Advanced Algorithms

Homework 3

April 15, 2026

Instructions

There are usually many different algorithms one can think of to solve a given problem, especially NP-hard ones. The first goal of this segment of the course is to learn how to reason about such algorithms, and systematically compare their effectiveness. The second goal is to design such algorithms yourself. All five problems below will touch on both of these important goals.

You are encouraged to work together on these problems or come to me if you are stuck. However, you should write up your solutions yourself. Please list the people you worked with at the top of your submission. Looking for answers on the internet is not allowed, nor is working with an AI-powered system for any part of this assignment. You must understand everything you submit and I reserve the right to ask you to orally explain your answer to me. You may write your solutions by hand or in latex. Either way, submit them on gradescope by 10:00pm on April 21st, 2026.

Problem 1 - Number packing (20 points)

Suppose you are given a set of positive integers $A = \{a_1, a_2, \ldots, a_n\}$ and a positive integer B. A subset $S \subseteq A$ is called feasible if the sum of the numbers in S does not exceed B:

$$\sum_{a_i \in S} a_i \leq B.$$

The sum of the numbers in S will be called the total sum of S. You would like to select a feasible subset S of A whose total sum is as large as possible. For example, if $A = \{8, 2, 4\}$ and $B = 11$, then the optimal solution is the subset $S = \{8, 2\}$.

(a) Look up the decision problem called "Subset Sum". Explain briefly why the NP-hardness of Subset Sum implies the NP-hardness of the above number packing problem.

(b) Here is an algorithm for this problem.

(i) Initially $S = \emptyset$ and define $T = 0$.

(ii) For $i = 1, \ldots, n$: **if** $T + a_i \leq B$, **then** let $S \leftarrow S \cup \{a_i\}$ and let $T \leftarrow T + a_i$.

(iii) **Return** S

Give an instance in which the total sum of the set S returned by this algorithm is less than half the total sum of some other feasible subset of A.

(c) Give a polynomial-time approximation algorithm for this problem with the following guarantee: It returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$. Your algorithm should have a running time of at most $O(n \log n)$.

Problem 2 - Fun with Vertex Cover (30 points)

The *weighted* minimum Vertex Cover problem is as follows. We are given a graph $G = (V, E)$ with positive costs $c : V \to \mathbb{R}_{\geq 0}$ on vertices. The goal is to select a subset $S \subseteq V$ of minimum cost so that every edge is touched by a vertex in S. That is, a set $S \subseteq V$ is feasible if for every edge $e = (u, v) \in E$, we either have $u \in S$ or $v \in S$ (or both).

Consider the following natural approximation algorithms for weighted Vertex Cover. For each of them, do the following:

(a) Briefly explain why the algorithm runs in polynomial time.

(b) Explain why the algorithm is guaranteed to output a feasible Vertex Cover.

(c) Give the best bounds you can on the approximation ratio of the algorithm (both lower and upper bounds).

Some of these algorithms may do better in the special case when all costs are 1. If that is the case, explain why this is true as well.

(a) **Super-Greedy.** Consider all the edges in some order. If the edge $\{u, v\}$ being considered is not covered yet, pick whichever of u or v has smaller cost.

(b) **Greedy.** Consider all the edges in some order. If the edge $\{u, v\}$ being considered is not covered yet, pick *both* vertices u and v.

(c) **LP Rounding.** The standard vertex-cover Linear Program is

$$\min \sum_{v \in V} c_v x_v \quad \text{s.t.} \quad x_u + x_v \geq 1 \ \ \forall \{u, v\} \in E, \qquad x \geq 0.$$

Solve the LP to obtain the optimal fractional solution x^*. Then, return

$$S \ = \ \big\{ v \in V \mid x_v \geq 1/2 \big\}.$$

(d) **Local Search.** Two solutions $S, S' \subseteq V$ are *neighbors* if S' can be obtained from S by adding, deleting, or swapping a vertex (a swap simultaneously adds one vertex and drops another). Start with any solution $S \subseteq V$. While there exists a neighboring solution S' with strictly lower cost, move to S'. When a local optimum is reached—i.e. all neighbors are no cheaper—output that solution. (Don't worry about the running-time for this question.)

Problem 3 - Additive Traveling Salesperson (10 points)

So far, we have only worked with approximation guarantees in a multiplicative sense. But sometimes an additive guarantee is also reasonable. We say that an algorithm is an **additive** α-approximation if the solution produced by the algorithm is at most $OPT(I) + \alpha$ for every instance of the problem I.

For example, in the Traveling Salesperson Problem, an additive 100-approximation is an algorithm which always returns a Hamiltonian cycle with cost at most 100 more than the cheapest one. Show that there is no polynomial-time additive 100-approximation algorithm for **metric** TSP unless P = NP.

Problem 4 - Ordered Load Balancing (10 points)

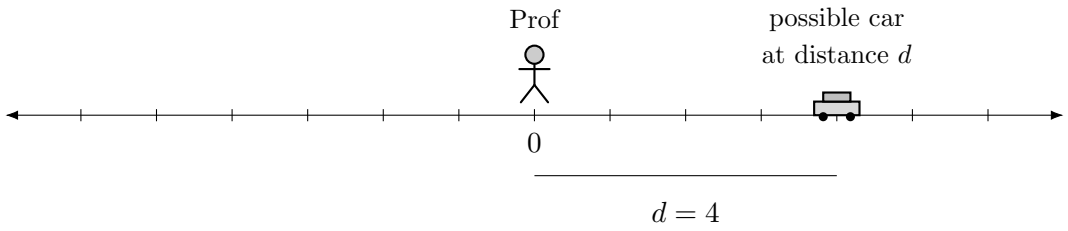
Recall the Least-Loaded-Machine-First algorithm for job scheduling. We showed in class that this is a 2-approximation for the load balancing problem (see the posted notes on job scheduling or Section 11.1 of KT). I also mentioned in class that if you perform this same algorithm but first sort the jobs in non-increasing order of size, then the algorithm is actually a 4/3-approximation.

Is it true that for every instance of the load balancing problem, there exists an order of the jobs so that when Least-Loaded-Machine-First processes the jobs in this order, it produces an optimal solution? Decide whether you think this is true or false, and give either a proof or a counterexample.

Hint: consider the optimal assignment, and order the jobs based on their start time.

Problem 5 - The Absent-Minded Professor (25 points)

Professor Z. Latin leaves his building at the Edmunds Institute of Technology one dark and foggy night and can't remember where he parked his car. Imagine that Prof. Latin is standing at the origin of a line (infinite in both directions). Every integer point on that line corresponds to a parking spot where Prof. Lai might find his car. Because it's dark and foggy, the professor can only see the car at the spot at which he is currently standing.



(a) Your first objective is to describe an algorithm for Prof. Latin to use to find his car. The algorithm should cause him to travel no more than some constant times the distance that he would have traveled had he known where his car was located. This constant (the competitive ratio of the algorithm), should be strictly less than 10. You should state the competitive ratio of your algorithm and show the derivation.

[Hint: Use a doubling search].

(b) Now imagine that, instead of a line of cars, there are k rays of cars which go infinitely outward from the origin where Professor Latin is standing (part (a) handled the $k = 2$ case). Give an algorithm for Professor Latin to find his car. Derive and state the competitive ratio of your algorithm as a function of k.