Sheetal Parikh
Foundations of Algorithms
EN.605.621.81.SU19
Homework #6

1.) (*a) Describe a bipartite graph G such that G has a perfect matching if and only if there is a feasible dinner schedule for the co-op.*

A bipartite graph is an undirected graph with 2 disjoint sets of vertices $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$, such that all edges $(v_i, v_j) \in E,\ v_i \in V_1\ and\ v_j \in V_2$. If the bipartite graph had perfect matching than every edge would have one endpoint in $V_1$ and one endpoint in $V_1$ 7 meaning that every person is matched with one specific night and that each vertex has one edge incident. We know that $V_1$ represents the set of people, $V_1 = \{P_1, .....P_n\}$, $V_2$ represents the set of nights, $V_2 = \{d_1, .....d_n\}$, and that for a person $p_i$ we associate a set of nights $S_i \subset \{d_1, .....d_n\}$ for when he or she is not available to cook. If we have exactly one person to cook on one specific night and $p_i$ cooks on night $d_j$ than $d_j \notin S_i$. We can say that $A_i \subset \{d_1, .....d_n\}$ is the set of nights for when $p_i$ is available to cook.

As stated earlier, a bipartite graph $G$ with perfect matching would mean that for every edge(or matchup between a person and a cooking night), we have exactly one endpoint in $V_1$ or set of people and one endpoint in $V_2$ or set of nights. This implies that we have edge $(v_i, v_j)$ in which $v_i = p_i \in V_1$, $v_j = d_j \in A_j$ and $A_j \subseteq V_2$. Also, because we have a bipartite graph then, $V_1 \cap V_2 = \emptyset$. Both these statements mean that a schedule exists in which one person can be responsible for cooking on a particular night in which he or she is available and all the nights can be covered. Therefore, a feasible dinner schedule must be possible because a bipartite graph $G$ with perfect matching can be used to describe the dinner schedule.

*(b) Show that it is possible, using her "almost correct" schedule, to decide in only O(n2 ) time whether there exists a feasible dinner schedule for the co-op. If one exists, your algorithm should also provide that schedule.*

We are given that $n-2$ people have been assigned different nights on which they are available. However, people $p_i$ and $p_j$ were both assigned to cook on $d_k$ and no one to cook on $d_l$. As per Alanis' schedule we know that both $p_i$ and $p_j$ are at least available on $d_k$. To fix the "almost correct" schedule, we can first check to see if either $p_i$ or $p_j$ are available on $d_l$. We would need to check sets $A_i$ and $A_j$ to see whether $d_l$ is included in those sets to determine the availability of $p_i$ and $p_j$ respectively. This would be the best case scenario. If either are available, than the available person could easily be switched to cover night $d_l$. If both are available, then it doesn't matter whether $p_i$ or $p_j$ switches to $d_k$.

However, the worst case scenario would be if neither $p_i$ and $p_j$ are not available on $d_l$ or that $d_l$ is not in sets $A_i$ or $A_j$. In this situation, we need to find another person say, $p_l$, that is available on $d_l$ and has an available day in common with $p_i$ or $p_j$ in order to swap with either person. Overall, we would need to check the available sets of all n people and check whether $d_l$ is included on that day and whether a day is in common with sets $A_i$ or $A_j$. If such a day exists than, we would switch the common day with $p_i$ or $p_j$ and assign $p_l$ to $d_l$. Because we would need to check the available sets of n people for n nights (since one person will always match to one night as confirmed by part a), it would take $O(n * n)$ or $O(n^2)$ time to decide whether there is a feasible dinner schedule for the co-op.

