CS170 Fall 2013 Solutions to Homework 7

Zackery Field, section Di, 103, `cs170-fe`

October 25, 2013

1. (20 pts.) A greedy algorithm - so to speak

We can formalize this as a graph problem. Let the undirected graph $G = (V, E)$ denote LinkedIn's relationship graph, where each vertex represents a person x who has an account on LinkedIn. There is an edge $\{u, v\} \in E$ if u and v have listed a professional relationship with each other on LinkedIn (we will assume that relationships are symmetric). We are looking for a subset $S \subseteq V$ of vertices so that every vertex $s \in S$ has edges to at least 20 other vertices in S. And we want to make the set S as large as possible, subject to these constraints. Design an efficient algorithm to find the set of super-schmoozers (the largest set S that is consistent with these constraints), given the graph G.

We can begin by identifying any vertices that have degree < 20. Let the set of these vertices be called $\bar{S}$. Since the definition of an edge $s \in S$ depends on its neighbors recursively, there has to be a way tracking the number degree of the edges that fall into S and then that information must be returned to all other nodes in the path.

This can be accomplished with a DFS with ties being broken on back edges. Simply mark each edge with pre- and post-visit numberings. As you run the search you will run into three cases:

- You will arrive at a vertex v that is a sink in the sense that it is connected to < 20 vertices $\in S$. By definition, $v \in \bar{S}$.
- You wil arrive at a vertex v that is connected to ≥ 20 vertices $\in S$. In this case you can not be sure whether or not $v \in S$ because it is not yet known whether the vertices connnected to it are $\in S$.
- Arrival at a vertex v reveals a back edge whose vertex l. If the relationship $post(v) - pre(l) \geq 20$ holds, then the vertex v is in a cycle of length ≥ 20 containing vertices

To be continued : sorry!

2. (20 pts.) A funky kind of coloring

Let $G = (V, E)$ be an undirected graph where every vertex has degree ≤ 51. Let's find a way of coloring each vertex blue or gold, so that no vertex has more than 25 neighbors of its own color.

Consider the following algorithm, where we call a vertex 'bad' if it has more than 25 neighbors of its own color:

1. Color each vertex arbitrarily

2. Let $B := \{v \in V : v \text{ is bad}\}$

3. While $B \neq \emptyset$:

4. Pick any bad vertex $v \in B$

5. Reverse the color of v.

6. Update B to reflect this chang, so that it again holds the set of bad vertices.

Notice that if the algorithm terminates then, it is guranteed to find a coloring with the desired property.

(a) Prove that this algorithm terminates in a finite number of steps. I suggest that you define a *potential function* that associates a non-negative integer (the potential) to each possible way of coloring the graph, in such a way that each interation of the while-loop is guranteed to strictly reduce the potential.

Following the hint, allow $p(G) : G \mapsto \mathbb{N}_0$ to denote the potential of a graph G. Let this potential be explicitly defined as the number of edges in E that have the same color vertex on either end (same-color edges), which by definition always takes on a value $\in \mathbb{N}_0$. Each time that the loop runs, a vertex $v \in B$ is changed from one color to the other. $\forall v \in B$ there are > 25 vertices adjacent to v and of the same color, and there are at most 25 vertices adjacent to v of the opposite color. This shows that even with the maximal number of non-similar and minimal similar adjacent vertices there is still a decrease in the number of same-color edges by at least one (namely, one of the edges with v as an end) when you switch the color of v. Thus, each iteration of the loop leads to a decrease of the value of $p(G)$ by *at least* one.

(b) Prove that the algorithm terminates after at most $|E|$ iterations of the loop. Hint: You should figure out the largest value the potential could take on.

If $\forall v \in V : v \in B$ then the potential function $p(G)$ described above takes on its maximum value, $p(G) = |E|$. It was shown in (a) that for each iteration of the loop for a chosen $v \in B$ the decrease in $p(G)$ is at least one. Therefore, even in this worst case scenario, the algorithm will terminate after $|E|$ iterations.

