# COS 330: Great Ideas in Theoretical Computer Science

Fall 2026

## Problem Set 6

*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.** Your performance on this PSet will not directly contribute to your grade, but will indirectly improve your ability to do well on the exams.
- **Your performance does not directly impact your grade**, so you may use any resources you like (collaboration, AI, etc.) to help you complete the PSet.
- **AI is not equally welcome in every part of this PSet**, and we say so explicitly where it matters. Problems 1 and 2 are exam-style (except Problem 1(a), as explained there) and should be attempted alone first. Problem 3 explicitly calls for AI assistance in part (e).
- **None of this is an integrity rule** – PSets are not graded directly. It's advice about what will actually develop your skills and in the process prepare you well for the exams, which are closed-book and closed-AI.
- **See the course website** for a full walkthrough of AI tools and recommendations, as well as the input files needed for Problem 3.
- **An AI-helper is available for Problem 1, Problem 2, and Problem 3(a)–(d)** – unlike a general assistant, it gives hints and pushes back instead of handing you the answer. See the link above for more information.

---

## Problem 1 : Greedy Set Cover

In this problem you will study a new optimization problem, Set Cover, and analyze a greedy algorithm for it. Let's start with some definitions.

You are given a universe $U = \{1, \ldots, n\}$ of $n \geq 1$ elements and $m \geq 1$ subsets $S_1, \ldots, S_m$ of $U$. Assume that $\bigcup_{i=1}^{m} S_i = U$, so a cover exists. A set cover is a choice of indices $I \subseteq \{1, \ldots, m\}$ such that $\bigcup_{i \in I} S_i = U$. The Set Cover problem is to find a set cover of minimum size, that is, to minimize $|I|$.

This is another example of an optimization problem, where feasible solutions are set covers and the objective is to minimize the number of sets used. It's also an example of a problem that is NP-hard, so we do not expect to find a polynomial-time algorithm that always finds an optimal solution. Instead, we will consider the following greedy algorithm:

- While there are uncovered elements remaining (i.e., elements not yet in any chosen set):
  - Select the set covering the largest number of uncovered elements (breaking ties arbitrarily).
  - Mark all elements in this set as covered.

You will analyze the approximation ratio of this greedy algorithm using duality[1].

Consider the following Integer Programming formulation of the Set Cover problem. Consider variables $x_i$ for $i = 1, \ldots, m$ (intuitively, you can think of $x_i$ as indicating whether set $S_i$ is chosen). The IP is:

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} x_i \\ \text{s.t.} \quad & \sum_{i:\, v \in S_i} x_i \;\geq\; 1 \qquad \forall v \in U, \\ & x_i \;\in\; \{0,1\} \qquad \forall i \in \{1, \ldots, m\}. \end{aligned}$$

We relax the above into a Linear Program (LP) by replacing the integrality constraint $x_i \in \{0, 1\}$ with $x_i \geq 0$ for every $i$. Let OPT denote the size of an optimal (minimum-size) set cover, $\text{OPT}_{\text{LP}}$ denote the optimal value of the above LP relaxation, and GREEDY denote the size of the set cover returned by the greedy algorithm.

*Note:* It's important to emphasize that the greedy algorithm is not solving the LP relaxation. In fact, for the purpose of this problem one doesn't even need to know how to solve LPs. They will be used only as a tool to analyze the greedy algorithm, through duality.

### Problem Solving Tips

There are a lot of definitions in this problem, and the main difficulty at the start is making sense of all of them. Before trying to prove anything, take some time to build intuition:

- Rewrite the main definitions in your own words: what a set cover is, what the greedy algorithm does, what the IP and its LP relaxation are, and what OPT, $\text{OPT}_{\text{LP}}$, and GREEDY measure. How do you expect these three quantities to compare to each other, and why?

[1] This should sound very surprising to you. The algorithm we just described doesn't use linear programming at all! But we can use LPs and duality as an analysis tool.

- Work through a small example by hand. For instance, take $U = \{1, \ldots, 6\}$ with $S_1 = \{1, 2, 3, 4\}$, $S_2 = \{1, 3, 5\}$, and $S_3 = \{2, 4, 6\}$. Run the greedy algorithm, keeping track of how many new elements each chosen set covers. Then find OPT, and write down the IP and the LP for this instance.

Don't try to work on the parts below until you understand the problem statement.

**Part (a) Is Not Exactly Exam-Style**

Writing down a dual is useful practice, but it is mostly a mechanical exercise. On an exam, we would provide the dual. Since the later parts use your answer, check it with the AI-helper or ask on Ed before continuing. The reason to include this is that taking a dual will help you better understand the concept of duality, which you really need if you want to solve the later parts.

**(a)** Write the dual of the LP above, using a variable $y_v$ for each element $v \in U$.

**(b)** List the elements as $v_1, \ldots, v_n$ in the order greedy covers them: list all elements first covered in the first iteration (in any order), then all elements first covered in the second iteration, and so on. For each $j$, let $c_j$ be the number of new elements covered in the iteration that first covers $v_j$. Thus, elements first covered in the same iteration have the same value of $c_j$.

