# COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

## Problem Set 4

*Module: Optimization and Approximation*

Below is a reminder of key aspects of the PSet:

- The only goal of this PSet is to help you develop your problem-solving skills in preparation for the exams. Your performance on this PSet will not directly contribute to your grade, but will indirectly improve your ability to do well on the exams.
- Because your performance does not directly impact your grade, you may use any resources you like (collaboration, AI, etc.) to help you complete the PSet.
- We strongly suggest taking a serious stab at the PSet alone, to help self-evaluate where you're at. But, we also suggest collaborating with friends, visiting office hours, asking on Ed, and/or using AI tools to help when stuck. Even when able to complete the entire PSet on your own, you may still find any of these methods useful to discuss the PSet afterwards.
- Throughout the PSet, we've included some general tips to help put these into broader context. Exams will not have these, and future PSets may have fewer.
- We strongly suggest treating this like any other PSet, and writing up your solutions as if you were handing them in for a grade. At minimum, we very strongly suggest writing up sufficiently many solutions to discuss with your Coach.

**Aligning Expectations**

The following problem is an "exam-style" problem, meaning that it is designed to mimic the style and difficulty of problems you might see on an exam. This means that it is a problem that we could have included on an exam (except for the optional part a), and it is designed to be solved within the time constraints of an exam. We will try to include more of these problems in future PSets, and we will mark them with the 🗎 symbol in the title.

Note, however, that much like the practice exams, it wasn't polished to the same degree as actual exam problems. In particular, some parts may be easier or harder than intended, or may require more or less time than intended. Also, it might take longer than the intended exam time to solve the entire problem since this is the first time you are practicing this topic.

We highly recommend that you write-up a full solution to this problem.

## Problem 1 : Another Binary String Problem

You are given a binary string $s \in \{0,1\}^n$ and a constant $C > 0$. The goal is to partition $s$ into contiguous segments. For a segment $S \subseteq \{1,\ldots,n\}$, let

$$\text{err}(S) = \min\{\#\text{zeros in } S,\ \#\text{ones in } S\},$$

where #zeros in $S$ is the number of $i \in S$ such that $s[i] = 0$, and similarly for #ones in $S$. Additionally, define the cost of a segment $S$ as $C + \text{err}(S)$.

The objective is to minimize the sum of segment costs. So formally, we want to find segments $S_1,\ldots,S_k \subseteq \{1,\ldots,n\}$ that don't overlap (so $S_i \cap S_j = \emptyset$ for $i \neq j$) and cover $s$ (so $\bigcup_{i=1}^k S_i = \{1,\ldots,n\}$), such that $\sum_{i=1}^k (C + \text{err}(S_i))$ is minimized.

**(a)** (Optional) Design an $O(n^2)$ dynamic program that computes an optimal partition into segments.

(Hint: This problem is quite similar to 1b from the midterm, so feel free to skip it if you don't want to redo similar work.)

Consider the following greedy algorithm (which runs in $O(n)$ time):

- Scan left to right. When you start a segment at position $L$, extend its right end $R$ as far as possible while the segment's error stays within the budget, i.e., while $\text{err}(s[L..R]) \leq C$.
- Close the segment at this maximal $R$. Formally, the chosen $[L,R]$ satisfies $\text{err}(s[L..R]) \leq C$ and either $R = n$ or $\text{err}(s[L..R+1]) = C+1$.
- Start the next segment at $L \leftarrow R+1$ and repeat until $R = n$.

Let the greedy algorithm produce segments $G_1,\ldots,G_g$ and an optimal solution produce segments $B_1,\ldots,B_b$[1]. The goal of the following questions is to prove that the greedy algorithm is a 2-approximation.

Consider an example with $C = 1$ and $s = 011011$. The optimal solution outputs the segment $[1,6]$ with total cost $(C + \text{err}(B_1)) = (1+2) = 3$. Greedy outputs the segments $[1,3],[4,6]$ with total cost $(C + \text{err}(G_1)) + (C + \text{err}(G_2)) = (1+1)+(1+1) = 4$.

**(b)** Show that

$$gC \leq \sum_{j=1}^{g} \text{err}(G_j) + C.$$

(Hint: Write a formula for the cost of the greedy solution and then apply the definition of the greedy algorithm to bound it.)

**(c)** Let $S$ be a segment and $S_1,\ldots,S_t$ be non-overlapping segments that are subsets of $S$, so $S_i \cap S_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^t S_i \subseteq S$. Show that

$$\sum_{u=1}^{t} \text{err}(S_u) \ \leq\ \text{err}(S).$$

[1] Think $G/g$ for "greedy" and $B/b$ for "best".

**(d)** Show that

$$\sum_{j=1}^{g} \mathrm{err}(G_j) \leq (b-1)C + \sum_{j=1}^{b} \mathrm{err}(B_j).$$

(Hint: Partition the $G_j$ intervals into two groups: those that are fully contained within a single optimal segment $B_i$, and those that span across multiple optimal segments. Analyze each group separately, and use parts b and c.)

**(e)** Conclude that the greedy algorithm is a 2-approximation.

## Problem 2 : A Worse Approximation

Recall the Vertex Cover problem which you saw in class. Given an undirected graph $G = (V, E)$ (with $|V| = n$ and $|E| = m$), a vertex cover is a subset of vertices $C \subseteq V$ such that for every edge $(u, v) \in E$, at least one of $u$ or $v$ is in $C$. The goal is to find a vertex cover of minimum size.

