CSL851: Algorithmic Graph Theory	Semester I 2013-14
Lecture 4: August 5	
Lecturer: Naveen Garg	*Scribes: Utkarsh Ohm*

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.*

4.1 Min-weight perfect matching

In this lecture, we discuss the problem of finding the minimum weight perfect matching in a weighted bipartite graph. Let us start with assuming that we have an algorithm to find the min-weight perfect matching. In the first section of the lecture, we learn how to use this algorithm in solving other problems of weighted general graphs. In the second section, we discuss the algorithm itself.

4.1.1 Min-weight perfect matching problem as a general problem

4.1.1.1 Finding the max-weight perfect matching

Consider a graph G with a min-weight perfect matching M and weight w(e) $\forall$ edge e in G. How can we find the max-weight perfect matching for this graph? Consider the algorithm shown in Figure 4.1.

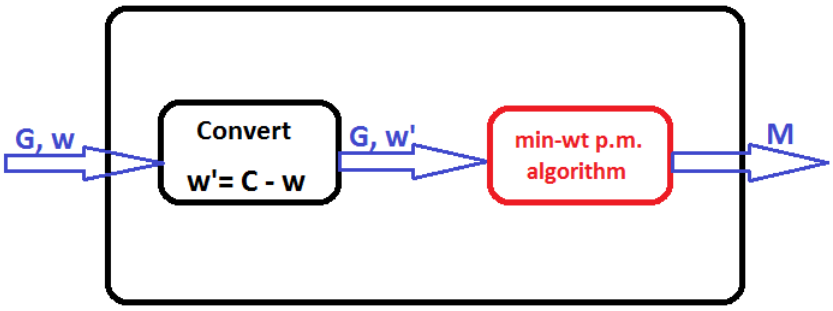

Figure 4.1: M: Max-weight perfect matching

Claim 4.1 *M is also the max-weight perfect matching of G*

Proof: Let C denote the maximum edge weight in G i.e. $C = \max_{\forall e \in G} w(e)$ and n be the size of M. The weights of the edges are changed such that $w'(e) = C - w(e)\ \forall e \in G$. Then, the algorithm to find the min-weight perfect matching is run on G with edge weights w'. Summing across edges in M, we get

$$\sum_{e \in M} w'(e) = C * n - \sum_{e \in M} w(e)$$

We know that $\sum_{e \in M} w(e)$ is the minimum possible weight for all perfect matchings. Since both weight summations are over the same matching M, if the minimum value is subtracted from a constant value, the resulting number would be the maximum possible value i.e. $\sum_{e \in M} w'(e)$ is the maximum weight for all perfect matchings. Hence, M is also the max-weight perfect matching in G. ■

4.1.1.2 Finding the min-weight maximum matching

Consider a graph G with non-zero weights w(e) $\forall$ edges e in G. Let C denote the maximum edge weight in G i.e. $C = \max_{\forall e \in G} w(e)$. How can we find the min-weight maximum matching for this graph? Consider the algorithm shown in Figure 4.2. M' is the min-weight perfect matching of G' as described in the figure and let $2 * n$ be the number of nodes in G'. Note that the *large* weight of the dummy edges refers to a weight $> C * n$ and all of them have the same weight.

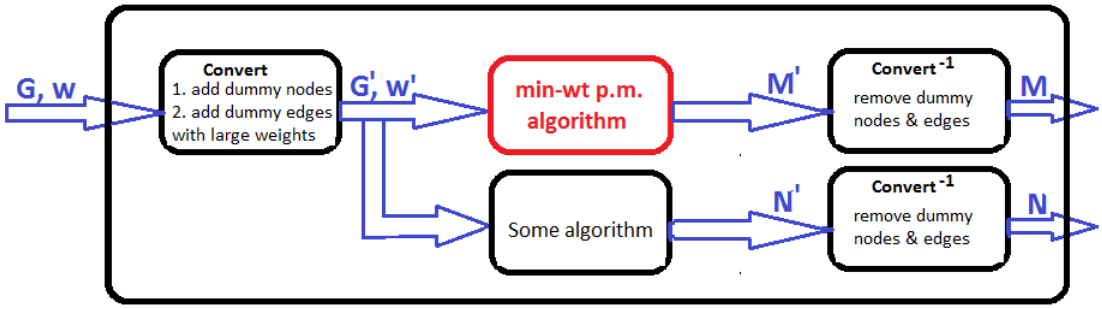

Figure 4.2: M: Min-weight maximum matching

Claim 4.2 *M, as obtained from Fig 4.2, is a maximum matching of G*

Proof: Assume that there is some matching N (different from M) which is the maximum matching of G obtained by some other algorithm as shown in Fig 4.2. So, N matches more nodes than M in graph G i.e.

