from above is called DPLL when used for CNF-SAT*
- *DPLL stands for Davis, Putnam, Logemann, and Loveland*
- *Modern implementations: Chaff, MINISAT*

DPLL

```
global problem
function search(model)
    model ← propagate(model)
    if is_solution(model) then return T
    if is_failed(model) then
        learn_constraints(model)
        return F
    var ← pick_var(model)
    vals ← sort_vals(var, model)
    for val in vals
        if search(model / var: val) then return T
```

propagate()

- *Constraint propagation becomes unit resolution:*

  If a clause w/ one remaining variable is unsatisfied, set variable to satisfy clause

- *In $(a \vee b \vee \neg c)$:*
  - *model (a: F, b: F) leaves ($\neg c$), set c: F*
  - *model (a: F, c: T) leaves (b), set b: T*
  - *model (a: F, c: F) leaves (T), do nothing*

Other deduction rules

- *Pure literal rule becomes*

  If a literal appears with only one sign in all remaining unsatisfied clauses, set it based on that sign

- *In $(a \lor b) \land (a \lor \neg b)$, sets a: T*
- *RN recommends it*
- *But Chaff paper says it is too slow*

pick_var()

- *Can't use most-constrained-variable heuristic from above*
- *This seems like a real pity*
- *Could imagine allowing clauses like*

  exactly-one-of(a, b, c, d)

  at-most-k-of(3, a, b, c, d)
- *Not sure why it isn't implemented more often*

pick_var()

- *One possibility: MOMS (maximum occurrence in minimum-sized clauses)*
- *Want to satisfy lots of clauses immediately*
- *Failing that, want lots of length-1 clauses*
- *Find smallest clause (say, 3 vars)*
- *Pick a variable which occurs maximally often in size-3 clauses*

MOMS discussion

- *Chaff authors say: MOMS doesn't choose good variables on non-random problems*
- *Recommend activity heuristics instead*
- *Chaff also prefers literals in most recently added clause*

Clause learning

- *New-clause learning rule is an example of resolution-based theorem proving*
- *Uses conflict cause to focus resolution*

Clause learning

- *Conflict clause has all unsatisfied literals*
  - $(a \lor b \lor \neg c)$, *model* $(a{:}F, b{:}F, c{:}T, d{:}F)$
- *Say c is most recent non-decision variable*
  - *from clause* $(b \lor c \lor d)$
  - *b and d must be in conflict too*

Clause learning

- *So, resolving these two clauses yields another conflict clause*
  - *in this case* $(a \lor b \lor d)$
- *Keep doing resolutions for all implied variables, in reverse chronological order*

Other tricks

WalkSAT

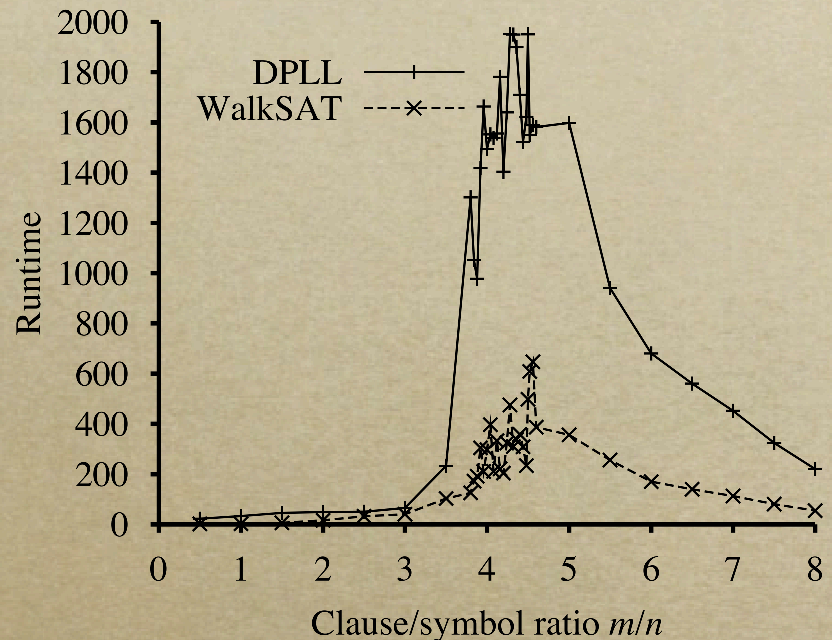
- *Very simple randomized search algorithm*
- *State space: complete models*
- *No formula changes (except perhaps initial simplification)*