For every $j$, consider the following dual solution:

$$y_{v_1} = \cdots = y_{v_{j-1}} = 0, \qquad y_{v_j} = \cdots = y_{v_n} = \frac{1}{c_j}.$$

Show that this assignment is feasible for the dual program. That is, show that for every set $S_i$, we have $\sum_{v \in S_i} y_v \leq 1$.

**Small Hint**

How many of $v_j, \ldots, v_n$ can be in $S_i$ when the iteration that first covers $v_j$ begins?

**(c)** Using weak duality and the dual solution from part (b), show that for all $j$:

$$\text{OPT} \geq \frac{n - j + 1}{c_j}.$$

**(d)** Prove that

$$\text{GREEDY} = \sum_{j=1}^{n} \frac{1}{c_j}.$$

**(e)** Using parts (c) and (d), prove that the greedy algorithm achieves an $H_n$-approximation, where $H_n = \sum_{i=1}^{n} \frac{1}{i}$[2].

## Problem 2 : Maximum Acyclic Subgraph

In the Maximum Acyclic Subgraph problem, we are given a directed graph $G = (V, E)$ with no self-loops or repeated edges; both $(u, v)$ and $(v, u)$ may be present. A subset $E' \subseteq E$ is called acyclic if the graph $(V, E')$ contains no directed cycles. The goal is to find an acyclic subset of maximum size.

Let OPT denote the size of a maximum acyclic subset of edges.

Let $n = |V|$ and fix any ordering $v_1, \ldots, v_n$ of the vertices. An edge $(v_i, v_j)$ is forward if $i < j$ and backward if $i > j$. Let

$$E_f = \{(v_i, v_j) \in E : i < j\}, \qquad E_b = \{(v_i, v_j) \in E : i > j\}.$$

**(a)** Prove that $(V, E_f)$ and $(V, E_b)$ are both acyclic. Deduce that $\text{OPT} \geq |E|/2$.

**(b)** Use part (a) to design an algorithm that returns an acyclic subset of at least $|E|/2$ edges. Describe the algorithm and give a bound on its running time.

**(c)** Give an upper bound on OPT in terms of $|E|$. Use it to prove that the algorithm from (b) is a 2-approximation: its output contains at least $\text{OPT}/2$ edges.

[2] $H_n$ is known as the $n$-th harmonic number. One can show that $H_n \sim \ln n$, so the greedy algorithm is an $O(\ln n)$-approximation. This approximation ratio is known to be essentially optimal, unless P = NP.

## Problem 3 : Let's Play a Game

A zero-sum matrix game is played by two players using a known matrix $A \in [0,1]^{m \times n}$, where $m, n \geq 1$. Player 1 chooses a row $i$ and Player 2 chooses a column $j$, simultaneously: neither sees the other's choice before making their own. Player 1 receives $A_{ij}$ points and Player 2 receives $-A_{ij}$ points. Each player wants to maximize their own expected score. Equivalently, Player 2 wants to minimize Player 1's expected score.

A player can choose an action at random. A strategy for Player 1 is a probability vector $p = (p_1, \ldots, p_m)$, where $p_i \geq 0$ and $\sum_i p_i = 1$: row $i$ is chosen with probability $p_i$. Similarly, Player 2 uses a probability vector $q = (q_1, \ldots, q_n)$ over columns. Always choosing one action is the special case where its probability is one.

The players draw their actions independently, so the probability of choosing row $i$ and column $j$ is $p_i q_j$. Thus Player 1's expected score is

$$u(p, q) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_i A_{ij} q_j,$$

and Player 2's expected score is $-u(p, q)$.

**(a)** Consider the game

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1/2 \end{pmatrix}.$$

Suppose both players choose uniformly among their two actions. What is each player's expected score? For each player separately, find a strategy that improves their expected score if the other player keeps choosing uniformly.

**(b)** Let's elaborate on the thread of finding strategies that improve expected score against a fixed opponent strategy. Suppose Player 2 chooses a strategy where they always choose column $j$ (so $q_j = 1$ and $q_{j'} = 0$ for $j' \neq j$). Player 1's expected score is $\sum_{i=1}^{m} p_i A_{ij}$.

Suppose Player 1 doesn't know which column Player 2 will choose, but wants to guarantee the best possible expected score against any fixed column. In other words, Player 1 wants to choose a strategy $p$ that maximizes $\min_j \sum_{i=1}^{m} p_i A_{ij}$, the worst-case expected score against any fixed column. We can formulate this as an LP:

$$\begin{aligned} \text{maximize} \quad & v \\ \text{subject to} \quad & \sum_{i=1}^{m} A_{ij} p_i \geq v && \text{for every } j, \\ & \sum_{i=1}^{m} p_i = 1, \quad p_i \geq 0 && \text{for every } i. \end{aligned}$$

The variables $p_i$ are the probabilities of choosing row $i$, and $v$ is a lower bound on the worst-case expected score against any fixed column. The variable $v$ is an auxiliary variable that is unrestricted in sign (it can be negative). The first set of constraints ensures that $v \leq \min_j \sum_{i=1}^{m} A_{ij} p_i$; maximization makes this an equality at an optimum. The second set ensures that $p$ is a valid probability vector.