$$|N| > |M| \tag{4.1}$$

Let N' and M', respectively be the corresponding perfect matchings in G' such that M' is the min-weight perfect matching. Since $n - |M|$ dummy edges are picked in M' and $n - |N|$ in N', we get

$$(n - |M|) * C * n < \sum_{e \in M'} w'(e) < (n - |M| + 1) * C * n \tag{4.2}$$

$$(n - |N|) * C * n < \sum_{e \in N'} w'(e) < (n - |N| + 1) * C * n \tag{4.3}$$

From definition of N' and M' and because M' is min-weight perfect matching of G', we get

$$\sum_{e \in M'} w'(e) < \sum_{e \in N'} w'(e) \tag{4.4}$$

By (4.2),(4.3),(4.4) we get

$$(n - |M|) * C * n < (n - |N| + 1) * C * n \Rightarrow |M| > |N| - 1 \Rightarrow |N| < |M| + 1 \Rightarrow |N| \geq |M| \tag{4.5}$$

We have reached a contradiction with (4.1). Hence, M and not N, is a maximum matching of G.

■

Claim 4.3 *M is a min-weight maximum matching of G*

Proof: After Claim 4.2 we only need to prove that M is min-weight among all possible maximum matchings. Assume that N (different from M), as obtained from another algorithm in Fig 4.2, is another maximum matching like M but is in fact, the min-weight maximum matching in graph G.

$$\sum_{e \in M} w(e) > \sum_{e \in N} w(e) \tag{4.6}$$

$$|M| = |N| \tag{4.7}$$

Let N' and M', respectively be the corresponding perfect matchings in G'. The algorithm ensures that M' is the min-weight perfect matching, we get

$$\sum_{e \in M'} w'(e) < \sum_{e \in N'} w'(e) \tag{4.8}$$

$$|M'| = |N'| = n \tag{4.9}$$

By (4.7) and (4.9), M' and N' both consist of equal number of dummy edges which have the same large weight value i.e.

$$\sum_{e \in M' - M} w'(e) = \sum_{e \in N' - N} w'(e) \tag{4.10}$$

From definition of w'(e) we know

$$w'(e) = \begin{cases} w(e) & \text{if } e \in G \\ \text{large weight} > C * n & \text{if e is a dummy edge} \end{cases} \tag{4.11}$$

Using above definition, we can add (4.6) and (4.10) to get $\sum_{e \in M'} w'(e) > \sum_{e \in N'} w'(e)$

We have reached a contradiction with (4.8). Hence, M and not N, is a min-weight maximum matching of G.

■

4.1.1.3 Finding the min-weight perfect matching in a graph with both positive and negative weights

Consider a graph G with both positive and negative edge weights. Remove the positive weight edges from G and make the weights of the negative weight edges positive by miltiplying by -1. Now add dummy nodes as

in previous problem and dummy edges between all nodes such that their weights are 0. Let this new graph be G'.

Exercise: Prove that the max-weight perfect matching of G' (which can be found using Section 4.1.1.1) can be used to find the min-weight perfect matching of G.

4.1.2 The min-weight perfect matching algorithm for bipartite graphs

Consider graph G with 2 partitions U and V. If $|U| \neq |V|$ then dummy edges with a large weight as defined in section 4.1.1.2 are added to make the partitions equal in size and let this graph be G'. This ensures that G' has perfect matching(s) between U and V. Let us denote a perfect matching in G' by M. We can now model the min-weight perfect matching problem into an optimisation problem.

$$\min \sum_{e \in G'} x_e w_e \tag{4.12}$$

$$\text{s.t. } x_e = \begin{cases} 1 & \text{if } e \in M; \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

$$\sum_{e \in \delta(v)} x_e = 1 \; ; \forall v \in U, V \tag{4.14}$$

$$0 \leq x_e \; ; \forall e \in G \tag{4.15}$$

The dual for this L.P is as follows.

$$\max \sum_{w \in U \cup V} y_w \tag{4.16}$$

$$\text{s.t. } y_u + y_v \leq w_e \; ; \forall e \equiv (u, v) \; ; \forall u \in U, \forall v \in V \tag{4.17}$$

Definition 4.4 *An edge $e \equiv (u, v)$ is called a* **tight edge** *if $w_e = y_u + y_v$*

