Name: Prasham Dhaneshbhai Sheth
UNI: pds2136

HW:4 — Theoretical part

1. **Solution to problem 1**

   In formulating flow problems, we assumed that the input graphs are simple directed graphs, that is, there is at most one directed edge between any pair of nodes. Now suppose that we allow multi-edges, that is, there may be be multiple directed edges going from node i to node j, each possibly with different capacities and weights.

   We need to reduce the problem of computing a min-cost circulation in a flow network with demands, costs and multi-edges to the problem of computing a min-cost circulation in a standard flow network with demands and costs.

   For this we propose following reduction by which we aim to reduce the input for multi-edge problem to that of the standard flow network.

   Input to problem of computing a min-cost circulation in a flow network with demands, costs and multi-edges: A graph having multi-edges, demands corresponding to each node, cost corresponding to each of the edges.

   Input to problem of min-cost circulation in a standard flow network with demands and costs: A graph with single edge between two vertices, demands corresponding to each node, cost corresponding to each of the edges.

   Thus, the reduction aims at converting the Input to problem of computing a min-cost circulation in a flow network with demands, costs and multi-edge to Input to problem of min-cost circulation in a standard flow network with demands and costs such that equivalence of instances is maintained.

   Equivalence of instances can be given as follows:

   "A circulation f' of cost c' that satisfies demands capacities and minimizes the total cost of flow exists in the graph G' having multi-edges if and only if a circulation f with cost c' exists in the equivalent standard flow network G that satisfies demands capacities and minimizes the total cost of flow"

   Steps for Reduction:

   For every edge (i,j) in the input network G' add 2 edges and a node in the new network G, first of which goes from i to the new node and has capacity equal to the capacity of the edge (i,j) and cost also has to be kept the same as (i,j). The second edge goes from the new node to j and has capacity same as that of the edge (i,j) and cost is set to 0 for this particular edge. The demand of the new node has to be set to 0 as we require all the flow entering that new node to exit the node.

   Thus G has all the nodes in G' plus one new node corresponding each edge in the network G' and the edges in the network G are added as described in the previous paragraph.

Proof for equivalence of instances:

(a) Forward direction (=>): Given that we have a valid circulation f' having minimum cost c' in network G', we have a valid circulation f with the minimum cost of c' in the equivalent flow network G.

As we have the valid circulation f' in the multi-edge network G' we can construct a valid circulation f in network G by allowing the flow value in 2 particular edges in network G be equal to that in the corresponding edge in G'.

The flow values such assigned would have cost same as that of the circulation in G' as the flow in edges(i,j) of G' would result in same flow value in edges (i,k) and (k,j) where k is the newly added node in network G for the edge (i,j) in network G'. The total cost to flow from (i,j) in both networks would be same as in network G (i,k) has same cost and same flow value whereas the edge (k,j) would have the same flow value but has zero cost so doesn't contribute to overall cost in the network.

The capacity constraint would also be satisfied as we send the flow of equal value and capacities are set to the same values as in network G and G'

The demands would also be satisfied as the same amount of total flow enters and exits any given node in G as that in the network G'. For all the newly added nodes, the amount of flow entering would be equal to amount of flow exiting as we set the flow values in both the edges to be the same and hence the f(in) = f(out) for all of the newly added nodes and hence demand for those node that is 0 would be satisfied.

Thus, we can have an equivalent circulation f in G given that we have a valid circulation f' in G' and both of them would have same minimum cost of c'

(b) Backward direction (<=): Given that we have a valid circulation f having minimum cost c' in network G, we have a valid circulation f' with the minimum cost of c' in the equivalent flow network G'.

As we have the valid circulation f in the standard circulation network G we can construct a valid circulation f' in network G' by allowing the flow value in a particular edge in network G' be equal to that in the corresponding edges in G.

