Bipartite Matching & the Hungarian Method

Last Revised: August 30, 2006

These notes follow formulation developed by Subhash Suri *http://www.cs.ucsb.edu/˜suri*.

We previously saw how to use the Ford-Fulkerson Max-Flow algorithm to find Maximum-Size matchings in bipartite graphs. In this section we discuss how to find Maximum-Weight matchings in bipartite graphs, a situation in which Max-Flow is no longer applicable.

The $O(|V|^3)$ algorithm presented is the Hungarian Algorithm due to Kuhn & Munkres.

- Review of Max-Bipartite Matching
  Earlier seen in Max-Flow section
- Augmenting Paths
- Feasible Labelings and Equality Graphs
- The Hungarian Algorithm for
  Max-Weighted Bipartite Matching

Application: Max Bipartite Matching

A graph $G = (V, E)$ is *bipartite* if there exists partition $V = X \cup Y$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$.

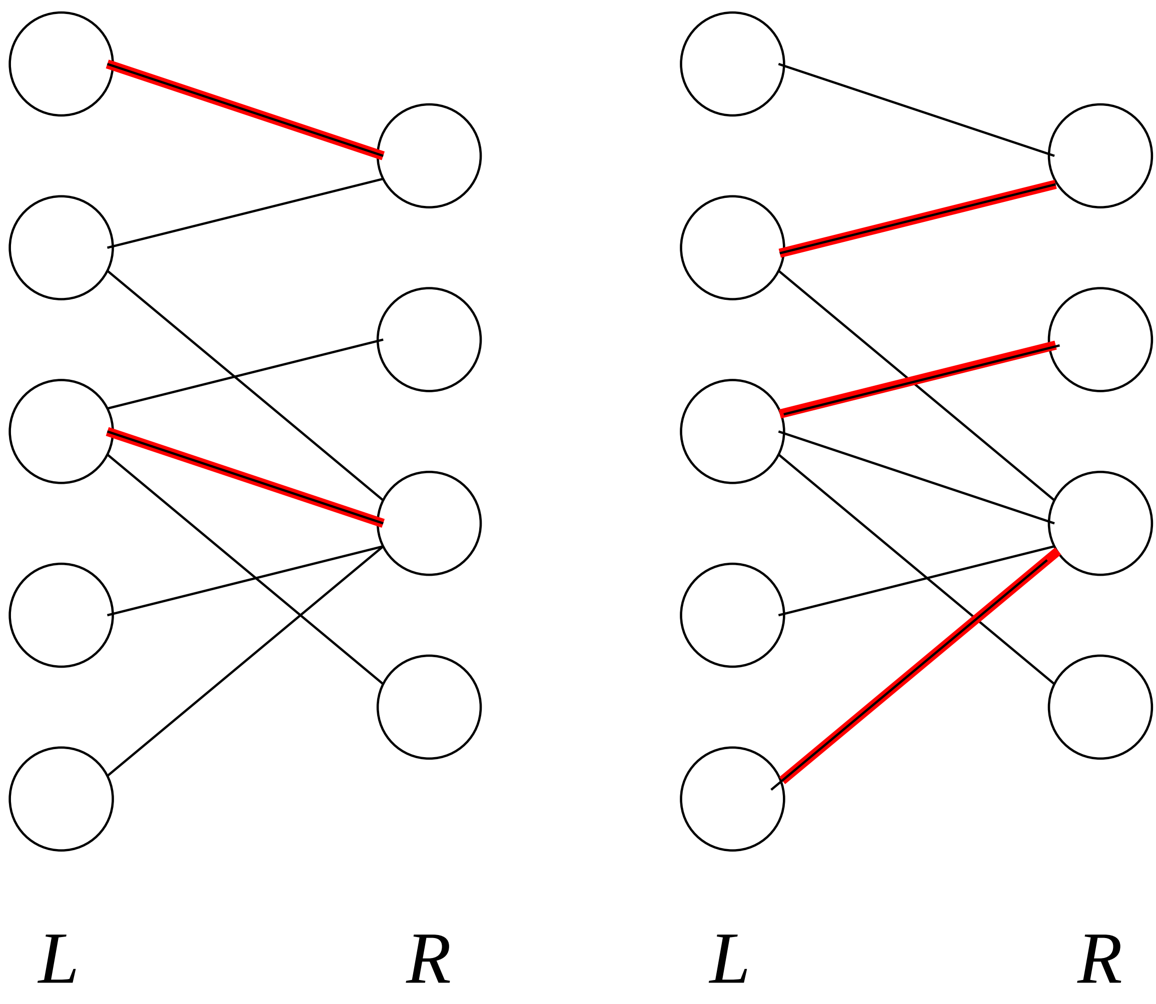
A *Matching* is a subset $M \subseteq E$ such that $\forall v \in V$ at most one edge in M is incident upon v.

