CSL 865: Algorithmic Graph Theory **Semester-I 2013-14**

Lecture 5: July 7,2013

Lecturer: Naveen Garg *Scribes: Ankit Anand*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

Minimum Cost perfect matching in general graphs

Given a general graph G(V,E) with weights on edges (w(e) ϵ R), we intend to find minimum cost perfect matching in the graph.

Note: If graph is not complete and perfect matching does not exist, we will make it complete by adding extra edges and vertices as discussed in previous lecture.

5.1 Linear Program

In the case of bipartite graph, the minimum cost perfect matching is given by following Linear Program:

Minimize:

$$F_{obj} = \sum_{e \epsilon E} w_e * x_e \tag{5.1}$$

where w_e are weights on edges and x_e is a variable corresponding to each edge.

Constraints:

$$\sum_{e \epsilon \delta(v)} x_v = 1 \quad \forall v \, \epsilon \, V \tag{5.2}$$

$$x_e \geq 0 \tag{5.3}$$

But the above constraints are not sufficient for general graph. There is a big gap between LP and ILP solution. We explain the same with a counter example.

Counter example: Consider the graph shown in figure 5.1. The LP will assign each of $x_e = 1/2$ but this does not give us an integral perfect matching solution . These fractional values arise because of odd cycles in graph. To eliminate such solutions and get a perfect matching in the graph, we need to have an additional constraint in the graph.

A key observation in this case is all odd subsets of vertices should have an edge going outside to have a perfect matching in general graph.

Linear Program for General Graph :

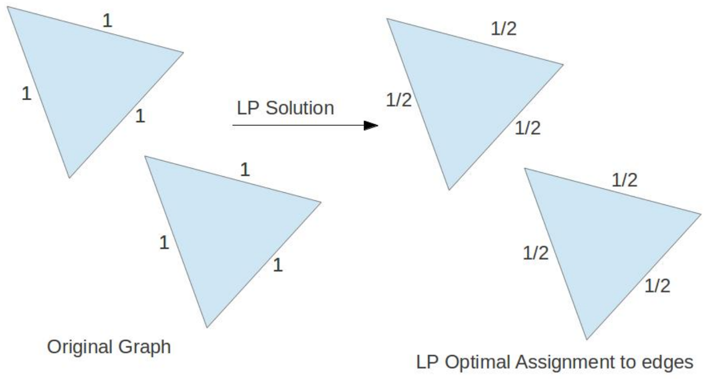

Figure 5.1: Counter example for General Graph

Minimize:

$$F_{obj} = \sum_{e \epsilon E} w_e * x_e \tag{5.4}$$

Constraints:

$$\sum_{e \epsilon \delta(v)} x_v = 1 \quad \forall v \, \epsilon \, V \tag{5.5}$$

$$\sum_{e \epsilon \delta(S)} x_e \geq 1 \, \forall S \, \subseteq \, V \, , |S| \, is \, odd \tag{5.6}$$

$$x_e \geq 0 \tag{5.7}$$

The optimal solution to this LP will be integral (for details refer Chandra Chekuri Notes)

We will give a Primal-Dual algo for this linear program.

Dual Linear Program:

Maximize:

$$F_{obj} \; = \sum_{v \epsilon V} Y_u + \sum_{S \subseteq V \; |S| is odd, |S| \geq 3} Y_s \tag{5.8}$$

Constraints:

$$Y_s \; \geq \; 0 \tag{5.9}$$

$$\forall e \,\epsilon\, E\,, E=(u,v)\ ;\ \left[Y_u+Y_v+\sum_{S\epsilon\delta(S),|S| is odd,|S|\geq 3} Y_S\right]\leq w_e \tag{5.10}$$

The point to be noted is that Dual variable of individual vertices can be +ve as well as -ve where as dual values corresponding to odd sets should be +ve only.

Edmonds Algorithm for Minimum Cost Perfect Matching

We will explain the new edmonds algorithm with the help of an example. Given a graph shown in figure 5.2, we have to find minimum cost perfect matching in this graph. We will mark tight edges as green and

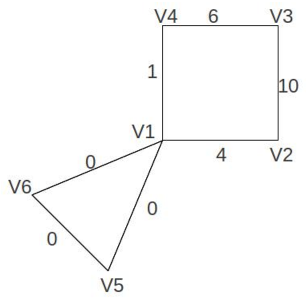

Figure 5.2: Original Graph

matched edges as red through out the algorithm.

Initally, we will put zero dual values on all vertices and check if any edge becomes tight. At each step,we will form a graph of tight edges and run edmonds algorithm if a perfect matching exists in this graph. If a perfect matching exists, then we are done and report that matching as minimum cost perfect matching. In each iteration, we will modify this graph as explained below.

As shown, after first iteration we will obtain the graph shown in figure 5.3 where V1- V6, V1-V5 and V5-V6 have become tight. We find a matching in this graph, V5- V6 and trying to augment this matching, we find a blossom, so will shrink it into a single vertex V1 as shown in figure 5.4.

