Problem Set 9

Due: Wednesday, November 9, 2016.

Collaboration policy: collaboration is *strongly encouraged.* However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 people in a given week.
4. **No bibles. This includes solutions posted to problems in previous years.**

NONCOLLABORATIVE Problem 1. Sometimes, approximation techniques are applied to tractable problems, when it isn't worth spending the (polynomial) time to find the optimum solution. Consider the problem of finding a maximum bipartite matching.

(a) Argue that the greedy algorithm (repeatedly take any edge that does not conflict with previous choices) can be implemented in linear time and gives a 2-approximation to the maximum (number of edges) bipartite matching.

(b) Generalize to argue that when edges have positive "weights," the greedy algorithm (consider edges in decreasing order of weight) can be implemented in $O(m \log n)$ time and is a 2-approximation algorithm for maximum (total) weight bipartite matching.

(c) Show the same holds for the general (non-bipartite) max-weight matching problem (even though we haven't studied the exact polynomial time algorithm for this problem).

Problem 2. You are given a directed graph representing a network of roads where traversing road (edge) e takes time t_e but also consumes g_e gallons of gasoline. Due to varying speed limits and hills the two quantities are unrelated. Given a starting point s, a destination d, and a *gas limit* G, develop a FPAS to for finding a path minimizing the travel time without consuming more than G gas (exact solution is NP-hard).

(a) Suppose first that the graph is acyclic and travel times (but not gas usage) are all small integers. Give an exact algorithm (it should also be able to detect when there is no solution).

(b) Extend the algorithm of the previous part to an FPAS for general travel times.

(c) Extend the algorithm of the previous part to general graphs (with cycles).

Problem 3. $1 \mid prec \mid \sum w_j C_j$. In this scheduling problem there are *precedence constraints*: job j cannot be started until after all jobs in its *precedence set* $A(j)$ have been completed. Each job also gets a *weight* w_j. Our goal is to minimize the *weighted average completion time* $\sum w_j C_j$ (where C_j is the time job j completes). Assuming that the p_j are polynomially-bounded integers, we will give a constant-factor approximation for this problem via a linear programming relaxation. Define variables x_{jt} for each (integer) time step t, denoting the "indicator" that job j completed at time t.

(a) Write down constraints forcing the ILP to solve the problem. In particular, enforce that every job completes, that a job is not processed before its predecesors, and (most subtly) that the total processing time of jobs completed before time t is at most t.

(b) The LP relaxation of this ILP is a kind of "timesharing" schedule for jobs. Define the *fractional completion time* of job j to be $\overline{C}_j = \sum_t t x_{jt}$. To turn it into an actual order, consider the *halfway point* h_j of each job: this is the time at which half the job is completed. Prove that $\overline{C}_j \geq h_j/2$.

(c) Consider the schedule that processes jobs in order of their halfway points. Prove that no job runs before its predecessors.

(d) Prove that for the given order, the actual completion time for job j is at most $4\overline{C}_j$.

(e) Conclude that you have a constant-factor approximation for $1 \mid prec \mid \sum w_j C_j$.

OPTIONAL (f) Suppose that the processing times are not polynomially bounded integers. Show how rounding can reduce the problem to this case while adding a negligible additional error.

OPTIONAL (g) Suppose that each job comes with a *release date* r_j before which it cannot be processed. Generalize your algorithm to handle this case (with a slightly worse constant).

Problem 4. The following is the NP-hard problem of **bin packing**: Given n items with sizes $a_1, \cdots, a_n \in (0, 1]$, find a packing of the items into unit-sized bins that minimizes the number of bins used. Let B^* denote the optimum number of bins for the given instance. Bin packing is a lot like $P||C_{\max}$, but somewhat more difficult because you have no flexibility to increase the bin sizes (completion time).

(a) Suppose that there are only k distinct item sizes for some constant k. Argue that you can solve bin-packing in polynomial time. Sizes can be arbitrary. **Hint:** no new algorithm needed here!

(b) Suppose that you have packed all items of size greater than ϵ into B bins. Argue that in linear time you can add the remaining small items to achieve a packing using at most $\max(B, 1 + (1 + 2\epsilon)B^*)$ bins (note that the second term involves B^*, not B).

(c) For $P||C_{\max}$, we reduced to the previous case in (a) by rounding each job size up to the next power of $(1 + \epsilon)$. Why doesn't that work for bin packing? (Saying $1 + \epsilon$ is bigger than 1 is not a good enough reason since you can take negative powers.)

(d) Consider instead the following *grouping* procedure. Fix some constant k. Order the items by size. Let S_1 denote the largest n/k items, S_2 the next largest n/k, and so on. Suppose that you increase the size of each item to equal the largest size in its group, so that there are only k distinct sizes. Argue that this increases the optimal number of bins by at most n/k. **Hint:** imagine setting aside the jobs in S_1. Argue that the remaining items, with their increased sizes, can still fit into the bins used by the original packing.

(e) Devise a polynomial time scheme that uses at most $1 + (1 + \epsilon)B^*$ to pack all the items.

Observe that the algorithm above just misses the definition of polynomial approximation scheme, because of the additive error of 1 bin. In practice, of course, this is unlikely to matter. The above scheme is known as an *asymptotic PAS* since its approximation ratio is $(1 + \epsilon)$ in the limit as the optimum value grows.

The techniques above were augmented to give an algorithm that finds a packing using $B^* + O(\log^2 B^*)$ bins—giving asymptotic approximation ratio 1. In 2013, the bound was improved to $B^* + O(\log B^* \log\log B^*)$. Indeed, at present it remains concievable that some algorithm might achieve $B^* + O(1)$ bins in polynomial time!

Problem 5. You have reached a river (modelled as a straight line) and must find a bridge to cross it. The bridge is at some integer coordinate upstream or downstream.

(a) Give a 9-competitive deterministic algorithm for optimizing the total distance travelled up and downstream before you find the bridge. This is optimal for deterministic strategies.

(b) Give a randomized 7-competitive algorithm for the problem

OPTIONAL (c) Give a randomized algorithm with competitive ratio strictly better than 7 (one can actually do better than 5-competitive).

OPTIONAL Problem 6. The graph coloring problems is to assign a *color* to every vertex of a graph such that no two neighbors have the same color. The objective is to minimize the number of colors.

(a) Prove that a 2-colorable graph can be colored with 2 colors in polynomial time.

(b) Show that a graph where every vertex has degree at most d can be colored with $d + 1$ colors using the obvious greedy algorithm.

(c) Give a polynomial-time approximation algorithm that will color any 3-colorable graph with $O(\sqrt{n})$ colors. **Hint:** how many colors do you need to color the *neighbors* of a given vertex?

Sadly, this kind of polynomial approximation is currently the best known for 3-coloring, although you can slightly improve the polynomial using sophisticated techniques. Some believe that you should be able to do much better—perhaps an $O(\log n)$-approximation.

Problem 7. How long did you spend on this problem set? Please answer this question using the Google form that is located on the course website. This problem is mandatory, and thus counts towards your final grade. It is due by the Monday 2:30pm after the pset due date.