COL352 HW-4

Sarthak Agrawal
2018CS10383

Piyush Gupta
2018CS10365

Navneel Singhal
2018CS10360

May 15, 2021

Read the instructions carefully.

The deadline for this homework will be announced later, and at least one week in advance. The deadline will be no earlier than May 15, 23:00. However, **if your group is not seriously affected by the pandemic and is in a position to submit early, you are requested to do so.** Please bear in mind that since we have our exam on the last day of the major exam week, the TAs will already be under pressure to grade your exams in the following week (May 17-23). The incentive for submitting the homework by May 15, 23:00 is that we will grade it and provide feedback in the next 24 hours (unless we get too many submissions on the same day). **Once your submission is ready to be graded, please ping the instructor and all the TAs on Teams so that we know we can start grading, and don't submit thereafter.**

Problem 1

Let U be a finite set and let $\mathcal{S} \subseteq 2^U$ be a collection of subsets of U. We say that a set $H \subseteq U$ is a *hitting set* of $\mathcal{S}$ if for all $A \in \mathcal{S}$, we have $H \cap A \neq \emptyset$, or in other words, H intersects every set in $\mathcal{S}$. In the `HittingSet` problem, we are given U, $\mathcal{S} \subseteq 2^U$, and an integer k as input. We are required to determine whether there is a hitting set of $\mathcal{S}$ having size at most k. Prove that `HittingSet` is `NP`-complete.

Solution. We break the proof into two parts.

Claim 1.1

`HittingSet` is in `NP`.

Proof. First note that if there is a hitting set of size k, and $k \leq |U|$, then there is a hitting set of size k' where $k \leq k' \leq |U|$ (by adding in superfluous elements). A non-deterministic Turing machine can be constructed for the following algorithm that solves the problem in polynomial time:

1. Choose a set of size k non-deterministically.
2. If this set has a non-empty intersection with every set in $\mathcal{S}$, then accept the input, else reject the input.

Since each step can be done in polynomial time non-deterministically, we are done. □

Claim 1.2

`VertexCover` is polynomial-time reducible to `HittingSet`.

Proof. Suppose $(G = (V, E), k)$ is a pair of a graph and an integer. Consider the tuple $(V, \{\{u, v\} \mid (u, v) \in E\}, k)$ as an input for the `HittingSet` problem. Note that constructing this input from the original input is possible in deterministic polynomial time. Now we claim the following.

Claim 1.3

$(G = (V, E), k) \in$ `VertexCover` $\iff (V, \{\{u, v\} \mid (u, v) \in E\}, k) \in$ `HittingSet`

Proof. First, suppose that $(G = (V, E), k) \in$ `VertexCover`. Let F be a vertex cover of G of size $\leq k$. Then for each edge (u, v), one of (u, v) is in F. Hence, $F \cap \{u, v\} \neq \emptyset \forall (u, v) \in E$, which implies that $(V, \{\{u, v\} \mid (u, v) \in E\}, k) \in$ `HittingSet`, since F is such a *hitting set* of the collection of subsets (the problem is well-formed trivially).

Now suppose that $(V, \{\{u, v\} \mid (u, v) \in E\}, k) \in$ `HittingSet`. This implies that there is a set H of size $\leq k$ such that $H \subseteq V$, and $H \cap \{u, v\} \neq \emptyset \forall (u, v) \in E$. This means H is a vertex cover of size $\leq k$ for the graph G, and thus $(G, k) \in$ `VertexCover`, as needed. □

This claim shows that this reduction is indeed valid, and hence Claim 1.2 holds. □

The second claim implies, from the fact that `VertexCover` is `NP`-hard, that `HittingSet` is also `NP`-hard. The first claim shows that it is in `NP`, which when combined with the previous sentence implies that it is indeed `NP`-complete.

Problem 2

Recall that two graphs $G = (V, E)$ and $G' = (V', E')$ are called *isomorphic* if there exists a bijection $h : V \longrightarrow V'$ such that for all vertices $u, v \in V$, $(u, v) \in E$ if and only if $(f(u), f(v)) \in E$, and such a bijection is called an *isomorphism* between G and G'. In other words, an isomorphism is an adjacency-preserving bijection. Also recall that a subgraph of a graph G is any graph that is obtained by deleting some vertices and edges of G. In the `SubgraphIsomorphism` problem, the input consists of two graphs, say G and G', and we are required to determine whether G' is isomorphic to some subgraph of G. Prove that `SubgraphIsomorphism` is `NP`-complete.

