CSC 2427 Spring 2008, Assignment #3

Due: May 1 before midnight.

You may consult:

- the two texts that are recommended for this course
- an introductory graph theory text, such as the one by West
- the papers posted on the course web page.

You may not consult any other materials.

You cannot work with any other students on this assignment.

1. **20 pts**

   Consider the following model for a random *constraint satisfaction problem*. There are n variables. Each can take one of three values $\{0, 1, 2\}$. A constraint on k variables consists of a k-tuple that those variables are not allowed to jointly take. For example, a constraint on x_1, x_7, x_{12} might be: "we cannot set $x_1 = 0, x_7 = 0$ and $x_{12} = 2$." A constraint is satisfied if it is not the case that all k variables are assigned those values. For example, that constraint is satisfied by the assignment $x_1 = x_7 = x_{12} = 0$, but not by $x_1 = x_7 = 0, x_{12} = 2$. A satisfying assignment is one that satisfies every constraint.

   The random model chooses cn constraints independently at random. For each constraint, it chooses three distinct variables, where each of the $\binom{n}{3}$ choices is equally likely, and then it chooses the forbidden triple of values, where each of the 3^3 choices is equally likely. (It is possible for the same constraint to be chosen twice, although it is easy to show that this a.s. does not occur.)

   Consider applying the following greedy algorithm to look for a satisfying solution:

   UNIT-CONSTRAINT

   1 If there is a constraint of size 1 then pick, uniformly at random, one of the two assignments to its variable that will satisfy the constraint. Else pick a variable uniformly at random and pick, uniformly at random, one of the three possible values to assign it.

   2 For every constraint C containing the variable that is set: If the variable is given the value that it would get in C's forbidden assignment, then reduce C to a constraint on the other $|C| - 1$ variables. Else remove C.

   Determine a constant c^* such that if $c < c^*$ then UNIT-CONSTRAINT will a.s. (i.e. with probablity $1 - o(1)$) find a satisfying assignment while if $c > c^*$ it will a.s. fail by producing a constraint of size zero.

   You will probably find it helpful to read a paper by Achlioptas that I posted on the course web page.

2. **20 pts** Find a deterministic polytime algorithm to produce the colouring guaranteed by Problem 5(a) on Assignment 2. You may assume that Δ is a sufficiently large *constant*. So your running time can be exponentially high in Δ.

   Do not simply cite Theorem 25.7 from the Molloy-Reed book. Instead, give the algorithm and the proof of correctness.

3. **25 pts** In this problem, you will prove the following:

   You are given a graph G where each vertex v has a list $L(v)$ of colours. For every vertex v and every colour $c \in L(v)$, c also appears in the lists of at most d neighbours of v. For every $\epsilon > 0$, there is some D_ϵ such that if $d > D_\epsilon$ and every $L(v)$ has size at least $(1+\epsilon)d$ then the vertices of G can be properly coloured using colours from their lists.

   Recall that, in class, we saw how to prove this when $1+\epsilon$ is replaced by 8.

   To solve this problem, carry out the following procedure. At each iteration you:

   (1) Activate each uncoloured vertex with probability $1/\ln d$.

   (2) Assign to each activated v, a colour chosen uniformly at random from $L(v)$.

   (3) For each activated v, remove the colour assigned to v from the list of every neighbour of v.

   (4) Uncolour every vertex that receives the same colour as a neighbour.

   Let $N(v,c)$ denote the set of neighbours u of v with $c \in L(u)$. Recursively define two parameters L_i, T_i such that, at the beginning of the ith iteration: (i) every list has size at least L_i and (ii) for every vertex v and colour $c \in L(v)$, $|N(v,c)| \leq T_i$.

   It will be useful if you define these parameters so that L_i/T_i is increasing with i.

   When bounding the change in $|N(v,c)|$ during an iteration, it might be useful to focus on the number of vertices of $N(v,c)$ which are successfully coloured during that iteration.