15-780: Graduate AI

Lecture 4. Logic, SAT, and CSPs

Geoff Gordon (this lecture)
Ziv Bar-Joseph
TAs Michael Benisch, Yang Gu

Admin

- *15-780 and 16-731 are the same course, cross listed in CS and Robotics*
- *If your email address is not yourID@cs.cmu.edu, please contact the TAs to make sure you're on the mailing list*

Last episode, on *Grad AI*

What you should know

- *IDA* definition*
- *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)*
- *SAT problem*
  - *its search graph*
  - *reductions (e.g., 3-coloring to SAT)*
- *Structure of a theorem prover*
  - *proof trees, knowledge bases*
  - *compare/contrast search graph w/ SAT*

Direction of reduction

- *If A reduces to B then*
  - *if we can solve B, we can solve A*
  - *so B must be at least as hard as A*
- *E.g., could take an easy problem and reduce it to a hard one*

Not-so-useful reduction

- *Path planning reduces to SAT*
- *Variables: is edge e in path?*
- *Constraints:*
  - *exactly 1 path-edge touches start*
  - *exactly 1 path-edge touches goal*
  - *either 0 or 2 touch each other node*

Reduction to 3SAT

- *We saw that search problems can be reduced to SAT*
  - *is CNF formula satisfiable?*
- *Can reduce even further, to 3SAT*
  - *is 3CNF formula satisfiable?*
- *Useful if reducing SAT/3SAT to another problem (to show other problem hard)*

Reduction to 3SAT

- *Must get rid of long clauses*
- *E.g.,* $(a \vee \neg b \vee c \vee d \vee e \vee \neg f)$
- *Replace with*

$$(a \vee \neg b \vee x) \wedge (\neg x \vee c \vee y) \wedge$$
$$(\neg y \vee d \vee z) \wedge (\neg z \vee e \vee \neg f)$$

A note on reductions

- *May be many reductions from problem A to problem B*
- *May have **wildly** different properties*
    - *e.g., search on transformed instance may take seconds vs. days*
- *Example will show up when we get to Planning topic*

Citation

- *"Using Inaccurate Models in Reinforcement Learning." Pieter Abbeel, Morgan Quigley, Andrew Y. Ng*

  http://www.icml2006.org/icml_documents/camera-ready/001_Using_Inaccurate_Mod.pdf

Comparing representations

- *All search algorithms presented so far use a discrete representation of the world*
- *If world is continuous, they divide it into blocks*
- *This works great for some domains, terribly for others*

Real vs. discrete

- *Discrete works well, e.g., for deciding which way to go around an obstacle*
- *But it would be really bad to discretize to the level required for precision position servoing*

Position servoing

- *E.g., if state is $(x(t) - x_{tgt}(t))$, discretization will allow bang-bang control (or, slightly better, control with k fixed levels of effort)*
- *If state is $(x(t), x_{tgt}(t))$, axis-parallel splits won't even allow accurate bang-bang control without very fine discretization*

Smooth control

- *Couldn't implement a smooth controller like PID without a **really** fine grid*
- *Probably so fine as to make it infeasible to search for control recommended by logical formula*

Theorem provers

Soundness and completeness

- *An inference procedure is **sound** if it can only conclude things entailed by KB*
  - *common sense; we already required it*
- *A set of rules is **complete** if it can conclude everything entailed by KB*
- *Modus ponens by itself is **incomplete***

Completeness of resolution

- *Inference procedure: put KB in CNF, add $\neg B$ to KB, apply resolution until*
  - *we get a False as a consequence (and conclude $KB \models B$),* ***or***
  - *we run out of inferences (and conclude $KB \not\models B$)*
- *This inference procedure is complete*

Variations

- *Horn clause inference (faster)*
- *Ways of handling uncertainty (slower)*
- *CSPs (sometimes more convenient)*
- *Quantifiers / first-order logic (say more about this later)*

Horn clauses

- *Horn clause:* $(a \wedge b \wedge c \Rightarrow d)$
- *Equivalently,* $(\neg a \vee \neg b \vee \neg c \vee d)$
- *Disjunction of literals,* ***at most one*** *of which is positive*
- *Positive literal =* ***head****, rest =* ***body***

Use of Horn clauses

- *People find it easy to write Horn clauses (listing out conditions under which we can conclude head)*

  $$happy(John) \wedge happy(Mary) \Rightarrow happy(Sue)$$

- *No negative literals in above formula; again, easier to think about*

Why are Horn clauses important

- *Inference in a KB of propositional Horn clauses is linear*
- *Forward chaining or backward chaining (see RN reading, or discussion of unit resolution below)*

Handling uncertainty

- *Fuzzy logic / certainty factors*
  - *simple, but don't scale*
- *Nonmonotonic logic*
  - *also doesn't scale*
- *Probabilities*
  - *may or may not scale—more in Part II*
  - *Dempster-Shafer theory*