4.1.2.1 The algorithm inspired by the dual LP

The idea is to pick any dual solution such that subgraph of tight edges has a perfect matching. Formally the algorithm is:

```
Start with a dual solution
until perfect matching found in subgraph of tight edges do
    if tight edges have no perfect matching then
        find Hall set and modify dual values accordingly, expanding the subgraph
    endif
```

Let us consider the example given in Fig 4.3(a). Start with dual values such that $y_v = 0 \; \forall v \in V$ and $y_u = \min\{w_e : e \in \delta(u)\} \; \forall u \in U$. This gives us a graph which has atleast one tight edge for each vertex in U. Start from a vertex in U and start building alternating trees (as done in first lecture) to match edges in the subgraph of tight edges. Say, we first picked u_2 and matched it to v_2. Next we picked u_3 and matched it to v_3. Next we picked u_1 but found a hall set. So far we have got Fig 4.3(b) and we can cannot find an alternating path in Fig 4.4(a). There is no perfect matching in the subgraph of tight edges and we have

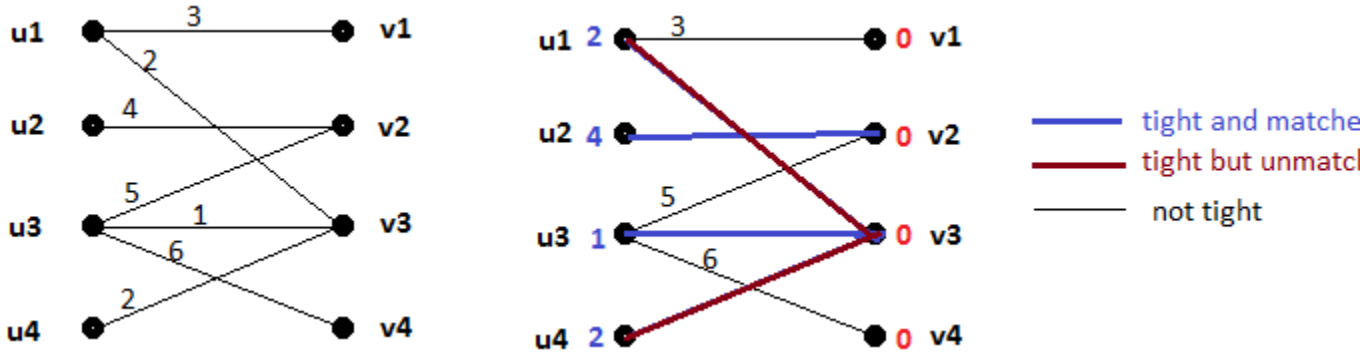

Figure 4.3: (a): original graph; (b): graph with a dual solution

found a hall set (in this case $u_1 \cup u_3$). So now, we modify dual values in such a way that we can grow our subgraph of tight edges. Starting from root of alternating tree we alternately add and subtract δ. We start by trying very small values for δ and stop as soon as value of δ has caused us to make an edge tight from among all the untight edges incident on vertices of alternating tree. Fig 4.4(b) demonstrates this event for $\delta = 1$. Since u_1 now has a tight but unmatched edge, it is matched to v_1. This leaves our graph at Fig 4.4(d). Continuing this approach, we finish with Fig 4.5(g) as the min-weight perfect matching of weight 15.

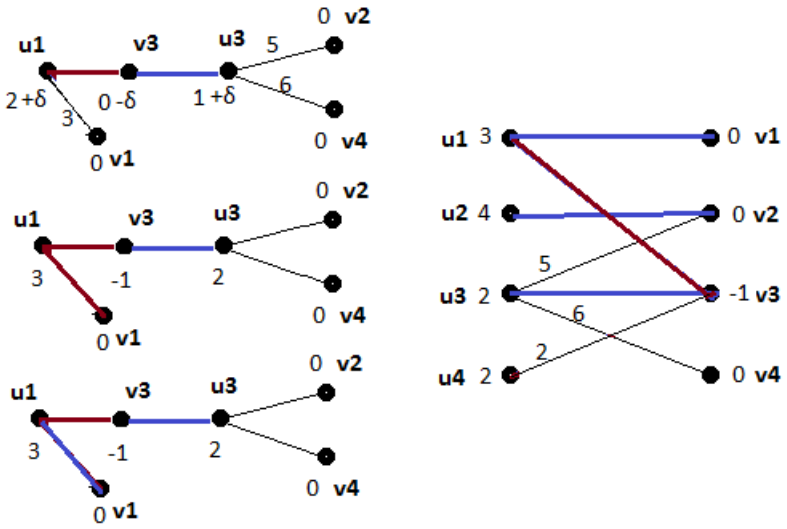

Figure 4.4: (a),(b),(c): modification of dual values in alternating tree; (d): improved dual solution

