Instructor:
Eleni Drinea
CSOR W4246–Fall, 2021

Student:
Vishweshwar Tyagi
vt2353@columbia.edu

Homework 4 Theoretical (105 points)

Out: Monday, November 22, 2021
Due: 11:59pm, Monday, December 13, 2021

Homework Problems

1. (12 points) Stingy SAT is the following problem.

   On input a SAT formula ϕ, return a truth assignment that satisfies ϕ and minimizes the number of variables that are set to 1, or **no** if ϕ is unsatisfiable.

   Give a polynomial time algorithm for Stingy SAT, or prove that no such algorithm exists unless $P = NP$.

   Solution: We will show that no such algorithm exists unless $P = NP$ by reducing SAT problem to Stingy SAT. To do so, we will first consider the decision version of Stingy SAT.

   Stingy SAT - Optimization Version, SS

   Input: ϕ, a CNF formula with m clauses and n literals

   Problem: Return No if ϕ is not satisfiable, else return the truth assignment with minimum number of literals set to 1.

   Stingy SAT - Decision Version, SS(D)

   Input: ϕ, a CNF formula with m clauses and n literals and an interger k

   Problem: Return Yes if and only if ϕ is satisfiable with a truth assignment that assigns atmost k literals equal to 1.

   SAT

   Input: ϕ, a CNF formula with m clauses and n literals

   Problem: Return Yes if and only if ϕ is satisfiable.

   SS(D) is clearly NP. Given an instance (ϕ, k) of SS(D) (where ϕ is a CNF formula with m clauses and n literals, and k is an integer) with candidate solution S, we can simply plug in the assigned literal values and evaluate the formula and return Yes if and only if it evaluates to 1. This is clearly done in polynomial time in size of input. Hence, S is a succinct certificate that $(\phi, k) \in$ SS(D)

   It suffices to show that SAT $\leq_p$ SS(D)

Outline: For given ϕ of m clauses and n literals, we want to transform it to (ϕ', k) such that ϕ is a Yes instance for SAT if and only (ϕ', k) is a Yes instance of SS(D), equivalently, *phi* is satisfiable if and only if ϕ' is satisfiable with a truth assignment that sets atmost k literals to 1.

Let, $\phi' = \phi$ and $k = n$, then our transformation is:

Transformation:

$$\phi \rightarrow (\phi, n) \tag{1}$$

Time-Complexity: Clearly this transformation is constant time.

Equivalence:

($\Rightarrow$) Let ϕ be a Yes instance for SAT, i.e, ϕ is satisfiable. Hence, $\exists$ a truth assignent for ϕ. Now, ϕ has n literals, hence, this truth assignment must set at most n literals to 1. Hence, (ϕ, n) is a Yes instance for SS(D).

($\Leftarrow$) Let (ϕ, n) be a Yes instance of SS(D).This implies that ϕ is satisfiable. Hence, ϕ is a Yes instance of SAT.

Hence, SAT $\leq_p$ SS(D). We know that SAT is NP-Complete, and therefore, SS(D) is NP-Complete. We conclude that there does not exist any polynomial time algorithm for SS(D) unless P=NP, and equivalently, there does not exist any polynomial time algorithm for Stingy SAT unless P=NP.

2. (25 points)

Suppose you had a polynomial-time algorithm A for the decision version of the maximum independent set problem IS(D). That is, on input a graph $G = (V, E)$ and an integer k, A answers yes if and only if G has an independent set of size at least k, and A runs in worst-case polynomial time.

Design a polynomial-time algorithm that takes as input a graph G and an integer k, and returns an independent set of size at least k if one exists in G, or no otherwise.

Solution: We have $A(G, k) =$ yes if and only if $G = (V, E)$ has an independent set of size atleast k.

Let $|V| = n$ and $|E| = m$. We know that size of an independent set is bounded, i.e., $1 \leq |S| \leq n$ for all S independent sets of G. We can use this fact to perform a binary search and get the size of a maximum independent set as follows:

```
procedure max-IS-size(G=(V,E)):
    lo = 1
    hi = |V|

    while hi > lo:
        mid = lo + (hi-lo+1)/2

        if A(G, mid) == Yes:         # TTTFFF, last T
            lo = mid
        else:
            hi = mid-1

    return lo
```

This is a fairly standard implementation of binary search and we will not prove its correctness.

