EECS 376: Foundations of Computer Science **Due 8:00pm, April 3**
University of Michigan, Winter 2024 **(accepted until 9:59 pm, no credit after)**

Homework 9

This homework has 8 questions, for a total of 100 points and 5 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't turn in the previous homework, then (1) state that you didn't turn it in, and (2) pick a problem that you think is particularly challenging from the previous homework, and explain the answer in your own words. You may reference the answer key, but your answer should be in your own words.

Solution:

(12 pts) 2. **Undecidable vs. NP-hard vs. NP-complete.**

Show that the undecidable language L_{ACC} is NP-hard, but is not NP-complete.

You may use the fact that $\mathsf{NP} \subseteq \mathsf{EXP}$ without proof. (This was the result of the extra-credit problem from HW7. Recall that EXP is the class of all languages that are decidable in exponential time, i.e., in time $O(2^{n^k})$ for some constant k where n is the input size.)

Solution:

(16 pts) 3. **Search vs. decision: any Hamiltonian path.**

Define the language

$$\textsc{AnyHamPath} = \{G : G \text{ is an undirected graph with a Hamiltonian path}\}\ .$$

Suppose that there exists an efficient algorithm D that decides AnyHamPath. Give an efficient algorithm that, on input an undirected graph G, outputs a Hamiltonian path in G if one exists, otherwise it outputs "No Hamiltonian path exists." Prove that your algorithm is correct and runs in polynomial time.

Solution:

(20 pts) 4. **The fire-station problem.**

The state government is building fire stations in a region of Northern Michigan that has many small towns connected by roads. Since it's expensive to install a fire station in every town, the government has decided that it wants to build as few fire stations as possible, so that each town either has a fire station, or is directly connected by a road to a neighboring town that has a fire station.

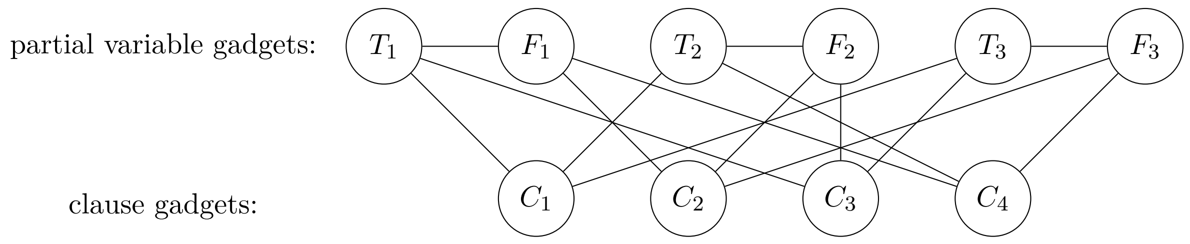
Formally, the decision problem FireStation is defined as follows: given an undirected graph $G = (V, E)$ and a positive integer "budget" k, does there exist a subset $S \subseteq V$ of at most k vertices such that for every $v \in V$, either $v \in S$ or there is an edge $(u, v) \in E$ such that $u \in S$? (Make sure to understand how this problem differs from the similar-looking VertexCover problem!) In this problem you will show that FireStation is NP-hard, by proving that 3SAT $\leq_p$ FireStation.

Since FireStation has some resemblance to VertexCover, a good source of inspiration is the 3SAT-to-VertexCover reduction from lecture. To start, we modify the reduction by making each "clause gadget" a *single vertex* (rather than a triangle of three vertices). To compensate for the simpler clause gadgets, the "variable gadgets" will need to be a little more complex, in a way that you will determine. In addition, the "budget" k will need to be different.

As an *incomplete* example of the modified reduction, consider the Boolean formula

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)\ .$$

The reduction starts by building the following *partial* graph; you will define appropriate other nodes and edges for the variable gadgets:



Show the following two things:

1. that FireStation is NP-hard, by proving that 3SAT $\leq_p$ FireStation;
2. a diagram depicting the full output of your reduction for the formula φ given above.

As always, your proof will need to include all of the steps outlined in Handout 3: NP-Hardness Proofs.

Solution:

5. **Approximate knapsack.**

Recall the 0-1 *knapsack* problem, in which we are given a set of items having weights and values, and wish to select a subset of the items so that their total weight does not exceed a specified capacity, and their total value is maximized.

More specifically, an instance is a vector of weights $W = (W_1, \ldots, W_n)$, a vector of values $V = (V_1, \ldots, V_n)$, and a knapsack capacity C, all of which are non-negative integers. (Without loss of generality, each $W_i \leq C$, because otherwise the ith item cannot be selected, so it is irrelevant.) The desired output is a subset $I \subseteq \{1, \ldots, n\}$ of the items that maximizes the total value $\sum_{i \in I} V_i$, subject to the constraint that the total weight $\sum_{i \in I} W_i \leq C$.

Recall from lecture that the Combined-Greedy algorithm for this problem is defined as follows:

1. Run the Single-Greedy algorithm,
2. Run the Relatively-Greedy algorithm,
3. Select one of the outputs that has the most total value (breaking a tie arbitrarily).

