CS 36/236: Approximation Algorithms (Winter 2022) Problem Set

This is a living document which will refresh frequently. Check every weekend for newer problems.

Can be done in groups of size ≤ 2.

Instructions

- *Credit Statements:* At the beginning of each problem you must write who all (including the teaching staff) you discussed this problem set with. This is ***important.*** Even if you did not talk with anyone about any of the problems, you need to mention "No one". *Without a credit statement, you may be awarded* 0 *for the problem.*
- *External Sources:* You are not allowed to consult any sources other than the notes and the text-books. Many of these problems have solutions out there on the web. Don't go hunting for them. If you stumble upon them by chance, then cite it. Uncited solutions will be an ***honor code violation.***
- *Presentation*: Your presentation should satisfy the following three C's: they should be *Correct, Clear, Concise*. Do not *ramble*. Ideally, we should be able to read and completely understand any answer to a single problem in ***less than five minutes***.
- *LaTeX*: You ***have*** to use LaTeX. Style files will be provided in Canvas. The main reason is that I will often choose the best solution among you all as official solutions.
- *Problems*: Not all problems are of the same level of difficulty. Those marked with a single ☕ should be like an exercise and quick to do, those with two ☕☕ should be do-able in a couple of hours or so, and those with three ☕☕☕ may take longer. Instructive problems which teach you a concept/algorithm are marked with ♥. I may also add some problems which I may not know the answer to myself. That doesn't mean they are difficult, rather, it means I haven't thought too much about them. I will mark them with a ☯.

Problem 1. [Dominating Set] ☕

Given an undirected graph $G = (V, E)$, a ***dominating set*** U is a subset of V such that every vertex of V is either in U or has a neighbor in U. Explain why the minimum cardinality dominating set is a set cover problem. Concisely describe the greedy algorithm for finding a minimum cardinality dominating set and state what its approximation factor is. Note: you don't have to re-do the analyses done in class.

Problem 2. [Unweighted Set Cover Analysis] ☕

Read the analysis of the *unweighted* set cover in the notes. This is the exercise following Claim 1. If k is the cardinality of the optimum set cover and I is the set cover returned by the greedy algorithm, then argue that $|I| \leq k \cdot (1 + \ln(n/k))$. Use this to argue that if every set $|S_i| \leq d$, then GREEDY SET COVER is an $(1 + \ln d)$-approximation algorithm.

Hint: The algorithm must be covering at least one element in each iteration. This may be useful...

Problem 3. ♥ [Maximum k-Coverage] ☕☕

In the MAX-COVERAGE problem, we are given m sets $S_1, \ldots, S_m$ each a subset of a universe U. We are also given an integer parameter $k \geq 1$. The objective is to pick k out of these m sets, indexed by I with $|I| = k$ such that $|\bigcup_{j \in I} S_j|$ is maximized. Describe a natural greedy algorithm for this problem and prove that is an $(1 - \frac{1}{e})$-approximation. For an extra ☕, also show that the analysis is near tight as $k \to \infty$.

Hint: Hopefully, the greedy algorithm is clear to you. Let $V \subseteq U$ be the elements covered by the optimal solution. Modify the analysis of Theorem 1. Perhaps define x_j as the number of elements of V not covered in the jth loop. You may need the inequality, $(1 + z) \leq e^z$ for all z, handy.

Hint: More hints: as in the set cover analysis, we need to relate some quantities like x_j's and n_j's. But the *definition* is important here. So, as in the previous hint, let x_j be the number of elements of V, that is the elements that the optimum solution covers. What should n_j be? Since the algorithm may completely ignore V and pick elements in $U \setminus V$, we should not define n_j as the number of new elements of V that is covered by the algo. Instead, we should define n_j to be the number of new elements covered by the algorithm in the jth step. However, one now **cannot** say $x_{j+1} = x_j - n_j$; after all, the algo might not have covered any elements from V at all, and x_{j+1} could still be x_j even when n_j is positive. So you should be careful of that.

You can still say $x_{j+1} \geq x_j - n_j$ (do you see why?) and you can still say $n_j \geq \frac{x_j}{k}$. Note again that you **cannot** combine these to say $x_{j+1} \leq x_j(1 - 1/k)$; please don't make this mistake. So how should you proceed?

