Advanced Algorithms

Homework 3

November 2, 2025

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 four 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 11/18/2025.

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 lower and upper bounds you can on the approximation ratio of the algorithm.

Some of these algorithms may do better in the special case when all costs are 1. If that is the case, please point it out.

(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 - Evening Optimizer (25 points)

Imagine you're a popular babysitter with a single evening free. Throughout the day, parents text you asking if you can watch their kids during different time windows. Because you can only be in one house at a time, you must choose a subset of jobs that don't overlap. Everyone pays the same flat rate, so your goal is to maximize the number of families you help. In scheduling terms:

- Each babysitting request is an open interval on a time line (you leave right at the finish time).
- Two requests are compatible if their intervals don't overlap.
- You want the largest possible collection of pair-wise non-overlapping intervals.

This is exactly the unweighted interval-scheduling problem: we are given a collection of intervals on the line, and the goal is to find a maximum cardinality subset of non-overlapping intervals. See Figure 1.

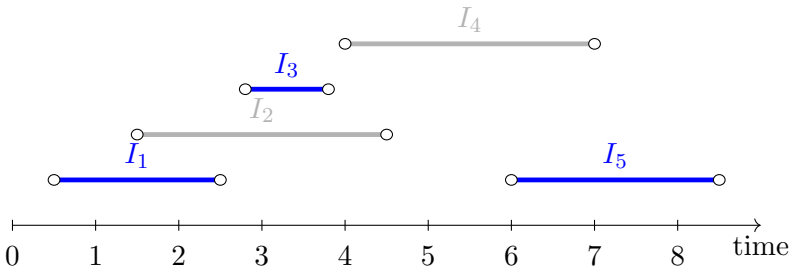


Figure 1: The bold blue intervals form a maximum subset of non-overlapping intervals.

(a) Consider the "Earliest Start Time First" algorithm for the interval-scheduling problem. We begin by selecting the interval whose left endpoint is earliest. We include this interval into the solution, eliminate all overlapping intervals, and repeat until there are no intervals remaining. Give an instance showing that this algorithm can have a bad approximation ratio.

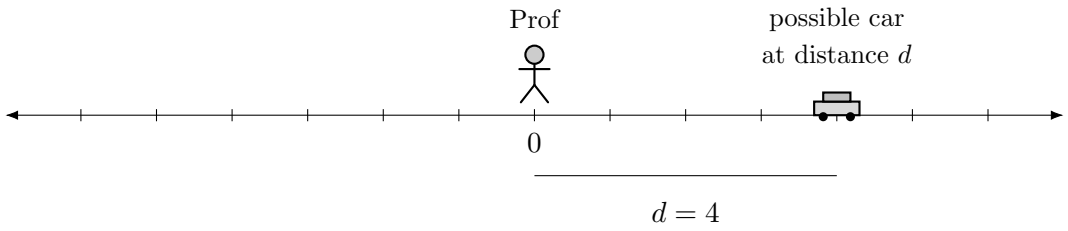
(b) Now consider the "Earliest Finish Time First" algorithm. We iteratively select the interval whose *right* endpoint is earliest, include it into the solution, eliminate overlapping intervals and continue until there are no intervals remaining. Show that this algorithm always returns an optimal solution.

(c) Consider the algorithm that orders the requests in increasing order of duration and greedily selects them while maintaining feasibility. Show that this algorithm is a 1/2-approximation.

(d) What if prices are involved? This can be modeled by associating a positive weight to each interval and asking for a maximum weight non-overlapping set. How would you handle this generalization?

Problem 4 - 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.