EE 382V: Social Computing **Fall 2018**

Lecture 2, Session 1: August 25

Lecturer: Vijay Garg *Scribe: Matt Wey*

2.1 Introduction

In this section we will be solving the Assignment Problem specifically using the *Kuhn-Munkres* (Hungarian) Algorithm.

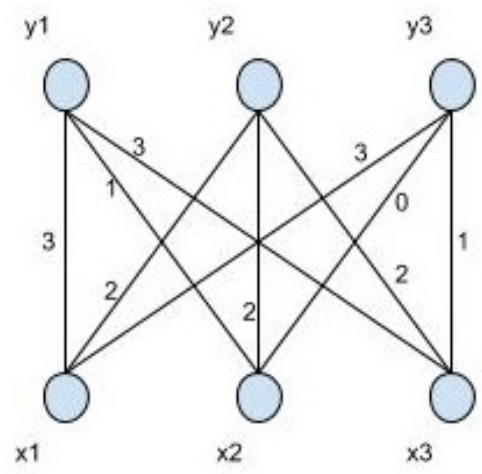


Figure 2.1: For the Assignment Problem we will consider this of bipartite graph.

2.2 Assignment Problem

The goal for the Assignment Problem is to find perfect matching with the maximum weight or the minimum cost. For our example we will be considering the bipartite graph shown in Figure 2.1. The key idea to solving this problem is duality.

2.2.1 Feasibility Constraints

Similar to assigning a weight to an edge, to add a feasibility constraint to the problem we give a label to each vertex. To be considered a feasibility constraint it must meet the following condition.

$$l(x_i) + l(y_j) \geq w(x_i, y_j); \qquad i = 1, 2, \ldots, n; j = 1, 2, \ldots, n \tag{2.1}$$

Where $l(x_i)$ is the label at vertex x_i, and $w(x_i, y_j)$ is the weight from vertex x_i to vertex y_j. The feasibility constraint is considered to create a "tight" edge when

$$l(x_i) + l(y_j) = w(x_i, y_j) \tag{2.2}$$

2.2.2 Equality Graph

When all edges in a given labeling l are tight we get an Equality Graph for that labeling, E_l, which is a subgraph of the original graph (Figure 2.1), an example of which is shown in Figure 2.2.

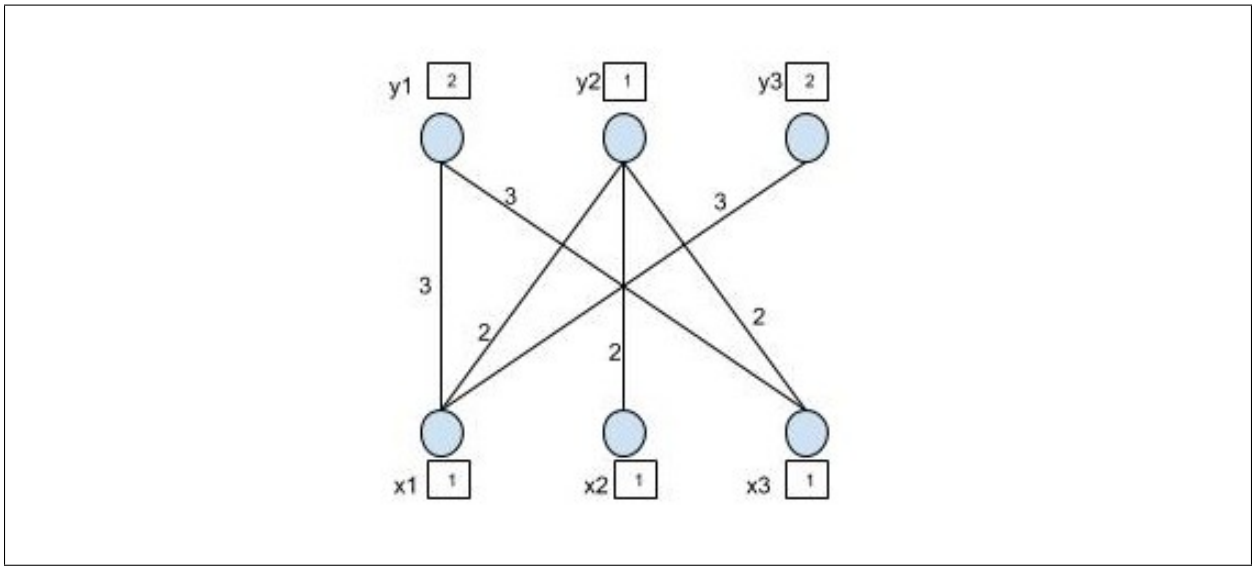


Figure 2.2: An equality graph. This particular equality graph is a subgraph of Figure 2.1.