Hint: argue that for any j, $n_1 + \ldots + n_j$ is large as compared to $\mathsf{opt} = x_1$. In particular, $n_1 \geq \frac{\mathsf{opt}}{k}$, $n_1 + n_2 \geq \frac{\mathsf{opt}}{k} \cdot (1 + \left(1 - \frac{1}{k}\right))$, and more generally

$$n_1 + \ldots + n_j \geq \frac{\mathsf{opt}}{k} \cdot \left(1 + \left(1 - \frac{1}{k}\right) + \left(1 - \frac{1}{k}\right)^2 + \cdots + \left(1 - \frac{1}{k}\right)^{j-1}\right)$$

Problem 4. ♥ [Maximum Colorful Coverage] ☕☕

In the MAX COLORFUL COVERAGE problem, we are given m sets $S_1, \ldots, S_m$. Each set S_j now has a color c_j which is one out of k colors $\{1, 2, \ldots, k\}$. The objective is to select k sets, each of a different color, so as to maximize the number of elements in the union. Formally, we need to find $I \subseteq m$ such that for all $j, j' \in I$, we have $c_j \neq c_{j'}$, and $\left| \bigcup_{j \in I} S_j \right|$ is maximized.

Describe a natural greedy algorithm for this problem and prove that is an $\frac{1}{2}$-approximation. For an extra ☕, also show that the analysis is near tight as $k \to \infty$.

Hint: Let A be the set of elements covered by your greedy algorithm. Suppose the optimum solution picks the sets $O_1, \ldots, O_k$ where set O_j has color c_j. When you pick the jth colored set in your algorithm, can you compare the number of *new* elements this set covers with $|O_j \setminus A|$? And then, can you complete the proof?

Problem 5. ♥ [Facility Location] ☕☕

In the FACILITY LOCATION problem, there are two sets: F, a set of facilities, and C, a set of clients. The objective is to "open" a subset of these facilities and "connect" every client to some open facility. There is a cost $f_i \geq 0$ associated with opening facility $i \in F$. For every client $j \in C$ and facility $i \in F$, there is a connection cost $d(i, j)$ in case i is opened.

First argue that the FACILITY LOCATION problem is a generalization of the Set Cover problem. Then describe a natural greedy $O(\log n)$-approximation algorithm and argue about its running time. Please note that you have to describe an *efficient* (polynomial time) algorithm. If not obvious, you will have to argue why it is polynomial time.

Hint: Try the charging argument. In the Supplementary reading, we actually describe a 2-approximation algorithm when the $d(\cdot, \cdot)$ satisfy triangle inequality. Note, in this problem, there is no such assumption. Nevertheless, reading the initial portion of those notes may give you an idea on how to solve this problem.

Problem 6. [Steiner Tree approximation factor.] ☕

In the lecture notes, we proved that MST-STEINER is a 2-approximation algorithm. Prove that, in fact, it is a $2\left(1 - \frac{1}{|R|}\right)$-approximation algorithm. That is, if T is the tree returned, then $\mathsf{cost}(T) \leq 2 \cdot \left(1 - \frac{1}{|R|}\right) \mathsf{opt}(G)$, where $\mathsf{opt}(G)$ is the minimum cost Steiner tree.

Problem 7. [Bad examples Local Search in Cut Problems] ☕

a. Describe an undirected graph with a locally optimal cut whose value is $= \frac{\mathsf{opt}}{2}$.

b. Describe a directed graph where DIRECTED MAX-CUT LS's solution has value $= \frac{\mathsf{opt}}{3}$.

You are allowed to have parallel edges.

Problem 8. ♥ [Local Search and Submodular Functions.]☕☕☕

A set function f defined over a universe U defines a value $f(S)$ on every subset $S \subseteq U$. A set function is **submodular** if it satisfies the following:

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

Such functions arise in many settings. For instance, if $G = (V, E)$ is a directed graph with non-negative weights $w(e)$ on edges, then the function $f(S) := \sum_{e \in \partial^+ S} w(e)$ is a submodular function. In the *submodular function maximization* problem, you have to find a subset $S \subseteq U$ which maximizes $f(S)$. We assume that f is non-negative, that is, $f(A) \geq 0$ for all $A \subseteq U$.

A set S is a *local maximum* if $f(S) \geq f(S \cup e)$ for all $e \notin S$, and $f(S) \geq f(S \setminus x)$ for all $x \in S$. In this exercise, you will prove that a locally maximum set S satisfies $\max\left(f(S), f(\overline{S})\right) \geq \frac{\mathsf{opt}}{3}$, where $\mathsf{opt} = \max_{A \subseteq U} f(A)$ and $\overline{S} := U \setminus S$. This generalizes the analysis of DIRECTED MAX-CUT LS. Let O be the set with $f(O) = \mathsf{opt}$.

a. Use the local maximality of S and submodularity of f to prove that $f(S) \geq f(O \cup S)$.

> **Hint**: Submodularity implies that for any set A, a superset B, and $e \notin B$, one has $f(A \cup e) - f(A) \geq f(B \cup e) - f(B)$. The quantity $f(A \cup e) - f(A)$ is e's marginal on A, and the above says submodular functions have "decreasing marginality". Local maximality of S implies any items marginal is non-positive. Now can you finish the proof.

b. Use above and submodularity to claim $f(S) + f(\overline{S}) \geq f(O \cap \overline{S})$.