The *size* of a matching is $|M|$, the number of edges in M.

A *Maximum Matching* is matching M such that every other matching M' satisfies $|M'| \leq M$.

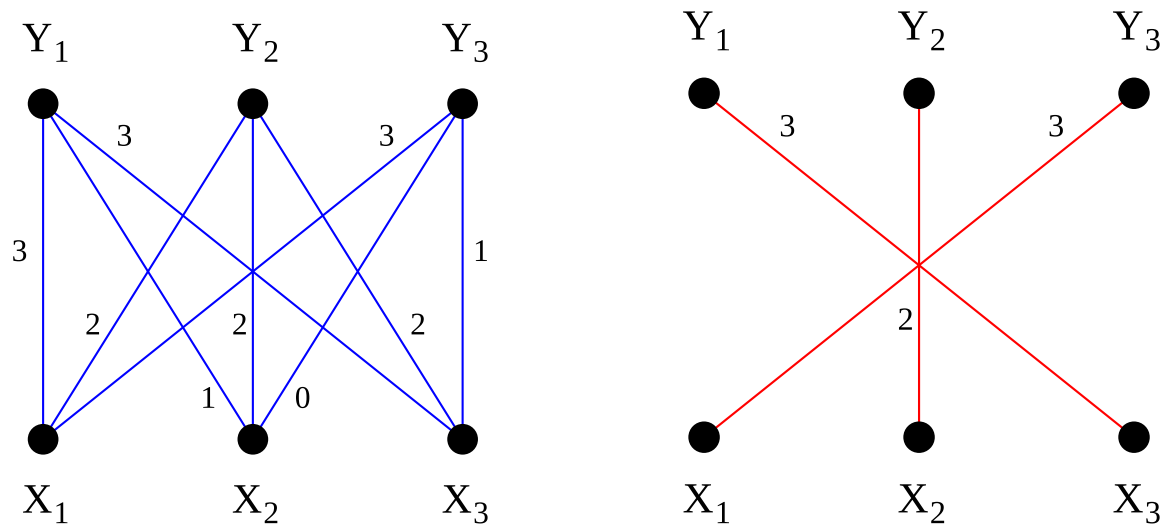
Problem: Given bipartite graph G, find a maximum matching.

A bipartite graph with 2 matchings



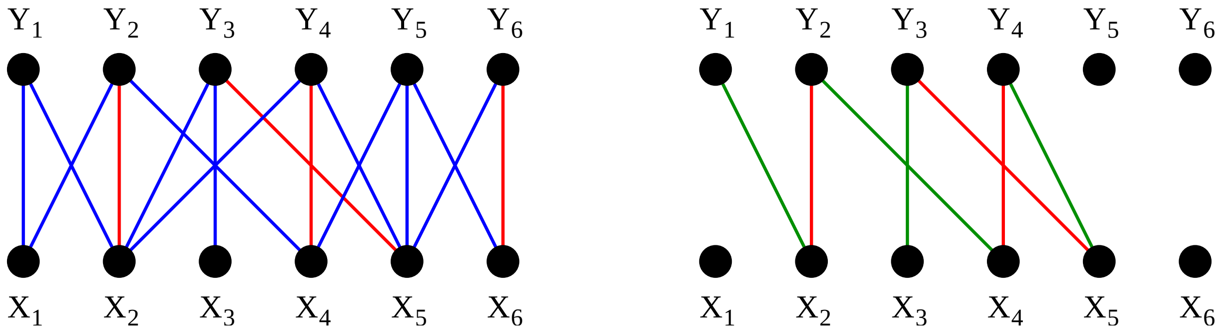
We now consider *Weighted* bipartite graphs. These are graphs in which each edge (i, j) has a weight, or value, $w(i, j)$. The *weight* of matching M is the sum of the weights of edges in M, $w(M) = \sum_{e \in M} w(e)$.

