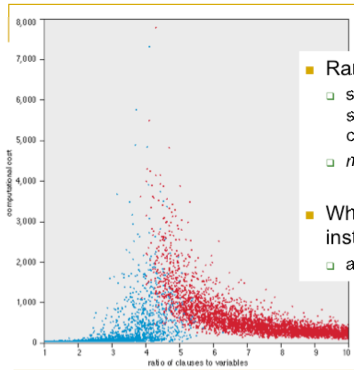
Satisfiability Solvers

Part 2: Stochastic Solvers

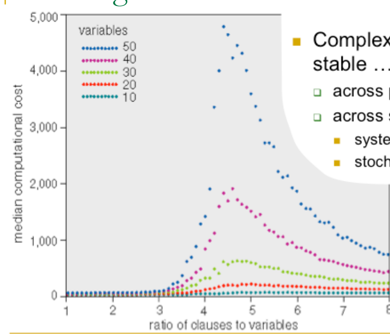
Structured vs. Random Problems

- So far, we've been dealing with SAT problems that encode other problems
- Most not as hard as # of variables & clauses suggests
  - Small crossword grid + medium-sized dictionary may turn into a big formula … but still a small puzzle at some level
  - Unit propagation does a lot of work for you
  - Clause learning picks up on the structure of the encoding
- But some **random** SAT problems really are hard!
  - zChaff's tricks don't work so well here



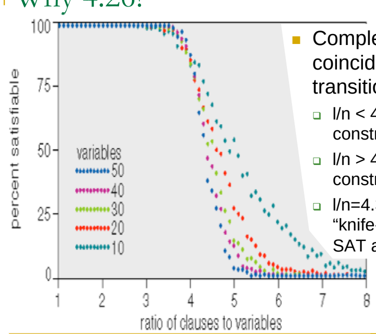
- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - *n* variables, *l* clauses
- Which are the hard instances?
  - around *l*/*n* = 4.26

The magic ratio 4.26



- Complexity peak is very stable …
  - across problem sizes
  - across solver types
    - systematic (last lecture)
    - stochastic (this lecture)

Why 4.26?



- 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

That's called a "phase transition"

- Problems < 32˚F are like ice; > 32˚F are like water
- Similar "phase transitions" for other NP-hard problems
  - job shop scheduling
  - traveling salesperson (instances from TSPlib)
  - exam timetables (instances from Edinburgh)
  - Boolean circuit synthesis
  - Latin squares (alias sports scheduling)
- Hot research topic:
  - predict hardness of a given instance, & use hardness to control search strategy (Horvitz, Kautz, Ruan 2001-3)

Methods in today's lecture …

- Can handle "big" random SAT problems
  - Can go up about 10x bigger than systematic solvers! ☺
  - Rather smaller than structured problems (espec. if ratio ≈ 4.26)
- Also handle big structured SAT problems
  - But lose here to best systematic solvers ☹
- Try hard to find a good solution
  - Very useful for approximating **MAX-SAT**
  - Not intended to find **all** solutions
  - Not intended to show that there are **no** solutions (UNSAT)

GSAT vs. DP on Hard Random Instances

today (not today's <u>best</u> algorithm) | yesterday (not yesterday's <u>best</u> algorithm)

form.	GSAT			Davis-Putnam		
vars	m.flips	retries	time	choices	depth	time
50	250	6	0.5 *sec*	77	11	1 *sec*
70	350	11	1 *sec*	42	15	15 *sec*
100	500	42	6 *sec*	10^3	19	3 *min*
120	600	82	14 *sec*	10^5	22	18 *min*
140	700	53	14 *sec*	10^6	27	5 *hrs*
150	1500	100	45 *sec*	—	—	—
200	2000	248	3 *min*	—	—	—
300	6000	232	12 *min*	—	—	—
500	10000	996	2 *hrs*	10^{30}	> 100	10^{19} *yrs*

Notes: Define "Hard" later
Only "satisfiable" formulae
(else GSAT does not terminate)

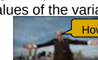
slide thanks to Russ Greiner and Dekang Lin

Local search for SAT

- Make a guess (smart or dumb) about values of the variables
- 

- Try flipping a variable to make things better
  - **Algorithms differ on <u>which</u> variable to flip**

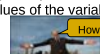
Local search for SAT

- Make a guess (smart or dumb) about values of the variables
- 

- Try flipping a variable that makes things better

- Flip a randomly chosen variable?
  - No, blundering around blindly takes exponential time
  - Ought to pick a variable that *improves* … what?
    - Increase the # of satisfied clauses as much as possible
    - Break ties randomly
    - Note: Flipping a var will repair some clauses and break others
  - This is the "GSAT" (Greedy SAT) algorithm (almost)

Local search for SAT

- Make a guess (smart or dumb) about values of the variables
- 

- Try flipping a variable that makes things better

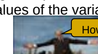
- Flip most-improving variable. Guaranteed to work?

A v C
A v B
~C v ~B
~B v ~A

- What if first guess is A=1, B=1, C=1?
  - 2 clauses satisfied
  - Flip A to 0 ➔ 3 clauses satisfied
  - Flip B to 0 ➔ all 4 clauses satisfied (pick this!)
  - Flip C to 0 ➔ 3 clauses satisfied

example thanks to Monaldo Mastrolilli and Luca Maria Gambardella

Local search for SAT

- Make a guess (smart or dumb) about values of the variables
- 

- Try flipping a variable that makes things better

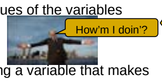
- Flip most-improving variable. Guaranteed to work?

A v C
A v B
~C v ~B
~B v ~A

- But what if first guess is A=0, B=1, C=1?
  - 3 clauses satisfied
  - Flip A to 1 ➔ 3 clauses satisfied
  - Flip B to 0 ➔ 3 clauses satisfied
  - Flip C to 0 ➔ 3 clauses satisfied
  - Pick one anyway … (picking A wins on <u>next</u> step)

example thanks to Monaldo Mastrolilli and Luca Maria Gambardella

Local search for SAT

- Make a guess (smart or dumb) about values of the variables
- 

- Try flipping a variable that makes things better

Repeat until satisfied

- Flip most-improving variable. Guaranteed to work?
- Yes for 2-SAT: probability → 1 within $O(n^2)$ steps
- No in general: can & usually does get locally stuck
- Therefore, GSAT just restarts periodically
  - New random initial guess

Believe it or not, GSAT outperformed everything else when it was invented in 1992.

example thanks to Monaldo Mastrolilli and Luca Maria Gambardella

Discrete vs. Continous Optimization

- In MAX-SAT, we're maximizing a real-valued function of n boolean variables
  - SAT is the special case where the function is –infinity or 0 everywhere
  - This is <u>discrete</u> optimization since the variables can't change gradually
- You may already know a little about <u>continuous</u> optimization
  - From your calculus class: maximize a function by requiring all its partial derivatives = 0
  - But what if you can't solve those simultaneous equations?
  - Well, the partial derivatives tell you which direction to change each variable if you want to increase the function

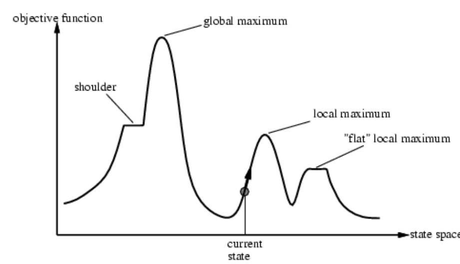
Gradient Ascent (or Gradient Descent)

nice smooth and convex cost function

nasty non-differentiable cost function with local maxima

- GSAT is a greedy local optimization algorithm: Like a discrete version of gradient ascent.
- Could we make a continuous version of the SAT problem?
  - What would happen if we tried to solve it by gradient ascent?
- Note: There are alternatives to gradient descent …
  - conjugate gradient, variable metric, simulated annealing, etc.
  - Go take an optimization course: 550.{361,661,662}.
  - Or just download some software!