> **Hint**: Apply the definition of submodularity on two particular sets A and B. You will also need to use non-negativity of f.

c. Use the local maximality of S and submodularity of f to prove that $f(S) \geq f(O \cap S)$.

> **Hint**: Local maximality implies that for any item $x \in S$, the marginal of x on $S \setminus x$ is non-negative. Submodularity implies that x's marginal should be non-negative for any subset of $S \setminus x$ as well. Now can you finish the proof?

d. Again using submodularity, and the above three parts, prove that $3 \max\left(f(S), f(\overline{S})\right) \geq f(O)$.

> **Hint**: This should now be easy by adding the above three...

Problem 9. ◕[Local Search for k-Center] ☕☕

In the k-center problem, one is given n points X and a distance $d(\cdot, \cdot)$ satisfying triangle inequality. That is, for any three points x, y, z, we have $d(x, z) \leq d(x, y) + d(y, z)$. For technical reasons, assume that

each $d(x,y)$ is a distinct number. The objective is to select k points $C \subseteq X$ such that the distance of any point to a point in C is minimized. Formally, we want to minimize

$$\mathsf{cost}(C) := \max_{x \in X} d(x,C) \quad \text{where} \quad d(x,C) := \min_{y \in C} d(x,y)$$

Consider the following local search algorithm for the problem : start with an arbitrary k-points C. If there exists a *swap* (c,x) for $c \in C$ and $x \notin C$ such that $\mathsf{cost}(C - c + x) < \mathsf{cost}(C)$, then swap and continue. Let A be the final k-points obtained that way. Comment on the approximation factor of this local search algo.

Hint: I haven't thought too much, but it seemed to be if the initial k points chosen are "close-by" the local optimum solution may be bad. But I am not 100% sure.

Problem 10. ♥ [Farthest Center Algorithm for k-Center] ☕☕

Consider the following "greedy" algorithm for k-center : initialize A with an *arbitrary* point $p_1 \in X$. Then, in $k-1$ subsequent iterations, pick the point $p \in X$ which *maximizes* $d(p,A)$ and add it to A. At the end, A has k points. Prove that $\mathsf{cost}(A) \leq 2\mathsf{opt}$.

Hint: Let O be the optimal k-center solution. For an $o \in O$ let $N_o := \{p \in X : d(o,p) = d(p,O)\}$ be the points in X nearest to o. First, prove that if $|A \cap N_o| = 1$ for all $o \in O$ then $\mathsf{cost}(A) \leq 2\mathsf{opt}$. Next prove that if at any time of the algorithm $|A \cap N_o| \geq 2$, then at that point $d(p,A) \leq 2\mathsf{opt}$ for all $p \in X$. Then complete the argument.

Problem 11 (Local Search for Max 3SAT.)**.** ☕☕

Consider the generalizations of the local search algorithms for Max 2SAT for Max 3SAT. In Max 3SAT, we are given a 3SAT formula where each clause has 3 literals. The objective is to find an assignment maximizing the number of clauses satisfied. First consider the (obvious) local search algorithm which starts with an arbitrary assignment, and negates a variable if it leads to an increase in the number of clauses satisfied. What is the approximation factor of this algorithm? For an extra ☕, suggest a non-oblivious local search algorithm for the same and analyze.

Hint: As in Max 2SAT, given an assignment $\mathbf{x}$ and variable x_i divide the clauses containing this variable into classes : those with all three variables false, those with all three true, etc. The only thing that changes in the analysis is the "counting argument"; the number of such sets a single clause may appear changes. To get the non-oblivious algorithm, I suggest giving "different coefficients" and then use the analysis to figure these coefficients out.

Problem 12. ♥[Greedy List Scheduling.] ☕☕

Consider the GREEDY LIST SCHEDULING algorithm done in class where the jobs are considered in decreasing order of processing times. Suppose there is a schedule which assigns at most two jobs in each machine and has makespan $\leq L$ and *every* individual job has processing time $> L/3$. Then prove the greedy list scheduling algorithm's makespan is $\leq L$.

Hint: Consider the schedule σ with at most two jobs per machine. For every machine i, consider the (at most) two jobs assigned to it, and rename the job with the larger processing time as i. Rename the other job to i', and $i' = \perp$ if there was only one job allocated to the machine i. Furthermore, rename the machines (and thus also the jobs again) such that $p_1 \geq p_2 \geq \ldots \geq p_m$. Note by definition, for any i, we have $p_i + p_{i'} \leq L$.

Now prove that if $i < j$, and thus $p_i \geq p_j$, show that if $p_{i'} > p_{j'}$, then one can swap i' and j' and still have an assignment with makespan $\leq L$. So, we may assume $i < j$ implies $p_i \geq p_j$ and $p_{i'} \leq p_{j'}$. What is the schedule that GREEDY LIST SCHEDULING leads to?