Certainty factors

- *Instead of just T/F, a model assigns a certainty factor in [0, 1] to each proposition*
- *And, KB assigns a certainty to each rule*
- *Interpret as "degree of belief"*

Certainty factors

- *Logical connectives are interpreted as arithmetic operations, e.g.,* $\wedge$ *as min,* $\vee$ *as max, and* $\neg$ *as (1-x)*
- *E.g., if KB has (*$\neg$*rains* $\vee$ *pours) @ 0.8 and rains @ 0.7, conclude*

  $max(0.3, pours) \geq 0.8$

  $pours \geq 0.8$

Problems w/ certainty factors

- *Hard to separate a large KB into mostly-independent chunks that interact only through a well-defined interface*
- *Certainty factors are not probabilities (i.e., do not obey Bayes' Rule)*

Nonmonotonic logic

- *Suppose we believe all birds can fly*
- *Might add a set of sentences to KB*

$$bird(Polly) \Rightarrow flies(Polly)$$

$$bird(Tweety) \Rightarrow flies(Tweety)$$

$$bird(Tux) \Rightarrow flies(Tux)$$

$$bird(John) \Rightarrow flies(John)$$

…

Nonmonotonic logic

- *Fails if there are penguins in the KB*
- *Fix: instead, add*

  $bird(Polly) \land \neg ab(Polly) \Rightarrow flies(Polly)$

  $bird(Tux) \land \neg ab(Tux) \Rightarrow flies(Tux)$

  …
- *ab(Tux) is an "abnormality predicate"*
- *Need separate $ab_i(x)$ for each type of rule*

Nonmonotonic logic

- *Now set as few abnormality predicates as possible*
- *Can prove flies(Polly) or flies(Tux) with no ab(x) assumptions*
- *If we assert ¬flies(Tux), must now assume ab(Tux) to maintain consistency*
- *Can't prove flies(Tux) any more, but can still prove flies(Polly)*

Nonmonotonic logic

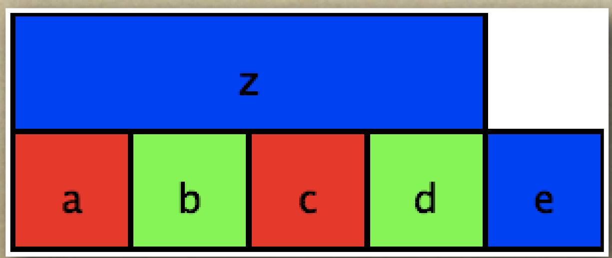
- *Works well as long as we don't have to choose between big sets of abnormalities*
  - *is it better to have 3 flightless birds or 5 professors that don't wear jackets with elbow-patches?*
  - *even worse with nested abnormalities: birds fly, but penguins don't, but superhero penguins do, but …*

Dempster-Shafer

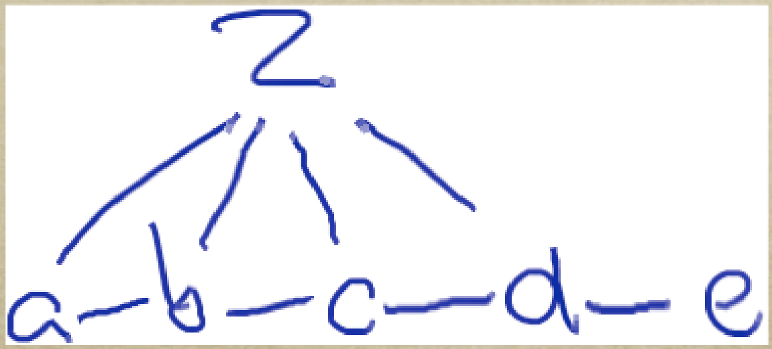
- *Allows additional worst-case uncertainty beyond probabilities*
- *Maintains lower, upper bounds on probabilities; assumes world is adversarial within those bounds*
- *Like probabilities, inference is guaranteed correct*
- *May be overly conservative*