(Refer to the lecture for the definitions of the two sub-algorithms.)

In this question, you will prove that the Combined-Greedy algorithm is a 1/2-approximation algorithm for the 0-1 knapsack problem.

(8 pts) (a) Recall from lecture that one way to prove the correctness of an approximation algorithm for a maximization problem is to show a lower bound on ALG, the value obtained by the algorithm, and show a related upper bound on OPT, the value of an optimal fractional solution. In this part, you will show an upper bound on OPT.

To do this, it will help to recall from Homework 5 the *fractional* knapsack problem, a variant of the 0-1 problem that allows for taking any *partial* amount of an item. That is, for an item of weight W and value V, we may select some fractional amount $t \in [0, 1]$, which has weight $t \cdot W$ and value $t \cdot V$. Recall that the optimal value for 0-1 knapsack is at most the optimal value for the fractional version (on the same knapsack instance).

Prove that the optimal value OPT for the 0-1 knapsack problem is at most the *sum* of the values obtained by the two sub-algorithms.

Hint: It may help to recall from Homework 5 the greedy algorithm for the fractional knapsack problem, which is closely related to Relatively-Greedy (make sure to understand the difference!). You showed that this "Fractional-Greedy" algorithm obtains an optimal solution for fractional knapsack. You can use that fact here without repeating the proof.

Solution:

(6 pts) (b) Using the previous part, and by establishing a suitable lower bound on the value ALG obtained by Combined-Greedy, show that it is a 1/2-approximation algorithm for the 0-1 knapsack problem.

Solution:

6. **Approximate f-SetCover.**

In this question, you will consider a variant of the SetCover problem where each element of the universe is in a limited number of subsets. Formally, in the f-SetCover problem, we are given a "universe" (set) U and subsets $S_1, \dots, S_n \subseteq U$ *where each universe element appears in at most f of the subsets.* The goal is to find a smallest collection of the subsets that "covers" U, i.e., an $I \subseteq \{1, \dots, n\}$ of minimum size such that $\bigcup_{i \in I} S_i = U$. We assume that $\bigcup_{i=1}^{n} S_i = U$, otherwise no solution exists.

You will analyze an approximation algorithm "f-cover" for the f-SetCover problem. The algorithm is a generalization of the "double cover" algorithm for VertexCover from lecture, and it works essentially as follows: while there is some uncovered element u in the universe, add to the cover *all* the subsets to which u belongs. The formal pseudocode is as follows. The notation $I(u) = \{i : u \in S_i\}$ for $u \in U$, that is, $I(u)$ indicates the subsets to which u belongs.

```
function f-COVER($U, S_1, \dots, S_n$)
    $I = C = \emptyset$                              ▷ selected indices $I$, covered elements $C$
    while $C \neq U$ do                              ▷ not all elements are covered
        choose an arbitrary $u \in U \setminus C$                ▷ element $u$ is not yet covered
        $I = I \cup I(u)$, $C = C \cup \bigcup_{i \in I(u)} S_i$    ▷ add all subsets $S_i$ containing $u$ to the cover
    return $I$
```

Fix some arbitrary f-SetCover instance, and let I^* denote an optimal set cover for it. Let E denote the set of elements u chosen in Step 4 during an execution of the algorithm, and let I denote the algorithm's final output.

(7 pts) (a) Prove that $I(u) \cap I(u') = \emptyset$ for every distinct $u, u' \in E$. In other words, prove that if u and u' are each selected as the uncovered element in different iterations, none of the S_i contains both u and u'.

Solution:

(7 pts) (b) We want a lower bound on OPT $= |I^*|$. Using the previous part, prove that $|E| \leq |I^*|$.

Solution:

(7 pts) (c) We want an upper bound on ALG $= |I|$. Prove that $|I| \leq f \cdot |E|$, and conclude that the f-cover algorithm is an f-approximation algorithm for the f-SetCover problem.

Solution:

(7 pts) (d) Prove that for every positive integer f, there is an input for which the f-cover algorithm necessarily outputs a cover that is *exactly* f times larger than an optimal one.

Solution:

(5 EC pts) 7. **Optional extra-credit question: disk storage**

You have been hired as a summer intern to work on a disk storage system. There are two identical disks, each with storage capacity of L. There are n files $F = \{f_1, \dots, f_n\}$, where file f_i has size ℓ_i. A file cannot be split between the two disks, it must be stored entirely on one or the other (if it is stored at all).

The Disk Storage Optimization Problem is to store the *maximum number* of files from F on the two disks. It turns out that the decision version of this problem is NP-complete (maybe you can prove it!).

Give a polynomial-time algorithm that produces a solution—i.e., a selection of files and which disk to store each of them on—that stores a number of files *within just one* of optimal.

This result is neat, since all of the approximation algorithms we've seen so far come within some *multiplicative* factor of optimal. Here, the algorithm finds a solution that is only an *additive* amount worse than optimal!

Solution: