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

Structured vs. Random Problems

- 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
  - \( \frac{l}{n} < 4.3 \) problems under-constrained and SAT
  - \( \frac{l}{n} > 4.3 \) problems over-constrained and UNSAT
  - \( \frac{l}{n} = 4.3 \), 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

<table>
<thead>
<tr>
<th>form.</th>
<th>vars</th>
<th>m. flips</th>
<th>GSAT</th>
<th>time</th>
<th>David-Putnam</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>250</td>
<td>0.5 sec</td>
<td>77</td>
<td>11 sec</td>
<td>10^3 22 18 min</td>
</tr>
<tr>
<td>70</td>
<td>350</td>
<td>1 sec</td>
<td>42</td>
<td>15 sec</td>
<td>10^3 19 5 min</td>
</tr>
<tr>
<td>100</td>
<td>500</td>
<td>2 sec</td>
<td>10^4</td>
<td>19 sec</td>
<td>10^3 27 5 hrs</td>
</tr>
<tr>
<td>120</td>
<td>700</td>
<td>3 sec</td>
<td>10^4</td>
<td>18 sec</td>
<td>10^3 — —</td>
</tr>
<tr>
<td>140</td>
<td>100</td>
<td>4 sec</td>
<td>—</td>
<td>—</td>
<td>10^3 — —</td>
</tr>
<tr>
<td>150</td>
<td>2000</td>
<td>248 sec</td>
<td>—</td>
<td>—</td>
<td>10^3 — —</td>
</tr>
<tr>
<td>300</td>
<td>6000</td>
<td>12 min</td>
<td>—</td>
<td>—</td>
<td>10^3 — —</td>
</tr>
<tr>
<td>500</td>
<td>10000</td>
<td>2 hrs</td>
<td>10^3</td>
<td>&gt; 100</td>
<td>10^19 yrs</td>
</tr>
</tbody>
</table>

Notes: Define “Hard” later
- Only “satisfiable” formulas
  - GSAT does not terminate

Slide thanks to Russ Greiner and Dekan Leitsch

Local search for SAT

- Make a guess (smart or dumb) about values of the variables
  - How’m I doin’?
  - Try flipping a variable to make things better
- Algorithms differ on which variable to flip

Flip most-improving variable. Guaranteed to work?
- 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

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 next step)

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?
  - 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.

Repeat until satisfied.

Discrete vs. Continuous 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 discrete optimization since the variables can’t change gradually.
- You may already know a little about continuous 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.

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 at once.
- Ignorance of the peak: Am I done?

Gradient Ascent (or Gradient Descent)

- 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: 550.1661, 662.
  - Or just download some software!

Problems with Hill Climbing

- 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: Confining 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 alg: Refuse to flip back a var flipped in past t steps.
  - Novelty alg 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/SC, WalkSAT/Tabu, Novelty, Novelty+, R-Novelty(+), Adaptive Novelty(+), …

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/SC
(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  
    - \( \Delta \) = damage to score  
    - What is p for \( \Delta = 0 \)? For large \( \Delta \)?  
    - \( T \) = “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.

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”)

Parent 1

Parent 2

Child 1

Child 2

Mutation

Good idea?

slide thanks to Russ Greiner and Dekang Lin (unmodified)