CSPs

Constraint satisfaction



=

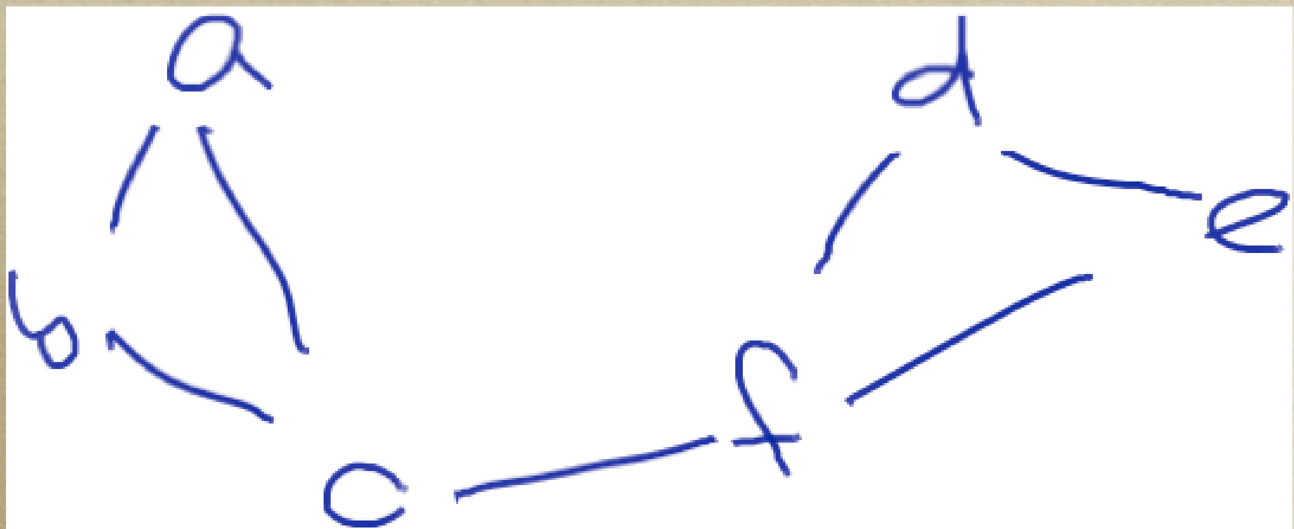


- *Recall 3-coloring*
- *Turned map into graph (same size) then into SAT problem (constant factor blowup)*
- *Did we have to do that?*