Reference: http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec5.pdf

2.) *Collaborative: (a) Given G,X, and S, show how to decide in polynomial time whether a set of evacuation routes exists.*

We will create a flow network as the escape problem closely resembles a network flow problem. We are given that each vertex in $X$ (the populated vertices) is the tail of one path, the last vertex on each path lies in $S$ (the set of safe vertices), and the paths do not share any edges. We can have the source connect to all the vertices in set $X$ and have the sink connect to all the vertices in set $S$. Because, the paths cannot share any edges, we can assign 1 as the weight of each edge the source to vertices in set $X$ to satisfy the evacuation route definition. In order to determine whether a set of evacuation routes exist we can calculate the max flow of the network we created. If an evacuation route exists, than the max flow will equal the number of vertices in set $X$ or $|X|$.

The path would flow as follows: Source → Vertex in $X$ → Vertex in $S$ → Sink

Because the weight or capacity of each edge is 1 from the source to the vertices in set $X$, a capacity of $|X|$, would flow from the source to the vertices in set $X$ from which the capacity of $|X|$ would flow from the vertices in set $S$ to the sink. We can determine whether a set of evacuation route exists by finding the max flow using the Ford-Fulkerson Algorithm which runs in polynomial time.

*(b) Suppose we have exactly the same problem as in (a), but we want to enforce an even stronger version of the "no congestion" condition (iii). Thus, we change (iii) to say, "the paths do not share any vertices." With this new condition, show how to decide in polynomial time whether such a set of*

*evacuation routes exists. Also provide an example with the same G, X, and S in which the answer is "yes" to the question in (a) but "no" to the question in (b).*

If the evacuation routes cannot share any vertices than, we could split the vertices not included in set $X$ or set $S$ into two vertices in which one vertex would have the original incoming edges and the other vertex would have the original outgoing edges.. For example if vertex, A is not included in either set, than we would split A into vertices: $A_1 and A_2$. An edge can be added in between $A_1 and A_2$. One of the split vertices, let's say vertex $A_1$ would have the incoming edges from vertex A and so vertex $A_2$ would have the outgoing edges from original vertex A. This would ensure that all paths remain unique. Afterwards, we could create a similar flow network as the one explained in part a.

The algorithm would still run in polynomial time because the Ford-Fulkerson algorithms could be used to find the max flow. We would have a greater number of vertices than from part a, but this would not cause the complexity to change from polynomial time.

An example in which the answer would be "yes" to the question in part a but "no" to the question in part b would be the following:

- We have a set of vertices: $(X_1, X_2, A, S_1, S_2)$ in which $X_1 and X_2$ are from set $X$ and $S_1$ *and* $S_2$ are from set $S$.
- We have the following edges: $(X_1, A)$, $(X_2, A)$, $(A, S_1)$, $(A, S_2)$

Using this example we can see that we have unique paths to get from X to S:

- $X_1 \rightarrow A \rightarrow S_1$
- $X_2 \rightarrow A \rightarrow S_2$

However, both paths use vertex A. Because the same vertex was used, we do not satisfy the requirement from part b causing the answer to be "no" for part b.

3.) *(a) What do the variables in this problem represent? How many are there?*

*Because the evaluator and detector component of the tracking system have already been defined, we are only examining the tracker or 0-1 integer linear program component of the system. Many factors can affect whether an object is detected in a track such as the object's position, size, velocity, etc.* Since the variables have to be boolean, we will only consider the 1 binary decision variable which determines whether or not an object has been detected on a particular track. As given, we also know that a particular track is defined as *j* and the detection of an object is defined by *i*. We can also define *t*, as the number of the scan that the system has completed.

*(b) Define the objective for this 0-1 integer linear program.*

The objective for the 0-1 integer linear program is to determine whether an object is detected on a particular path. The tracking system will take multiple scans of the area we are evaluating and through the detector and evaluator component, returns a number of detections and tracks. The 0-1 integer linear program of the tracker component decides whether a particular detection of an object was on a particular path. If an object was detected in the specified trajectory, then decision variable is assigned a 1, meaning that an instance of detection of an object occurred on one of the tracks we are considering. If an object is not detected in the trajectory, than the decision variable is assigned a 0. Overall, this problem is similar to an assignment problem in which we are assigning an instance of detection to a particular track if the decision variable is 1. If the decision variable is 0, the instance of detection is not assigned.

