H7.

- The assignment is due on Gradescope on Friday, March 12 at 5pm.
- You can either type your homework using LaTeX or scan your handwritten work. We will provide a LaTeX template for each homework. If you writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.
- **NP-completeness guidelines:** You can cite and use the NP-completeness of any problem shown NP-complete in class or in K-T Chapter 8 (including the solved exercises, not that they are recommended for use). You are encouraged to consult Chap. 8 Sec. 10, "A Partial Taxonomy of Hard Problems", when looking for a good candidate problem to reduce to the problem being studied.

Lewis Arnsten, Collaborator: Brain McManus
This problem set is identical to the problem set I submitted at 1am, except that I fixed a typo involving the ¬ not appearing in problem 3.

Problem 1:

Zero weight cycle in NP: Zero weight cycle is NP because, for a cycle in G, we can check if the sum of the edge weights is equal to 0. It is clear that this behavior can be implemented in polynomial time.

We will show that subset sum is polynomial time reducible to zero weight cycle.

Reduction: Given weights $w_1,..,w_n$, we want to find a subset that adds up to exactly W. We construct an instance of Zero weight cycle in which the graph has nodes 0,1,..,n and edges (i,j) for all pairs i<j. The weight of each edge (i,j) is equal to w_j. Out last edge (n,0), is assigned a weight -W.

NP-completeness: We claim that $\exists S \subset \{w_i, 1 \leq i \leq n\}$ whose sum of all values w_i is equal to W if and only if G has a zero weight cycle. If there is such a subset S, then we define a cycle starting at 0 that goes through the nodes in S and then returns to 0 on the edge (n,0). The weight -W on the edge (n,0) cancels the sum of the other edge weights. Additionally, all cycles in G use the edge (n,0). Thus, if there exists a zero-weight cycle, then the other edges must cancel -W (the other edges must form a set that adds up to exactly W).

Problem 2:

We will show that vertex cover is polynomial time reducible to monotone satisfiability with few true variables.

Monotone Satisfiability with few true variables in NP: Monotone Satisfiability with few true variables is NP because any certificate can be evaluated as true or false by checking the output of $C_1,...,C_k$. It is clear that this behavior can be implemented in polynomial time.

Reduction: Given a graph G = (V,E) and a number k, we aim to find a vertex cover in G of size at most k.

To create an instance of monotone satisfiability with few true variables we define a variable x_i for each vertex v_i. We create a clause $C_j = (x_a \vee x_b)$ for each edge $e_j = (v_a, v_b)$. Thus for our instance, we define clauses $C_1,C_2,...C_m$, one for each edge of G. We want to know if all $C_1,C_2,...C_m$ can be evaluated as true by setting at most k variables to 1.

NP-completeness: The vertex cover instance returns yes if and only if the answer to the monotone satisfiability with few true variables instance is yes. Consider a vertex cover S in G of size at most k, and set all corresponding variables to 1 as well as all other variables to 0. We know that all clauses are satisfied for, since each edge is covered by an element of S, each clause contains at least one variable set to 1. Now, we will try to satisfy all our clauses by setting a subset T of at most k variables to 1. If we consider the corresponding vertices in G, each edge must have at least one end equal to one of these vertices as the clause corresponding to this edge contains a variable in T. Thus the nodes corresponding to the variables in T form a vertex cover of size at most k.

PROBLEM 3 (35 points). Consider a CNF formula like so

$$F(x_1, \ldots, x_n) \;=\; \bigwedge_{j \in [m]} C_j$$

We call a CNF formula "(w,k)-restricted" if every clause C_j has width at most w, and every variable x_i occurs in at most k clauses (in positive or negated form). As usual, say that such a formula is *satisfiable*, if there exists an assignment $y \in \{0,1\}^n$ to the variables x such that $F(y) = 1$. For example, the formula

$$F = (x_1 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_4} \vee x_5)$$

is $(3,2)$-restricted. It is also $(3,10)$-restricted, since the former immediately implies the latter.

Fix a reasonable natural encoding $\langle F \rangle$ of such formulas and let (w,k)-RESTRICTED SAT denote the language consisting of all strings of form $\langle F \rangle$, where F is a satisfiable $(3,10)$-restricted CNF formula. Prove that (w,k)-RESTRICTED SAT is NP-complete.

We note that the constant 10 in the statement is chosen to give some slack in the proof; the statement can be proved with a smaller number in place of 10.

Solution: Your solution goes here.

Problem 3:

(w, k)-restricted SAT in NP: (w, k)-restricted SAT is NP because any certificate can be evaluated as true or false by checking the output of F(y). It is clear that this behavior can be implemented in polynomial time.

NP-completeness: For our reduction, we will start with a 3-SAT instance with variables $x_1, \ldots, x_n$. For each x, if either x or $\neg x$ appears in more than 10 clauses, we will perform the following procedure: If x appears in k clauses, we will create k new variables $x_1, \ldots, x_k$. To construct our instance, we will replace the i−th occurrence of x with x_i, where $i = 1, \ldots, k$. Then, we add the clauses $\{x_i \vee \neg x_{i+1}\}$ to $\langle F \rangle$ for i = l, ..., k − 1 and the clause $\{x_k \vee \neg x_1\}$. In our new instance, the clause $\{x_i \vee \neg x_{i+1}\}$ will ensure that, if x_i is 0, x_{i+1} must also be 0. The cyclic structure of the clauses forces the x_i to be either all 1 or all 0, so the new instance is satisfiable if and only if the original one is. The transformation clearly runs in polynomial time, so (3, k)-restricted SAT (as well as (3, 10)-restricted SAT) is NP-complete.