WalkSAT

function WALKSAT($clauses, p, max\_flips$) **returns** a satisfying model or *failure*
inputs: $clauses$, a set of clauses in propositional logic
p, the probability of choosing to do a "random walk" move, typically around 0.5
max_flips, number of flips allowed before giving up

$model \leftarrow$ a random assignment of $true/false$ to the symbols in $clauses$
for $i = 1$ **to** max_flips **do**
if $model$ satisfies $clauses$ **then return** $model$
$clause \leftarrow$ a randomly selected clause from $clauses$ that is false in $model$
with probability p flip the value in $model$ of a randomly selected symbol from $clause$
else flip whichever symbol in $clause$ maximizes the number of satisfied clauses
return $failure$

Discussion



- *Pros: easy to implement, very fast on satisfiable formulas*
- *Cons: can't ever prove unsatisfiable*

Randomness

- *Both WalkSAT and DPLL are random*
- *Result is a significant variance in solution times for same formula (Chaff authors report seconds vs. days)*

We can be very lucky or unlucky

The GWYDYR CASTLE in the Doldrums, while sharks hang around her, awaiting the next offering of galley slops

Doldrums: One Of Murphy's Yarns

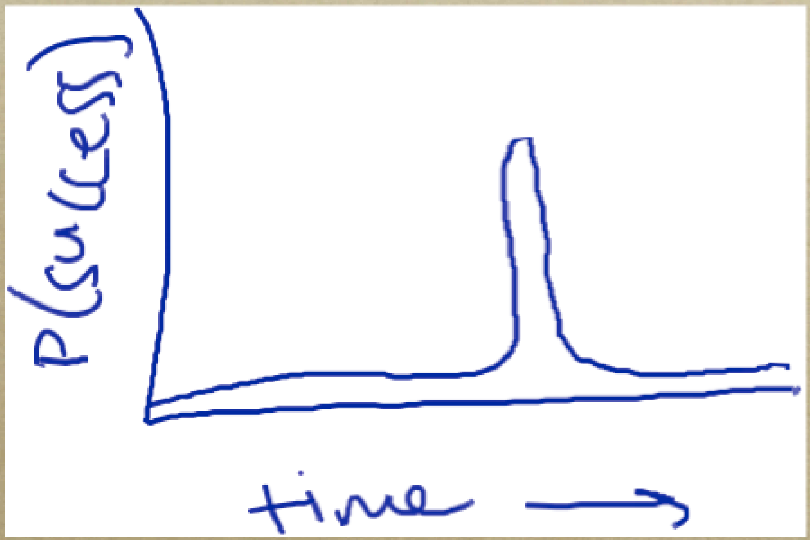
http://oldpoetry.com/opoem/56157 **Cicely Fox Smith**

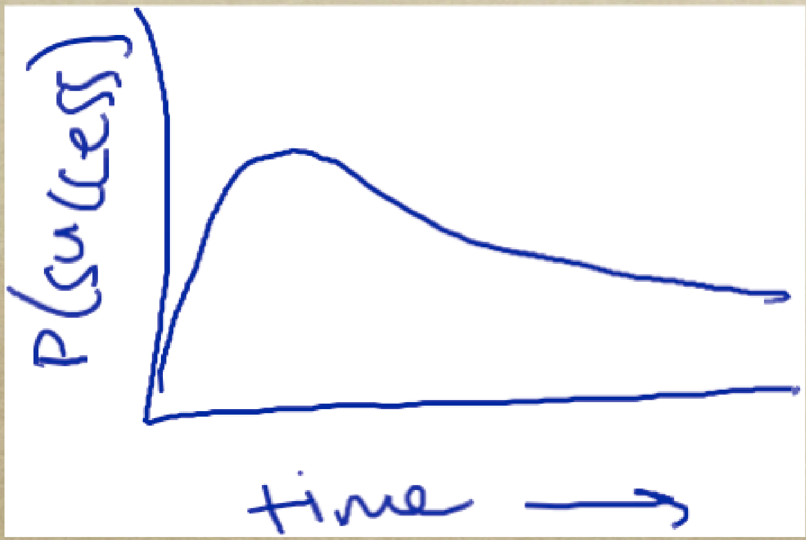
“I heard onst of a barque,” said Murphy.
“Becalmed, that couldn’t get a breath,
Till all the crowd was sick with scurvy
An’ the skipper drunk himself to death.”