*(c) Define the entire 0-1 integer linear program, including constraints, in standard form. How many constraints are there in the program, total?*

As explained in part a, the binary decision variable indicates whether an instance of detection should be assigned to a particular track. We will define this binary decision variable as $y_{itj}$ . As given by the problem, we know that a particular track is defined as *j* and the detection of an object is defined by *i*. We can also define *t* , as the number of the scan on which an object was or was not detected.

$$y_{itj} = \begin{matrix} 1 \ , \ \textit{if instance of detection occurred on track j} \\ 0 \ , \ \textit{otherwise} \end{matrix}$$

Since this problem closely resembles an assignment problem, we can specify 2 constraints for the 0-1 integer linear program:

1. *Each instance of detection has to be assigned to one track j for each scan t*

$$\sum_{j=1} y_{itj} = 1 \qquad \forall\, i, t$$

2. *Each track j has to be assigned one instance of detection for each scan t*

$$\sum_{i=1} y_{itj} = 1 \qquad \forall j, t$$

References:
http://www.robots.ox.ac.uk/~az/lectures/b1/lect3.pdf
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.171.360&rep=rep1&type=pdf
https://apps.dtic.mil/dtic/tr/fulltext/u2/1033655.pdf

4.) *(a) What is the expected loss for Player 1 when Player 1 plays a mixed strategy x = (x1, x2, x3) and Player 2 plays a mixed strategy y = (y1, y2, y3)?*

We are given the loss matrix for Player 1 in the description of the problem and that both players 1 and 2 will be playing a mixed strategy. In order to find the total expected loss of player 1, add together the expected loss of player 1 choosing rock, the expected loss of player 1 choosing paper, and the expected loss of player 1 choosing scissors.

$$E(P1_{Rock}) = x_1\left[(0)(y_1) + (1)(y_2) + (-1)(y_3)\right]$$
$$= x_1(y_2 - y_3)$$

$$E(P1_{Paper}) = x_2\left[(-1)(y_1) + (0)(y_2) + (1)(y_3)\right]$$
$$= x_2(y_3 - y_1)$$

$$E(P1_{Scissors}) = x_3\left[(1)(y_1) + (-1)(y_2) + (0)(y_3)\right]$$
$$= x_3(y_1 - y_2)$$

$$Total\ Expected\ Loss\ of\ Player\ 1 = E(P1) = E(P1_{Rock}) + E(P1_{Paper}) + E(P1_{Scissors})$$
$$= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

*(b) Show that Player 1 can achieve a negative expected loss (i.e., and expected gain) if Player 2 plays any strategy other than y = (y1, y2, y3) = (1/3 , 1/3 , 1/3 ).*

In part a we found the equation for the total expected loss of Player 1. If we use the given strategy for $y = (y_1, y_2, y_3) = (1/3,\ 1/3,\ 1/3)$ , we would get the following expected loss for player 1:

$$Total\ Expected\ Loss\ of\ Player\ 1 = E(P1) = E(P1_{Rock}) + E(P1_{Paper}) + E(P1_{Scissors})$$
$$= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$= x_1(1/3 - 1/3) + x_2(1/3 - 1/3) + x_3(1/3 - 1/3)$$
$$= 0$$

We can determine whether it is possible for Player 1 to have a negative expected loss by finding the expected loss using a different example strategy for Player 2.

For example, let's use the following strategy : $y = (y_1, y_2, y_3) = (1/3, 2/3, 0)$

$$Total\ Expected\ Loss\ of\ Player\ 1 = E(P1) = E(P1_{Rock}) + E(P1_{Paper}) + E(P1_{Scissors})$$
$$= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$
$$= x_1(2/3 - 0) + x_2(0 - 1/3) + x_3(1/3 - 2/3)$$
$$= 2/3x_1 - 1/3x_3$$

It is possible for player 1 to have a negative expected loss if $x_3$ is a value greater than $x_1$. For example if Player 1 used the strategy $x = (x_1, x_2, x_3) = (0, 1/3, 2/3)$, Player 1 would experience a negative expected loss.