Problem: Given bipartite weighted graph G, find a maximum weight matching.



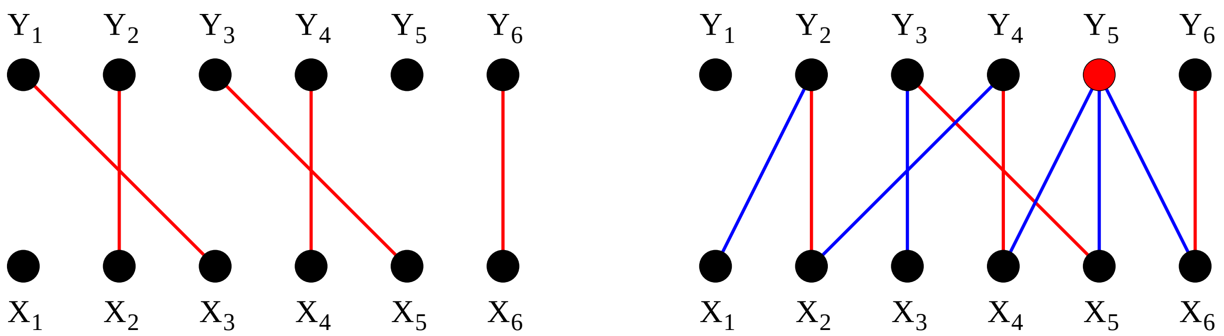
Note that, without loss of generality, by adding edges of weight 0*, we may assume that G is a* complete *weighted graph.*

Alternating Paths:



- Let M be a matching of G.
- Vertex v is matched if it is endpoint of edge in M; otherwise v is free
  $Y_2, Y_3, Y_4, Y_6, X_2, X_4, X_5, X_6$ are matched, other vertices are free.
- A path is alternating if its edges alternate between M and $E - M$.
  $Y_1, X_2, Y_2, X_4, Y_4, X_5, Y_3, X_3$ is alternating
- An alternating path is augmenting if both end-points are free.
- Augmenting path has one less edge in M than in $E - M$; replacing the M edges by the $E - M$ ones increments size of the matching.

Alternating Trees:



An alternating tree is a tree rooted at some free vertex v in which every path is an alternating path.

Note: The diagram assumes a *complete* bipartite graph; matching M is the red edges. Root is Y_5.

The Assignment Problem:

Let G be a (complete) weighted bipartite graph.

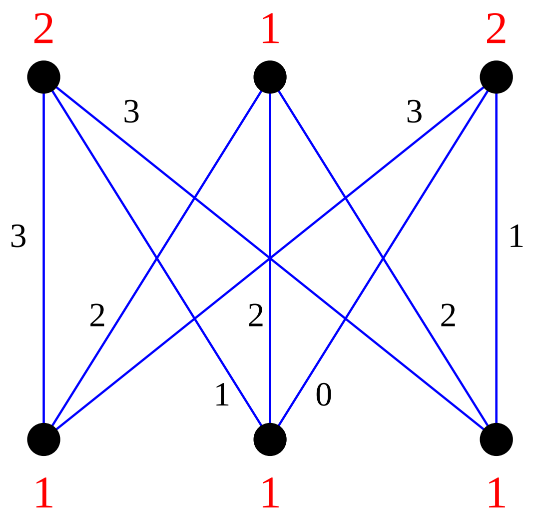
The Assignment problem is to find a max-weight matching in G.

