INTRO TO APPROXIMATION, CLASS 4

Is greedy scheduling into a knapsack satisfiable?

A selection from homework:

EXERCISE ONE **Scheduling on machines with speeds:**

We have m machines with speeds $s_1 \geq s_2 \geq \cdots \geq s_m$ on which we want to schedule n jobs (so that one machine processes at most one job at a time). Now, processing the j-th job on machine i takes p_j/s_i units of time, where p_j is the length of the job. (The schedule starts at time 0 and two jobs cannot of course run at the same time on one machine.)

We say that a polynomial-time algorithm is a ρ-relaxed decision procedure if it gets a number D in addition to the intput and either (I) creates a schedule so that all jobs are finished till time ρD, or (II) outputs that there is no schedule that can finish all jobs till time D.

Show that one can use a ρ-relaxed decision procedure to find a ρ-approximation algorithm.

EXERCISE TWO Consider SCHEDULING WITH DEPENDENCIES: we schedule jobs on m computers, but in addition to length p_j each job j has a set $\mathcal{D}_j$ of dependencies and we can start job j only if all jobs in $\mathcal{D}_j$ are already finished.

a) Prove the following lower bound on the optimum:
"$OPT \geq$ length of any chain in the input. A *chain* is a sequence of jobs where each one depends on the previous one. Its length is then the total processing time of all the jobs in the chain."
b) Design a greedy 2-approximation algorithm for this problem.

EXERCISE THREE Consider the classic NP-hard KNAPSACK PROBLEM, where we have n objects $a_1, \ldots, a_n$, each object has a weight w_i and cost c_i, and our bag has a weight limit of B.

a) Explain why "naive greedy algorithm", i.e. "we put the most expensive item (that fits) into the knapsack and continue the same way" is a bad one.
b) OK, let us try the following: "we sort the items according to their density (ratio price/size), go through them in decreasing order and insert only those that fit in the knapsack."
Spoiler alert: this algorithm also fails. Show an input where it does.
c) Finally, design a 2-approximation algorithm for this problem. This algorithm does not need to be greedy.
Hint: When you iterate over the items based on the density, at some point it may happen that object P does not fit with the items you have already selected into the knapsack. What should you do then?

EXERCISE FOUR The k-CENTER PROBLEM is another example of an interesting metric problem. On input we get a set $V, |V| = n$ of points in a metric space and the goal is to select k centers out of them (a k-element subset of V) so that the points from V are as close to the centers as possible – so that the point which is furthest away from any center is as close as possible. Formally we minimize the function $u(S) = \max_{p \in V} d(p, S)$, where $d(p, S)$ is the distance from p to its closest point in S.

Design and analyze a 2-approximation algorithm for the k-CENTER PROBLEM.

Tip: Note that both the algorithm and the optimum are choosing exactly k points, the factor that is relaxed by 2 is the distance function. This must be present in the analysis somehow – it makes sense to start by considering the k points that optimum chooses, the k points that you choose, and compare the two sets.

EXERCISE FIVE Recall the integer program for MAX SAT and its linear relaxation. During the lecture you have seen a 3/4-approximation algorithm for MAX SAT based on choosing a better of two solutions, one of which was created by rounding a solution of this relaxation. By a better rounding one can avoid choosing a better of two solutions and still maintain the approximation ratio 3/4.

Find an instance, that is, a set of clauses, such that the optimum of the relaxation OPT_r and the optimum of the instance OPT satisfies $\mathrm{OPT} = (3/4)\mathrm{OPT}_r$.

This shows that using the linear relaxation one cannot obtain a better than 3/4-approximation algorithm. (The worst case ratio between OPT_f and OPT is called integer gap.)

Hint: you can use, for example, just 2 variables and 4 clauses.