CSCI 570 - HW 5

1. **Maximizing Profit**
   A furniture company produces three types of couches. The first type uses 1 foot of framing wood and 3 feet of cabinet wood. The second type uses 2 feet of framing wood and 2 feet of cabinet wood. The third type uses 2 feet of framing wood and 1 foot of cabinet wood. The profit of the three types of couches is \$10, \$8, and \$5, respectively. The factory produces 500 couches each month of the first type, 300 of the second type, and 200 of the third type. However, this month there is a shortage of cabinet wood and the supply to the factory will have to be reduced by 600 feet, but the supply of framing wood is increased by 100 feet. How should the production of the three types of couches be adjusted to minimize the decrease in profit? Formulate this problem as a linear programming problem.

   Solution: The above problem can be reformulated in a

Type of Couch	Framing Wood	Cabinet Wood	Profit
Couch A	1ft	3ft	\$10
Couch B	2ft	2ft	\$8
Couch C	2ft	1ft	\$5

To Do: Minimize the decrease in profit
As we need to figure out the change in the production of each type of couch, Let us say x_1, x_2 *and* x_3 be the change in the production for couch A, couch B and couch C produced each month.
Based on the problem statement, we can formulate our constraints as below:

$x_1 + 2x_2 + 2x_3 \leq 100$, as there is an increase in the framing wood supply by 100ft, this equation tell us about the change in framing wood(positive means increase in production)

$3x_1 + 2x_2 + x_3 \leq -600$, as there is a shortage of cabinet wood supply by 600ft, this equation tells us about the change in cabinet wood(negative means decrease in production)

$x_1 \geq -500,\ x_2 \geq -300,\ x_3 \geq -200$

Objective function: $max(10x_1 + 8x_2 + 5x_3)$

Subject to, $x_1 + 2x_2 + 2x_3 \leq 100$

$3x_1 + 2x_2 + x_3 \leq -600$

$x_1 \geq -500,\ x_2 \geq -300,\ x_3 \geq -200$

In order to reformulate the above as a standard maximum problem,
As we are calculating the change in the production,

$$X = X' + \begin{pmatrix} -500 \\ -300 \\ -200 \end{pmatrix}$$

Where $X' \geq 0$ *and* $X \geq n$ where,

$$n = \begin{pmatrix} -500 \\ -300 \\ -200 \end{pmatrix}$$

We can now go ahead and reframe our objective equation we get,

$c^T X = c^T(X' + n)$

$= c^T X' - 8400$

and A$(X' + n) \leq b$

$AX' \leq b - Ad$

$$= \begin{pmatrix} 1600 \\ 1800 \end{pmatrix} = b'$$

Hence, at the end we get,

$maximize(c^T X')$

subject to, $AX' \leq b'$

$X' \geq 0$

2. Dual Program

Consider the following linear program:

$$max(3x_1 + 2x_2 + x_3)$$

Subject to,

$x_1 - x_2 + x_3 \leq 4$

$2x_1 + x_2 + 3x_3 \leq 6$

$-x_1 + 2x_3 = 3$

$x_1 + x_2 + x_3 \leq 8$

$x_1, x_2, x_3 \geq 0$

Write the dual problem.

Solution: The dual problem for the above question,

All constraints apart from this constraint: $-x_1 + 2x_3 = 3$ are in standard form, hence this equality constraint needs to be converted to an inequality as below:

$$-x_1 + 2x_3 \leq 3 \text{ and } x_1 - 2x_3 \leq -3$$

Now using the below equations we formulate the matrices,

$$x_1 - x_2 + x_3 \leq 4$$
$$2x_1 + x_2 + 3x_3 \leq 6$$
$$-x_1 + 2x_3 \leq 3$$
$$x_1 - 2x_3 \leq -3$$
$$x_1 + x_2 + x_3 \leq 8$$

$$A = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 1 & 3 \\ -1 & 0 & 2 \\ 1 & 0 & -2 \\ 1 & 1 & 1 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & 2 & -1 & 1 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ 1 & 3 & 2 & -2 & 1 \end{pmatrix}$$

$$b = \begin{bmatrix} 4 \\ 6 \\ 3 \\ -3 \\ 8 \end{bmatrix}$$

$$c = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

$$b^T = \begin{bmatrix} 4 & 6 & 3 & -3 & 8 \end{bmatrix}$$

Now based on the above matrices we formulate our dual as,

$min(b^T y)$ subject to,

$A^T y \geq c$

$y \geq 0$

Dual:

Objective function: $min(4Y_1 + 6Y_2 + 3Y_3 - 3Y_4 + 8Y_5)$

subject to, $Y_1 + 2Y_2 - Y_3 + Y_4 + Y_5 \geq 3$

$-Y_1 + Y_2 + Y_5 \geq 2$

$Y_1 + 3Y_2 + 2Y_3 - 2Y_4 + Y_5 \geq 1$

$Y_1 \geq 0, Y_2 \geq 0, Y_3 \geq 0, Y_4 \geq 0, Y_5 \geq 0$

3. **Spectrum Management**

Spectrum management is the process of regulating the use of radio frequencies to promote efficient use and gain a net social benefit. Given a set of broadcast emitting stations $s_1,...,s_n$, a list of frequencies $f_1,...,f_m$, where $m \geq n$, and the set of adjacent stations $E = \{(s_i,s_j)\}$. The goal is to assign a frequency to each station so that adjacent stations use different frequencies and the number of used frequencies is minimized. Formulate this problem as an integer linear programming problem.

Solution:

To Do: To assign a frequency to each station so that adjacent stations use different frequencies and the number of used frequencies is minimized.

Let F=$\{f_1,f_2,f_3...f_n\}$ and in our case all vertices v that belong to V are stations.

For every vertex v(*station s*) and available frequency f, we give associate each vertex to a frequency in the below manner

$s_{vf} = 1,\ if\, frequency\, f\ \in F\ is\ assigned\ to\ station\ s$

0, otherwise.

We also bring in another variable m_f, in order to denote the use of a frequency *f.*

$m_f = 1$, if frequency $f \in F$ is used up

0, otherwise.

As we need to minimize the number of frequencies used, we formulate the below objective function:

$$min \sum_{f \in F} m_f$$

subject to below constraints,

$$\sum_{f \in F} s_{vf} = C_v,\ For\ all\ v(stations\ s)\ belongs\ to\ S$$

We have another constraint, wherein we need to make sure that the adjacent stations, don't have the same frequency assigned, hence

$s_{vf} + s_{wg} \leq 1,\ For\ \{v,w\}\ that\ belongs\ to\ E, f\ \in F, g\ \in F, f \neq g,\ v \neq w$

$s_{vf} \leq m_f$, For all vertices v*(stations s)* $\in S, f \in F$

$s_{vf} \in \{0,1\}\ For\ all\ vertices\ v(stations\ s)\ that\ belong\ to\ S,\ f\ \in F$

$m_f \in \{0,1\}\ For\ all\, f\ \in F$

Here, C_v frequencies need to be assigned each edge connecting vertex v(*ie each station s*) ie the set of stations in E.

4. **Short Questions**

True or false? Shortly explain your answers.

a. $If\ A \leq_p B\ and\ B\ is\ in\ NP-Hard,\ then\ A\ is\ in\ NP-Hard.$

Solution: False

If $A \leq_p B$, then this would mean that B is at least as hard as A or reframing it in other words, A is not harder than B and hence it's not necessary for A to be *NP-hard.*

b. $If\ A \leq_p B\ and\ B\ is\ in\ NP,\ then\ A\ is\ in\ NP.$

Solution: True

If $A \leq_p B$ and B is in NP then it would mean that B is at least as hard as A, then this would mean that A needs to belong to at least NP class as we are reducing A to B using a polynomial time reduction. Hence, a Non-Deterministic Polynomial time Turing machine that solves A will be in NP iff B has a Non-Deterministic Polynomial time Turing machine that has a certificate that accepts f(w) $\in B\ if\ w \in A$, such that it has a certificate, that can be checked by a polynomial time deterministic Turing Machine.