2.2.3 Kuhn-Munkres Algorithm

As a method to solving the assignment problem we will explore the Kuhn-Munkres (Hungarian) Algorithm.

Theorem 2.1 *If l is feasible and M is a perfect matching in E_l, then M is a max-weight matching.*

Note that this method uses Equality Graphs which require tight edges so for each graph formed in the process the edges should remain tight.

Proof: Let M' be any perfect matching (not necessarily in E_l)

$$w(M') = \sum_{e \in E} w(e) \leq \sum_{e \in M'} l(e_x) + l(e_y) = \sum_{v \in V} l(v)$$

M is a perfect matching in E_l

$$w(M) = \sum_{e \in M} w(e) = \sum_{v \in V} l(v)$$

and finally

$$w(M') \leq w(M)$$

■

We can use Theorem 2.1 in creating the algorithm that will solve the Assignment Problem. It is important to note that the input for our algorithm is a bipartite graph with the connected edges and the weights for the edges. Any non-existent edges can also be considered edges zero weight.

Invariants

l is feasible;

M is some matching in E_l, where E_l is an equality graph with labeling l

The following pseudocode explains how our algorithm will solve the Assignment Problem.

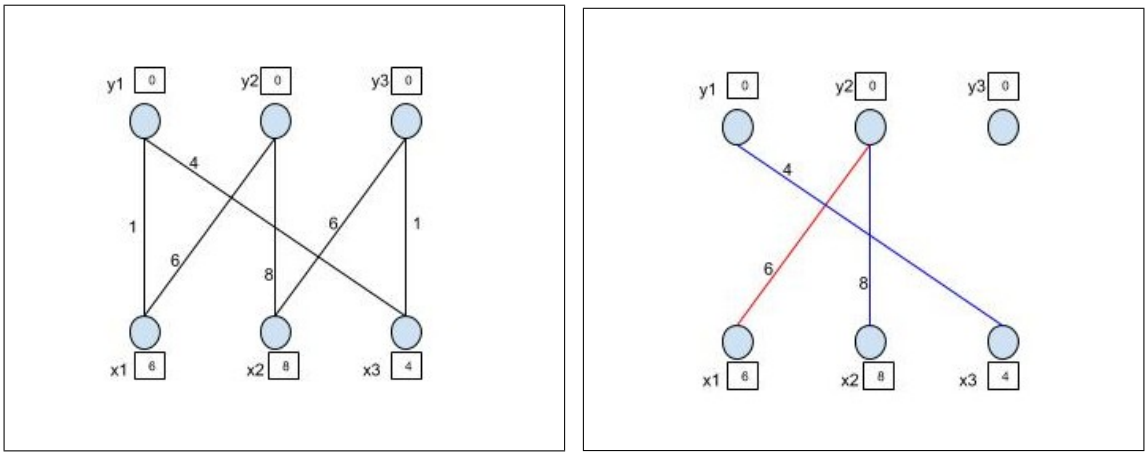
```
while(|M| < n) {
  - if there is an augmenting path in E_l
    - use the augmenting path to augment and increase the size of M
  - else
    - improve l to l' such that E_l exists in E_l'
}
```

Notice each iteration through the while loop will each increase the size of M OR change E_l to include an additional augmenting path. Note that $N_l(S)$ represents the neighbors of set S in labeling the l. The steps of the algorithm are as follows:

1. Initialization: l is initial labeling, $M = \{\}$
2. if M is a perfect matching stop. Otherwise pick a free vertex u such that $u \in X$.
   Assign $S = \{u\}, T = \emptyset$
3. if $N_l(S) = T$
$$\alpha_l = M_{x \in S, y \notin T}(l(x) + l(y) - w(x, y))$$
$$l'(v) = \begin{cases} l(v) - \alpha_l; & v \in S \\ l(v) + \alpha_l; & v \in T \\ l(v); & otherwise \end{cases}$$
4. if $N_l(S) \neq T$ pick $y \in (N_l(S) - T)$
   if y is free then u to y is an augmenting path.
   Augment path and go back to 2.
   if y is match to z
   $S = S \cup \{z\}, T = T \cup \{y\}$. Go to 3.