A *Perfect Matching* is an M in which every vertex is adjacent to some edge in M.

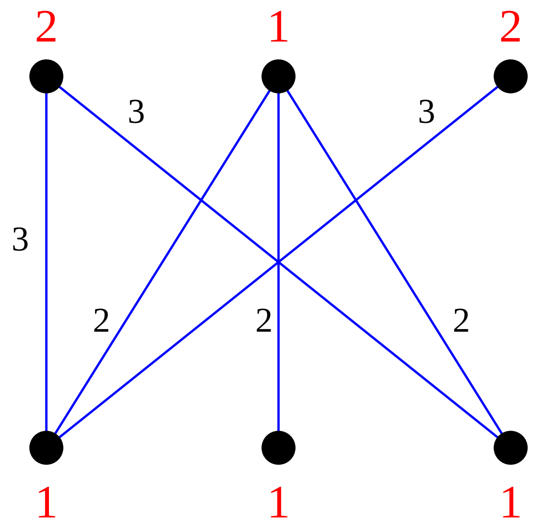
A max-weight matching is perfect.

Max-Flow reduction dosn't work in presence of weights. The algorithm we will see is called the Hungarian Algorithm.

Feasible Labelings & Equality Graphs



A feasible labeling ℓ



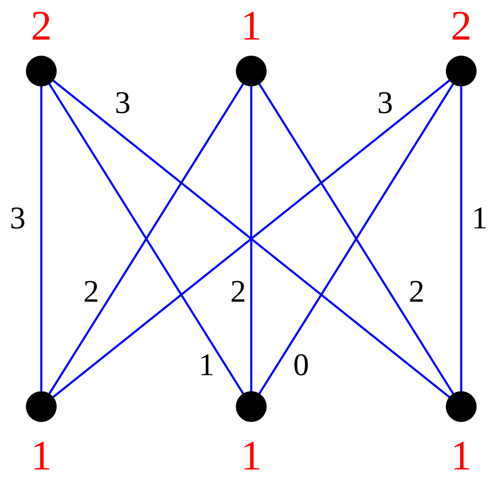
Equality Graph G_ℓ

- A vetex *labeling* is a function $\ell : V \to \mathcal{R}$
- A *feasible* labeling is one such that

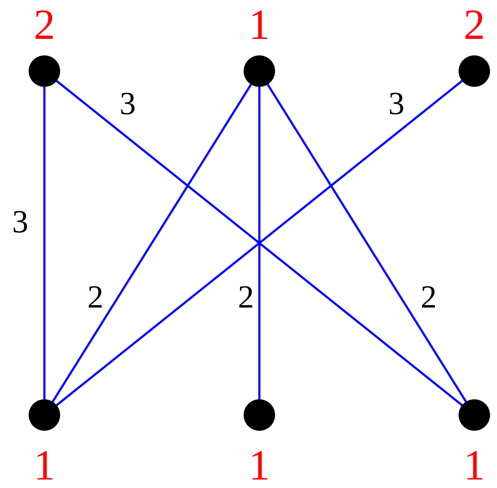
$$\ell(x) + \ell(y) \geq w(x, y), \quad \forall x \in X,\ y \in Y$$

- the *Equality Graph* (with respect to ℓ) is $G = (V, E_\ell)$ where

$$E_\ell = \{(x, y) \ : \ \ell(x) + \ell(y) = w(x, y)\}$$



A feasible labeling ℓ



Equality Graph G_ℓ

Theorem: If ℓ is feasible and M is a Perfect matching in E_ℓ then M is a max-weight matching.