c. $If\ 3 - SAT \leq_p 2 - SAT,\ then\ P = NP.$

Solution: True

We know that $2 - SAT$ belongs to P class and if $3 - SAT \leq_p 2 - SAT$, this means that there is a polynomial time reduction from 3-SAT to 2-SAT, and as we know that $3 - SAT$ is a NP-Complete problem it must 3-SAT must be NP hard and must belong to NP class. Hence, we are reducing a NP problem to a problem that belongs in P class(2-SAT) ie we know that 2 - SAT is easy and $\in P,\ hence\ Y = 3 - SAT\ becomes\ easy$ and we can can tell that $P = NP$.

d. $Any\ NP\ problem\ can\ be\ solved\ in\ time\ O^{(2poly(n))},\ where\ n\ is\ the\ input\ size\ and\ poly(n)\ is\ a\ polynomial.$

Solution: True

Any NP problem can be solved in $time\ O^{(2poly(n))}$ as every NP-Complete problem can be solved in exponential time complexity and since every NP-Complete problem is in NP ,which would mean that there is a certificate that can be verified in polynomial time by a deterministic Turing machine and as we also know that, $PSPACE \subseteq EXPTIME$ we can tell that any NP problem can be solved in $O^{(2poly(n))}$.

e. $If\ a\ problem\ A \leq_p B\ then\ it\ follows\ that\ B \leq_p A.$

Solution: False

This is not true as if A is polynomially reducible to B, it means that A is simpler than B, ie. not as hard as B which means that if we could solve B then we can solve A. Now assuming, A is hard then, A $\in NP$, then B is at least as hard as A. If it would follow that $B \leq_p A$, it might be the case that some problems are easier and it's not necessary for them to be hard hence the reduction doesn't hold. Another analogy would be, assuming B is in NP-Complete class that would mean it belongs to NP and is NP-Hard. Now if we assume that if $B \leq_p A$ was true, then it would mean that, any NP-Complete problem would be able to be solved in Polynomial time, which nobody has been able to prove.

5. **Finding a satisfying assignment**

Assume that you are given a polynomial time algorithm that given a 3-SAT instance decides in polynomial time if it has a satisfying assignment. Describe a polynomial time algorithm that finds a satisfying assignment (if it exists) to a given 3-SAT instance.

Solution:

Given: A polynomial time algorithm that decides in polynomial time if a given 3-SAT has a satisfying assignment.

To Do: Polynomial time algorithm that finds a satisfying assignment(if it exists) to a given 3-SAT instance.

Let P be the polynomial time algorithm that decides in polynomial time if a given 3-SAT has a satisfying assignment.

Let P' be the Polynomial time algorithm that finds a satisfying assignment(if it exists) to a given 3-SAT instance.

Let $x_1, x_2, x_3, x_4 .....x_m$ *be input variables.*

We start with setting $x_1 = True$ and compute the boolean formula. Then, P' can call P on the boolean formula. Then, P decides if the formula(*original SAT*) is satisfiable or not. If it is satisfiable, then we have reduced the problem to finding a satisfying assignment for the boolean formula(*SAT*) with $x_2, x_3, x_4, x_5 .....x_m$ (let's call this *x'*) ie P' recursively calls itself. In case the boolean formula was not satisfied we set $x_1 = False$ and call P with these values. Hence, P' recursively calls itself again. Our algorithm, P' takes polynomial time as it makes a recursive call once for every input variable and as there are m variables there would be m recursive calls reducing the input size, thereby making the overall runtime complexity polynomial. Specifically, the time complexity would be $T(n) = T(n^k) + T(n')$; where $n' = x'$ *and some k where* $T(n)$ *denotes the runtime of* P' *on a SAT of length n.*

6. **Shipping Goods**

Given n positive integers $x_1,...,x_n$. The Partition Problem asks if there is a subset S ⊆ [n] such that:

$$\sum_{i \in S} x_i = \sum_{i \in S} x_i$$

It can be proven that the Partition Problem is NP-complete. You do not prove it, but rather use it in the following problem.

