

CS 670 (Spring 2011) — Final Exam (Takehome)

Unless you are specifically asked to reprove a known fact, you can use all facts from class, homeworks and the textbook without proof. There are four questions on this exam. The only legitimate sources in solving this exam are (1) your textbook and course notes, (2) any handouts provided by us, (3) discussions with the TA or instructor. You cannot discuss the problems with anyone else (whether in the class or not), nor can you look for solutions in other textbooks, on the Internet, or in other sources. The exam is due in David's office by noon on Friday, 05/06. If you submit after that, your exam will automatically be graded as 0.

You can view your graded exam on Wednesday, 05/11, between 10:30-12:00 and 2:00-4:00. If you can't make those times yourself, you can designate (by e-mail to David) another student who will be allowed to view the exam on your behalf. Grades will be finalized at 5:00pm on 05/11; after that, no regrades are possible, and the only reason for grade changes are administrative errors.

G O O D L U C K

(1) [10 points]

Suppose that we generate a “social network” as follows. There are four parameters, k, p, q, n . Out of these, k is the number of “groups”, a small absolute constant, and $0 \leq q < p \leq 1$ are edge probabilities, also absolute constants¹. n is the number of people in each “group”, and we will look at n getting large. There are k disjoint groups of n individuals each. If two individuals u, v are in the same group, they will have an edge between them with probability p . If they are in different groups, the probability of having an edge is q . All edges are generated independently of each other. (Obviously, this model is not completely realistic, as individuals will be friends with a constant fraction of all people in the world.)

Now suppose that someone shows you a social network (graph) G that has been generated in this way, and you want to infer the groups from this. Obviously, there will be lots of edges between people who are not actually in the same group. However, there is a simple heuristic for figuring out whether they are in the same group: count their common friends². If that number is greater than some threshold value, then they are likely to be in the same group. Otherwise, they are not.

Prove that there is a choice of threshold (possibly depending on k, p, q, n) such that for large enough n (meaning n larger than some constant that may depend on k, p, q), this heuristic will reconstruct the original group memberships perfectly (i.e., not misclassify *any* individual), with probability at least $1 - 1/n^2$. (The probability is taken over the random process of generating the graph.)

(2) [10 points]

You may remember from the midterm that the key step for Problem 2(c) was to show that if you have a bipartite graph G with maximum degree d , you could add edges (and perhaps nodes) to make all nodes have degree exactly d . The easiest way to do that was to allow yourself to create a multigraph (with parallel edges); another way chosen by some was to add enough other nodes. It turns out that it is not always possible to make the graph d -regular without adding new nodes or parallel edges. Here, you are to come up with an algorithm for deciding whether it is possible.

Let $G = (X \cup Y, E)$ be a bipartite graph with $|X| = |Y|$ with maximum degree d . The goal is to add edges to G without adding nodes and without adding parallel edges such that in the resulting graph, each node has degree exactly d . Give and analyze a polynomial-time algorithm for deciding whether this is possible, and for finding the edges to add in case it is possible.

(3) [15 points]

Consider the following online version of the SET COVER problem. There is a universe U of n elements

¹That means that these three numbers will not vary with n .

²This is what Facebook does for friend recommendations, for instance.

(which the algorithm knows ahead of time), and a collection \mathcal{C} of sets S_1, \dots, S_m (also known to the algorithm ahead of time). Now, some (or maybe all) of the elements will arrive online one by one. The algorithm has to select sets (online) in order to ensure that at every point in time, all elements that have arrived so far are covered. The goal is to do so with as few sets as possible. (So there are no costs associated with the sets this time.)

- (a) [6 points] Give an online algorithm and prove that it is $O(\sqrt{n})$ competitive³. (Hint: The obvious algorithm works.)
 - (b) [3 points] Prove a lower bound on the competitive ratio of your algorithm matching your upper bound from part (a). (Notice that if you came up with a better than $O(\sqrt{n})$ competitive algorithm in (a), you will need a better lower bound here, as $\Omega(\sqrt{n})$ is obviously not a lower bound then.)
 - (c) [6 points] Prove that *every* deterministic online algorithm has a lower bound of $\Omega(\log n)$ on its competitive ratio. (Hint: Think binary expansion of numbers.)
- (4) [10 points]

In class, we saw an $O(\log n / \log \log n)$ approximation for the problem of routing paths with minimum congestion in an undirected graph. Recall that the setup was that we had k pairs (s_i, t_i) , and for each i had to select an s_i - t_i path. The goal was to minimize the maximum load (number of paths using it) of any edge.

Suppose now that the graph is actually an undirected cycle. (If it helps you, you may assume that the number n of nodes is odd.) To make things a bit more interesting, each pair i has a *demand* $d_i \geq 0$, which is the amount of traffic it will put on the selected path. The load of an edge e is now not the *number* of paths that use e , but rather the *total load* of all paths using e . We still want to minimize the maximum load of any edge.

Give and analyze a polynomial-time 2-approximation algorithm for this problem. (Hint: There is a simple algorithm with a slightly more complicated analysis, and a slightly more complicated algorithm with a simple analysis.)

³One can do significantly better, essentially matching the lower bound from part (c), but that's significantly more complicated.