Advanced algorithms: CSL758
Maximum Weight bipartite graph matching

Vishesh Sharma(2004cs50224)

17th April, 2008

1 Introduction

Given a bipartite Graph G=(V,E), and a weight function that assigns weights to all the edges e ϵ E, our aim is to find a maximum weight matching of the graph G and formulate this problem as a linear programming problem.

2 Algorithm

Step 1

We start by assigning weights to all the vertices such that for any edge e=(u,v), we have: $w_u + w_v \geq w(u, v)$.

Clearly there always exists a feasible weight assignment for the vertices since we can put arbitrarily large weights on them.

Claim: Sum of weights of all the vertices $\geq$ sum of weights of all the edges of the maximum weight matching.

Proof : For any feasible weight assignment we have $w_u + w_v \geq w(u, v)$. Thus we can easily say that for each edge in the maximum weight matching the sum of weights of its end vertices is at least equal to its own weight. Also since in a matching no two edges share a vertex, it implies that the sum of weights of all the vertices is at least as much as the sum of weight of all the edges of the matching.

Therefore we can say that any feasible weight assignment to the vertices provides an upper bound on the weight of the maximum weight matching . Thus in order to find the maximum weight matching we try to find such a weight assignment which is not only feasible but provides the minimum sum of weights of the vertices.

A **tight edge** is defined as an edge (u,v) such that $w_u + w_v = w(u, v)$

Step 2

Once the feasible weight assignment has been made we then consider the subgraph consisting of the **tight edges** only. We try to find a perfect matching in this subgraph. If we succeed we are done as shown below.

Claim: If a perfect matching is found in the subgraph of tight edges it is the maximum weight matching of the original graph.

Proof: The sum of the weight of the edges of the matching will be equal to the sum of the weights of all the vertices since the edges are tight. Also since the matching is perfect no vertex is left out. Now since the feasible weight assignment to the vertices always provide an upper bound on the weight of the maximum weight matching and in this case the upper bound is exactly equal to the weight of matching we can safely say that this is the maximum weight matching.

In case we are unable to find a perfect matching in the subgraph we continue with the following steps.

Note that we can assume the given graph to be a complete bipartite graph by adding the missing edges and giving them a weight of '0'. Therefore we can always find a perfect matching.

Definitions

Given a Graph G = (V,E) and a matching M,

An **alternating path** is a path in which the edges belong alternatively to the matching and not to the matching.

An **augmenting path** is an alternating path that starts from and ends on unmatched vertices.

An **alternating tree** is tree rooted at an unmatched vertex and containing only alternating paths.

Let us say that the set of vertices in the bipartite subgraph on the left side is 'U' and on right side is 'V'.

Step 3

Since we couldn't find a perfect matching in our subgraph there should be an unmatched vertex left from whatever maximal matching we found. We use this unmatched vertex as a root and build an alternating tree as explained below.

Taking the unmatched vertex as a root we take all the edges incident on it in the 'tight subgraph'. Let us say that this root belongs to 'U'. All the edges incident on this vertex have their other endpoints in 'V'. Therefore we get a set of vertices in 'V'. Now we consider only the matched edges from each of these vertices. If anyone of the vertices is unmatched we consider the path from the root to this unmatched vertex. Clearly this is an augmenting path. Since in an augmenting path the number

of unmatched edges are one more than the number of matched edges we reverse all the edges i.e. change the matched edges to unmatched and vice versa. Clearly this increases the size of matching by one.

In case all the vertices in 'V' are matched we get a set of vertices in 'U'. We continue the same procedure on each of these vertices that we performed on the root vertex i.e. considering all the incident tight edges on these vertex. In this way we keep building the alternating tree till we find an unmatched vertex in 'V'.

If we are unable to find any unmatched vertex in the alternating tree we do the following:

Step 4

In the alternating tree we decrease the weights of all the vertices of 'U' by 'δ' and simultaneously increase the weights of all the vertices of 'V' by 'δ'.