As a second example, let's also use the following strategy for Player 2:
$y = (y_1 + y_2 + y_3) = (1/2, 0, 1/2)$

$$Total\ Expected\ Loss\ of\ Player\ 1 = E(P1) = E(P1_{Rock}) + E(P1_{Paper}) + E(P1_{Scissors})$$
$$= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$
$$= x_1(0 - 1/2) + x_2(1/2 - 1/2) + x_3(1/2 - 0)$$
$$= 1/2x_3 - 1/2x_1$$

It is possible for player 1 to have a negative expected loss if $x_1$ is a value greater than $x_3$. For example if Player 1 used the strategy $x = (x_1, x_2, x_3) = (2/3, 1/3, 0)$, Player 1 would experience a negative expected loss.

Therefore, we can see that it is definitely possible for Player 1 to have a negative expected loss if player 2 plays any strategy other than $y = (y_1, y_2, y_3) = (1/3, 1/3, 1/3)$.

*(c) Show that x = (1/3 , 1/3 , 1/3) and y = (1/3 , 1/3 , 1/3) form a Nash equilibrium.*

We know that we have a nash equilibrium when neither player has an incentive to changing his strategy. We found in part b that the expected loss of Player 1 using the strategy $(1/3, 1/3, 1/3)$

results in an expected loss of 0. If we find the expected loss of Player 2 using the same strategy of $(1/3\,,\ 1/3\,,\ 1/3\,)$, it also will result in 0.

Loss Matrix Player 2:

$$A = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$\begin{aligned} Total\ Expected\ Loss\ of\ Player\ 2 \ = \ E(P2\,) &= E(P2_{Rock}) + E(P2_{Paper}) + E(P2_{Scissors}) \\ &= y_1(x_2 - x_3) + y_2(x_3 - x_1) + y_3(x_1 - x_2) \\ &= y_1(1/3 - 1/3\,) + y_2(1/3 - 1/3\,) + y_3(1/3 - 1/3\,) \\ &= 0 \end{aligned}$$

As both players using a mixed strategy of $(1/3\,,\ 1/3\,,\ 1/3\,)$ would result in an expected loss of 0, they wouldn't have an incentive to change strategies.

As shown in part b, we already have seen how if a mixed strategy other than $(1/3\,,\ 1/3\,,\ 1/3\,)$ is used for one of the players, than the remaining player also has an incentive to change his strategy, meaning the strategy is not at a nash equilibrium. For example, if player 1 has a strategy to always choose paper, player 2 will have the incentive to use the strategy of always choosing scissors. Regardless of the combination used, if both players do not use the strategy of having an equal probability of choosing rock, paper, or scissors, one player will always have the incentive of switching to beat the other player. Therefore, only the strategy of both players having equal probabilities for rock, paper, or scissors (meaning that $y = (1/3\,,\ 1/3\,,\ 1/3\,)$ and $x = (1/3\,,\ 1/3\,,\ 1/3\,)$) forms a nash equilibrium.

*(d) Let x = (1/3 , 1/3 , 1/3) as in part (c). Is it possible for (x, y) to be a Nash equilibrium for some mixed strategy y' ≠ (1/3 , 1/3 , 1/3) ? Explain.*

As we explained in part b, if a strategy other than (1/3, 1/3, 1/3) is used, it is possible for the expected loss of of player 1 to be negative. Therefore, Player 1 would have an incentive to make the expected loss even more negative (i.e. have a greater expected gain), which would be possible if he changed his strategy. Overall, part b showed that when Player 2 uses a strategy other than (1/3, 1/3, 1/3), then player 2 also has an incentive to use a strategy other than (1/3, 1/3, 1/3). This means that there is no nash equilibrium if a mixed strategy other than (1/3, 1/3, 1/3) is used.