Proof:
Denote edge $e \in E$ by $e = (e_x, e_y)$.
Let M' be *any* PM in G (not necessarily in in E_ℓ).
Since every $v \in V$ is covered *exactly* once by M we have

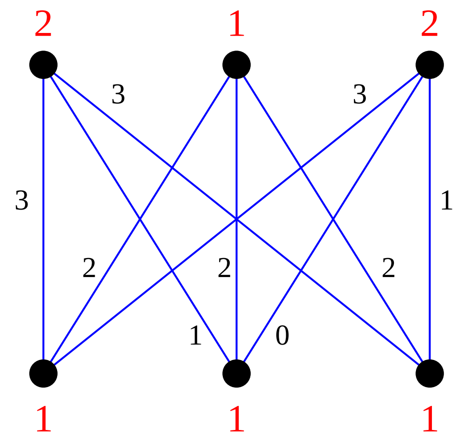
$$w(M') = \sum_{e \in M'} w(e) \le \sum_{e \in M'} (\ell(e_x) + \ell(e_y)) = \sum_{v \in V} \ell(v)$$

so $\sum_{v \in V} \ell(v)$ is an upper-bound on the cost of any perfect matching.

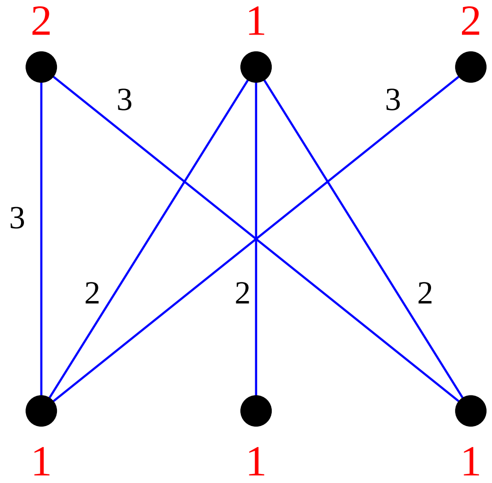
Now let M be a PM in E_ℓ. Then

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

So $w(M') \le w(M)$ and M is optimal.



A feasible labeling ℓ



Equality Graph G_ℓ

Theorem[Kuhn-Munkres]: If ℓ is feasible and M is a Perfect matching in E_ℓ then M is a max-weight matching.

The KM theorem transforms the problem from an *optimization* problem of finding a max-weight matching into a combinatorial one of finding a perfect matching. It combinatorializes the weights. This is a classic technique in combinatorial optimization.

Notice that the proof of the KM theorem says that for any matching M and any feasible labeling ℓ we have

$$w(M) \leq \sum_{v \in V} \ell(v).$$

This has very strong echos of the max-flow min-cut theorem.

Our algorithm will be to

Start with any feasible labeling ℓ
and some matching M in E_ℓ

While M is not perfect repeat the following:

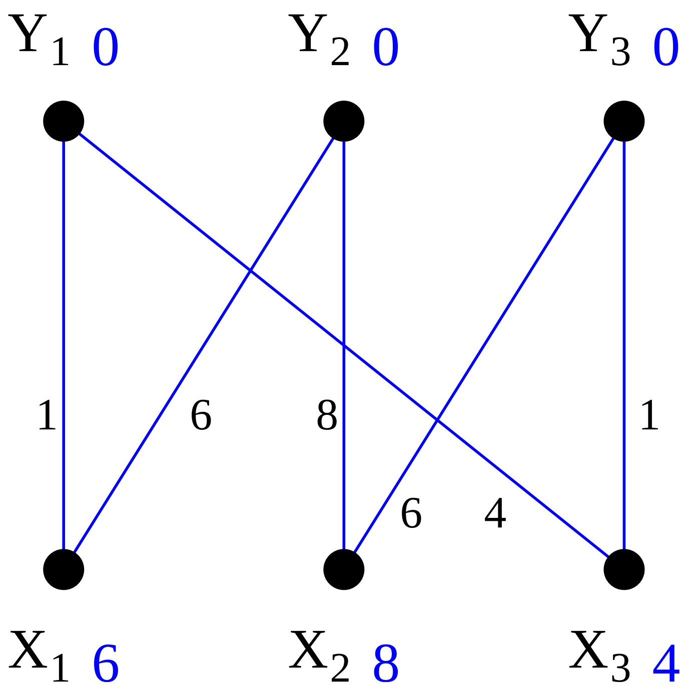
1. Find an augmenting path for M in E_ℓ;
   this increases size of M

2. If no augmenting path exists,
   improve ℓ to ℓ' such that $E_\ell \subset E_{\ell'}$.
   Go to 1.

Note that in each step of the loop we will either be increasing the size of M or E_ℓ so this process must terminate.