Solution. We give a polynomial time reduction from the `Clique` problem to this problem, and then exhibiting a non-deterministic algorithm that solves `SubgraphIsomorphism` in polynomial time.

Claim 2.1

`Clique` $\leq_P$ `SubgraphIsomorphism`

Proof. Consider the reduction as follows. Denote by K_r the complete graph of size r (by size we mean the number of vertices here). Then for input (G, k) to `Clique`, let (G, K_k) be the corresponding input to `SubgraphIsomorphism`.

Note

We assume that for (G, k) is a non-trivial input to the `Clique` problem, i.e., $k \leq |V(G)|$ (other inputs are necessarily not in `Clique` due to size reasons). This is necessary, else our reduction won't be polynomial time for large k.

Now that we have $k \leq V(G)$, we can build a clique in time polynomial in k, which is at most polynomial in $|V(G)|$, and thus at most polynomial in the input size. Hence, this reduction (including the copying of the graph G) is polynomial-time. Now we show that it is indeed a valid reduction.

Claim 2.2

$(G, k) \in$ `Clique` $\iff (G, K_k) \in$ `SubgraphIsomorphism`.

Proof. First, suppose that $(G, k) \in$ `Clique`. Then there is a clique of size $\geq k$ in G, and hence there is also a clique of size k in G (removing the extra vertices gives rise to a clique too). Hence, there is a subgraph of G which is a complete graph of size k. Since all complete graphs of size k are isomorphic, there exists a subgraph of G isomorphic to K_k, from where we have that $(G, K_k) \in$ `SubgraphIsomorphism`.

Now suppose that $(G, K_k) \in$ `SubgraphIsomorphism`. Then there exists a subgraph which is isomorphic to the complete graph of size k. Hence, there exists a clique in the graph of size $\geq k$ (formed by the vertices of this subgraph), which implies that $(G, k) \in$ `Clique`, as needed. □

This shows that this reduction is indeed valid, and hence we have proven the claim. □

Now we exhibit a non-deterministic algorithm that solves `SubgraphIsomorphism` in polynomial time. Consider the following algorithm:

1. Non-deterministically choose a subset V' of V and a subset E' of E, and remove the edges from E' that are incident to a vertex not in V'.
2. Now call the non-deterministic Turing machine that solves the graph isomorphism problem for the graphs $(G', (V', E'))$.

This clearly works in non-deterministic polynomial time, so `SubgraphIsomorphism` is in NP, which when combined with the first claim (which implies that `SubgraphIsomorphism` is `NP`-hard), implies that `SubgraphIsomorphism` is `NP`-complete.

Problem 3

A *dominating set* of a graph is a subset D of vertices such that every vertex that is not in D is adjacent to some vertex in D. The input to the `DominatingSet` problem is a graph G and an integer k, and the output is whether G has a dominating set of size at most k. Prove that the following algorithm is a Karp reduction from `VertexCover` to `DominatingSet`.
`VCtoDS`(G, k):

1. Construct a vertex set V' by taking the vertex set V of G, and then adding a vertex v_e for each edge e of G.
2. Construct an edge set E' as follows. For each edge $e = (a, b)$ of G, add three edges – (a, v_e), (b, v_e), and (a, b) to E'.
3. Return the instance (G', k) of `DominatingSet`, where $G' = (V', E')$.

Solution. The error in the reduction stated in the question is that it does not account for the fact that there may be *isolated* vertices in G, i.e., vertices with degree $= 0$ which have no incident edges. The reason these vertices can cause a mishap is because such vertices are not necessary to be included in vertex cover but they must necessarily be included in the dominating set. We shall further emphasize this fact in the proof below. Hence, we will make the correction that while constructing the vertex set V' we will not include such *isolated* vertices.
Clearly, the reduction can be performed in polynomial-time in the input size. So, we now show the following two claims which shall finish the problem.

Claim 3.1

$(G = (V, E), k) \in$ `VertexCover` $\implies (G' = (V', E'), k) \in$ `DominatingSet`.