Problem 13 (Makespan Minimization with Few Types of Jobs.)**.** ☕☕

Consider the makespan minimization problem on identical machines. Suppose there are only t kinds of jobs where the sth kind of jobs B_s have processing times q_s, for $1 \leq s \leq t$. Furthermore, suppose you are given a guess opt. Describe a $n^{O(t)}$ time algorithm to decide if all jobs can be allocated to m machines (assume $m \leq n$) such that the load on every machine is $\leq \mathsf{opt}$.

For this problem, clearly *define* the function (which will be stored in a table), the base cases, which value of the function you are interested, and most importantly, the recursive formula. Also, write the precise running time (as in, the exponent of n).

Problem 14. ♥(Knapsack Problem) ☕☕☕

In the knapsack problem we are given n items and each item j, for $1 \leq j \leq n$, has a *profit* p_j and a *weight* w_j. We are also given a knapsack of total capacity B. The objective is to select a subset S of the items such that (a) their total weight is at most B, and (b) their total profit is maximized. If you remember your undergraduate algorithms, this problem can be solved in $O(nB)$ time, which is **not** a polynomial time algorithm (since B can be exponentially large). Indeed, the knapsack problem is NP-hard. In the following, we use opt to denote the profit of the maximum profit subset that fits in the knapsack, and assume you know this value.

a. Consider the following *greedy* algorithm: order items in decreasing bang-per-buck order, that is, $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}$. Keep selecting items in this order till the knapsack overflows, and then throw away the last item picked. Prove that if $p_j \leq \varepsilon\mathsf{opt}$ for all j, then the above algorithm returns a solution with profit $\geq (1-\varepsilon)\mathsf{opt}$.

   Hint: Show that the algorithm's profit *before* the last item is thrown is at least opt. This should use the greedy property, but make sure you write a complete proof for this. The proof is perhaps easier if you assume the items picked by the optimum algorithm is disjoint from those picked by the greedy algorithm. And what if they aren't? Write this one well.

   Now use the assumption given in the problem to complete the proof.

b. Using part (a), design a PTAS for the knapsack problem. For any constant $\varepsilon > 0$, your algorithm's running time should be upper bounded by a polynomial in n and $\log B$.

Hint: Guess the items with $p_j > \varepsilon \mathsf{opt}$? After guessing, what should you do? And how will the full analysis proceed?

Problem 15. ♥[Scheduling with Precedence Constraints]☕☕☕

In this problem, we have n jobs and each job has $p_j = 1$. We also have m machines, and for what it's worth, think of $m = 3$. We are also given a DAG (directed acyclic graph) on the n jobs with the following semantic : if (j, k) is an edge, then job k **cannot be begun** till job j has finished processing. That is, job j is a *pre-req* for job k. The goal is to schedule the jobs on machines satisfying these precedence constraints, so as to minimize the makespan, that is, the time at which the last job finishes processing.

Note that unlike the problem done in class, in this problem you need to specify for each job the machine and the time t at which it begins, and this job would end at time $t + 1$ (since $p_j = 1$). It may so occur that there are "gaps" in a machine, that is, machine 1 runs job j then sits idle for 3 units, and then runs k. This may occur, for instance, if k has 4 prereqs which have been assigned contiguously on a second machine.

Consider the following algorithm. Take the jobs in a topological order of the DAG (where the first vertex is a source, with indegree 0) and schedule them the earliest you can given the schedules of the earlier jobs. Prove that this is a 2-approximation.

Remark: *In fact, the above gives a* $2-\frac{1}{m}$*-approximation. So, on* 3 *machines, it gives* $5/3$*-approximation. I don't think anything better is known for* 3 *machines. On the other hand, this problem is one of the very few problems which we don't know to be NP-complete.* **You** *can perhaps find a polynomial time algorithm for the* 3 *machine case.*

Hint: First, what can you say about opt and the *longest path* in the DAG?

Suppose the algorithm takes t time to finish. Divide these t time slots into two categories : a *busy* slot defined as one in which *every* machine is processing some job, and *underutilized* slots where at least one machine is not processing anyone.

Can you upper bound the number of busy slots by opt? (This should be something similar to what was done in class)?

Can you upper bound the number of underutilized slots as well by opt? To do this, can you use these underutilized slots to get a path in the DAG?

Problem 16 (Improving the Integrality Gap of Vertex Cover.)**.** ☕☕

Consider the vertex cover problem when all costs $c_v = 1$and the graph is d-regular, that is, all vertices have degree exactly d. Consider the normal LP-relaxation for vertex cover (as in the notes). Prove that one can find a vertex cover with $\leq 2\left(1 - \frac{1}{d+1}\right) \cdot \mathsf{lp}$ vertices.

Hint: Prove that $\mathsf{lp} \geq \frac{n}{2}$, and one can always find a vertex cover with at most $n \cdot \left(1 - \frac{1}{d+1}\right)$ vertices. For the former, you will need to use that all degrees are the same. For the latter, recall that a vertex cover's complement is an independent set. Think of the naivest algorithm to pick an independent set — that may just work.