Consider the following greedy algorithm, GREEDY, for the vertex cover problem:

- Initialize the vertex cover $C$ to be the empty set.
- While there are edges remaining in the graph:
  - Pick the vertex $v$ with the maximum degree (i.e., the largest number of incident edges), breaking ties arbitrarily.
  - Add $v$ to the vertex cover $C$.
  - Remove $v$ and all edges incident to $v$ from the graph.

**(a)** Prove that GREEDY produces a vertex cover of $G$.

**(b)** In class, we saw the following 2-approximation, ALG:

- Find any maximal matching in $G$, $M$ ($M$ is a set of edges).
- Add both endpoints of $M$ to the vertex cover.

For every even $n > 1$, construct an example of a graph with $n$ vertices where the size of the vertex cover produced by GREEDY is exactly half the size of the vertex cover produced by ALG.

**(c)** Let OPT be the size of the minimum vertex cover of $G$. Let $m_i$ be the number of edges remaining in the graph at the start of iteration $i$ of the while loop ($m_0 = m$). Show that $m_i \leq m\left(1 - \frac{1}{\mathrm{OPT}}\right)^i$.

(Hint: Try to find a lower bound on the degree of the vertex selected in iteration $i$ in terms of $m_i$ and OPT.)

**(d)** Show that GREEDY has an approximation ratio of $1 + 2\ln n$. That is, show that for every graph, if that graph admits a vertex cover of size $C$, GREEDY produces a vertex cover of size at most $(1 + 2\ln n) \cdot C$.

(Hint: Use the inequality $1 - x \leq e^{-x}$ for all $x \in [0, 1]$.)

**(e)** (Bonus, hard!): Prove that GREEDY does not guarantee a $o(\log n)$-approximation. That is, construct a family of graphs where there is a vertex cover of size $f(n)$, but GREEDY produces a vertex cover of size $g(n)$, and $g(n)/f(n) = \Omega(\log n)$.

## Problem 3 : Rounding with Coin Tosses

Consider the following problem, known as the Set Cover problem. You are given a universe of $n$ elements $U = \{1, 2, \ldots, n\}$ and a collection of $m$ subsets $S_1, S_2, \ldots, S_m$ of $U$. A set cover of $U$ is a collection of these subsets whose union is equal to $U$, or formally, a set $I \subseteq \{1, 2, \ldots, m\}$ such that $\bigcup_{i \in I} S_i = U$. The goal is to find a set cover of minimum size.

**(a)** Let $x_i$ be a binary variable for each subset $S_i$, where $x_i = 1$ if subset $S_i$ is included in the set cover and $x_i = 0$ otherwise. Write an integer linear program that captures the set cover problem using these variables.

**(b)** Now consider the linear programming relaxation of the integer linear program you wrote in part (a), where the variables $x_i$ are allowed to take any value in the interval $[0, 1]$.

Consider the following rounding scheme: take any $\vec{x}$ satisfying the constraints of your LP, and include in your cover every set $i$ with $x_i \geq 1/2$.

Prove that this rounding scheme does not necessarily guarantee a valid set cover. That is, come up with an instance of sets, and a valid $\vec{x}$ satisfying the constraints of your LP, such that rounding $\vec{x}$ in the manner described above does not produce a valid set cover.[2]

Let's consider a different rounding scheme, based on randomization. We will prove step by step that this randomized rounding scheme produces a set cover of size at most $O(\log n)$ times the minimum set cover with probability close to $1/2$. So, in a probabilistic sense, this rounding scheme yields an approximation algorithm for the set cover problem with approximation ratio $O(\log n)$.

Assume that $\{x_i^*\}_{i=1}^m$ is an optimal solution to the linear programming relaxation. Now, for each subset $S_i$, include it in the set cover with probability $x_i^*$, independently of the other subsets.

**(c)** Let $u \in U$ be an arbitrary element of the universe. Show that the probability that $u$ is not covered by the selected subsets in the randomized rounding scheme is at most $1/e$.

(Hint: Use the fact that $1 - x \leq e^{-x}$ for all $x \in [0, 1]$.)

The above problem shows that each element is covered with some constant probability, but it does not guarantee that all elements are covered, which is our goal. To address this, we can repeat the randomized rounding process multiple times and take the union of the selected subsets from each iteration. In other words, we repeat the following procedure $k$ times: for each subset $S_i$, include it in the set cover with probability $x_i^*$, independently of the other subsets. If a subset is selected in any of the $k$ iterations, we include it in the final set cover. Refer to this procedure as ROUND(k).

**(d)** Show that the probability that all elements in $U$ are covered by ROUND($2 \ln n$) is at least $1 - \frac{1}{n}$.

**(e)** Show that with probability at least $\frac{1}{2}$, the size of the set cover obtained from ROUND($2 \ln n$) is at most $4 \ln n$ times the minimum set cover.

[2] To be clear, you do not need to ensure that your chosen $\vec{x}$ solves the LP – it just needs to satisfy the feasibility constraints.

**(f)** Conclude that the randomized rounding scheme with $k = 2 \ln n$ iterations produces a set cover of size at most $4 \ln n$ times the minimum set cover with probability at least $\frac{1}{2} - \frac{1}{n}$.