Assume that you are consulting for a shipping company that ships a large amount of packages overseas. The company wants to pack n products with integer weights $p_1, . . . , p_n$ into a few boxes as possible to save on shipping costs. However, they cannot put any number of products into a box due to the ship- ping capacity restriction. The total weight of products in each box should not exceed W ? You may assume that for each i-th product $p_i \leq W$. Your task is to advise the company if it can be placed into at most $k < n$ boxes. Show that the problem is NP-Complete by reduction from the Partition Problem.

Solution:

Given: Partition problem is NP-Complete; Package weights={ $p_1, . . . , p_n$};There are n products to be shipped wherein each i-th product has $p_i \leq W$.

To Do: Advise the shipping company if all the products can be shipped in at most $k<n$ boxes and show that Shipping Goods problem is NP-Complete.

In order to show that Shipping Goods problem is NP-Complete, we need to prove two things,

1. *Shipping Goods problem* $\in NP$
   We can formulate this as a decision problem, wherein given that we have placed all products in some *x* amount of boxes keeping the weight constraint in consideration, we can now go count and verify if the number of boxes used are *at most k < n (Assuming we fix up k and W).* This verification would take polynomial time as we need to count and compare a constant k(boxes) to the total number of products n.
2. *Shipping Goods problem* $\in NP-Hard$
   In order to prove *Shipping Goods problem* $\in NP-Hard$, we need to find a reduction,
   Below is the reduction we consider,

$Partition\ Problem \leq_p Shipping\ goods\ problem$

Here, as k and W are not given in the partition problem we can choose certain specific k to match our needs so that we can reduce the partition problem to the shipping good problem.
We can now go ahead and frame a claim for the above reduction,

Claim: Partition problem returns a valid subset S such that $\sum_{i \in S} x_i = \sum_{i \in S} x_i$ with respect to W iff shipping goods problem can be solved with at most k<n boxes.

(=>)

Given the partition problem has a valid solution, that would mean that there is a valid division of products keeping in mind the weight constraint such that $\sum_{i \in S} x_i = \sum_{i \in S} x_i$. Let us consider a specific case if we have we weights of 6 products as of $P = \{3,1,1,2,2,1\}$, we can split them into $S_1 = \{1,1,1,2\}$ and $S_2 = \{2,3\}$ using the Partition problem keeping W=6 into consideration. Hence, in this case we can see that we are able to place the products in k=2 boxes which means that we have arrived at a solution for the shipping problem for packaging products with at most $k < n$ boxes. Hence, we can tell that if there exists a solution for the shipping problem for this specific case of weights then we can generalize it to any given set of weights and use the partition problem to solve the shipping goods problems.

(<=)

Given the shipping problem has a valid solution ie. we can package *n* products into at most *k<n* boxes, which means that we are place all products { $p_1, \ldots, p_n$} into at most k boxes which would mean that we have chosen the packages with weights in such a way that given that we know that each box cannot exceed weight W, Considering if we package our products in the below way: $S_1 = \{1,1,1,2\}$ and $S_2 = \{2,3\}$ with W=6, we can see that this splitting into at most *k<n* boxes will will rise only when we are able to partition all our products in this fashion $\sum_{i \in S} x_i = \sum_{i \in S} x_i$ keeping in mind the W in consideration. If we partition the products any other way we would end up using many boxes ie. not k<n boxes thereby not helping in minimizing our shipping costs.

7. **Longest Path**

Given a graph G = (V,E) and a positive integer k, the longest-path problem is the problem of determining whether a simple path (no repeated vertices) of length k exists in a graph. Show that this problem is NP-complete by reduction from the Hamiltonian path.

Solution:

Given: G = (V,E) and a positive integer k.

To Do: Determining whether a simple path (no repeated vertices) of length k exists in a graph and showing the problem is NP-Complete.

In order to show Longest Path problem is NP-Complete, we need to prove two things,

1. $Longest\ Path \in NP$

   We formulate this as a decision problem, wherein we are looking for a Hamiltonian Path of given length k. Hence, we can verify this in polynomial time whether the path that we get is a simple path (no repeated vertices ie. visit every vertex once) and go ahead and check if *path length k=V-1* . This verification takes polynomial time as we need to traverse and count the edges in the path.