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*

Search

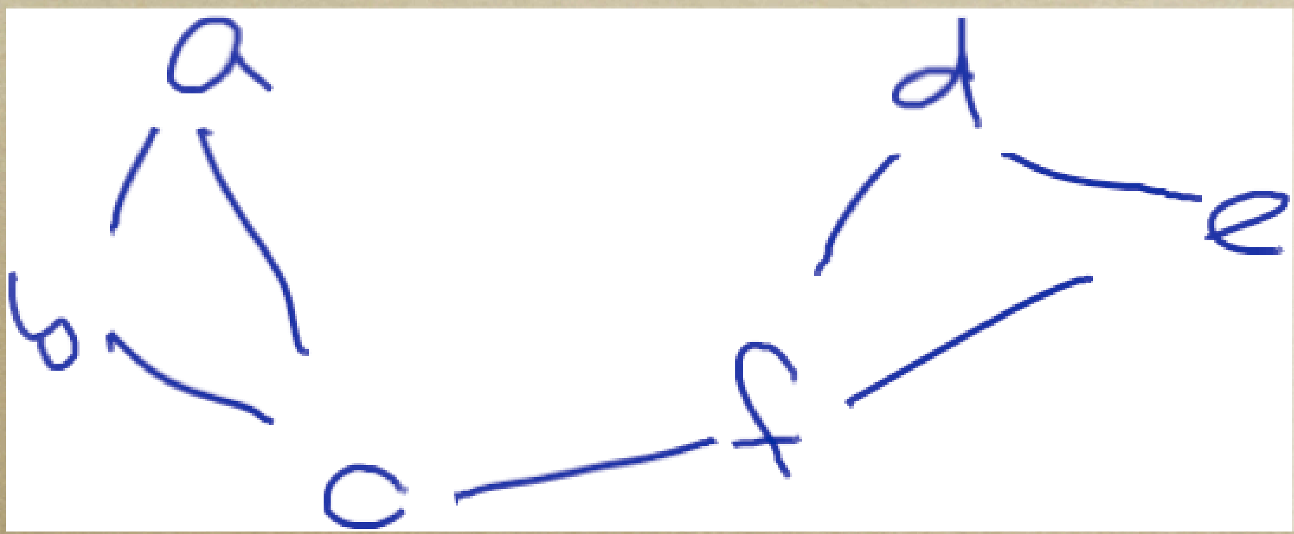


- *Obviously a search problem*
- *Let's try DFS—top to bottom, RGB*

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

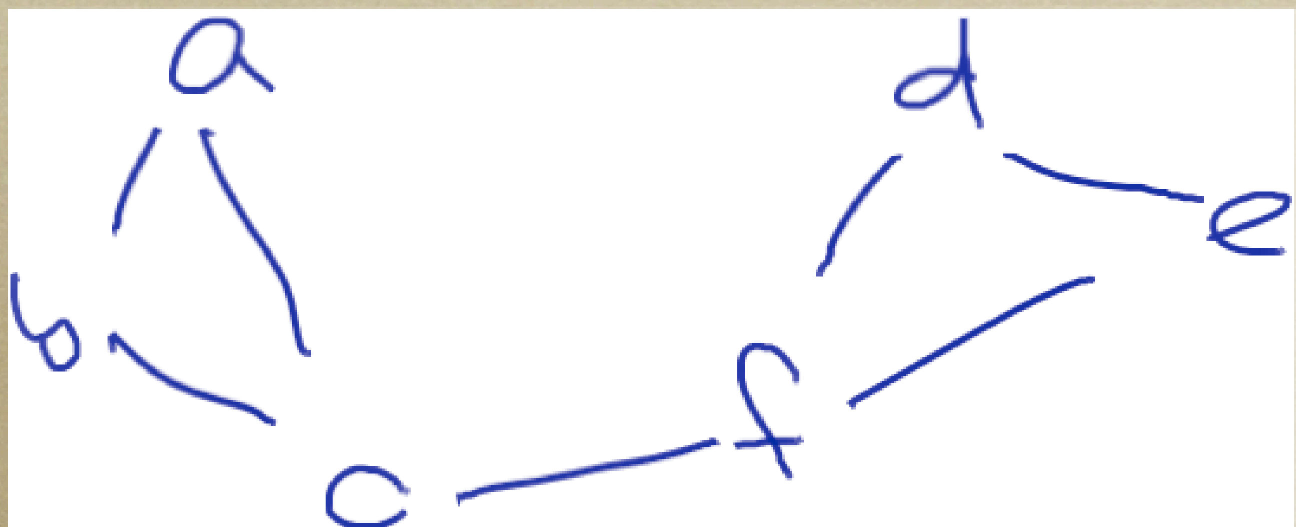


- *DFS with consistency checking*

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

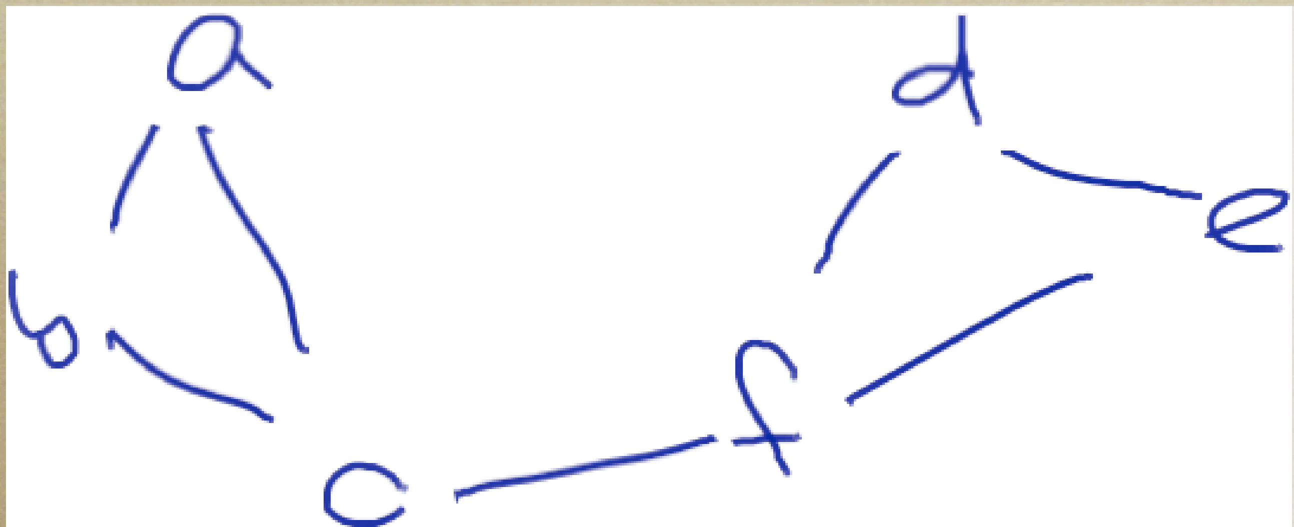


- *Try again with forward checking*

Can we do even better?

- *Constraint propagation*
- *E.g., once we notice a variable has just one consistent value, delete that value from its neighbors' domains*
- *Even fancier: arc consistency, k-consistency (see RN)*

Search



- *Constraint propagation solves it without backtracking!*

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 CSP*
- *Saves backtracking later*
- *But useless constraints slow us down*
- *See RN Ch 5 for more detail*