3. (20 pts.) Job Scheduling

You are given a set of n jobs, each runs in unit time. Job i has an integer-valued deadline time $d_i \geq 0$ and a real-valued penalty $p_i \geq 0$. Jobs may be scheduled to start at any non-negative integer time $(0, 1, 2, etc)$, and only one job may run at a time. If job i completes at or before time d_i, then it incurs no penalty; otherwise, it is late and incurs penalty p_i. The goal is to schedule all jobs so as to minimize the total penalty incurred. For each of the following greedy algorithms, either prove that it is correct, or give a simple counterexample (with at most three jobs) to show that it fails.

(a) Among the unscheduled jobs that can be scheduled on time, consider the one whose deadline is the earliest (breaking ties with the highest penalty), and schedule it at the earliest available time. Repeat.

counterexample:

$$j_a = \{d_a \quad = 1, p_a = 5\}$$
$$j_b = \{d_b \quad = 2, p_b = 15\}$$
$$j_c = \{d_c \quad = 2, p_c = 10\}$$

The minimized total penalty for this set is: $p = 5$, where j_b and j_c are compelted.

However, the algorithm at time $t = 0$ will schedule job j_a because it has the earliest deadline, and then it will schedule j_b at time $t = 1$ breaking ties with the highest penalty. This leaves the total incurred penalty as being $p_{total} = 10$.

(b) Among unscheduled jobs that can be scheduled on time, consider the one whose penalty is the highest (breaking ties with the earliest deadline), and schedule it at the earliest available time. Repeat.

counterexample:

$$j_a = \{d_a \quad = 1, p_a = 5\}$$
$$j_b = \{d_b \quad = 2, p_b = 10\}$$
$$j_c = \{d_c \quad = 3, p_c = 15\}$$

The minimal penalty incurred for this example is: $p = 0$, with all jobs being completed.

For this algorithm, the scheduling is based on penalty so the order of scheduling will be j_c start@ $t = 0$, j_b start@ $t = 1$, which will incur a penalty of $p = 5$.

Continued on Page 6

4. (20 pts.) Timesheets

Suppose we have N jobs labelled $1, ..., N$. For each job, you have determined the bonus of completing the job, $V_i \geq 0$, a penalty per day that you accumulate for not doing the job, $P_i \geq 0$, and the days required for you to successfully complete the job $R_i > 0$.

Every day, we choose one unfinished job to work on. A job i has been finished if we have spent R_i days working on it. This doesn't necessarily mean you have to spend R_i contiguous sequence of days working on job i. We start on day 1, and we want to complete all our jobs and finish with maximum reward. If we finish job i at the end of day t, we will get reward $V_i - (tP_i)$. Note, this value can be negative if you choose to delay a job for too long.

Given this information, what is the optimal job scheduling policy to complete all of the jobs?

The objective function for N jobs that we are trying to maximize is:

$$reward(N) = \sum_{i=1}^{N} V_i - (tP_i)$$

The optimal algorithm involves solving a linear system that maximizes this reward function. An approach that is inspired by problem 3(c) would be to take each job that has the lowest value to penalty ratio and schedule it for as late as possible with respect to maximizing the value of $V - (tP)$.

5. (20 pts.) Weighted Set Cover

In class we looked at a greedy algorithm to solve the *set cover* problem, and proved that if the optimal set cover has size k, then our greedy algorithm will find a set cover of size at most $k \log_e n$.

Here is a generalization of the set cover problem.

- *Input:* A set of elements B of size n; sets $S_1, \ldots, S_m \subseteq B$; positive weights $w_i, \ldots, w_m$.
- *Output:* A selection of the sets S_i who union is B.
- *Cost:* The sum of the weights w_i for the sets that were picked.

Design an algorithm to find the set cover with approximately the smallest cost. Prove that if there is a solution with cost k, then your algorithm will find a solution with cost $O(k \log n)$.

Continued on Page 7

Continued from Page 3

(c) Among unscheduled jobs that can be scheduled on time, consider the one whose penalty is the highest (breaking ties arbitrarily), and schedule it at the latest available time before its deadline. Repeat.

For the highest penalty jobset J' with deadline $t = i$, the time slots that the jobs will be scheduled in are $t = i - 1$ to $t = i - |J'|$ (arbitrarily). The only case that is of any concern is when there is another set of jobs J with deadlines within the timespan $t = \{i - 1, i - |J'|\}$ and maximized penalty s.t. penalty$(J) <$ penalty(J'). In this case, if there there was a more optimal solution that involved a job $j \in J$ then this job would belong to J' and would have already been scheduled.