Proof. Let F be a vertex cover of size at most k of the graph G. We claim that F is a dominating set of size at most k in graph G', which will prove this claim.

To this end, consider any vertex in V' not in F. It can be of two types.

1. It is v_e for some edge $e \in E$. Since $e = (u, v)$ and F is a vertex cover of G, at least one of u, v is in F, wlog, say u. Then since $u \in F$, and $(v_e, u) \in E'$, v_e is adjacent to some vertex in F, finishing off this case.
2. It is $v \in V$. Consider any edge of the form $e = (u, v)$. Since F is a vertex cover, and $v \notin F$, we must have $u \in F$, and since $(u, v) \in E'$, we have v adjacent to some vertex in F, finishing off this case as well. Note that here it could have been possible that there is no such e in the first place, and this is the error with the original reduction as mentioned in the problem statement. But as we have stated before, we have modified reduction to delete such vertices hence this anomaly does not arise.

Hence, for each vertex not in F, there is a vertex in F which is adjacent to it, implying that F is a dominating set. □

Claim 3.2

$(G = (V, E), k) \in$ `VertexCover` $\impliedby (G' = (V', E'), k) \in$ `DominatingSet`.

Proof. Consider a dominating set D of G' of size $\leq k$. We exhibit a vertex cover as follows: Let S be initially empty. For each vertex v_e in $D \cap (E' \setminus E)$, where $e = (u, v)$, add u to S. Then for each vertex $v \in D \cap E$, add v to S. This gives a set of size at most k (since we have at most k additions).

Now we show that S is indeed a vertex cover of the graph. To this end, consider any edge $e = (u, v)$ in the graph.

1. If v_e is in D, then at least one of u, v is in S, so S covers the edge e in G.
2. If v_e is not in D, then there must be some vertex w in D that it is adjacent to, and since u, v

are the only vertices adjacent to v_e, at least one of u, v is in D, and due to our construction of S, S must have that vertex. So S covers the edge e in G.

We have hence shown that S is a vertex cover of size $\leq k$ in G, as needed. $\square$

Problem 4

Recall that a boolean formula is said to be a 3CNF formula if it is an AND of clauses, where each clause is an OR of at most 3 literals. Let us call a boolean formula a *strict 3CNF* formula if it is an AND of clauses, where each clause is an OR of exactly 3 literals. In the `Strict3SAT` problem, we are given a strict 3CNF formula, and we need to accept it if it is satisfiable, and reject it otherwise. Prove that `3SAT` is Karp-reducible to `Strict3SAT`.

Solution. Consider the following reduction, where we convert an input 3CNF formula ϕ for the `3SAT` problem to an input struct 3CNF formula φ for the `Strict3SAT` problem. Here, x, y, z are extra *dummy* variables. Instantiation of a variable x is either x or $\overline{x}$.

1. Let φ be initially empty (and-ing a clause with an empty clause leads to replacement of the empty clause with that clause).
2. Break ϕ into clauses, and do the next 3 steps for each clause.
3. If the clause is of the form $(a \vee b \vee c)$, set $\varphi \leftarrow \varphi \wedge (a \vee b \vee c)$.
4. Else if the clause is of the form $(a \vee b)$, set $\varphi \leftarrow \varphi \wedge (a \vee b \vee x)$.
5. Else, the clause is of the form (a), so set $\varphi \leftarrow \varphi \wedge (a \vee x \vee y)$.
6. Finally, add the following $2^3 - 1$ extra clauses to φ : For each possible *instantiation* of variables x, y, z (x', y', z') except the instance (x, y, z), set $\varphi \leftarrow \varphi \wedge (x' \vee y' \vee z')$.

Clearly, this reduction takes time polynomial in the input size. So it suffices to show that this is a valid reduction.

Claim 4.1

$\phi \in$ `3SAT` $\implies \varphi \in$ `Strict3SAT`.

Proof. For any clause C in ϕ, the corresponding clause in φ is of the form C or $C \vee x$ or $C \vee x \vee y$. Since ϕ is satisfiable, C must evaluate to true, and hence since or-ing true with anything gives true, the corresponding clause in φ must also be satisfiable. The only remaining clauses in φ that need to be satisfied are those of form $x' \vee y' \vee z'$. But these clauses can be satisfied by selection $x, y, z = 0$ as each of these clause contains at least one of either $\overline{x}$ or $\overline{y}$ or $\overline{z}$. Hence we are done. □

Claim 4.2

$\phi \in$ `3SAT` $\impliedby \varphi \in$ `Strict3SAT`.