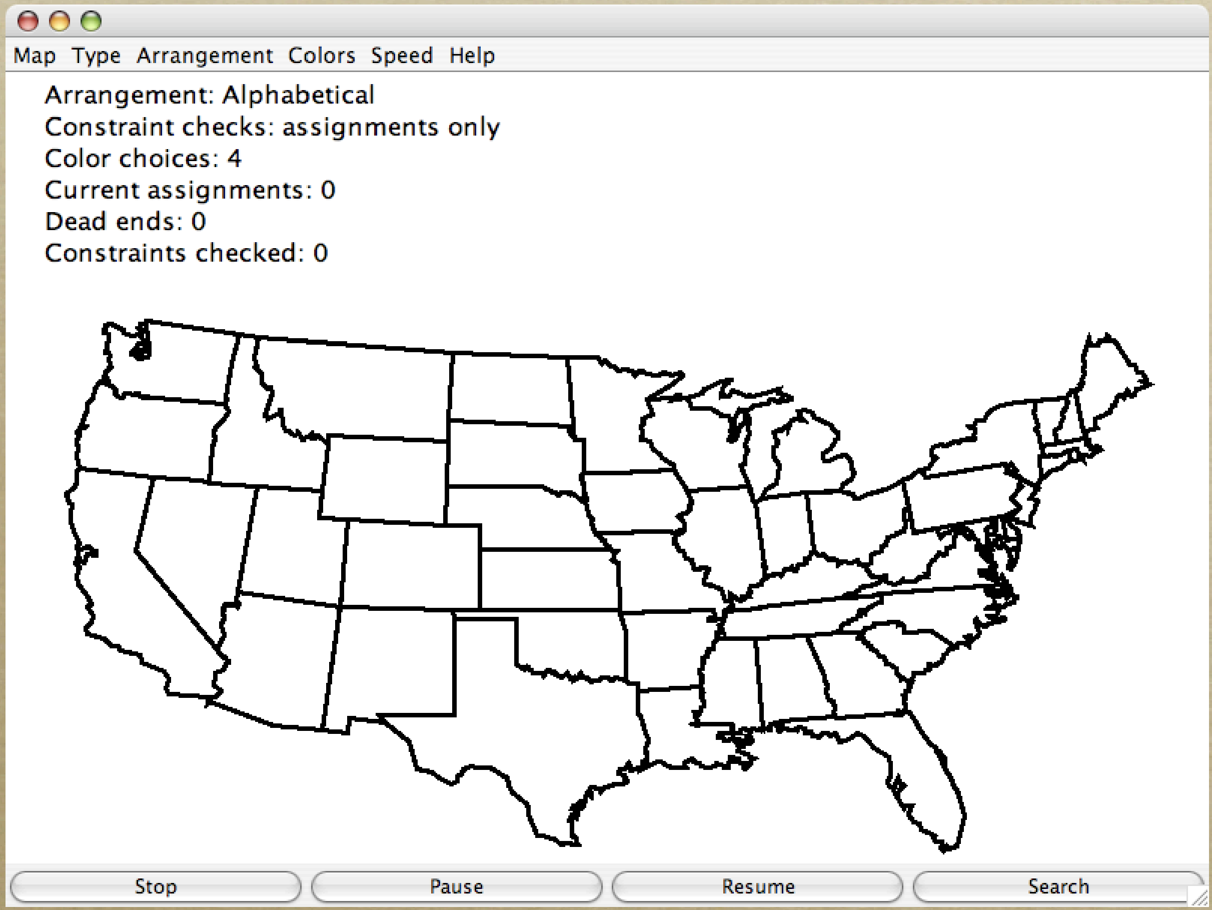
Orderings

- *Big choices: which variable to try next? What value to assign to it?*
- *So far, fixed order—can do better*
- *Most constrained variable first*
  - *natural generalization of propagation*
  - *tends to find inconsistencies quickly*
  - *cheap to do, often a big win*

Orderings

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

Example



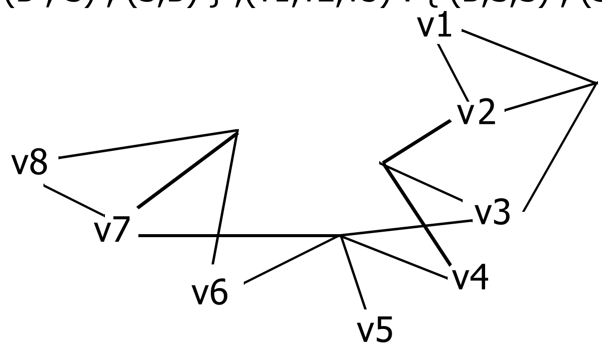
http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-034Artificial-IntelligenceFall2002/Tools/detail/mapresalloc.htm