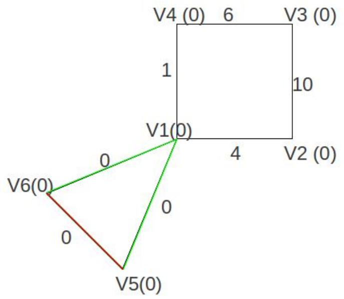

Figure 5.3: Graph after first iteration

We increase the dual values of the remaining vertices as shown in figure 5.5 and 5.6 to increase tight edges and update the matching in tight edge sub-graph.

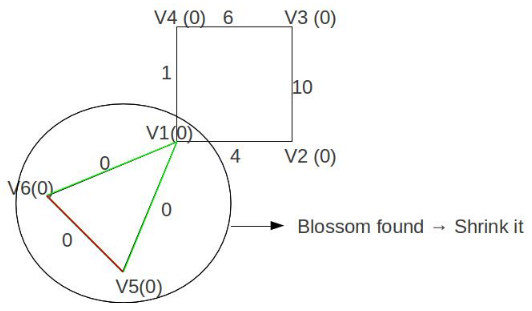

Figure 5.4: Graph on finding a blossom

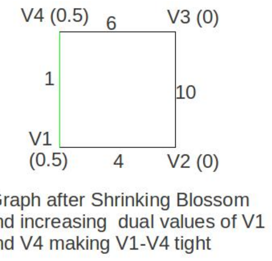
V4 (0.5)
6
V3 (0)
1
10
V1
(0.5)
4
V2 (0)
Graph after Shrinking Blossom and increasing dual values of V1 and V4 making V1-V4 tight

Figure 5.5: Making V1-V4 tight

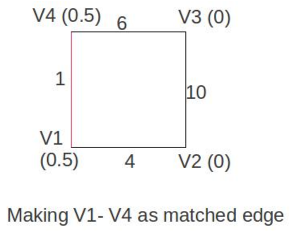

Figure 5.6: Making V1-V2 tight

In figure 5.8, we observe that neither there is an augmenting path, nor there is a blossom in sub graph, so there must be a Tutte Set. Here, we will reduce dual values of vertices in tutte set and increase dual values of odd components. This can lead to three situations:

- An edge between two odd components becomes tight
- An edge between an odd an even component becomes tight
- If dual value of a vertex representing a blossom touches zero, it will lead to unshrinking of blossom.

In the example shown in our case, it will lead to unshrinking of blossom at V1 as shown in figure 5.9

After unshrinking the blossom, we will find augmenting path in new sub-graph of tight edges, but again we encounter the same situation where we will neither find an augmenting path nor a blossom, so there is a Tutte set consisting of V1 where as V2, V3 form odd components of this tutte set (Figure5.10). We will reduce the dual value of V1 (can be made -ve -singleton) and increase dual value of V2 and V3 (Figure 5.11).

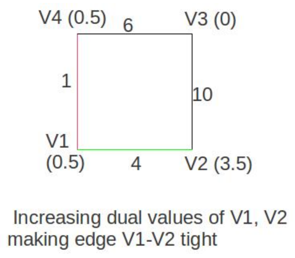

Figure 5.7:

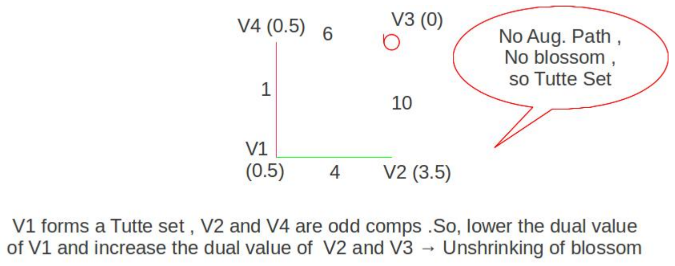
V4 (0.5)
6
V3 (0)
1
10
V1 (0.5)
4
V2 (3.5)
No Aug. Path , No blossom , so Tutte Set
V1 forms a Tutte set , V2 and V4 are odd comps .So, lower the dual value of V1 and increase the dual value of V2 and V3 → Unshrinking of blossom

Figure 5.8: No augmenting No blossom case

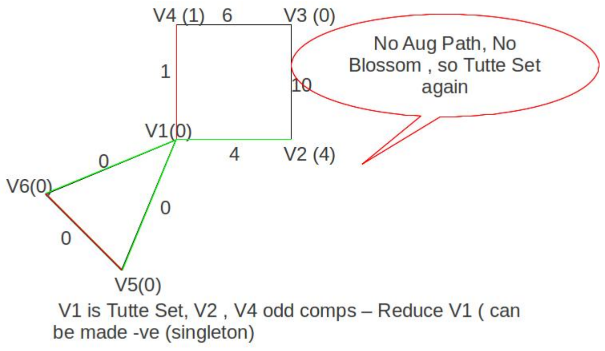
V4 (1)
6
V3 (0)
1
10
V1(0)
4
V2 (4)
No Aug Path, No Blossom , so Tutte Set again
V6(0)
0
0
0
V5(0)
V1 is Tutte Set, V2 , V4 odd comps – Reduce V1 (can be made -ve (singleton)