The flow values such assigned would have cost same as that of the circulation in G as the flow in edges(i,j) of G' would be having same flow value as that in edges (i,k) and (k,j) where k is the newly added node in network G for the edge (i,j) in network G'. The total cost to flow from (i,j) in both networks would be same as in network G (i,k) has same cost and same flow value whereas the edge (k,j) would have the same flow value but has zero cost so doesn't contribute to overall cost in the network and as we allow the flow value to be same in G' as well the costs of both networks would be equal.

The capacity constraint would also be satisfied as we send the flow of equal value and capacities are set to the same values as in network G and G'

The demands would also be satisfied as the same amount of total flow enters and exits any given node in G' as that in the network G.

Thus, we can have an equivalent circulation f' in G' given that we have a valid circulation f in G and both of them would have same minimum cost of c'

Thus, once we have the network G we can use the algorithm to find circulation with the minimum cost that satisfies the demand and capacity constraints. Lets say the cost for the circulation comes out to be c' then we can say get the valid circulation with the cost of c; in original input network G' by allowing the flow between (i,j) equal to the flow in corresponding edges in G. By the equivalence of constraints we know that this circulation would be a valid circulation having a minimum cost.

2. **Solution to problem 2**

   Stingy SAT is the following problem: on input a SAT formula ϕ and an integer k, is there a satisfying assignment for ϕ in which at most k variables are True

   To prove: Prove that Stingy SAT is NP-complete.

   For proving Stingy SAT is NP-complete we need to show:

   (1) Stingy SAT is in NP.
   (2) A known problem in NP-complete reduces to Stingy SAT.

   For the first (i.e) Stingy SAT is in NP, we need to show that an efficient certifier for Stingy SAT exists which on a given certificate can verify the validity of the instance. Here a certificate would be the permutations of values for each of variables and the certifier would check that in the given certificate, is there at most only k variables that are true and also whether the values result into satisfactory assignment. This can definitely be done efficiently as we can directly plug in the values to check the satisfiability and counting the number of variables to check whether at most k variables have true values both the tasks can be done efficiently.

   For the second part (i.e) A known problem in NP-complete reduces to Stingy SAT, we show that SAT (which is known to be in NP-Complete) reduces to Stingy SAT.

   Reduction Transformation:

   - The input to SAT problem is a formula ϕ with n variables.

   - The input to Stingy SAT problem is a formula ϕ' with n' variables and a value k.

   We need to show SAT reduces to Stingy SAT and hence need to convert the input to SAT problem to an equivalent input to Stingy SAT problem and prove the equivalence of instance between both of them.

   Transformation of input: Given a SAT formula ϕ with n variables we simply choose (ϕ ; n) as an input of Stingy SAT.

   Equivalence of instances: ϕ is a yes instance of SAT if and only if (ϕ,n) is a yes instance of Stingy SAT.

   To prove equivalence of instances we show:

   (a) Forward direction (=>): Suppose that ϕ is a yes-instance of SAT. The formula has n variables and out of these n variables at most n can be true. Hence, we can say that no more than n variables are true for the given satisfiable formula ϕ. So any satisfying assignment of instance ϕ for SAT will be a satisfying assignment of instance (ϕ,n) for Stingy SAT.

       So (ϕ,n) is a yes-instance of Stingy SAT, given that ϕ is a yes-instance of SAT.

(b) Backward Direction (<=): Suppose that (ϕ,n) is a yes-instance of Stingy SAT. Any satisfying assignment of that instance will be also a satisfying assignment of instance ϕ for SAT. So ϕ is a yes-instance of SAT. This follows intuitively as we can set the same values for each of the variables and the formula would result into true as that condition is being checked in the Stingy SAT problem already.

So ϕ is a yes-instance of SAT, given that (ϕ,n) is a yes-instance of Stingy SAT.

Hence we can say that as Stingy SAT is in NP and SAT which is a NP-Complete problem can be reduced to Stingy SAT, Stingy SAT is a NP-Complete problem.

