Problem Set 8

Due: Wednesday, October 31, 2012.

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. You are given a collection of n points in some metric space (i.e., the distances between the points satisfy the triangle inequality). Consider the problem of dividing the points into k clusters so as to minimize the maximum diameter of (distance between any two points in) a cluster.

(a) Suppose the optimum diameter d is known. Devise a greedy 2-approximation algorithm (an algorithm which gets k clusters each of diameter at most $2d$). **Hint:** consider any point and all points within distance d of it.

(b) Consider the algorithm that (k times) chooses as a "center" the point at maximum distance from all previously chosen centers, then assigns each point to the nearest center. By relating this algorithm to the previous algorithm, show that you get a 2-approximation.

Problem 2. 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 3. The *Steiner Tree Problem* presents an undirected graph G with edge costs c_e, and a subset T of the vertices called *terminals*. The goal is to construct a minimum cost tree spanning all the terminals (it may also include any desired subset of the non-terminals; these included vertices are called *Steiner points*).

(a) Suppose you compute all-pairs shortest paths in G, and create a complete graph G' on the terminals T where each edge cost is equal to the shortest path between its (terminal) endpoints in G. Relate the cost of the minimum spanning tree in this graph to the cost of the optimum steiner tree in G.

(b) Give a 2-approximation algorithm for the Steiner tree problem.

Problem 4. 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 know 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 5. Consider the problem of scheduling, on one machine, a collection of jobs with given *processing times* p_j, *due dates* d_j, and *lateness penalties* (*weights*) w_j paid for jobs that miss their due dates, so as to minimize the total lateness penalty. (If we let U_j denote the indicator variable for job j completing after its due date, then our problem is $1 \mid\mid \sum w_j U_j$.)

(a) Argue that any *feasible* subset of jobs (that can all together be completed by their due dates) might as well be scheduled in order of increasing deadline (so it is sufficient to find a set without worrying about order). **Hint:** if two adjacent jobs in the sequence are out of order, swap them.

(b) Assuming the lateness penalties are polynomially bounded integers (i.e. $\log w_j$ is polynomial in n), give a polynomial-time dynamic program that finds the fastest-completing maximum-weight feasible subset.

(c) Give a fully polynomial-time approximation scheme for the original problem of minimizing lateness penalty with arbitrary lateness penalties.

Problem 6. 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.

(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 $(1 + \epsilon)B^*$ 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 have been augmented to give an algorithm that finds a packing using $B^* + O(\log^2 B^*)$ bins—giving asymptotic approximation ratio 1. Indeed, at present it remains concievable that some algorithm might achieve $B^* + O(1)$ bins in polynomial time!

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