Figure 5.9: Unshrinking of blossom

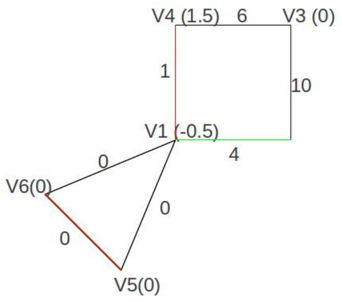

Figure 5.10: Finding augmenting Path

Then, in the next iteration we will increase the dual values to make V3-V4 tight. Finding the new augmenting path and matching, we find the Maximum matching as shown in figure 5.12.

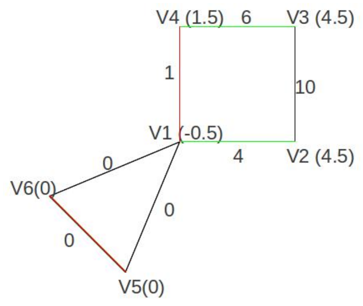

Figure 5.11: Updated Dual values

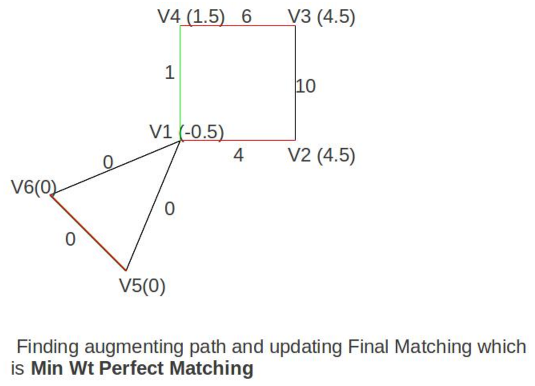
V4 (1.5)
6
V3 (4.5)
1
10
V1 (-0.5)
4
V2 (4.5)
0
V6(0)
0
0
V5(0)
Finding augmenting path and updating Final Matching which is **Min Wt Perfect Matching**

Figure 5.12: Final Min Cost Max Matching

Definition 5.1 *A group of sets U will form a Laminar family if* $\forall S1, S2 \epsilon U$ *, either S1 and S2 are disjoint or* $S1 \subseteq S2$ *or* $S2 \subseteq S1$ *. The same is explained in figure 5.13.*

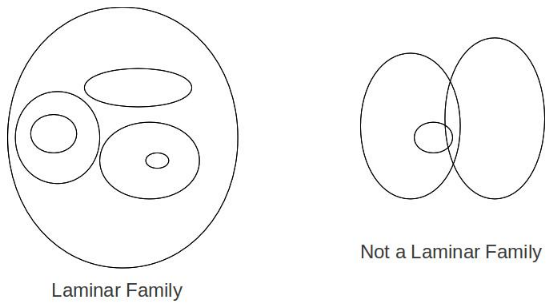

Figure 5.13: Laminar Family Concept

Claim 5.2 *All those odd sets which have +ve dual variables form a Laminar Family.*

Proof: Apart from singleton vertices, all odd sets get +ve values when they are shrinked to Supervertex (Set of Vertices in odd set), then either they are unshrinked and remain as such or these Supervertices become part of another odd set which is shrinked again forming in all a laminar family. ■

Claim 5.3 *No of odd sets S having dual value* ≥ 0 *is at max n-1.*

Proof: All the odd sets form a Laminar family and all the sets in a Laminar family can be arranged in a tree structure. If individual vertices(n) form the leaves of tree, then maximum nubmer of internal nodes can be n-1. ■

Time Complexity of Algorithm

An important point to note is that size of matching can't decrease in the iterations of algorithm since a matched edge never becomes untight. Now, consider an iteration where matching increases from M to M+1. We analyse all operations that can happen in course of increasing the matching size by 1 :

- We have to build an alternating tree which can be done in O(m) time where m is maximum number of edges.
- Also, we note that we can have a series of shrinking and unshrinking of edges in each iteration but set of all shrink sets won't be unshrink in same iteration. At maximum, there are O(n) those sets , so maximum number of shrinkings and unshrinkings are bounded by O(n).
- Thirdly, certain edges may become tight which may lead to either increasing the size of matching (in direct case - no blossom-edges between odd and even components), may lead to formation of blossom (shrinking bounded by O (n)-edges between odd and odd components.)

Overall, even if we have to build the whole alternating tree after every shrinking, unshrinking , we will take O(mn)(m - time to build tree and n - no of shrinks and unshrinkings). Also, perfect matching size is n/2 (perfect matching), so total no of incremental iterations is bounded by O(n). Hence, total time for the whole algorithm is $O(mn^2)$ and hence, a polynomial time algorithm.