Now consider Player 2. We can formulate a similar LP for Player 2, who wants to minimize the

maximum expected score of Player 1 against any fixed row. The LP is:

$$\begin{aligned} \text{minimize} \quad & w \\ \text{subject to} \quad & \sum_{j=1}^{n} A_{ij} q_j \le w \qquad \text{for every } i, \\ & \sum_{j=1}^{n} q_j = 1, \quad q_j \ge 0 \quad \text{for every } j. \end{aligned}$$

The variable $w$ is unrestricted in sign.

Show that for any feasible solutions $(p, v)$ and $(q, w)$ to these LPs,

$$v \le u(p, q) \le w,$$

and hence $v \le w$.

**(c)** The LPs in (b) are in fact dual to each other[3]. You may use this fact without proving it, and you may apply strong duality to LPs with equality constraints and unrestricted variables. Use strong duality to show that there are strategies $p^*, q^*$ such that, for every pair of strategies $p, q$,

$$u(p, q^*) \le u(p^*, q^*) \le u(p^*, q).$$

Explain why these inequalities mean neither player can improve their expected score by changing strategy alone.

**(d)** Let $v^*$ be the common optimal value of the LPs in (b), and let $p^*$ be an optimal strategy for Player 1. This strategy guarantees expected score at least $v^*$ against every opponent. An opponent with predictable (potentially suboptimal) behavior might allow us to earn more.

Suppose $\widehat{q}$ is our prediction of the opponent's action distribution. Fix $\delta \ge 0$, the amount of the guarantee we are willing to give up. Formulate an LP that chooses a strategy $p$ maximizing $u(p, \widehat{q})$, subject to guaranteeing expected score at least $v^* - \delta$ against every opponent strategy. Prove that your LP is feasible and that any optimal solution $\widetilde{p}$ satisfies both

$$u(\widetilde{p}, q) \ge v^* - \delta \quad \text{for every strategy } q, \qquad u(\widetilde{p}, \widehat{q}) \ge u(p^*, \widehat{q}).$$

Thus the safety guarantee holds even if our prediction is wrong, while the predicted score is at least that of $p^*$.

[3] In lecture we only saw how to take duals of maximization LPs with $\le$ constraints and nonnegative variables, but any valid LP has a dual.

**Aligning Expectations – Using AI in This Question**

Part (e) asks you to use AI assistance for implementation and experiments. Decide what you want to implement and measure, then ask the assistant to help carry out those plans. You are responsible for checking that the code implements your methods.

(e) The goal of this question is to play against bots acting as Player 2 by providing a probability vector $p$ to use as Player 1's strategy. You will try to learn how each bot behaves by playing against it, then use that information to obtain the highest average score you can against each bot.

The course website will provide `pset6_bots.zip`, containing six payoff matrices in `.txt` format, the Python simulator `simulate.py`, and usage instructions in `README.md`. The matrices have sizes $2 \times 2$, $4 \times 4$, $6 \times 6$, $12 \times 12$, $30 \times 30$, and $100 \times 100$. Each file begins with its numbers of rows and columns, followed by the payoffs row by row.

Select a matrix with `--matrix` followed by its file path, and a bot with `--bot` followed by an integer from 1 to 6. All six bots can play on every supplied matrix. Provide one row probability per row of your chosen matrix with `--p`. The script simulates 1000 games, drawing Player 1's action from the same vector $p$ in each game, and reports only Player 1's average score. Every invocation resets the bot's state. You may run the script repeatedly with different vectors to investigate the bots; repeated scores can vary because of randomness.

**Do not read the simulator's source code, and explicitly instruct your AI assistant not to read it either.** Reading the implementation would spoil the task of learning what the bots do by playing them. State this restriction in your prompts; an AI assistant may otherwise inspect the supplied script.

First, have an AI assistant implement Player 1's LP from (b), using a Python LP library, to find an optimal strategy $p^*$ for each matrix, with the guarantee established in (c). Test these strategies against all six bots and use their scores as your baselines.

Then investigate the bots through further runs of the simulator. Form hypotheses about their behavior, choose vectors $p$ that help test those hypotheses, and use what you learn to improve your score against each bot. Use AI to help implement and test your ideas. The LP from (d) may be useful, but your strategies need not retain its safety guarantee. You may choose a different vector for each combination of matrix and bot.

Report what you think each bot does, the evidence supporting your conclusions, and your recommended vectors and average scores, compared with the corresponding baselines. Include unsuccessful attempts and explain what remains uncertain; you need not beat the baseline against every bot. Write your explanations in your own words.