Furthermore, when the process terminates, M will be a perfect matching in E_ℓ for some feasible labeling ℓ. So, by the Kuhn-Munkres theorem, M will be a max-weight matching.

Finding an Initial Feasible Labelling



Finding an initial feasible labeling is simple. Just use:

$$\forall y \in Y,\ \ell(y) = 0, \qquad \forall x \in X,\ \ell(x) = \max_{y \in Y}\{w(x, y)\}$$

With this labelling it is obvious that

$$\forall x \in X, y \in Y,\ w(x) \leq \ell(x) + \ell(y)$$

Improving Labellings

Let ℓ be a feasible labeling.
Define *neighbor* of $u \in V$ and set $S \subseteq V$ to be

$$N_\ell(u) = \{v : (u,v) \in E_\ell, \}, \quad N_\ell(S) = \cup_{u \in S} N_\ell(u)$$

Lemma: Let $S \subseteq X$ and $T = N_\ell(S) \neq Y$. Set

$$\alpha_\ell = \min_{x \in S, y \notin T} \{\ell(x) + \ell(y) - w(x,y)\}$$

and

$$\ell'(v) = \begin{cases} \ell(v) - \alpha_\ell & \text{if } v \in S \\ \ell(v) + \alpha_\ell & \text{if } v \in T \\ \ell(v) & \text{otherwise} \end{cases}$$

Then ℓ' is a feasible labeling and
(i) If $(x,y) \in E_\ell$ for $x \in S, y \in T$ then $(x,y) \in E_{\ell'}$.
(ii) If $(x,y) \in E_\ell$ for $x \notin S, y \notin T$ then $(x,y) \in E_{\ell'}$.
(iii) There is some edge $(x,y) \in E_{\ell'}$ for $x \in S, y \notin T$

The Hungarian Method

1. Generate initial labelling ℓ and matching M in E_ℓ.

2. If M perfect, stop.
   Otherwise pick free vertex $u \in X$.
   Set $S = \{u\}$, $T = \emptyset$.

3. If $N_\ell(S) = T$, update labels (forcing $N_\ell(S) \neq T$)

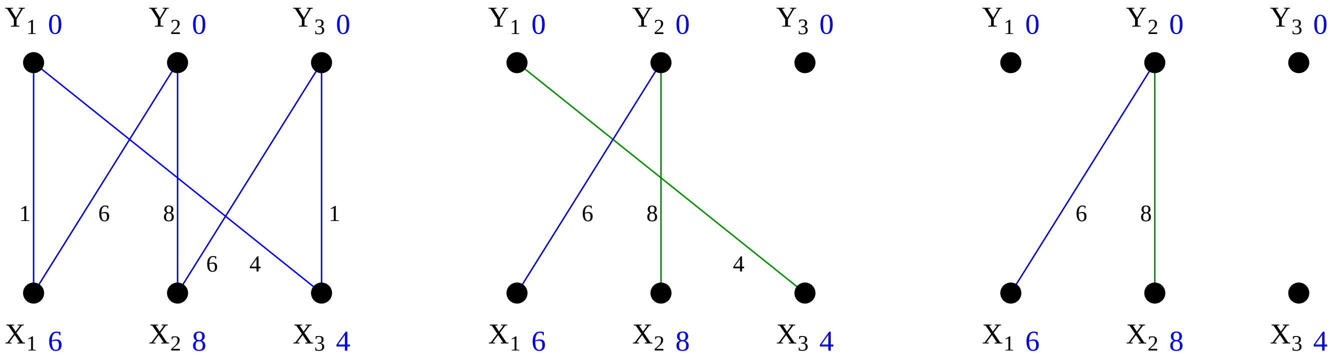
   $$\alpha_\ell = \min_{s \in S,\ y \notin T} \{\ell(x) + \ell(y) - w(x, y)\}$$

   $$\ell'(v) = \begin{cases} \ell(v) - \alpha_\ell & \text{if } v \in S \\ \ell(v) + \alpha_\ell & \text{if } v \in T \\ \ell(v) & \text{otherwise} \end{cases}$$