3. **Solution to problem 3**

Suppose we had a polynomial-time algorithm that, on input a graph, answers yes if and only if the graph has a Hamiltonian cycle.

We want to show that on input a graph G = (V;E), we can return in polynomial time - a Hamiltonian cycle in G, if one exists, - no, if G does not have a Hamiltonian cycle.

This can be done by following the below mentioned steps with an idea that removing an edge which is not a part of Hamiltonian Cycle would result in a graph with Hamiltonian cycle and removing an edge that is a part of the Hamiltonian cycle would result in a graph having no Hamiltonian cycle.

Let P be a algorithm which takes graph G as an input and returns True if the graph has a Hamiltonian cycle and returns False if it doesn't have a Hamiltonian cycle.

We here run the algorithm P for graph by removing an edge from the graph and check whether the resultant graph has a Hamiltonian cycle or not. If P return False we know that the edge is a part of Hamiltonian cycle and hence we append it to set of edges forming Hamiltonian cycle and if the algorithm P return True we know that the specific edge e is not a part of Hamiltonian cycle and we ignore the edge then.

Algorithm 1

```
Hamiltonian_Cycle(G(V, E))
1: if P(G(V,E)) == False then
2:     return No Hamiltonian Cycle
3: else
4:     E' = { }
5:     for every edge e in E do
6:         if P(G(V, E - {e})) == False then
7:             $E' = E' \bigcup \{e\}$
8:         end if
9:     end for
10:    return The set E' edges which would form the Hamiltonian Cycle.
11: end if
```

The given algorithm calls algorithm P once for each and every edge and hence we can consider the given algorithm to be polynomial in order of number of edges of the given input graph provided that the algorithm P is a polynomial time algorithm.

Proof of correctness of given algorithm follows from the fact that removing an edge which is not a part of Hamiltonian Cycle would result in a graph with Hamiltonian cycle and removing an edge that is a part of the Hamiltonian cycle would result in a graph having no Hamiltonian cycle. We here are checking for each edge whether it is a part of Hamiltonian cycle or not.

In the case that there is no Hamiltonian Cycle we return No by default which is determined by a call for Algorithm P on the original graph. In the other case we follow the else part of the algorithm whose proof of correctness is as shown below:

- Initialization: The set E' is initialized as an empty set.
- Loop Termination: As the for loop iterates through each and every edge and we have a fixed number of edges it is for sure to terminate.
- Loop invariant: At end of any for loop E' would have all the edges which are checked and are a part of Hamiltonian Cycle.

At the termination of the loop we would have checked all edges and would have all the edges that form the Hamiltonian cycle in E' and hence the algorithm would correctly return the edges that form Hamiltonian cycle.

4. **Solution to problem 4**

The given problem does not seem to be solvable in polynomial time. We hence give the decision version for the problem and prove that it is a NP=Complete problem.

The given problem: Given a set of ground elements E = {e1; e2;...; en} and a collection of m subsets {S1; S2; ... ; Sm} of the ground elements we need to select a minimum cardinality set A of ground elements such that A contains at least one element from each subset Si.

The decision version of the problem: Given a set of ground elements E = {e1; e2;...; en} and a collection of m subsets {S1; S2; ... ; Sm} of the ground elements and an integer k, does a set A(subset of U), with size at most k which has intersection with all S1, S2, , Sm, exists?

For proving the decision version of the above problem is NP-complete we need to show:

(1) Decision version is in NP.
(2) A known problem in NP-complete reduces to the decision version as defined above.

For the first (i.e) Decision version is in NP, we need to show that an efficient certifier for Decision version exists which on a given certificate can verify the validity of the instance. Here the certificate would be a subset of ground elements(A) and the certifier would require to check that there exists an intersection between each of the Subsets S_i and A as we require that A contains at least one element from each subset S_i Also it checks for the size of the subset and performs a validation for its size not being greater than k. This can be done efficiently in polynomial time and hence we can say that the decision version of the problem is in NP.