Problem 17. ♥[Strengthening doesn't help Vertex Cover.] ☕☕☕

In this exercise, we describe an example which shows that even upon adding the inequalities of the form $x_u + x_v + x_w \geq 2$ for all triangles, the integrality gap of the vertex cover LP remains ≥ 1.99. Indeed, the idea is to come up with a graph which has no triangles and yet a large vertex cover. The idea is to use a random graph.

Imagine a graph on n vertices. For every pair of vertices (u, v) include this edge with probability p.

a. As a function of n and p, what is the expected number of triangles in this graph?

b. Pick a subset $S \subseteq V$ of size k. What is the probability that this set is independent? Write your answer as a function of n,p, and k.

c. Using union bound, as a function of k, n, p, what is the *upper bound* on the probability that there exists **any** independent set of size k?

d. Henceforth, set $p = n^{-0.9}$. Use part (c) to conclude the probability there is an independent set of size $n^{0.95}$ is $\leq \frac{1}{3}$ when n is large enough. (This is a much weaker statement that what you can show).

e. Use part (a) to conclude that the probability there are $> n^{0.3}$ triangles is at most $\frac{1}{3}$.

f. Remove all edges incident on these triangles thereby getting a triangle free graph. Use part (d) and part (e) to conclude that with probability $> 1/6$, you have obtained a triangle-free graph whose largest independent set is $\leq n^{0.95} + 3n^{0.3}$.

> **Hint**: The latter quantity is $\leq 0.005n$ for large enough n, and therefore, this graph has vertex cover $\geq 0.995n$ since complement of IS is VC.

g. Conclude that the integrality gap of the strengthened LP is ≥ 1.99.

Problem 18. ♥ [Set Cover LP Relaxation and Integrality Gap.]☕☕

Recall the set cover problem done in class. Write an integer programming formulation which covers the problem exactly, and obtain the LP relaxation by relaxing the integrality constraints. In the remainder of the exercise, you need to show that the integrality gap of this LP is $\Omega(\log n)$. In particular, this LP can't be used to give a better than $O(\log n)$-approximation.

Here is the set system you should use. Fix an integer $d \geq 2$. The universe U is the set of all $\{0,1\}^d$ bit-strings *except* the all zeros string. Thus, $|U| = 2^d - 1$. Thus, if we use n to denote $|U|$, we see $d \geq \log_2 n$. We now describe 2^d sets $S_{\mathbf{v}}$ for every $\mathbf{v} \in \{0,1\}^d$. The set

$$S_{\mathbf{v}} := \{\mathbf{u} \in U \ : \ \mathbf{u}^{\mathsf{T}}\mathbf{v} \text{ is an odd number}\}$$

It may help you to explicitly "draw" this set for $d = 2$ and 3. Each set's cost is 1. The following two parts prove that the integrality gap is $\Omega(\log n)$.

a. Describe an LP solution with cost = 2 thereby proving $\mathsf{lp} \leq 2$.

Hint: How many sets is an element $\mathbf{u}$ in?

b. Prove that any set cover must pick at least d sets.

Hint: Think of the sets this way : $S_{\mathbf{v}}$ contains all elements $\mathbf{x}$ such that $\sum_{i=1}^{d} \mathbf{v}_i \mathbf{x}_i \equiv 1 \bmod 2$. Or perhaps more illuminatingly, it **doesn't** cover the elements $\mathbf{x}$ such that $\sum_{i=1}^{d} \mathbf{v}_i \mathbf{x}_i \equiv 0 \bmod 2$.

For an extra ☕ describe a valid inequality which will "kill" this above cheating ☯.

Problem 19. ♥ 🖱 [Set Cover LP Relaxation and Coding Assignment.]☕☕

Recall the set cover problem done in class. Write an integer programming formulation which covers the problem exactly, and obtain the LP relaxation by relaxing the integrality constraints. The remainder of the exercise is a coding assignment which should done in a colab notebook. For this, you need to familiarize yourself with the `linprog` module in `scipy.optimize`. This is well worth everyone's time.

a. Look at the set system described in the above problem. Experimentally verify part (a) for $d = 5$ say. What's the largest d your code can handle before running out of memory?

b. For, say $d = 4$, can you see a valid inequality that you can add which will make the lp value > 2? ☯

Problem 20. ♥ 🖱 [A Planted Set Cover Experiment] ☕☕☕☯

After you have warmed your hands with the above, let me come to an experiment that I haven't done myself but I am quite curious as to what happens. For this, we have a parameter n which is going to be a large integer, say 1000 for now. Let d be a small positive integer, say 2 or 3.