Proof. Consider a satisfying assignment for literals in φ. First we claim that in any satisfying assignment the values of x, y, z must be all 0. Suppose not, and atleast one of them has value 1. Then, one of the additional $2^3 - 1$ clauses will become false, hence this is a contradiction.

Now, consider any clause C in ϕ, and let C' be the corresponding clause in φ. Note that C' evaluates to true. Then we have one of the following three cases:

1. $C = (a \vee b \vee c)$: in this case, since $C' = C$, C evaluates to true as well.
2. $C = (a \vee b)$: in this case, we have $C' = (a \vee b \vee x)$. Since C' evaluates to true and x is false as established before, at least one of a, b are true, so C evaluates to true as well.
3. $C = (a)$: in this case, we have $C' = (a \vee x \vee y)$. Since C' evaluate to true and both x and y are false, a must be true, so C evaluates to true as well.

In all cases, C evaluates to true, so each clause in ϕ evaluates to true, and thus ϕ is satisfiable using the same assignment. □

These two claims show that this is indeed a valid reduction.

Problem 5

The crazy instructor of COL352 has an equally crazy criterion for passing the course. The instructor gives n homework problems over the semester, and every student gets zero or one mark in every problem. For a student to pass the course, her/his marks must satisfy all of m constraints specified by the instructor in the beginning of the semester. Specifically, a student who gets marks $x[1], \ldots x[n]$ passes the course if and only if for each $i \in \{1, \ldots, m\}$, $\sum_{j=1}^{n} A[i][j] \times x[j] \geq b[i]$. Here $A[i][j]$ and $b[i]$ are integers specified in the beginning of the semester.
For example, if $m = n = 3$ and the constraints are $x[1]+x[2]+x[3] \geq 2$, $-2x[2] \geq -1$, and $-x[1]+3x[3] \geq 1$, then a student getting marks $x[1] = 0, x[2] = 0, x[3] = 1$ fails the course due to violation of the first constraint, while a student getting marks $x[1] = 1, x[2] = 0, x[3] = 1$ passes the course.
Given the matrix of integers $(A[i][j])$ where $i = 1, \ldots, m$ and $j = 1, \ldots, n$, and the array $b[1], \ldots, b[m]$, the students would like to know whether it is at all possible to pass the course. That is, they want to accept the input if and only if there exist $x[1], \ldots, x[n] \in \{0, 1\}$ such that for each $i \in \{1, \ldots, m\}$, $\sum_{j=1}^{n} A[i][j] \times x[j] \geq b[i]$. Prove that this task is NP-complete.

Solution. We first show that this task is in NP. For this, it is easy to non-deterministically choose an assignment of $x[1 \ldots n]$ and check all the conditions in polynomial time.

Now we show that 3SAT is reducible to this problem (we shall call the language of this problem LP).

Consider the following algorithm which takes as input a 3CNF formula ϕ and outputs $(A[][], b[])$.

1. Let n be the number of boolean variables featuring in the formula.
2. Let m be the number of clauses.
3. Set $b[i] = 1$ for all $1 \leq i \leq m$.
4. Set $A[i][j] = 0$ for all $1 \leq i \leq m$, $1 \leq j \leq n$.
5. For each literal X in the i^{th} clause, if $X = x_j$ for some j, then increment $A[i][j]$ by 1, else if $X = \overline{x_j}$ for some j, then decrement $A[i][j]$ by 1 and decrement $b[i]$ by 1.

This can be done in polynomial time. Now we show that this is indeed a valid reduction.

Claim 5.1

$\phi \in$ 3SAT $\implies (A, b) \in$ LP.

Proof. Let $x_i = x_i^*$ be a satisfying assignment for ϕ. Now we claim that $x[i] = x_i^*$ satisfies $\sum_{j=1}^{n} A[i][j] \times x[j] \geq b[i]$ for all $1 \leq i \leq m$.