Simple idea

- *Try multiple random seeds*
  - *influences order of expanding neighbors (when ordering heuristics are tied)*
  - *influences starting point in WalkSAT*
- *Interleave computation (or iterative lengthening)*
- *When does this work?*

Randomization cont'd





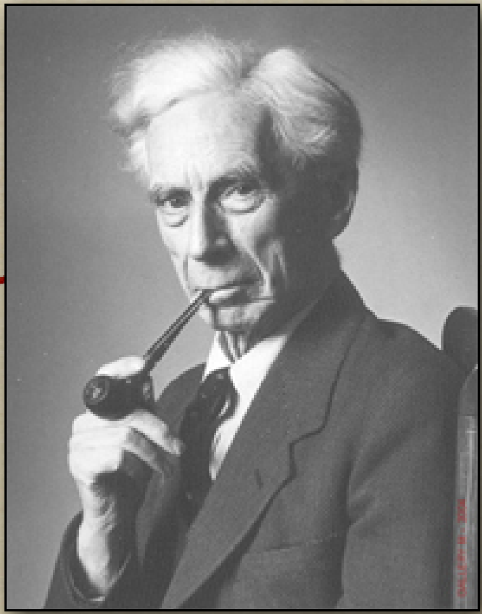
- *Randomization works well if search times are sometimes short but have heavy tail*

Randomness and clause learning

- *For DPLL-style algorithms, if clause learning was active, random restarts don't totally lose effort from previous tries*

First-order logic

First-order logic



Bertrand Russell
1872-1970

- *So far we've been using opaque vars like **rains** or **happy(John)***
- *Limits us to statements like "it's raining" or "if John is happy then Mary is happy"*
- *Can't say "all men are mortal" or "if John is happy then someone else is happy too"*

Predicates and objects

- *Interpret happy(John) or likes(Joe, pizza) as a **predicate** applied to some **objects***
- *Object = an object in the world*
- *Predicate = boolean-valued function of objects*
- *Zero-argument predicate plays same role that Boolean variable did before*

Distinguished predicates

- *We will assume three distinguished predicates with fixed meanings:*
  - *True, False*
  - *Equal(x, y)*
- *We will also write* $(x = y)$ *and* $(x \neq y)$
- *Equality satisfies usual axioms*

Functions

- ***Functions*** *map zero or more objects to another object*
  - *e.g., professor(15-780), last-common-ancestor(John, Mary)*
- *Zero-argument function is the same as an object—John v. John()*

The **nil** object

- *Functions are untyped: must have a value for **any** set of arguments*
- *Typically add a **nil** object to use as value when other answers don't make sense*

Definitions

- ***Term*** *= expression referring to an object*
  - *John*
  - *left-leg-of(father-of(president-of(USA)))*
- ***Atom*** *= predicate applied to objects*
  - *happy(John)*
  - *raining*
  - *at(robot, Wean-5409, 11AM-Wed)*

Definitions

- ***Literal*** *= possibly-negated atom*
  - *happy(John), ¬happy(John)*
- ***Sentence*** *= literals joined by connectives like* $\wedge \vee \neg \Rightarrow$
  - *raining*
  - *done(slides(780))* $\Rightarrow$ *happy(professor)*

Models

- *Meaning of sentence: model $\mapsto$ {T, F}*
- *Models are now much more complicated*
  - *List of objects*
  - *Table of function values for each function mentioned in formula*
  - *Table of predicate values for each predicate mentioned in formula*

Models

- *Function table includes referent for each object*
- *Predicate table includes value of each boolean-valued variable*

For example

KB describing example

- $alive(cat)$
- $ear\text{-}of(cat) = ear$
- $in(cat, box) \wedge in(ear, box)$
- $\neg in(box, cat) \wedge \neg in(cat, nil) \ldots$
- $ear\text{-}of(box) = ear\text{-}of(ear) = ear\text{-}of(nil) = nil$
- $cat \neq box \wedge cat \neq ear \wedge cat \neq nil \ldots$