CMU 15-252 **Spring 2017**

HOMEWORK 8 — DUE APRIL 21 IN CLASS

In this homework you will attempt to give a "physics proof" of a lower bound on the critical clause density for random 3SAT. Specifically, you will try to identify a number Δ_u (somewhere between 2 and 3?) such that a certain very simple algorithm has a good chance of finding a satisfying assignment for a random 3SAT formula with clause density $\Delta < \Delta_u$. This homework will require you to write code, produce some plots, and write some short explanations. You should turn all of this in on paper (including code printouts!). The TAs will briefly eyeball the code to make sure it looks plausible, but will mainly grade the plots, explanations, and quality of your findings. You may use any programming language, and any package/software for making diagrams.

1. Write code that takes as input a parameter $n \in \mathbb{N}$ and a parameter $\Delta \in \mathbb{R}^+$. It should first set $m = \lfloor \Delta n \rfloor$. Then it should choose and return (the representation of) a random 3CNF formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses. Each clause (independently) should consist of 3 randomly chosen variables, each possibly negated (randomly). I don't care whether you choose the 3 variables "without replacement" or "with replacement"; the former is usually the "official model", but the latter (which may lead to a clause containing a variable more than once) is slightly easier to implement and is perfectly fine for this problem.

   If you use "with replacement", your code should immediately "simplify" all clauses that contain repeated variables: Whenever x_i and $\neg x_i$ both appear in the clause, you should delete the whole clause, because any assignment will automatically satisfy this clause anyway. (This means you might end up with fewer than m clauses at the end, but that's okay.) Similarly, if x_i (or $\neg x_i$) appears multiple times in a clause, you should just include it once. (This means some clauses may have fewer than m literals; that's also okay, and it's going to happen in the course of the upcoming algorithm anyway.)

   By the way, one standard "format" for 3CNFs is a list of clause-representations, each of which is a set of (at most) 3 integers in the range $[-n, n]$, where $+i$ indicates that x_i appears in the clause and $-i$ indicates that $\neg x_i$ appears in the clause (0 is unused).

2. Write code for this algorithm, "attempting" to find a satisfying assignment for ϕ:

   for $T = 0 \ldots n-1$ // counts how many variables have been set so far
   - if the current 3CNF has no clauses, halt and output "satisfiable"
   - if the current 3CNF has at least one clause of size 1, then do a "forced step":
     - pick a random size-1 clause from among the set of all size-1 clauses
     - this size-1 clause "forces" a truth assignment for its variable[1]
     - set that forced truth value, and "simplify" the 3CNF accordingly[2]
   - else if the current 3CNF has no clauses of size 1, then do a "free step":
     - pick a random variable (that hasn't been set yet)
     - set that variable to a random truth value (0/1), and "simplify" the 3CNF accordingly
   - if the 3CNF now contains an empty clause, halt and output "fail"

   output "satisfiable"

[1] That is, if the clause is (x_i) then it's forcing $x_i = 1$; if the clause is $(\neg x_i)$, it's forcing $x_i = 0$.

[2] That is, go through each clause: do nothing if the clause doesn't contain the newly set variable, remove the clause if the newly set variable satisfies it, and delete the set variable from the clause if it doesn't help to satisfy it.

3. Try out your code for several choices of n (e.g., $n = 100, 200, 400, \ldots$), values of Δ (e.g., 1, 1.2, 1.4, ...), and lots of runs. Collect statistics on how often your algorithm succeeds (outputs "satisfiable"). Produce some plots of "success probability vs. Δ". Write some explanation of your findings. Does there seem to be some value Δ_u with the property that the algorithm is likely to fail when $\Delta > \Delta_u$ and has a good chance of succeeding when $\Delta < \Delta_u$? (For large n.)

4. For up to one extra point, do as much as you can of the following: Define $t = T/n$, so $t \in [0, 1]$. At all times in the algorithm, let $s_3 = s_3(t) = S_3/n$, where S_3 denotes the number of size-3 clauses in the formula. Similarly define s_2 for the number of size-2 clauses. (Initially, $s_3(0) = \Delta$ and $s_2(0) = 0$.) Each run of the algorithm essentially produces functions $s_2, s_3 : [0, 1] \to \mathbb{R}^+$. Make plots of "typical" outcomes for these functions, for various values of Δ. Can you identify what these functions tend towards?

   Clauses of size 1 behave somewhat differently. Whenever a free step of the algorithm generates a size-1 clause, it may actually generate several size-1 clauses. Doing all the subsequent "forced steps" (which themselves may in turn generate more size-1 clauses), before getting back down to zero size-1 clauses, is called doing a "cascade". Try to plot/analyze/describe quantities like the following: When a cascade begins at time t, what is the "average" number of size-1 clauses $\lambda(t)$ that are generated by the first free step? What is the "average" number of size-1 clauses generated in the whole cascade, $\Lambda(t)$? How do these quantities seem to depend on Δ?