ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 1 Statistical Physics for Communication and Computer Science
Homework 1 February 25, 2010, INR 113 - 9:15-11:00

This is the first in a sequence of homeworks. You can use any programming language you like. The idea is that over the following weeks we will build up a sequence of routines which we can reuse. We therefore recommend that you comment your code and write it carefully.

The aim of this weeks homework is to write programs which can sample a random bipartite graph and then to test this program for random 3-SAT instances by running a very simple routine called unit clause propagation.

Problem 1 (Configuration Model)**.** Let l be the *variable* node degree and r be the *check* node degree. Pick n variable nodes and $m = nl/r$ check nodes. Each variable node has l *sockets* and each check node has r sockets. Number these sockets in a fixed but arbitrary order from 1 to nl on both sides. Pick a random permutation from the set of permutations on nl letters uniformly at random. Construct a bipartite graph by connecting the variable node socket i to check node socket $\pi(i)$. This is called the *configuration* model.

Your program should take as input the parameters n, m, l and r. It should check that the input is valid and return a bipartite graph according to this configuration model. Think about the data structure. Later on we will run algorithms on such a graph. It will then be necessary to loop over all nodes, refer to edges of each node, be able to address the neighbor of a node via a particular edge and store values associated to nodes and edges.

Problem 2 (Poisson Model)**.** Pick two integers, n and m. As before, there are n variable nodes and m check nodes. Further, let r be the degree of a check node. For each check node pick r variables uniformly at random either with or without repetition and connect this check node to these variable nodes. For each edge store in addition a binary value chosen according to a Bernoulli(1/2) random variable.

This is called the Poisson model since the node degree distribution on the variable nodes converges to a Poisson distribution for large n.

Again, think of the data structure. We will use this model right away to run some simple algorithm on it.

Problem 3 (Unit Clause Propagation for Random 3-SAT Instances)**.** Generate random instances of the Poisson model. Pick $n = 10^5$ and let $r = 3$. Let α be a non-negative real number. It will be somewhere in the range $[0, 5]$. Let $m = \lfloor \alpha n \rfloor$.

For a given α generate many random bipartite graphs according to the Poisson model. Interpret such a bipartite graph as a random instance of a 3-SAT problem. This means, the variables nodes are the Boolean variables and the check nodes represent each a clause involving 3 variables. The binary variable associated to each edge indicates whether in this clause we have the variable itself or its negation.

For each instance you generate, try now to find a satisfying assignment in the following greedy manner. This is called the *unit clause propagation* algorithm.

(i) If there is a check node in the graph of degree one (this corresponds to a *unit*-clause), then choose among such check nodes uniformly at random. Remove it from the graph together with the connected variable node and all edges emanating from this variable node.

(ii) If no such check exists, pick a variable node uniformly at random from the graph and sample a Bernoulli(1/2) random variable, call it X. Remove this variable node from the graph. For each edge emanating from the variable node do the following. If X agrees with the variable associated to this edge then remove not only the edge but the associated check node and all its outgoing edges. If not, then remove only the edge.

Continue the above procedure until there are no variable nodes left. If, at the end of the procedure, there are no check nodes left in the graph (by definition all variable nodes are gone) then we have found a satisfying assignment and we declare success. If not, then the algorithm failed, although the instance itself might very well be satisfiable.

Plot now the probability of success for this algorithm as a function of α. You should observe a threshold behavior. Roughly at what value of α does the probability of success change from close to 1 to close to 0? Hand in this plot.