Construct the following *random* set-system. The sets S_1 to S_{n-d} each contain a *random* subset of the universe of size $\lceil n/d \rceil$. The sets S_{n-d+1} to S_n are formed as follows: pick a *random permutation* σ of the n elements. The first $\lceil n/d \rceil$ elements of this permutation go to S_{n-d+1}, the second $\lceil n/d \rceil$ elements go to S_{n-d+2}, and so on. The last set may be slightly smaller. After you have done so, *scramble* the names of the sets. You can choose to remember this scrambling, but the codes below should not.

By design, `opt` $= d$; in particular, we have *planted* a good solution (signal) in random noise.

a. Implement the greedy algorithm done in class. What is the value the greedy algorithm gives?

b. What is the value of the LP relaxation. It should be $\leq d$, and does it actually find the planted set.

All of the above should be repeated multiple times and averaged-and-error-barred before making a confident answer to the above questions. Your submission for this should be a CoLab notebook where you also introduce text to explain various choices, etc.

Problem 21. 🖱 [Strengthening Vertex Cover] ☕☕☕☯

This is an "experimental evaluation" for the strengthening inequalities for vertex cover that we saw in class/notes. For this, we are going to solve the LP on random graphs $G(n, p)$. You may use $n = 50, 100,$

and the values $p = 0.5, 0.1$. Repeat each experiment decent number of times to increase confidence in your conclusions.

First construct the random graph $G(n, p)$ where every pair (x, y) is an edge with probability p. On this graph, first solve the natural LP relaxation (all costs are 1). Use the LP solution to get a vertex cover. Note that you don't have to use the algorithm done in class to the word. You could, for instance, perform a greedy rule as was suggested in class. Whatever you do though, explain in the colab notebook. Note down your alg/lp.

Next, try to find triangles for which $x_u + x_v + x_w < 2$, and as soon as you find one, add it to the LP relaxation, and run your rounding algorithm again. Rinse and repeat. I would like to know how much does it help for the different values of n and p.

Problem 22. ♥ [Integrality Gap Example for GAP] ☕☕

For any constant $c < 2$, come up with an instance of GAP for which the LP studied in class has an integrality gap of $\geq c$. More precisely,

a. Clearly describe n jobs, m machines, the p_{ij}'s and w_{ij}'s for jobs j and machines i, and the machine loads B_i. The parameters can (and probably should) depend on c.

b. Describe a *feasible* LP solution x_{ij} with objective value lp. Recall, if $p_{ij} > B_i$ on any machine, then you better have $x_{ij} = 0$.

c. Argue why any integral solution's value is at most lp/c.

For partial credit, show the above for *any* $c > 1$.

Hint: Since there are so many parameters, let me tell you that an answer to the above question exists even when all $p_{ij} \in \{p_j, \infty\}$, that is, every job j has an intrinsic processing time p_j, but it can't be allocated to all machines. Let me also tell you that a gap exists even when $w_{ij} = 1$, that is, if the solution's value is just the *number* of jobs assigned. Final hint: the LP value is n, that is, all jobs are assigned.

Problem 23. ♥ [Makespan Minimization for Unrelated Machines.] ☕☕☕

Consider the "dual" problem of GAP. n jobs, m machines, every job j has a processing time p_{ij} on machine i. The objective now is to assign ***all*** jobs to the machines, so as to minimize the maximum load on any machine. When $p_{ij} = p_j$ for all i, then this we saw a PTAS for this problem. You need to give a complete description of a 2-approximation for this problem.

This exercise builds on the remark given in the lecture notes on GAP. However, there are certain steps missing in the remark. For instance, GAP doesn't need to allocate all jobs, in GAP the B_i's are input, there is nothing like this here, etc. You don't need to *reprove* all the things done in the notes; but you will have to state what theorems you use (they will be slightly different), and then state why the proofs in the lecture notes imply proofs of your theorem.

This is not a difficult problem if you've understood GAP; it just may be a bit long.

Remark: *Unlike GAP (for which we know a better than $1/2$-approximation), nothing better than a 2-approximation is known for this problem. This is true even when all $p_{ij} \in \{p_j, \infty\}$.*

Problem 24 (DeepC's mistake in GAP)**.** ☕☕

Recall the GAP algorithm from the lecture notes. The idea is to take a GAP instance and LP solution and move to a bipartite graph. Show that if jobs are taken in *increasing order* of p_{ij}'s, and then follows the same procedure as described in class, then the "tentative assignment" σ' can lead to a load of $> B_i + \Delta_i$ (and indeed $\approx B_i + 2\Delta_i$).

Indeed, you only need 1 machine for this, and can use any feasible fractional solution x to the GAP-LP. After you describe your example, also comment on what the tentative assignment is if we take jobs in the correct decreasing order.