Clearly this procedure runs in polynomial time because we are making atmost $lg(|V|)$ calls to $A()$ which is also polynomial-time.

Now, we can use this procedure to get a maximum independent set as follows:

Let for $v \in V$ of $G = (V, E)$, we define $n(v)$ to be the neighbors of v in G. Consider the following procedure:

```
procedure max-IS(G=(V,E), k):
    if A(G, k) = No, then return No.

    L = max-IS-size(G)
    Initialize S to be an empty list, S = []

    for node v in V:
        if max-IS(G\{v}) == L:
            G = G\{v}
        else:
            G = G\ [{v} U n(v)]
            S.append(v)

    return S
```

Polynomial Time: This procedure also runs in polynomial time because it makes atmost $|V|$ calls to max-IS(G') procedure where $G' = (V', E')$ such that $|V'| \leq |V|$ and $|E'| \leq |E|$ and only one call to $A(G, k)$. Note that we can also delete a given v and its neighbors $n(v)$ from the graph in $O(|E|)$ time.

Correctness: First of all, we note that the above procedure terminates because $[7 - 12]$ runs atmost $|V| + 1.$ times.

The idea is to iterate through all nodes of the graph while simultaneously forming a maximum independent set. More concretely, for a given node v in V of $G = (V, E)$, either there is a maximum independent set that does not contains v or there is no such maximum independent set. In the former case, we simply remove v from our graph and keep searching for a maximum independent set. In the latter case, v has to necessarily belong in a maximum independent set and hence we add v to our maximum independent set S, and because we include v, we cannot include its neighbors and therefore, we can safely remove v as well as its neighbors $n(v)$ from the graph and continue constructing S. That this is a safe move is demonstrated by the following claim:

Claim: Let L be the size of a maximum independent set in $G = (V, E)$. Then, for each node v in V, the size of a maximum independent set in $G \setminus \{v\}$ is equal to L if and only if there exists a maximum independent set in G that does not includes v in it.

Proof: ($\Rightarrow$) Suppose size of a maximim independent set in $G \setminus \{v\}$ is equal to L. Hence, there is an independent set S in $G \setminus \{v\}$ with $|S| = L$ that does not have v in it. However, every independent set of $G \setminus \{v\}$ is also an independent set of G, and therefore S is an independent set of size L that doesn't have v in it. Because size of a maximum independent set in G is L, this means that S is also a maximum independent set of G (that does not include v in it).

($\Leftarrow$) Suppose there is a maximum independent set S of G that does not include v in it. This means that $|S| = L$ and note that the size of a maximum independent set in $G \setminus \{v\}$ can atmost be L since every such set is also an independent set in G. If we can show that S is an independent set of $G \setminus \{v\}$, we'll be done.

3. (30 points) A large store has m customers and n products and maintains an $m \times n$ matrix A such that $A_{ij} = 1$ if customer 1 has purchased product j; otherwise, $A_{ij} = 0$.

   Two customers are called orthogonal if they did not purchase any products in common. Your task is to help the store determine a maximum subset of orthogonal customers.

   Give a polynomial-time algorithm for this problem or prove that no such algorithm exists unless P = NP.

Solution:

X - Optimization Version

> Input: Matrix A of size $m \times n$ with $A_{ij} = 1$ if and only if customer i has purchased product j, else 0
>
> Problem: Return a maximum orthogonal subset of customers.

X(D) - Decision Version

> Input: Matrix A of size $m \times n$ with $A_{ij} = 1$ if and only if customer i has purchased product j, else 0 and an integer k
>
> Problem: Output Yes if and only if there is an orthogonal subset of customers of size atleast k.

We will show that $X(D)$ is NP-Complete. First we will show that $X(D)$ is in NP.