4. If $N_\ell(S) \neq T$, pick $y \in N_\ell(S) - T$.

   - If y free, $u - y$ is augmenting path.
     Augment M and go to 2.

   - If y matched, say to z, extend alternating tree:
     $S = S \cup \{z\}$, $T = T \cup \{y\}$. Go to 3.



Original Graph Eq Graph+Matching Alternating Tree

- Initial Graph, trivial labelling
  and associated Equality Graph

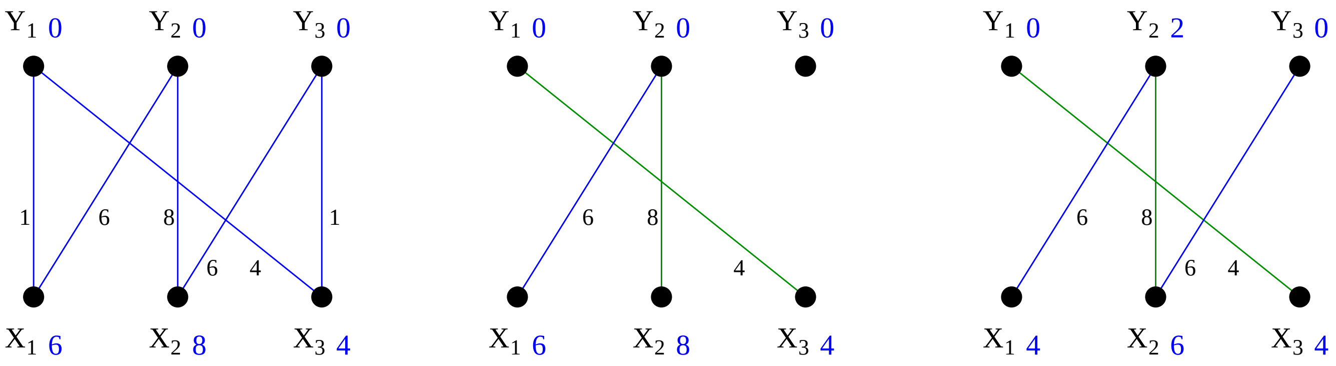
- Initial matching: (x_3, y_1), (x_2, y_2)

- $S = \{x_1\}$, $T = \emptyset$.

- Since $N_\ell(S) \neq T$, do step 4.
  Choose $y_2 \in N_\ell(S) - T$.

- y_2 is matched so grow tree by adding (y_2, x_2), i.e., $S = \{x_1, x_2\}$, $T = \{y_2\}$.

- At this point $N_\ell(S) = T$, so goto 3.



Original Graph Old E_ℓ and $|M|$ new Eq Graph

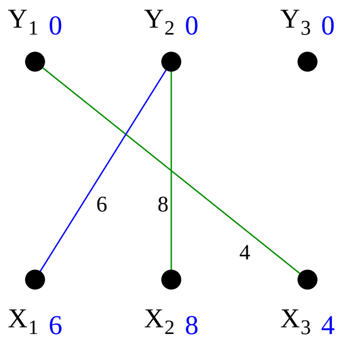
- $S = \{x_1, x_2\}$, $T = \{y_2\}$
  and $N_\ell(S) = T$