Fix i. Now we look at the i^{th} clause in ϕ. For the sake of convenience, we shall assume wlog that the clause has 3 literals X, Y, Z (the rest can be done analogously). Let $rep(x_i) = x[i]$, and $rep(\overline{x_i}) = 1 - x[i]$. Then, since the reduced clause evaluates to true, we get that at least one of $rep(X), rep(Y), rep(Z)$ is 1. The others can be at most 0. Adding them up, we get that $rep(X) + rep(Y) + rep(Z) \geq 1$. Now opening up and rearranging gives us the equation $\sum_{j=1}^{n} A[i][j] \times x[j] \geq b[i]$, as needed. Since all such equations arise from clauses, this shows that $(A, b) \in$ LP, which completes the proof of this claim. □

Claim 5.2

$\phi \in$ 3SAT $\impliedby (A, b) \in$ LP.

Proof. Suppose there exists a solution $x[1 \ldots n]$ to $\sum_{j=1}^{n} A[i][j] \times x[j] \geq b[i]$ for all $1 \leq i \leq m$ where $0 \leq x[i] \leq 1$ for all $1 \leq i \leq n$. Then consider the i^{th} clause. Wlog assume that there are exactly 3 j such that $A[i][j]$ are non-zero (there can be at most 3, and the other cases are completely analogous). The corresponding clause has 3 literals in this case, say X, Y, Z. Using the notation and the argument in the previous claim, the equation reduces to $rep(X) + rep(Y) + rep(Z) \geq 1$, so since $rep(\cdot)$ when evaluated at $0, 1$ always gives something non-negative, at least one of them is ≥ 1. Note that rep is a function that evaluates a literal and if the result is true, returns 1 and 0 otherwise. So at least one literal in the clause is true, so the clause itself evaluates to true. Since this is true for all clauses, all

clauses evaluate to true, so we have $\phi \in$ 3SAT, as needed. □

From these two claims, it follows that LP is NP-hard, and combining this with the first observation gives that it is NP-complete.

Problem 6

Recall the `CircuitSAT` problem discussed in class. The input to the problem is a boolean circuit over variables $x_1, \ldots, x_n$, where each input of each gate is a variable, a constant (`true` or `false`), or the output of an "earlier" gate. One gate is designated as the output gate, and we are required to determine whether there exists a boolean assignment to the variables that makes the output of the circuit `true` (a.k.a. satisfying assignment). The `CircuitSATSearch` problem has the same input as `CircuitSAT`, but we are required to output a satisfying assignment if such an assignment exists, and otherwise output "NO".
Suppose that you are given a "black box" algorithm B that solves `CircuitSAT`, that is, given a boolean circuit as input, the B accepts if the circuit is satisfiable, and rejects otherwise. Design an algorithm to solve `CircuitSATSearch` which makes at most a polynomial number of calls to B and spends at most a polynomial amount of time outside those calls. (This implies that to be able to search for a satisfying assignment in polynomial time, it is sufficient to be able to detect the presence or absence of such an assignment in polynomial time.)

Solution. We use the following algorithm to create a valid solution:

1. First run B on the circuit once to check if the circuit is satisfiable, and if it isn't, then indicate that there is no solution and exit. Else, carry on with the next steps.
2. Let $A = \{\}$, and C be the original circuit.
3. For $i \in \{1, \ldots, n\}$, do the following: For $v \in \{\texttt{true}, \texttt{false}\}$, make a copy of C and for all gates with input x_i, replace that input with a constant equal to v, and run B on this circuit. If this fails, check the next value of v, else set C to be this circuit, and $A \leftarrow A \cup \{x_i = v\}$, and we go to the next i.
4. Return the assignment A.

In the case where we exit on step 1, there is no solution, so consider the case where there is a solution. Suppose we are at iteration i. Then since we have not exited on step 1, so there must exist a solution. Using induction, we can show that at the end of iteration i, there is an assignment where the first i variables are assigned according to the current A, as follows:

1. Base case ($i = 0$) is true since we haven't exited on step 1.
2. Now if there is a solution with the first $i-1$ assignments, then consider such a solution. If $x_i = \texttt{false}$ in any such assignment, B shall return true, so we will be able to add $x_i = \texttt{false}$ to the assignment A and move on. In the other case, it must happen that $x_i = \texttt{true}$, so we will be able to add this to the assignment A and move on.

Now using this invariant for $i = n$, we are done.