2. $Longest\ Path \in NP-Hard$

   In order to prove that $Longest\ Path \in NP-Hard$, we need to find a reduction,

   Below is the reduction we consider:

   $$Hamiltonian\ Path \leq_p Longest\ Path\ Problem$$

   We go ahead and frame our claim for the above reduction,

   Let us construct a Graph *G'=Graph G*

   Claim: G has a simple path of length *k* iff there exists a Hamiltonian Path in *G'*.

   Proof:

   (=>)

   Given G has a simple path i.e. a path that means its number of edges would be E=V-1=k thereby implying path length of k exists then this would mean that it visits every vertex only once resulting in a Hamiltonian Path ie we essentially traverse the path to realize that we get E=V-1 edges which constitutes as a Hamiltonian Path..

   (<=)

Given there exists a Hamiltonian Path in *G',* which means the path would visit every vertex exactly once implying that the number of edges covered in this path would be V-1 which in fact would mean that there exists a simple path in G of length *k.*

8. Helping with the COVID-19 Crisis

Given an integer k, a set C of n cities $c_1,...,c_n$, and the distances between these cities $d_{ij} = d(c_i,c_j)$, for $1 \leq i < j \leq n$, where d is the standard Euclidean distance. We want to select a set H of k cities for building mobile hospitals in light of the coronavirus outbreak. The distance between a given city c and the set H is given by $d(c,H) = \min_{h\in H} d(c,h)$. The goal is to select a set H of k cities that minimizes $r = \max_{c\in C} d(c,H)$. Namely, the maximum distance from a city to the nearest mobile hospital is minimized. Give a 2-approximation algorithm for this problem.

Solution:

Given: Integer k, Set C={$c_1,...,c_n$}, distances between each city $d_{ij} = d(c_i,c_j)$, distance between a given city c and the set H is given by $d(c,H) = \min_{h\in H} d(c,h)$

To Do: To select a set H of k cities that minimizes $r = \max_{c\in C} d(c,H)$

Our 2-approximation algorithm(ALG) for this problem is as below:

A. Let us choose a city randomly and add to set H to act as a center..
B. Next we choose a city that is farthest away from the current cities in H to become the next center. Farthest away essentially means, we select that city has maximum:
   Min[dist(c, c1), dist(c, c2), dist(c, c3), …. dist(c, ci)] where H={c1,c2,...ci}

We continue the above step until we find all the k cities.

C. At the end, if length of set H is $\leq$ k, we output the set H, else this would mean that there exists no set of k cities such that all cities are within the radius to some other city.

Essentially, in the above algorithm, we are choosing the the city to be added in H based on the fact that it is farthest away from the already chosen cities(centers), hence we need to show that the distance from the city to closest city is at most 2 * OPT(as the farthest point will be on the circumference of a circle, if we plot circles and keep picking cities)

Suppose, we have an optimal placement of hospitals in cities, let us call it H*. All the cities that the optimal hospitals cover is within *2r* of the hospital picked by our algorithm. Also each city that our algorithm picks is within a distance *r* to a city in H*. Hence, our algorithm picks at most k cities, and the distance of any city is at most 2*r* from each city.

Proof that the above algorithm is a 2-approximation algorithm:

Let us prove it using contradiction,

Let us assume that the distance from the farthest city to all cities in set H is $> 2 * OPT$, this would hence imply that the distances between each city in set H is also $> 2 * OPT$. This would further imply that we have k+1 points who have distances $> 2 * OPT$, between each pair of cities. Each point now has a center of the optimal solution with distance <= OPT to it. This would mean that there exists a pair of points with the same center X in the optimal solution using the pigeonhole principle wherein k optimal centers(cities in H in our case) means k+1 points. Hence, the distance between them is at most 2 * OPT using the triangle inequality which states that:

$$d(u,w) \leq d(u, v) + d(v,w)$$

We have arrived at a contradiction.