2.2.4 Hungarian Algorithm Example

- Given a weighted graph, create a basic initialization labeling all the vertices with the max weight of its connected edges as shown in Figure 2.3 (a).
$$\forall y_j : \quad l(y_j) = 0, \quad l(x_i) = \max(w(x_i, y_j))$$
- Select an initial matching as shown in Figure 2.3 (b).



(a) Initialized labeled graph with trivial labeling (b) Initial equality graph E_l

Figure 2.3

- Assign $S = \{x_1\}, T = \emptyset$. Since $N_l(S) \neq T$ to go step 4.

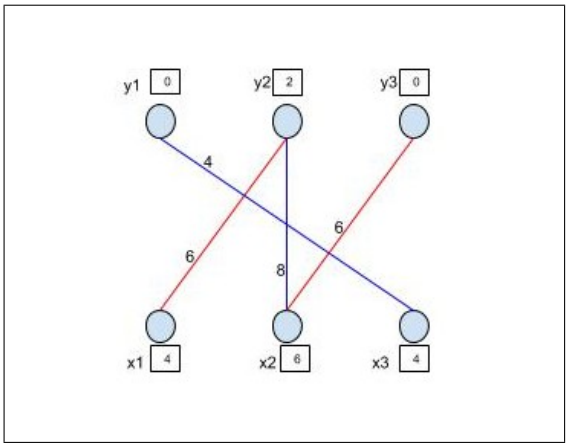
- Assign $y \in (N_l(S) - T)$. We choose y_2, which is matched, and increase the graph by assigning $S = \{x_1\} \cup \{x_2\}, T = \emptyset \cup \{y_2\}$. Go to 3.

- Now $S = \{x_1, x_2\}, T = \{y_2\}, N_l(S) = T$. Calculate α_l

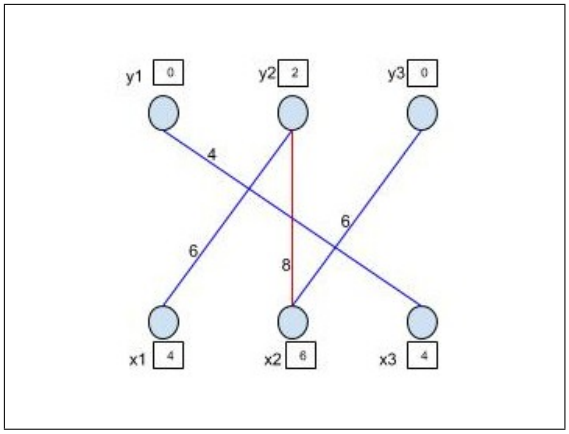
$$\alpha_l = min \begin{cases} (x_1, y_1) & 6 + 0 - 1 \\ (x_1, y_3) & 6 + 0 - 0 \\ (x_2, y_1) & 8 + 0 - 0 \\ (x_2, y_3) & 8 + 0 - 6 \end{cases} = 2$$

- Update Labels by reducing labels of S by 2 and increasing labels of T by 2 and add edge (x_2, y_3) to the equality graph. See Figure 2.4 (a).

- We now have a path starting and ending with unselected edges, so we augment the path and add the updated equality graph to the matching M as shown in Figure 2.4(b).



(a) Equality graph updated to include augmented path.



(b) Graph showing the final optimal matching with total weight 16.

Figure 2.4

- Now that we have complete vertex coverage (perfect matching), by Theorem 2.1 we know we have an optimal (max-weight) solution and can calculate the final value by summing the selected edges or summing the vertex labels.