Efficient Certifier for $X(D)$: Given an instance (A, k) of $X(D)$, where A is a matrix of size $m \times n$ with $A_{ij} = 1$ if and only if customer i has purchased product j, else 0 and an integer k and a candidate solution which is a subset S of customers, we can first easily check that $|S| \geq k$. Then, we can check that there exists no pair of customers (out of the $\binom{m}{2} = O(m^2)$ pairs) that have purchased a common product. That is, if c_i and c_j is the current pair of customers for which we want to check, we can iterate over the columns and check if A_{ik} and A_{jk} are not both equal to 1 for all $k = 1, \ldots n$ in $O(n)$ time. Hence, our verifier runs in $O(m^n)$ time which is polynomial in the input size.

Therefore, $X(D) \in$ NP.

Now, it suffices to show that IS(D) $\leq_p X(D)$, where, IS(D) is:

Independent Set - Decision Version, IS(D)

> Input: (G, k) where $G = (V, E)$ is a graph of $|V| = n$ vertices and $|E| = m$ edges and an integer k.
>
> Problem: Output Yes if and only if there is an independent set of size atleast k.

We already know that IS(D) is NP-Complete and if we can show IS(D) $\leq_p X(D)$, this will prove that $X(D)$ is also NP-Complete.

For convenience of notation, define $[n] = \{1, 2 \ldots n\} \ \forall \ n \in \mathbb{N}$

Transformation:

For an input instance (G, k) of IS(D), where $G = (V, E)$ with $|V| = n$ and $|E| = m$ and integer k, we can order the set of vertices and edges as $\{u_1, u_2 \dots u_n\}$ and $\{e_1, e_2 \dots e_m\}$ respectively. We then define A to be a matrix of size $m \times (m + n)$, where $\{c_1, c_2 \dots c_n\}$ are the customers and $\{p_1, p_2 \dots p_{n+m}\}$ are the products and $A_{ii} = 1 \ \ \forall\ i \in [n]$ and $A_{i,(n+j)} = 1$ if the edge e_j is incident on the node u_i for all $(i, j) \in [n] \times [m]$. Remaining A_{ij} are set to 0. Then,

$$(G, k) \to (A, k) \tag{1}$$

defines a transformation.

Time-Complexity: The matrix A can easily be filled in $O(|V| \cdot (|V| + |E|)|)$ time which is polynomial in the size of input.

Equivalence: We want to show that (G, k) is a Yes instance of IS(D) if and only if (A, k) is a Yes instance of $X(D)$.

($\Rightarrow$) Suppose (G, k) is a Yes instance of IS(D). Then, $\exists\ S' \subset V$ an independent set with $n \geq |S| \geq k$. This implies that $\exists\ S \subset S'$ an independent size of size exactly $k \leq n$.

WLOG assume $S = \{u_1, u_2 \dots u_k\}$. We claim that $T = \{c_1, c_2 \dots c_k\}$ form an orthogonal subset of customers.

Suppose T is not orthogonal, hence, we have $c_i,\ c_j \in S$ for some $i, j \in [k]$ and $t \in [m]$ such that $A_{i,(n+t)} = A_{j,(n+t)} = 1$. This is possible only if the edge e_k is incident on both u_i and u_j, and since an edge is incident on exactly two node, u_i and u_j are connected by an edge, which contradicts the fact that S is an independent set. Hence, we must have that T is orthogonal and since $|T| = k \geq k$, we have (A, k) as a Yes instance of $X(D)$.

($\Leftarrow$) Suppose (A, k) is a yes instance of $X(D)$. Hence, we have an orthogonal subset of customers of size atleast $k \leq n$. Therefore, we also have an orthogonal subset of size exactly k, WLOG call it $T = \{c_1, c_2 \dots c_k\}$. We claim that $S = \{u_1, u_2 \dots u_k\}$ is an independent set of G. Suppose not, then again for some $i, j \in [n]$, we have $t \in [m]$ such that e_t is an edge connecting u_i and u_j. Clearly in such case we must have $A_{i,(n+t)} = A + j, (n + t) = 1$ which contradicts the fact that T is orthogonal. Hence, we must have that S is independent set and because $|S| = k \geq k$, (G, k) is a Yes instance of IS(D).

