Toshi’s epic notes on assignment problem

<http://www.cse.ust.hk/~golin/COMP572/Notes/Matching.pdf>

Definitions

A graph is bipartite if there exists partition with and .

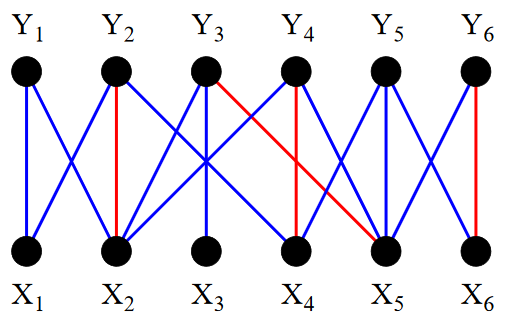
A matching is a subset such that at most 1 edge in is incident upon .

The size of a matching is , the number of edges in .

A maximum matching is matching such that every other matching satisfies

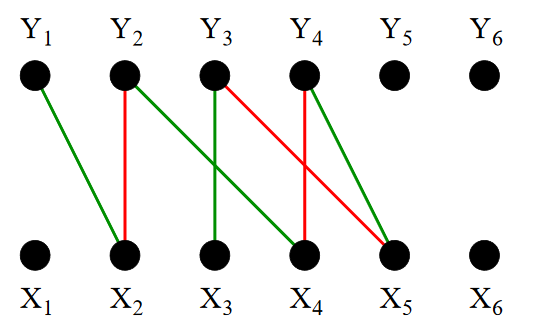
Weighted bipartite graphs are bipartite graphs in which each edge has a weight .

The weight of a matching is the sum of weights of edges in .

WLOG, we may assume is a complete bipartite weighted graph by adding edges of weight .

Vertex is matched if it is an endpoint of edge in . Otherwise, is free.

are matched, and others are free.

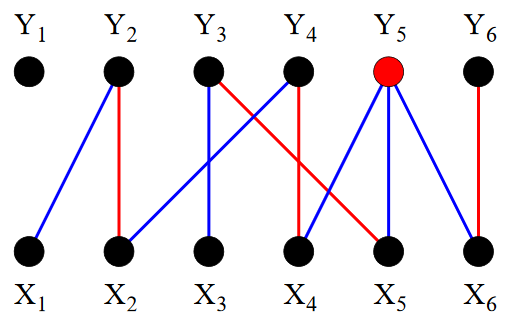
All paths in a bipartite graph alternate between and .

A path is alternating if its edges alternate between and .

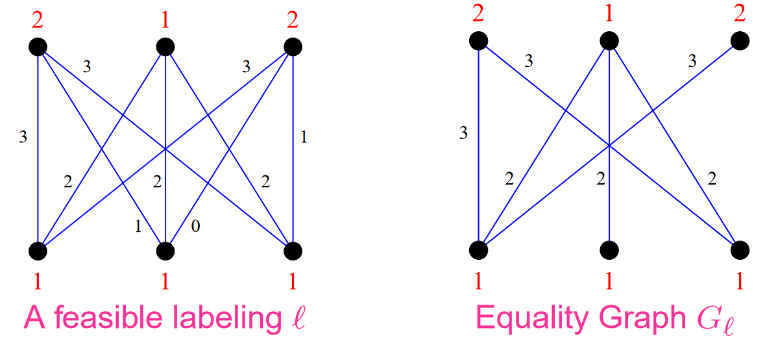
An alternating path is augmenting if both its endpoints are free.

The path is an augmenting path because it is alternating and and are free.

Important observation: Augmenting path can help us expand out matching. Swap the edges and the edges in the path, and will still be a valid matching. There is 1 more edges than edges in the path, so the new will increase in size by 1.

An alternating tree is a tree rooted at some free vertex in which every path starting from the root is an alternating path.

Let be a complete bipartite graph. The assignment problem is to find a max weight matching in . A perfect matching is an in which every vertex is adjacent to some edge in . A max weight matching is always perfect.

Labelings

A vertex labeling is a function .

A feasible labeling is a labeling such that .

The equality graph of a labeling is where .

Kuhn-Munkres theorem: if is feasible and is a perfect matching in , is a max weight matching.

Very big brain proof:

Let be any perfect matching in . Since every is covered exactly once by ,

Let be any perfect matching in .

Therefore, so is optimal.

The 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.

Basic outline

Start with any feasible labeling and a matching in .

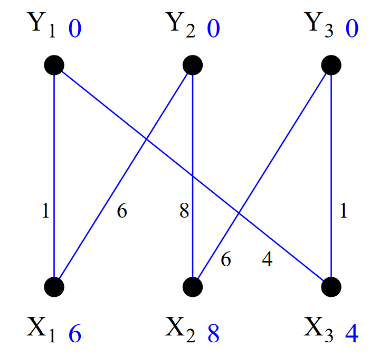
While is not perfect:

Find an augmenting path in , increase the size of .

If no augmenting path exists, improve to such that .

In each step or must be increasing so the algorithm will terminate.

When the process terminates, will be a perfect matching in , which means is an max weight matching by the Kuhn-Munkres theorem.



Initialization

It is obvious that this labeling is feasible. For the initial matching, we can just start with an empty one.

Pick a free vertex to be the root of an alternating tree. Set contains the vertices in and in the alternating tree, while set contains the vertices in and in the alternating tree.

Initially, .

Improving the labeling

If , the matching in cannot be further improved, so must be improved to such that .

After the change, we can see that edges originally in the alternating tree maintain the property that , but there will be new edges that also satisfy the condition, which are those that , because for those edges only decreased the right amout to satisfy the condition.

Extending the alternating tree and finding augmenting path

Now (if it is, improve the labeling). Pick a vertex .

If is free, an augmenting path from the root of the alternating tree to is found! Update the matching and build the alternating tree again.

If is matched, we can extend the alternating tree. Let’s say is matched to . We can add edges and to the tree. is in , and is in . will never be in before adding the edges because the root is free and matched edge of another vertex is already in the tree.

The Hungarian algorithm

Generate an initial labeling and a matching

while is not perfect:

pick a free vertex to be the root of alternating tree .

while true:

if , improve to

pick

if is free, update according to the augmenting path , break

else, extend the alternating tree