Claim : All the edges of the alternating tree remain tight.

Proof: For every edge in the alternating tree one end vertex is in 'U' while one is in 'V'. Thus the weight of one end vertex is increased by 'δ' while the weight of other is decreased by 'δ'. So the net effect is null.

Note that edges going out of the 'U vertices' can only be non tight edges while edges going out of the 'V vertices' can be both tight and non tight.

Also the weight assignment is not invalidated by increasing the weight of 'V vertices' because increasing the weight of any vertex cannot lead to violation of the weight condition. But increasing the weight of these vertices will make edges going out of these vertices non tight which previously were tight. But since these edges are not in our matching, this should not effect our algorithm.

All the edges going out of the 'U vertices' are non tight. So we choose a 'δ' such that only one of the outgoing edge becomes tight and the weight assignment also remains valid thereby obtaining an unmatched vertex i.e. the other end vertex of this newly 'tightened' edge. In this way we have found an unmatched vertex and hence can increase the size of the matching as explained above. Also as shown below we are bound to find one such unmatched vertex.

Claim: For an alternating tree which has no augmenting path, we can always find an unmatched vertex via a non tight edge going out of a 'U vertex' by reducing its weight and thereby tightening the edge.

Proof: For alternating tree with no augmenting path, $\mid U \mid = \mid V \mid +1$. Since the graph is bipartite the number of vertices both in 'U' and 'V' are the same.So there is atleast one vertex in 'V' which is not in the tree with an edge to a vertex in 'U' in the tree.

From the above analysis we can conclude that we can always find an unmatched vertex and thereby increase the size of our matching maintaining the validity of the

weight assignments and keeping all the edges in our matching tight.

Step 5

We keep on repeating the above procedure till we end up with a perfect matching and thereby finding the maximum weight matching.

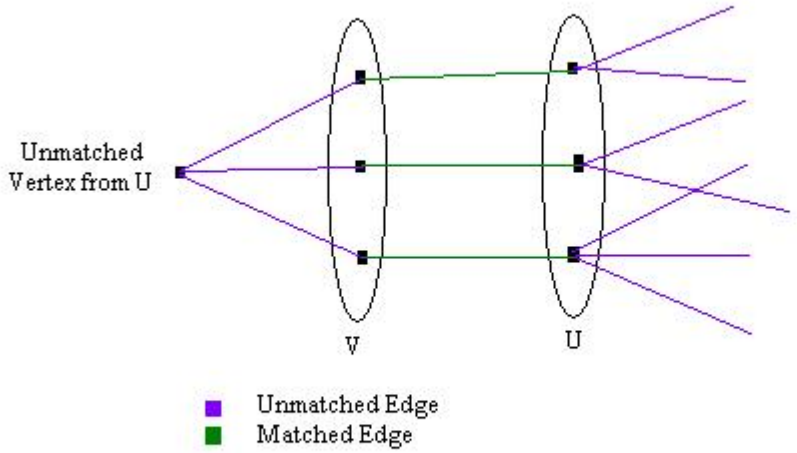

Figure 1: Alternating Tree with an unmatched vertex as root

3 Linear Programming Formulation

The ***Maximum weight matching*** problem can be formulated as an integer programming problem as done below:

Let x_e be the variable associated with and edge e and defined as:

$x_e = 1$, if e is a matched edge

$= 0$, otherwise

Objective: Max $\Sigma_e w_e x_e$, under the condition:

$\forall v \in U \cup V, \quad \Sigma_{e \in \delta(v)} x_e \leq 1,$

where $\delta(v)$ is the set of edges incident on vertex v

and $x_e \in \{0, 1\}$.

Since this Integer programming problem is NP-hard we need to reduce it to general Linear programming problem which can be done simply by relaxing the condition $x_e \in \{0, 1\}$ to $x_e \geq 0$

The solution to this LP is larger than the original integer problem because the constraints have been relaxed.