For the second part we would reduce the vertex cover problem which we know is a NP-Complete problem to the decision version of the problem on hand.

Transformation:

Input to the decision version of the problem: a set of ground elements E = {e1; e2;...; en} and a collection of m subsets {S1; S2; ... ; Sm} of the ground elements and an integer k, Input to decision version of vertex cover: A graph G(V,E), an integer k'

We are thus, required to transform the input to vertex cover to an equivalent input to decision version of the problem

We do it by the following steps:

(1) Set V = $\{v_1; ....; v_n\}$ to be the set of ground elements.

(2) $S_i = \{u, v\}$; where (u, v) is an edge of G.

(3) set k = k'

Equivalence of instances: A subset of ground elements of size at most k exists if and only if there exists a vertex cover of size k in the graph G.

Proof for equivalence of instances:

To prove the equivalence of instances we need to show

(a) Forward direction (=>): If C is a vertex cover for G of size k: by definition then, for every edge (u, v) in G either u C or v C. C is thus solution subset because it must intersect every set S_i.

    Thus, we can say that if we have a vertex cover of size k in the graph G,we also have a subset of ground elements having size k that has at least one common element with each of the subsets S_i. The elements in the subset A would be those corresponding to the vertices in the set of vertex cover.

(b) Backward Direction (<=): Now suppose A is subset of E of size k whose intersection with each of the subsets S_i is not a null set. Since A intersects every set, it has at least one endpoint of every edge. Hence we can say that A also represents a vertex cover for the graph G.

    Thus, we can say that given a subset A of ground elements having size k that has at least on element in common with each of the given subsets, we also have a vertex cover in graph G with size k. The set of vertex cover would have the vertices corresponding to the elements of set A.

Thus, we can say that decision version of the vertex cover can be reduced polynomially to decision version of our problem.

Hence, we say that the decision version of our problem is a NP-complete problem.

5. **Solution to problem 5**

We have m employees and n positions. Also, as given n ¿ m. The goal of the manager is to assign jobs to employees so that the sum of the scores of the employees who are assigned to jobs is maximized. A single job cannot be assigned to more than one employee and a single employee may not be assigned to more than one job.

He has assigned a score s_{ij} to each employee i for each position j they are willing to accept reflecting how qualified employee i is for position j. We assume that for all employees who are not interested in the jobs the value of s_{ij} is zero.

i We want to formulate an Integer Program for this problem of manager assigning the jobs to employees such that the score is maximized. We introduce an indicator variable that shows whether or not the j^{th} job was allocated for i^{th} employee. Here, $x_{ij} = 1$ if i^{th} employee was assigned j^{th} job and 0 otherwise.

  - The formulated integer program thus can be shown as below:

  $maximize\ \Sigma\ x_{ij} * s_{ij}$ where $1 <= i <= m; 1 <= j <= n$.

  subject to:

  (a) $x_{ij} \in \{0, 1\}$

  (b) $\Sigma\ x_{ij} <= 1$, Summation over j for each i.

  (c) $\Sigma\ x_{ij} <= 1$, Summation over i for each j.

ii We can view this problem as a problem of matching m employees to n jobs such that the summation of scores is maximized along with a constrain of matching one job to an employee as well as only one employee to a job.

  Thus we can see this thing as a formulation of bipartite graph with a goal to find the matching with maximum sum of weights for the edges in matching where weight of the particular edge represents the score.

  Thus, we would have m employees in one set which we call X and n jobs in another set which we call Y. Thus, the graph would have V = X $\bigcup$ Y vertices and set of edges E which would join each of the m employees to the jobs they qualify for and each of the edge would have a weight which represents the score for that employee and the job position. Thus edge between i^{th} employee and j^{th} job will have weight of s_{ij}.