Other important CSPs

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

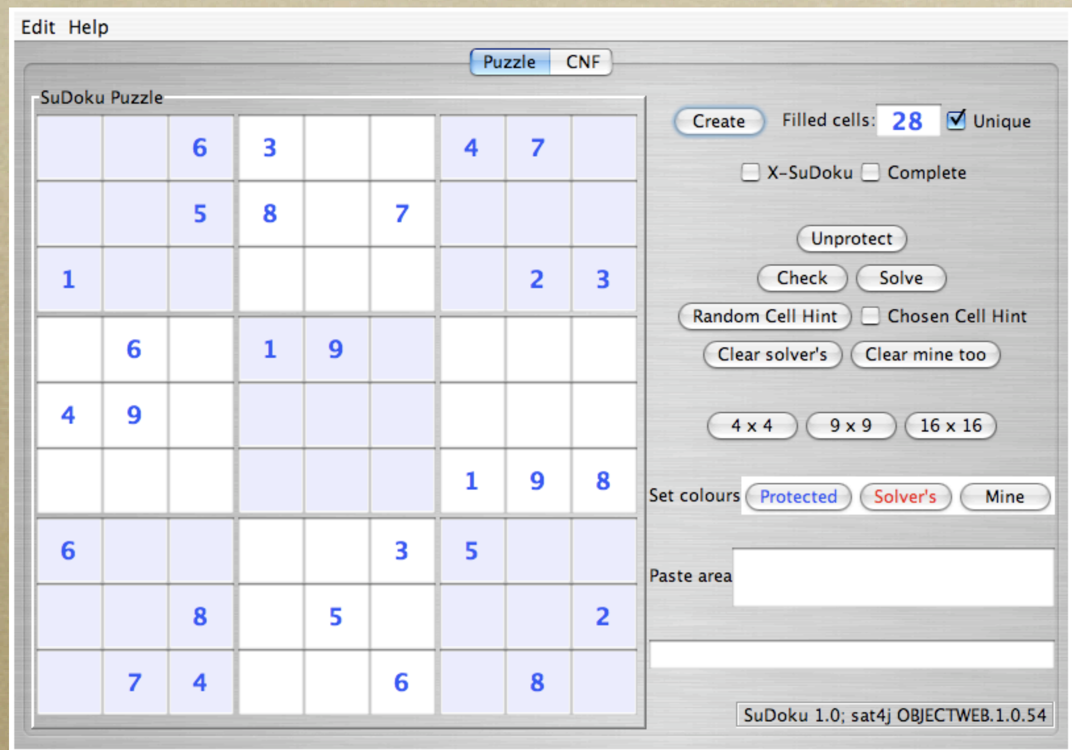
V = { v1 , v2 , v3 , v4 , v5 , v6 , v7 , v8 }, D = { B (bomb) , S (space) }

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*](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* $\leq k$?

SAT Solvers

SAT solvers

- *There are SAT solvers which routinely handle problems with 1,000,000 variables*
- *Such a SAT solver is a subroutine in one of the planning algorithms we'll discuss soon*
- *So, here's how to write one*

Hard instances

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

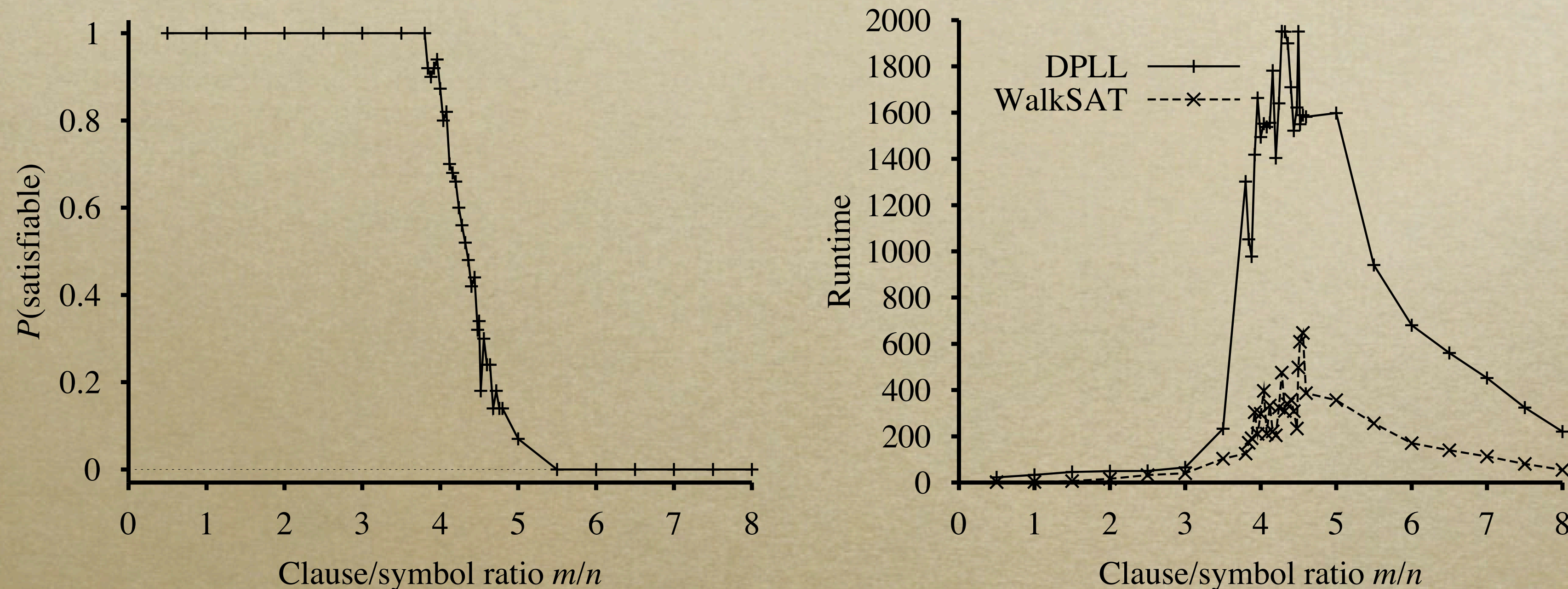
So where are the hard examples?