- Calculate α_ℓ

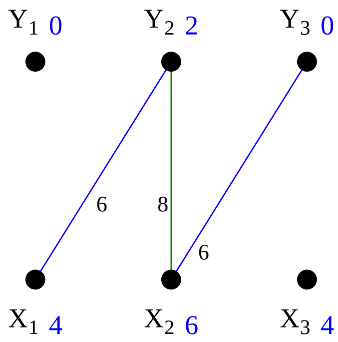
$$\begin{aligned}\alpha_\ell &= \min_{x \in S, y \notin T} \begin{cases} 6+0-1, & (x_1, y_1) \\ 6+0-0, & (x_1, y_3) \\ 8+0-0, & (x_2, y_1) \\ 8+0-6, & (x_2, y_3) \end{cases} \\ &= 2\end{aligned}$$

- Reduce labels of S by 2;
  Increase labels of T by 2.

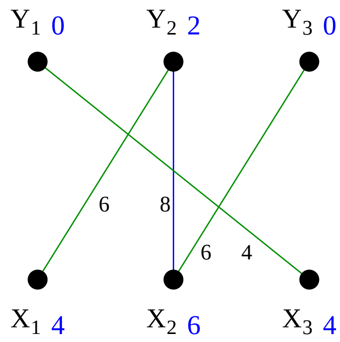
- Now $N_\ell(S) = \{y_2, y_3\} \neq \{y_2\} = T$.



Orig E_ℓ and M



New Alternating Tree



New M

- $S = \{x_1, x_2\}$, $N_\ell(S) = \{y_2, y_3\}$, $T = \{y_2\}$

- Choose $y_3 \in N_\ell(S) - T$ and add it to T.

- y_3 is **not** matched in M so we have just found an alternating path x_1, y_2, x_2, y_3 with two free end-points. We can therefore augment M to get a larger matching in the new equality graph. This matching is perfect, so it must be optimal.

- Note that matching (x_1, y_2), (x_2, y_3), (x_3, y_1) has cost $6 + 6 + 4 = 16$ which is exactly the sum of the labels in our final feasible labelling.

Correctness:

- We can always take the trivial ℓ and empty matching $M = \emptyset$ to start algorithm.

- If $N_\ell(S) = T$, we saw on that we could always update labels to create a new feasible matching ℓ'. The lemma on page 13 guarantees that all edges in $S \times T$ and $\bar{S} \times \bar{T}$ that were in E_ℓ will be in $E_{\ell'}$. In particular, this guarantees (why?) that the current M remains in $E_{\ell'}$ as does the alternating tree built so far,

- If $N_\ell(S) \neq T$, we can, by definition, always augment alternating tree by choosing some $x \in S$ and $y \notin T$ such that $(x, y) \in E_\ell$. Note that at some point y chosen *must* be free, in which case we augment M.

- So, algorithm always terminates and, when it does terminate M is a perfect matching in E_ℓ so, by Kuhn-Munkres theorem, it is optimal.

Complexity

In each phase of algorithm, $|M|$ increases by 1 so there are at most V phases. How much work needs to be done in each phase?

In implementation, $\forall y \notin T$ keep track of
$slack_y = \min_{x \in S}\{\ell(x) + \ell(y) - w(x,y)\}$

- Initializing all slacks at beginning of phase takes $O(|V|)$ time.

- In step 4 we must update all slacks when vertex moves from $\bar{S}$ to S.
  This takes $O(|V|)$ time; only $|V|$ vertices can be moved from $\bar{S}$ to S, giving $O(|V|^2)$ time per phase.

- In step 3, $\alpha_\ell = \min_{y \in T} slack_y$ and can therefore be calculated in $O(|V|)$ time from the slacks.
  This is done at most $|V|$ times per phase (why?) so only takes $O(|V|^2)$ time per phase.
  After calculating α_ℓ we must update all slacks. This can be done in $O(|V|)$ time by setting
  $$\forall y \notin T,\ slack_y = slack_y - \alpha_\ell.$$
  Since this is only done $O(|V|)$ times, total time per phase is $O(|V|^2)$.

There are $|V|$ phases and $O(|V|^2)$ work per phase so the total running time is $O(|V|^3)$.