  Our task is to find a matching M which would be a subset of E such that each vertex of X and Y both is incident to only one edge and the sum of weights for those edges in the subset is maximum.

  The above IP thus, finds a matching M which would be a subset of E such that each vertex of X and Y both is incident to only one edge and the sum of weights for those edges in the subset is maximum. The indicator variable x_{ij} denotes whether the particular edge is in the matching set M or not. The constraint (b) and (c) takes into consideration the fact that there should be only one edge between any vertex in set X and Y. The objective function represents the summation of weights of the edges in the matching M which we want to maximize.

iii Now suppose that $s_{ij} = 1$ for every employee i that qualifies for a job j. Thus the above IP then would simplify to the following:

  $maximize\ \Sigma\ x_{ij}$ where $1 <= i <= m; 1 <= j <= n$.

  subject to:

  (a) $x_{ij} \in \{0, 1\}$

  (b) $\Sigma\ x_{ij} <= 1$, Summation over j for each i.

  (c) $\Sigma\ x_{ij} <= 1$, Summation over i for each j.

  Hence, as can be shown the task now in terms of graph theoretic terms is to find a matching of size m in the bipartite graph of vertices V = X $\bigcup$ Y where vertices in X represent m employees and vertices in Y represents n jobs.

iv m = 2, n = 3, $s_{ij} = 1$ for all $1 <= i <= 2; 1 <= j <= 3$.

i The IP thus would be as formulated as shown below:
It would have 6 variables and 5 constraints.
$maximize\ x_{11} + x_{12} + x_{13} + x_{21} + x_{22} + x_{23}$ where $[x_{ij} \in \{0,1\}; 1 <= i <= 2; 1 <= j <= 3.]$
subject to:
(a) $x_{11} + x_{12} + x_{13} <= 1$
(b) $x_{21} + x_{22} + x_{23} <= 1$
(c) $x_{11} + x_{21} <= 1$
(d) $x_{12} + x_{22} <= 1$
(e) $x_{13} + x_{23} <= 1$
The relaxed LP for the above formulated IP can be shown as follows:
$maximize\ x_{11}+x_{12}+x_{13}+x_{21}+x_{22}+x_{23}$ where $0 <= x_{ij}; 1 <= i <= 2; 1 <= j <= 3.$
subject to:
(a) $x_{11} + x_{12} + x_{13} <= 1$
(b) $x_{21} + x_{22} + x_{23} <= 1$
(c) $x_{11} + x_{21} <= 1$
(d) $x_{12} + x_{22} <= 1$
(e) $x_{13} + x_{23} <= 1$
For finding the dual of the above LP we follow the 7-Step procedure that was introduced in the class.
Step-1: work with a minimization problem
$minimize\ -x_{11}-x_{12}-x_{13}-x_{21}-x_{22}-x_{23}$ where $0 <= x_{ij}; 1 <= i <= 2; 1 <= j <= 3.$
Step 2: rewrite the LP in a convenient form
$minimize\ -x_{11}-x_{12}-x_{13}-x_{21}-x_{22}-x_{23}$ where $0 <= x_{ij}; 1 <= i <= 2; 1 <= j <= 3.$
subject to:
(a) $x_{11} + x_{12} + x_{13} - 1 <= 0$
(b) $x_{21} + x_{22} + x_{23} - 1 <= 0$
(c) $x_{11} + x_{21} - 1 <= 0$
(d) $x_{12} + x_{22} - 1 <= 0$
(e) $x_{13} + x_{23} - 1 <= 0$
Step 3: introducing the dual variables
We introduce 5 dual variables for constraints (a) - (e):
y1, y2, y3, y4, y5 each of which would be non negative as they refer to an inequality constraint.
Step 4: maximizing the sum of everything
$maximize\ over\ y_k s\ and\ minimize\ over\ x_{ij} s:$

