CS–270 Combinatorial Algorithms and Data Structures, Spring 2017

Homework 2

Due February 17.

1. We will work with the fractional path routing problem in this question. Recall that in the path routing problem you are given a graph $G = (V, E)$, and pairs of terminals $(s_1, t_1), \dots, (s_k, t_k)$ and asked to find paths, $p_1, \dots, p_k$ where p_i connects s_i and t_i and the maximum congestion is minimized: that is, we minimize the maximum number of paths that use any edge. The "flow" or fractional version of the problem asks that a unit flow is routed between s_i and t_i and that the maximum flow across any edge is minimized.

(a) Give an example where the maximum congestion in optimal solutions to the two problems differ.

(b) Show that the value of the solution to the toll problem defined in class is a lower bound on the optimal solution to the fractional problem. (We've done something similar in class.)

2. Path routing runtime.

In class, we saw that the path routing algorithm must terminate eventually because there is a potential function $\Psi = \sum_e 2^{c(e)}$ whose value decreases every time a path is re-routed. Unfortunately, this analysis doesn't give a very good bound on the runtime — perhaps the potential function could start out exponentially high, and only decrease by a small amount on each iteration.

Show how to change the algorithm from class in order to run in polynomial time, and still achieve a maximum congestion $c_{\max} \le 3(1 + \frac{1}{m})c_{\text{opt}} + O(\log m)$. *(Hint: when there is more than one path that can be re-routed, which one should the algorithm choose? When should the algorithm stop?)*

We'll post one example way to modify the algorithm after the weekend, but try to figure it out on your own first. In any case, you will still need to do the analysis.

3. Equilibrium.

(a) Give an example of a two person game (non-zero sum) with two different Nash equilibria of different values. (Hint: at a Nash equilibrium neither has incentive to change.)

(b) Find an equilibrium for a version of the rock, paper, scissors, cheat game, where rock, paper, scissors payout normally and where row has a cheat strategy, row wins when column plays either paper or scissors and ties against rock. (Hint: write it as a linear program, see "Algorithms" by DasGupta, Papadimitriou, and Vazirani, and use a linear programming solver. Plenty are available online.)

(c) Implement the multiplicative weights algorithm for losses in the range $[0, 1]$.

Use the multiplicative weights algorithm with $\epsilon = .1$, with the game from the previous problem set, question 3(b). You will need to shift and scale the payoffs.

Turn in your translated and scaled game matrix.

Also turn in the corresponding game strategies from day 10, day 100, and day $\lceil 100 \ln 4 \rceil$ along with the smallest value of δ where your strategies are in δ-approximate equilibrium for each of these days. δ-approximate equilbria were defined in Lecture 6: $\delta = C(x) - R(y)$, where C and R denote the values of the best response of the column and row player, respectively. (Use the natural translation and scaling of the game matrix to ensure that the payoffs are in $[0, 1]$.)

4. (a) Given bipartite graph G with weight function w, and a solution, M, to the maximum weight matching problem and a solution, $p(\cdot)$, to the vertex cover problem of the graph. Find an algorithm to solve both problems after increasing the weight of one edge (u, v) in the graph (starting from the solution to the original graph). Your algorithm should run in time $O(m \log n)$, where m is the number of edges and n is the number of nodes in G. (You can assume that zero weight edges make perfect matchings possible.)

(b) Does the previous algorithm give an $O(m^2 \log n)$ algorithm for the maximum weight macthing problem?

5. For a multi-variate function $f(x)$ where x is a vector, we take the gradient $\nabla_x f(x)$ is the vector $(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_i}, \ldots)$.

(a) Compute the gradient of $f(x) = x^t Ax$ where A is an $n \times n$ symmetric matrix and x is an n-dimensional vector? Express your answer using A and x.

(b) Compute the gradient of $f(x) = \sum_i 2^{a_i \cdot x}$ where a_i is the ith row of an $m \times n$ matrix A, and x is an n-dimensional vector. Express your answer by reference to A and x.

6. Games and application.

In lecture, we showed how to find a $1 + \epsilon$ approximate solution to the fractional path routing problem in $O(k^2 m \frac{\log^2 m}{\epsilon^2})$ time using the experts framework. Use the framework to find an $O(km \frac{\log^2 m}{\epsilon^2})$ time *randomized* algorithm for fractional path routing. You may assume $k \leq n$. Your algorithm should work with probability $1 - O(1/k)$, and the **expected** congestion of the algorithm's output $E[c_{max}]$ should satisfy

$$E[c_{max}] \leq (1 + \epsilon)C^* + \epsilon$$

where C^* is the optimal congestion. (Hint: the solution I am thinking of uses the average of the column (routing) player's methods as its routing. The trick is to get the maximum payoff down to 1, which one path would give on average, and route the same number of paths for all k terminal pairs. A useful lemmas that you may use without proof follow.)

Lemma: If one chooses from k possibilities $T \geq \frac{8k \ln k}{\epsilon^2}$ times independently and uniformly at random, each possiblity will be chosen $\frac{T}{k}(1 \pm \epsilon)$ times with probability at least $1 - 1/k$.

7. Consider the following investing scenario. There are n possible instruments one can invest in in a period of T time intervals, where each instrument either pays one of 0 dollars or 1 dollar at each time. You have inside information that at least one of the instruments yields P dollars over the T time intervals, but you don't know which one.

One can only invest in one instrument. The problem is to decide which one to invest in and how long to wait before making the decision.

(a) Show that if $n > 1$, then for every P and T, any deterministic algorithm may yield zero dollars.

(b) (Extra for experts/theorists)

Give a randomized algorithm that yields $\Omega(P/\log n)$ dollars in expectation. Give a proof for the performance of your method. (For convenience, you may assume that $P = c \log n$ for some constant $c > 1$, say 2 or 10 or something, so that it's enough to get 1 unit of profit on average.)