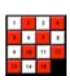
Problems with Hill Climbing



slide thanks to Russ Greiner and Dekang Lin

Problems with Hill Climbing

- Local Optima (foothills): No neighbor is better, but not at global optimum.
  - (Maze: may have to move AWAY from goal to find (best) solution)
- Plateaus: All neighbors look the same.
  - (15-puzzle: perhaps no action will change # of tiles out of place)



- Ridge: going up only in a narrow direction.
  - Suppose no change going South, or going East, but big win going SE: have to flip 2 vars **<u>at once</u>**
- Ignorance of the peak: Am I done?

slide thanks to Russ Greiner and Dekang Lin

How to escape local optima?

- Restart every so often
- Don't be so greedy (more randomness)
  - Walk algorithms: With some probability, flip a randomly chosen variable instead of a best-improving variable
  - WalkSAT algorithms: Confine selection to variables in a single, randomly chosen unsatisfied clause (also faster!)
  - Simulated annealing (general technique): Probability of flipping is related to how much it helps/hurts
- Force the algorithm to move away from current solution
  - Tabu algs: Refuse to flip back a var flipped in past t steps
  - Novelty algs for WalkSAT: With some probability, refuse to flip back the most recently flipped var in a clause
  - Dynamic local search algorithms: Gradually increase weight of unsatisfied clauses

How to escape local optima?

Lots of algorithms have been tried …

Simulated annealing, genetic or evolutionary algorithms, GSAT, GWSAT, GSAT/Tabu, HSAT, HWSAT, WalkSAT/SKC, WalkSAT/Tabu, Novelty, Novelty+, R-Novelty(+), Adaptive Novelty(+), …

How do we generalize to MAX-SAT?

Adaptive Novelty+ (current winner)

- with probability 1% (the "+" part)
  - Choose randomly among all variables that appear in at least one **unsatisfied** clause
- else be greedier (other 99%)
  - Randomly choose an **unsatisfied** clause C
    - Flipping any variable in C will at least fix C (the "novelty" part)
  - Choose the most-improving variable in C … except …
  - with probability p, ignore most recently flipped variable in C
- The "adaptive" part:
  - If we improved # of satisfied clauses, decrease p slightly
  - If we haven't gotten an improvement for a while, increase p
  - If we've been searching too long, restart the whole algorithm

WalkSAT/SKC

(first good local search algorithm for SAT, very influential)

- Choose an **unsatisfied** clause C
  - Flipping any variable in C will at least fix C
- Compute a "break score" for each var in C
  - Flipping it would break how many other clauses?
- If C has any vars with break score 0
  - Pick at random among the vars with break score 0
- else with probability p
  - Pick at random among the vars with min break score
- else
  - Pick at random among all vars in C

Simulated Annealing

(popular general local search technique – often very effective, rather slow)

- Pick a variable at random

  If flipping it improves assignment: do it.

  Else flip anyway with probability $p = e^{-\Delta/T}$ where
  - Δ = damage to score
    - What is p for Δ=0? For large Δ?
  - T = "temperature"
    - What is p as T tends to infinity?
    - Higher T = more random, non-greedy exploration
    - As we run, decrease T from high temperature to near 0.

slide thanks to Russ Greiner and Dekang Lin (modified)

Simulated Annealing and Markov Chains

- [discuss connection between optimization and sampling]

Evolutionary algorithms

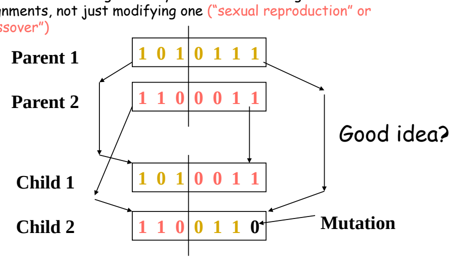
(another popular general technique)

- Many local searches at once
  - Consider 20 random initial assignments
  - Each one gets to try flipping 3 different variables ("reproduction with mutation")
  - Now we have 60 assignments
  - Keep the best 20 ("natural selection"), and continue

Sexual reproduction

(another popular general technique – at least for evolutionary algorithms)

Derive each new assignment by somehow combining **two** old assignments, not just modifying one ("sexual reproduction" or "crossover")



slide thanks to Russ Greiner and Dekang Lin (modified)