Therefore, we have shown that $X(D) \in$ NP and IS(D) $\leq_p X(D)$ and since IS(D) is NP-Complete, so is $X(D)$. Therefore, there does not exists a polynomial time algorithm for $X(D)$ unless P=NP, or equivalently, there is no polynomial time algorithm for original problem X unless P=NP.

4. (38 points) Formulate linear or integer programs for the following optimization problems.

   (a) (10 points) Min-cost flow: Given a flow network with capacities c_e and costs a_e on every edge e, and supplies s_i on every vertex i, find a feasible flow $f : E \to \mathbb{R}^+$ — that is, a flow satisfying edge capacity constraints and node supplies—that minimizes the total cost of the flow.

   Solution: Let $G = (V, E, c, s, a)$ be the given flow network with edge capacity and node supply constraints as c and s respectively and cost of unit flow across edges given by a. We want a flow f in G such that it is feasible and minimizes the cost.

   Let $f(e) \geq 0$ represent the flow across edge e, we want to minimize $\sum_{e \in E} a(e)f(e)$.
   From our knowledge of min-cost flow, we can formulate this problem as a linear program as follows:

$$\min_{f(e) \geq 0 \ \forall e \in E} \sum_{e \in E} a(e)f(e)$$

   such that,

$$f(e) \leq c(e) \quad \forall\, e \in E$$

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) = s(v) \quad \forall\, v \in V$$

$$\sum_{v : s(v) > 0} \Big(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in } v} f(e) \Big) = \sum_{v : s(v) < 0} \Big(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in } v} f(e) \Big)$$

   (b) (14 points) The assignment problem: There are n persons and n jobs that have to be matched on a one-to-one basis. There is a given set A of ordered pairs (i, j), where a pair (i, j) indicates that person i can be matched with job j. For every pair $(i, j) \in A$, there is a value a_{ij} for matching person i with job j. Our goal is to assign persons to jobs so as to maximize the total value of the assignment.

Solution: Let $\forall (i,j) \in A$

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned to job } j \\ 0 & \text{otherwise} \end{cases}$$

We want a valid assignment that maximizes $\sum_{(i,j)\in A} a_{ij}x_{ij}$. Hence, we can formulate this as an Integer Linear Program as follows:

$$\max_{x_{ij}:(i,j)\in A} \sum_{e\in E} a_{ij}x_{ij}$$

such that,

$$\sum_{j:(i,j)\in A} x_{ij} = 1 \;\; \forall i \in [n] \ni \exists l \in [n] \text{ with } (i,l) \in A$$
$$\sum_{i:(i,j)\in A} x_{ij} = 1 \;\; \forall j \in [n] \ni \exists l \in [n] \text{ with } (l,j) \in A$$
$$x_{ij} \in \{0,1\} \;\; \forall (i,j) \in A$$

(c) Exact-3SAT: Given a formula ϕ with n variables and m clauses where each clause consists of exactly three distinct literals, is the formula satisfiable?

Solution: Let ϕ be the CNF formula consisting of $\{x_1, x_2 \dots x_n\}$ literals and m clauses $C_1, \dots C_m$, with each clause consisting of exactly three literals (their negations included). For each literal x_i, introduce a variable $b_i \in \{0,1\}$. Then, for each clause consisting of x_i, x_j, x_k: $C_t = (p \cup \cup q \cup r)$, where p, q, r are placeholders for x_i, x_j, x_k respectively and including their negations, add the constraint:

$$c_i + c_j + c_k = 1$$

where $c_i = b_i$ if $p = x_i$ else $c_i = 1 - b_i$, and similarly for c_j, c_k
Then, whether or not ϕ is satisfiable can be reformulated as a feasibility ILP problem:

$$\max_{b_i \;\forall i\in[n]} 0$$

subject m constraints added by each clause explained above.
Additional constraint: $b_i \in \{0,1\} \;\; \forall i \in [n]$

Note that these are $m+1$ constraints, clearly polynomial in input size.