CSL 758: Advanced Algorithms

Scribe: Aditya(2004CS50207), Abhishek Arora(2004CS50206)
Lecturer: Naveen Garg, Kavitha Telikepalli

Date: April 17, 2008

Maximum Weight Bipartite Graph Matching

1 Introduction

In this lecture we will discuss the Hungarian algorithm to find a matching of maximum possible weight in a bipartite graph. We also discuss the integer programming formulation of the problem and its relaxation to Linear Programming(LP) problem.

2 Algorithm

Consider a bipartite graph $G = (V, E)$. We define weight assignment on vertices: Put weights w_v on vertices such that for all edges $e(u, v) \in E$, $w_u + w_v \geqslant w_{e(u,v)}$.

Note that there always exists a feasible weight assignment(of vertices) because we can always put arbitrary high weights while edge weights are finite. In particular, We can give 0 weights to all the vertices in right part of bipartite graph G and for each vertex in left part we give it a vertex weight equal to maximum of all the edge weights of the edges incident on it. Clearly, this weight assignment is a feasible one. We Consider this feasible weight assignment.

Claim: For a feasible weight assignment, Sum of vertex weights $\geqslant$ Weight of maximum weight matching.
Proof: We know that in a feasible weight assignment, for any edge $e(u, v) \in E$, $w_u + w_v \geqslant w_{e(u,v)}$. A matching, in particular a maximum weight matching, which contains finite number of edges, summation over all edges gives the result.

So finding any feasible weight assignment gives an upperbound on the weight of maximum weight matching. So to get a better bound, we should try to find a feasibile weight assignment such that total weight of vertices is smallest possible.

Definitions

Tight Edge: An edge $e(u, v) \in E$ is tight if $w_u + w_v = w_{e(u,v)}$.

Alternating path: A path is alternating if its edges alternate between M, a matching and $E - M$. An alternating path is *augmenting* if both endpoints are unmatched. 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. An *alternating tree* is a tree rooted at some unmatched vertex in which every path is an alternating path.

Now consider the subgraph of tight edges only. It may or may not contain a perfect matching. If it contains a perfect matching then we are able to find a matching in G such that it has its weight exactly equal to sum of vertex weights which in turn was upperbound for maximum weight

matching and so we can not find a higher weight matching. But, if no perfect matching exists in this tight subgraph then we update the vertices weights to move towards making this matching a perfect matching. So we concentrate on finding a perfect matching while maintaining feasible weight assignment condition.

Note that, without loss of generality, by adding edges of weight 0, we may assume that G is a complete bipartite graph, so that we can always find a perfect matching.

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

ALGORITHM

1. Start with a feasible weight assignment to vertices(as discussed above) and consider any matching M in tight subgraph.

If M is perfect then STOP else go to 2.

2. While M is not perfect repeat the following:

a. Start with some unmatched vertex from U as root and make out an alternating tree. Find an augmenting path for M in tight subgraph. This increases size of M.

b. If no augmenting path exists, update the weight assignment to all the vertices in tree such that we are able to tighten atleast one untight edge going out of tree from U side, so more edges can now be used to extend the tree (How to do this exactly is described below).Go to a.

Note that V side can have both tight and untight edges going out of the alternating tree. But U side can have only untight edges going out.

If no augmenting path exists, what we do is decrease the weights of all U side vertices in tree by δ and increase the weights of all vertices on V side in tree by δ. Note that for all the vertices in tree and all edges among them, state of tightness is not going to change by this operation. There can be tight and untight edges going out from vertices in V. For these, only some tight edges can now become untight since weight of other vertex of such an edge remains same. Now from U side there can only be untight edges going out of the tree. So, to keep weight assignment feasible, δ is constrained to have some maximum value. For a vertex in tree from U, δ can take a minimum of all the values required to make all the untight edges tight separately. So, it comes out that for all vertices from U side δ can not be greater than minimum of all the individual minimum values for all U vertices in the tree. So, by choosing this value of δ we convert atleast one untight edge to tight edge and this edge is also unmatched, hence giving a way to extend the alternating tree.

Claim: At step 2b we are able to find atleast a vertex outside of tree connected through untight edge, so that this can be tightened.
Proof: For alternating tree with no augmenting path, $\mid U \mid = \mid V \mid + 1$. Now graph being bipartite cardinality of left side is same as cardinality of right side. So there is atleast one vertex on right side which is not in tree, with an edge to any vertex in tree in U, hence with untight edge.

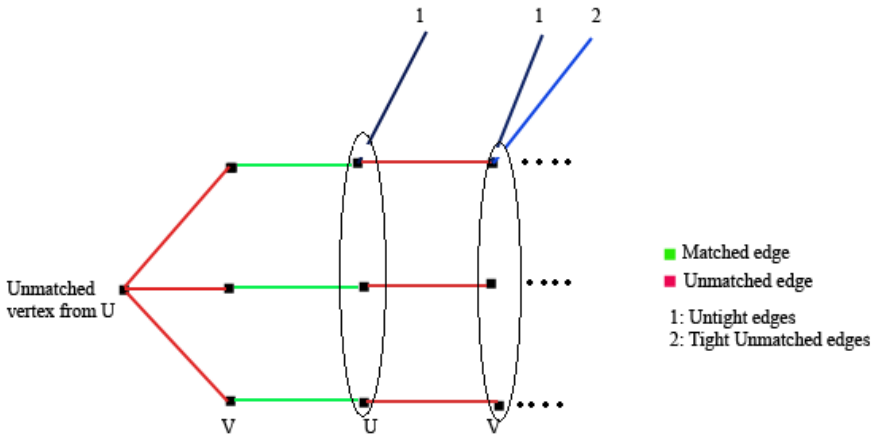

Figure 1: Alternating tree with unmatched vertex of U as root with egdes going out of tree as shown.

3 Integer Programming Formulation

This problem can also be formulated as Integer Linear programming as follows.
Let x_e is a variable associated with edge $e \in E$ such that

$x_e = 1$ if e is a matched edge, 0 otherwise.

Maximum weight matching problem can be formulated as ,

$$\begin{aligned} &\text{Max } \Sigma_e \; w_e * x_e = \textit{weight of matching} \\ &\quad \forall \; v \in U \cup V, \; \Sigma_{e \in \delta(v)} \; x_e \leq 1 \\ &\quad \forall \; e \in E, \; x_e \in \{0, 1\}. \end{aligned}$$

Optimum solution to this problem is the weight of maximum weight matching. But, since we can not solve this in polynomial time we relax integer constraint. Now we relax the integralty contraint so as to convert Integer Linear program to a LP problem

$$\begin{aligned} &\text{Max } \Sigma_e \; w_e * x_e = \textit{weight of matching} \\ &\quad \forall \; v \in U \cup V, \; \Sigma_{e \in \delta(v)} \; x_e \leq 1 \\ &\quad \forall \; e \in E, \; 0 \leq x_e \leq 1. \end{aligned}$$

This is called the *LP relaxation* of the original Integer programming problem. The solution to this LP is going to be larger than the original Integer program because of more relaxed constraints on x_e. We solve LP and get some upperbound for maximum weight matching.