

## CS 170 Homework 13

Due Monday 4/20/2026, at 10:00 pm (grace period until 11:59pm)

### Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write “none”.

### 1 NP Basics (Solo Question; 10 points)

Assume A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. What information can you derive of the other problem given each fact?

- (a) A is in **P**.
- (b) B is in **P**.
- (c) A is **NP**-hard.
- (d) B is **NP**-hard.

**Solution:** If A reduces to B, we know B can be used to solve A, which means B is at least as hard as A. Therefore if B is in **P**, then A is in **P** (part (b)), and if A is **NP**-hard, then B is **NP**-hard (part (c)). If A is in **P** (part (a)), or if B is **NP**-hard (part (d)), we cannot say anything about the complexity of B or A respectively.

## 2 TSP Variants (10 points)

In the Traveling Salesman Problem (TSP), we are given an undirected graph  $G$  with non-negative weights and asked to find a minimum weight cycle that visits each vertex exactly once.

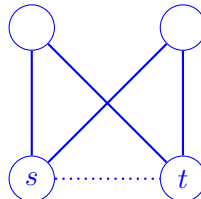
In the  $s$ - $t$  Traveling Salesman Problem ( $s$ - $t$  TSP), we are given an undirected graph  $G$  with non-negative weights, two vertices  $s$  and  $t$ , and are asked to find a minimum weight path that starts at  $s$ , visits all other vertices exactly once, and ends at  $t$ .

- Consider the following reduction from  $s$ - $t$  TSP to TSP: Take  $G$  and add a weight 0 edge between  $s$  and  $t$  to get a new graph  $G'$ . Show that if a  $s$ - $t$  TSP solution of weight  $w$  exists in  $G$ , then a TSP solution of weight  $w$  exists in  $G'$ .
- Despite what you proved in part (a), why is the reduction in part (a) not valid?
- Give a valid reduction from  $s$ - $t$  TSP to TSP. Prove correctness for your reduction. No runtime analysis needed.

### Solution:

- Given a weight- $w$   $s$ - $t$  TSP solution in  $G$ , we can add the weight-0 edge  $(s, t)$  to complete the cycle in  $G'$ , thereby obtaining a weight- $w$  TSP solution in  $G'$ .
- A TSP solution in  $G'$  is not forced to use the new edge, which means we may not be able to map a TSP solution on  $G'$  back to an  $s$ - $t$  TSP solution on  $G$ .

For example, the following graph  $G$  (solid edges) has no  $s$ - $t$  TSP solution, but  $G'$  (includes the dotted edge from  $s$  to  $t$ ) has a TSP solution, which does not use the new dotted edge:



- To fix the reduction from part (a), we need to replace  $G'$  with a graph  $G''$  on which the TSP solution is forced to go from  $s$  to  $t$ , without hitting any other vertices from  $G$  in between. To do so, we construct  $G''$  by adding a vertex  $q$  to  $G$ , and then connecting  $q$  to both  $s$  and  $t$  by weight-0 edges. A TSP solution on  $G''$  then must pass through  $q$ , and hence takes the new weight-0 edges  $(s, q)$  and  $(q, t)$ ; removing these edges yields an  $s$ - $t$  TSP solution on  $G$ . Meanwhile, an  $s$ - $t$  TSP solution on  $G$  still gives a TSP solution on  $G$ , by simply adding in the two new weight-0 edges.

### 3 SAT and Integer Programming (10 points)

Recall the following problems:

1. **3-SAT:** Given a set of boolean variables  $x_i$  with a set of clauses, where each clause is the OR of at most 3 literals, we want to decide if there exists an assignment of the variables that satisfies all the clauses.

Recall that a literal is a variable or its negation; an example clause is  $x_1 \vee \overline{x_4} \vee \overline{x_7}$ .

2. **Integer linear programming feasibility:** Given a set of variables  $x_i$  with a set of constraints, we want to decide if there exists an assignment of the variables that satisfies all the constraints. Each constraint is either a linear inequality, or a 0-1 constraint  $x_i \in \{0, 1\}$ . (We are not given an objective function.)

Give a reduction from 3-SAT to integer linear programming feasibility, and briefly justify its correctness.

**Solution:** Given a 3-SAT instance, we create an integer linear program with the same variables  $x_i$ , with constraints  $x_i \in \{0, 1\}$  restricting to boolean values. We then replace each 3-SAT clause  $a \vee b \vee c$  with the linear constraint  $a + b + c \geq 1$ , where each negated variable  $\overline{x_i}$  is replaced with  $1 - x_i$ . For instance,

$$x_1 \vee \overline{x_4} \vee \overline{x_7} \quad \text{becomes} \quad x_1 + (1 - x_4) + (1 - x_7) \geq 1.$$

Then each clause in the 3-SAT instance is satisfied if and only if the corresponding linear constraint is satisfied, so a solution to either problem (3-SAT or the integer program) is also a solution to the other.

## 4 Algorithms for NP Problems (10 points)

- (a) Give a  $O(2^n m)$ -time algorithm to solve a 3-SAT instance with  $n$  variables and  $m$  constraints (see Question 3).
- (b) Show that every problem in **NP** can be solved in time  $O(2^{n^c})$  for some constant  $c$  (that can depend on the problem). Here  $n$  denotes the size of (i.e. the number of bits needed to write down) the original problem instance.

*Hint: use part (a) along with the fact that 3-SAT is NP-hard.*

### Solution:

- (a) Enumerate all  $2^n$  assignments. For each assignment, we can check if all  $m$  clauses are satisfied in  $O(m)$  time. So we can find a satisfying assignment if one exists in  $O(2^n m)$  time.
- (b) Because 3-SAT is **NP**-hard, we can reduce any problem in **NP** to 3-SAT in polynomial time  $n^{O(1)}$ . Because it takes at least one unit of time to write down each variable/clause, the resulting 3-SAT instance has at most a polynomial number  $n^{O(1)}$  of variables and clauses. Then by part (a), we can solve this 3-SAT instance in time  $O(2^{n^{O(1)}} \cdot n^{O(1)}) = O(2^{n^{O(1)}})$ .

## 5 Halting (10 points)

The *halting problem* has input given by a program (as e.g. a .py file), and asks us to determine if the program runs forever or eventually halts.

- (a) Give a reduction from 3-SAT (or another **NP**-complete problem) to the halting problem. Your reduction algorithm, which *constructs* a program from the 3-SAT instance, should run in polynomial time, but the resulting program may run for arbitrarily long.

(Your reduction will imply that the halting problem is **NP**-hard.)

- (b) Read the box on page 262 of the textbook DPV, which shows that there is no algorithm that solves the halting problem in finite time. Using this fact, can you conclude whether or not the halting problem is in **NP**?

*Hint: it may be helpful to recall your solution to Question 4.*

### Solution:

- (a) Given a 3-SAT instance with  $n$  variables and  $m$  constraints, we construct a program that repeatedly loops through all  $2^n$  assignments to the variables, and halts if it finds an assignment for which all constraints are satisfied. If there is no satisfying assignment, our program loops forever.

While our program may run indefinitely, the reduction algorithm that *constructs* the program takes time  $\text{poly}(n, m)$ , as it simply must set up the loop over assignments to the  $n$  variables, while hardcoding in the  $m$  clauses.

- (b) By Question 4(b), every problem in **NP** can be solved in finite time. But the halting problem cannot be solved in finite time (by page 262 of DPV), so it is not in **NP**.