- *Why are practical examples easy?*
- *They are over- or under-constrained*
    - *under-constrained* ⇒ *succeed quickly*
    - *over-constrained* ⇒ *fail quickly*
- *Where are the hard examples?*

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 vars, 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 hardness*

SAT solvers

- *Many different search strategies*
- *Will mention two: WalkSAT (briefly) and DPLL / Chaff*
- *Both assume formula input in CNF*
- *Could do a simplification search before handing to algorithm*
- *Chaff paper claims this may not help much*

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 ← 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 ← 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*

DPLL

- *WalkSAT used complete assignments as its search space*
- *DPLL uses (partial assignment, formula)*
- *DPLL stands for Davis, Putnam, Logemann, and Loveland*
- *Refers to a family of algorithms; we will discuss the Chaff implementation*

DPLL

```
DPLL(formula, model)
    model = deduce(formula, model)
    if (all-assigned(formula, model))
        return evaluate(formula, model)
    x = choose-variable(formula, model)
    if (DPLL(formula, model / x: T))
        return T
    else
        return DPLL(formula, model / x: F)
```

Simple subroutines

- *all-assigned: checks whether all clauses have all variables assigned*
- *evaluate: evaluates a fully-assigned formula*

Clause learning

- *An optional feature of DPLL-style algorithms is **clause learning***
- *When we backtrack, we can analyze reasons for failure and try to add a clause that will cause us to notice the same type of failure sooner on the next branch*
- *More below*

deduce()

- *Does any inference it can do quickly to set more variables without searching*
- *Has to be fast, so will miss some inferences*
- *E.g, a Sudoku puzzle requires no search, but most deduce() implementations won’t solve it*

deduce()

- *Chaff uses only the following rule:*

  Unit resolution

  If a clause contains just one unknown variable, set it to satisfy the clause

- *In* $(a \vee b \vee \neg c)$*:*
  - *with (a: F, b: F), will set c: F*
  - *with (a: F, c: T), will set b: T*

Other deduction rules

- *RN recommends*

  Pure literal rule

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

- *In $(a \vee b) \wedge (a \vee \neg b)$, sets a: T*
- *Chaff paper says this rule is too slow*

Choose-variable

- *Can't use most-constrained variable heuristic from CSP*
- *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 this isn't implemented more often*

Choosing a branch variable

- *Want to satisfy lots of clauses immediately*
- *If we can't do that, want lots of length-1 clauses*
- *MOMS heuristic*
  - *find smallest clause (say 3 variables)*
  - *pick a variable that occurs maximally often in size-3 clauses*

MOMS discussion

- *Chaff authors say: MOMS doesn't choose good variables on non-random problems*
- *Recommend heuristics based on "activity" of a variable*
- *Each time a literal seems important, increment its score; decay all scores at a constant rate over time*

Important literals

- *"Important" literals are*
  - *ones in added clauses*
  - *ones in conflict clauses*
- *Chaff increments on conflict, restricts choice to literals in most recently added clause*

Clause learning

- *Try to add clauses which will let us detect failure sooner on other branches*
- *These clauses are redundant*
- *So if they don't help us prune, they slow us down*
- *Chaff paper recommends counting how often a clause is involved in a conflict*

Clause learning

- *Skipped conflict learning in CSPs; this is essentially the same idea*
- *Learned clauses are derived by resolution from clauses already in formula*
- *When we fail, there is a **conflict clause** which has all literals unsatisfied*
- *Use conflict cause to focus resolution*

Clause learning

- *Conflict clause has all unsatisfied literals*
  - *(a ∨ b ∨ ¬c) in model (a: F, b: F, c: T)*
- *Some assignments in model came from unit resolution—call these* ***implied vars***
  - *say c is most recent, from clause (b ∨ c)*
  - *all other literals in this clause must be in conflict too*

Clause learning

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

When should we stop?

- *As we back up through assignments, eventually we will hit a* ***decision variable*** *(i.e., one that wasn't assigned)*
- *Call it x*
- *Could skip x, continue with next assigned variable*
- *But Chaff 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*

Intuition

- *[Subset of previous decisions] ⇒ [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*

Randomness

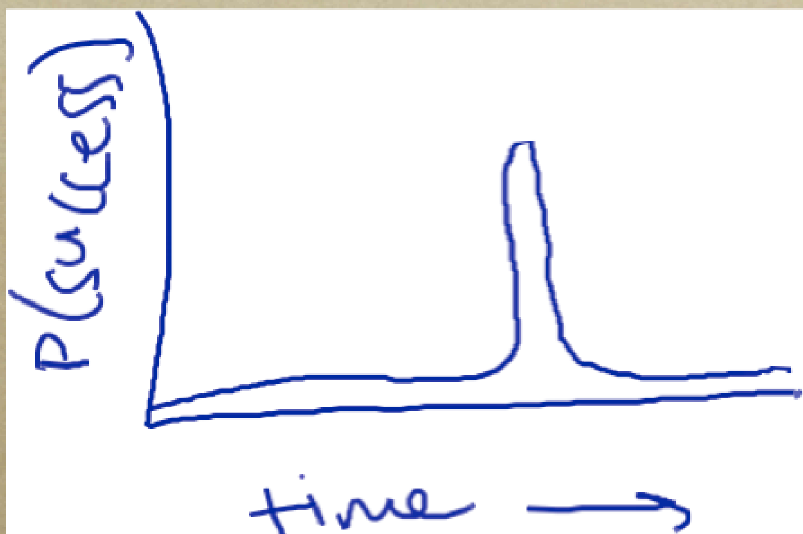
- *Both WalkSAT and Chaff are random*
  - *more randomness in WalkSAT*
- *Result is a significant variance in solution times for same formula (Chaff authors report seconds vs. days)*

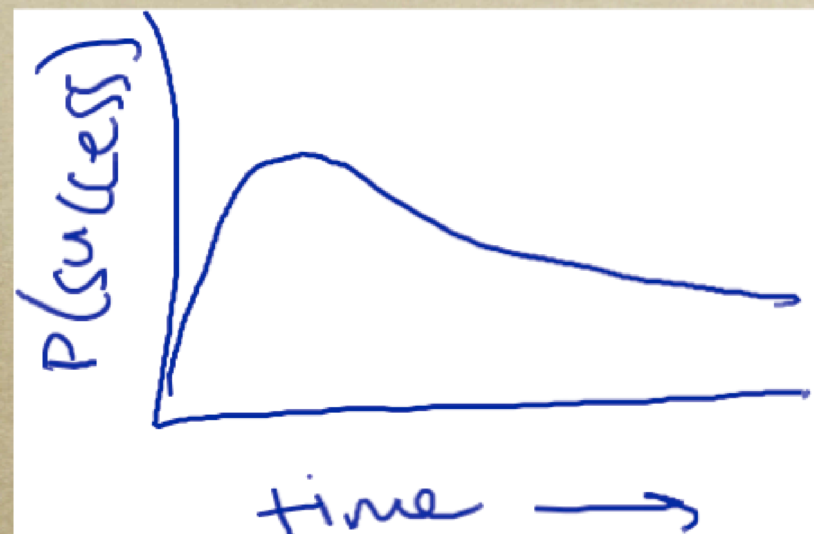
We can be very lucky or unlucky

Simple idea

- *Try different random seeds for breaking ties in variable ordering heuristic*
- *Let each seed run longer than the last*
- *Seems to help a lot*

Randomization cont'd





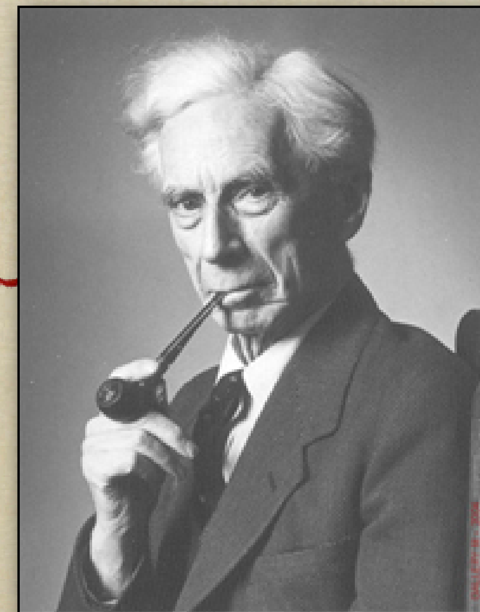
- *Randomization works well if search times are sometimes short but have heavy tail*

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*
- *predicate(object) plays same role that 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)*
- *Predicates and functions have fixed arity*
- *Zero-argument function is equivalent to an object variable*

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*

Model

- *Models are now much more complicated*
  - *List of objects*
  - *Table of function values for each function mentioned in formula*
    - *includes referent for each variable*
  - *Table of predicate values for each predicate mentioned in formula*

For example

KB describing example

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

Aside: typed variables

- *KB illustrates need for data types*
- *Don't want to have to specify ear-of(box) or ¬in(cat, nil)*
- *Could design a type system and allow only formulas which obey type rules (e.g., argument of happy() is of type animate)*

Model of example

- *Objects: C, B, E, N*
- *Assignments:*
  - *cat: C, box: B, ear: E, nil: N*
  - *ear-of(C): E, ear-of(B): N, ear-of(E): N, ear-of(N): N*
- *Predicate values:*
  - *in(C, B), ¬in(C, C), ¬in(C, N), …*

Failed model

- *Objects: C, E, N*
- *Fails because there's no way to satisfy inequality constraints with only 3 objects*

Another possible model

- *Objects: C, B, E, N, X*
- *Extra object X could have arbitrary properties since it's not mentioned in KB*
- *E.g., X could be its own ear*