Problem 25 (Finding a long path in an Hamiltonian Graph)**.** ☕☕

$G = (V, E)$ is a Hamiltonian graph if there exists a path (without repeating edges or vertices) from a vertex s to a vertex t which contains all the vertices of V. Even if you knew a graph is Hamiltonian, finding a long path in it is not trivial. Describe a randomized algorithm which finds a path of length at least $\geq \frac{C \log n}{\log \log n}$ for some constant C.

Hint: Finding the longest path in a DAG is easy (right?). Now consider a random ordering σ of the vertices of G, and for every undirected edge (u, v) replace it with a directed edge pointing from a lower σ value to a higher one. You get a directed graph D. Argue that D is a DAG. What's the expected length of the longest path in this DAG D.

Problem 26 (Randomized Rounding for Maximum Coverage)**.** ☕☕☕

In this exercise, we look at randomized rounding for the max coverage problem. Recall, in this problem we are given a parameter k, a set family $(U, (S_1, \ldots, S_m))$, and the goal is to pick k subsets so that the maximum number of elements in U are covered. Write an LP relaxation for this problem. Using this, design a randomized rounding algorithm which picks *exactly* k sets and covers $\geq (1 - \frac{1}{e}) \cdot \mathsf{lp}$ many elements?

You may find the following inequality handy: for any $0 \leq t \leq 1$, $1 - e^{-t} \geq (1 - \frac{1}{e})t$. This itself follows from the "concavity" of the function $1 - e^{-z}$.

Hint: For the LP relaxation you should have two classes of variables : one class which indicates whether a set is picked, and one class which indicates whether an element is covered. How would these variables be related?

Next, note that the algorithm cannot pick more than k-sets; and thus the "usual" independent rounding may be problematic. Rather, the hint is proceed in k rounds, and in each round pick exactly one set. An idea similar to this was indeed proposed by a student in class when we looked at Set Cover.

Problem 27. ♥ [Randomized Rounding and Set Packing]☕☕☕

In this problem we are given m sets $S_1, S_2, \ldots, S_m$ which are subsets of some universe U. Each set S_j has a weight w_j, which is a positive rational number. We are also told that each set S_j has at most D elements. The objective is to pick a collection of *pairwise disjoint sets* with maximum weight. Note that if U is the set of edges of a graph and each set S_j corresponds to the subset of edges incident on a vertex j, then the problem is precisely the maximum weight independent set problem that we did in class, and D is the maximum degree of a vertex.

a. Write an LP relaxation for this problem. This should, by now, be simple.

b. After solving the LP, you have a variable x_j for each set S_j. Sample each set independently with probability $\alpha \cdot x_j$, where α is a parameter you will set later. Let this collection of sets picked by $\mathcal{S}$. What is $\mathbf{Exp}[w(\mathcal{S})]$?

The collection $\mathcal{S}$ may not be pairwise disjoint. To fix this, in stage two we delete from $\mathcal{S}$ any set S_j if there exists a set $S_k \in \mathcal{S}$, $k \neq j$, such that $S_k \cap S_j \neq \emptyset$. The next set of calculations now puts a *lower bound* on the probability S_j appears in $\mathcal{S}$ and is not deleted in stage two.

c. Fix a set S_j. Suppose its elements are $\{1, 2, \ldots, d\}$ for $d \leq D$. Now fix an element $i \in S_j$. Conditioned on the event that $S_j \in \mathcal{S}$, what is the probability that there exists $S_k \in \mathcal{S}$, $k \neq j$, such that $i \in S_k$? Your answer should use the LP constraint here.

d. Using union bound, upper bound the probability:

$$\mathbf{Pr}[\exists S_k \in \mathcal{S}, k \neq j, \text{ such that } S_k \cap S_j \neq \emptyset \mid S_j \in \mathcal{S}]$$

Your upper bound should be in terms of α and D.

> **Hint**: The conditioning is a red herring : the sets are sampled independently. The conditioning is present to make the next part almost immediate.

e. For the fixed set S_j, upper bound

$$\mathbf{Pr}[S_j \in \mathcal{S} \textbf{ and } \exists S_k \in \mathcal{S}, k \neq j, \text{ such that } S_k \cap S_j \neq \emptyset]$$

Your upper bound should be in terms of α, x_j and D.

f. Using the above, lower bound

$$\mathbf{Pr}[S_j \in \mathcal{S} \textbf{ and } S_j \text{ isn't deleted in stage two.}]$$

Your lower bound bound should be in terms of α, x_j and D.

g. Put a lower bound on the expected weight of the sets picked after stage two. Your answer should in terms of α, D, and lp. Now pick the best α, and argue that the above algorithm is a $4D$-approximation.

Problem 28 (Vertex Cover in k-partite k-uniform hypergraphs)**.** ☕☕

In a k-partite k-uniform hypergraph $H = (V, E)$, the vertex set V is partitioned into k sets $V_1 \sqcup V_2 \sqcup \cdots \sqcup V_k$, and each hyperedge $e \in E$ has $|e \cap V_i| = 1$ for all $1 \leq i \leq k$. Every vertex v has a cost $c(v)$. Describe a $k/2$-approximation for finding the minimum cost vertex cover.