$$-x_{11} - x_{12} - x_{13} - x_{21} - x_{22} - x_{23}$$

$$+y_1 * (x_{11} + x_{12} + x_{13} - 1)$$

$$+y_2 * (x_{21} + x_{22} + x_{23} - 1)$$

$$+y_3 * (x_{11} + x_{21} - 1)$$

$$+y_4 * (x_{12} + x_{22} - 1)$$

$$+y_5 * (x_{13} + x_{23} - 1)$$

where $0 <= x_{ij}; 1 <= i <= 2; 1 <= j <= 3.$ and $0 <= y_k; 0 <= k <= 5$

Step 5: grouping terms by primal variables

maximize over y_k*s and minimize over* x_{ij}*s* :

$$-y_1 - y_2 - y_3 - y_4 - y_5$$

$$+x_{11} * (y_1 + y_3 - 1)$$

$$+x_{12} * (y_1 + y_4 - 1)$$

$$+x_{13} * (y_1 + y_5 - 1)$$

$$+x_{21} * (y_2 + y_3 - 1)$$

$$+x_{22} * (y_2 + y_4 - 1)$$

$$+x_{23} * (y_2 + y_5 - 1)$$

where $0 <= x_{ij}; 1 <= i <= 2; 1 <= j <= 3.$ and $0 <= y_k; 0 <= k <= 5$

Step 6: eliminating primal variables to get the dual LP

maximize : $-y_1 - y_2 - y_3 - y_4 - y_5$ where $0 <= y_k; 0 <= k <= 5$

subject to:

(a) $(y_1 + y_3 - 1) >= 0$

(b) $(y_1 + y_4 - 1) >= 0$

(c) $(y_1 + y_5 - 1) >= 0$

(d) $(y_2 + y_3 - 1) >= 0$

(e) $(y_2 + y_4 - 1) >= 0$

(f) $(y_2 + y_5 - 1) >= 0$

Step 7: As original LP was a maximization one we rewrite the dual as a minimization one.

minimize : $y_1 + y_2 + y_3 + y_4 + y_5$ where $0 <= y_k; 0 <= k <= 5$

subject to:

(a) $(y_1 + y_3 - 1) >= 0$

(b) $(y_1 + y_4 - 1) >= 0$

(c) $(y_1 + y_5 - 1) >= 0$

(d) $(y_2 + y_3 - 1) >= 0$

(e) $(y_2 + y_4 - 1) >= 0$

(f) $(y_2 + y_5 - 1) >= 0$

ii Assuming that the dual LP has an integral solution.

The dual LP finds a minimum vertex cover - it minimizes the number of vertices under the constraint that for each edge, at least one of its endpoints is in the cover.

The objective function *minimize* : $y_1 + y_2 + y_3 + y_4 + y_5$ can be thought of as being composed of two sets of variables where the y corresponding to k = 1,2 are representing the 2 employees whereas the ones with value of k = 3, 4, 5 represent the 3 jobs. Thus it is similar to problem of selecting as minimum number of vertices as possible that cover the edges that come up in the matching and make the graph a bipartite one. The selection of minimum number of vertices here thus, would give the same answer as the primal that is to find out the maximum matching from the given graph as we have here assumed that the weights of all edges is equal to 1.

The variables in the dual are encoding the selection of *ith* vertex where all the vertices are composed of union of two sets (1) set of vertices corresponding to all the employees and (2) set of vertices corresponding to all the jobs. The constraints take into consideration the necessity for existence of edge between two of the selected vertices as they need to have an edge between them to make them a part of the matching.
This is similar to vertex cover as we select the vertices from a graph such that they cover all the edges of the graph and here we need to find out the vertex cover of minimum size and hence we use y to be an indicator variable which takes value 1 with selection of a vertex to be a part of the vertex cover and 0 otherwise and minimize the sum of all y and hence would find the vertex cover of minimum size.
Thus, it is the case where we find minimum number of vertices such that for each edge at least one of the two vertices must be picked.