Hint: If you followed the lecture on vertex cover in bipartite graphs and tri-partite 3-uniform hypergraphs, you would realize we need to find a "good" distribution of $(r_1, \ldots, r_k)$ such that they sum to 1 with probability 1, and each individual r_i is uniform in $[0, \frac{2}{k}]$. I claim that this is actually almost immediate from the fact that we have done $k = 2, 3$. In particular, consider the case when $k = 4$; do you see a solution immediately from the $k = 2$ case?

Problem 29. ♥ [Multi-Set-Multi-Cover] ☕☕

Consider the multi-set-multi-cover problem where the input is same as the set cover problem, but now every element i has a positive integer demand $d(i)$ as to how many times it needs to be covered. More precisely, the algorithm can choose a set S_j multiple times, but if it chooses it k_j times then it pays cost $k_j c(S_j)$. For every element i, we should have $\sum_{j:i \in S_j} k_j \geq d(i)$. That is, the number of picked sets in which i appears is at least $d(i)$. Describe an LP relaxation and an $O(\log n)$ randomized rounding algorithm.

Hint: Note that the variable you use in your LP may not be between 0 and 1 since you are allowed to pick a set multiple times. Nevertheless, you can break any number into its floor (largest integer at most that number) and the remaining fractional part. Use that fractional part to do the rounding?

Use the Chernoff bound here to make life way easier.

Problem 30 (Practice with taking duals)**.** ☕

By now we all have seen the LP relaxation for the set cover problem many times. We have a variable x_j for every set S_j. In class, we looked at the LP relaxation and instead of having $0 \leq x_j \leq 1$, we only looked at the LP with $x_j \geq 0$. In this exercise, you need to write the dual when $x_j \leq 1$ is also in the primal. Note that the dual LP *will* look different, since it will have variables corresponding to these constraints.

This exercise is just to give you practice in writing duals.

Problem 31. ♥ [More practice with taking duals]☕☕

Consider the *maximum flow* problem in a graph. If you recall, we are given a *directed* graph $G = (V, E)$, and two special vertices s and t. Every edge $e \in E$ has a capacity $u(e) \geq 0$. The goal is to find a *flow*, that is $f(e)$ for every edge such that (a) for every vertex $v \notin \{s, t\}$, the total flow "coming into" v equals the total flow "going out" of v, and (b) the flow on any edge $f(e)$ is at most the capacity $u(e)$. The value of the flow is the total flow "going out" of s.

- Write the maximum flow problem as a *maximization* LP. Remember there will be two types of constraints corresponding to (a) and (b).
- Write the *dual* of the above LP. Note that this dual will have two types of variables corresponding to the two types of constraints. Also be wary which dual variables have non-negative constraints, and which are free.

Once again, this exercise is just to give you practice in writing duals. However, once you have written the dual, you should try and interpret it and connect it to the s, t-cut problem. You don't have to submit this.

Hint: I would recommend looking at the notes for how duals look like with equality constraints. And then deep breaths. Also, office hours may help.

Problem 32. ♥ [Vertex Cover with Penalties on Edges] ☕☕

This is a slight twist on the vertex cover problem. In the usual vertex cover problem, we are given a graph $G = (V, E)$ with costs $c(v)$ on vertices and the objective is to select a minimum cost subset of vertices such that every edge has at least one end point in S. In this problem, every edge has a *penalty* π_e for not being covered. So, the objective is to select a subset of vertices $S \subseteq V$. The cost of this solution is $\mathsf{cost}(S) := \sum_{v \in S} c(v) + \sum_{e : e \cap S = \emptyset} \pi_e$. That is, it is the total cost of the vertices plus the penalities on the edges which are not covered. The goal is to find the set S with the smallest cost.

- Write an LP relaxation for the problem.
- Write the dual.
- Describe a primal-dual approximation algorithm, and prove the approximation factor is 3.

For an extra ☕, describe a primal-dual approximation algorithm with approximation factor 2. (This is **not** much harder; one extra observation...one extra shot of espresso).

Hint: Add a variable z_e indicating whether e pays penalty or not. The objective of the primal LP should have an extra $\sum_e \pi_e z_e$ term. What would the constraint of the primal LP look like? Say $e = (u, v)$. What's the relation between x_u, x_v and z_e?

Write the dual mechanically. It'll still have variables corresponding to every edge.

To get a factor 3, one can really mimic the procedure for primal-dual vertex cover except there will be another condition when "dual variables" are stopped from rising. Try to understand what that means in the primal (the corresponding primal-variable should be picked?).

To get a factor 2, one looks at the wastefulness of the above algo, and then proceeds to say, "hey, why am i paying penalty on edges that i have already covered".