4.1.2.2 The algorithm's correctness

Claim 4.5 *The above algorithm terminates to give the min-weight perfect matching.*

Proof: Whenever we find a hall set, the number of vertices of U in the alternating tree is 1 more than the number of vertices in V. Since, in the alternating tree, the dual values of vertices of U is increased by δ and those of V is decreased by δ, and the dual values of vertices outside the alternating tree does not change, the change in sum of all dual vaues is $+\delta$. Hence the dual increases after every iteration of the alogrithm's loop.

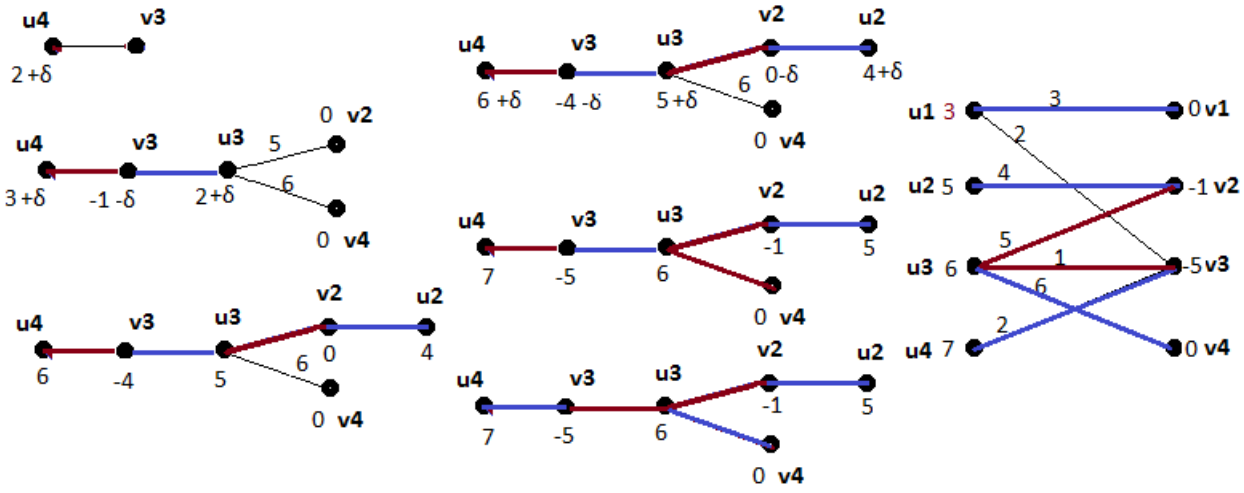

Figure 4.5: (a)-(f): modification of dual values in alternating tree; (g): min-weight perfect matching

Since the sum of the dual values is bounded by sum of weights of all edges, the algoithm has to terminate. Since this algorithm solves the dual LP and terminates, its correctness is guaranteed. ■

It must be noted that the primal-dual approach adopted here serves as a general framework to reduce weighted graph problem to a unweighted one.

4.1.2.3 The algorithm's time complexity

Assume that $|U| = |V| = n$ and there are m edges in the original graph.

1. The initial dual solution can be constructed in $O(m)$ time.
2. Starting from any one vertex, building an alternating tree, finding hall set and modifying dual values can take $O(n)$ time in the worst case, if we have to check all vertices. We assume that we can store the value of min weight incident on each vertex using data structures like heaps.

Step 1 is done once in the entire algorithm while step 2 is executed for each vertex in U i.e. n times. Hence, time complexity is $O(m + n^2)$.

4.1.3 The min-weight perfect matching algorithm for general graphs

The bipartiteness of the graph was exploited by the previous section's algorithm, only in confirming the existence of a hall set. So intuitively, we can see that this algorithm could be extended to work on general graphs too. But the algorithm fails in a graph like that in Fig 4.6.

The primal (also, the correct) solution is 2 + large number (for the dotted dummy edges) whereas the dual solution is 3 ($= \frac{1}{2} * 6$). This is due to the odd connected components (with more than 3 vertices) of the graph. A set of *Odd set constraints* are added to the LP to ensure that at least one vertex of each odd component in a graph is matched outside the component. Hence the LP formulation for general graph G with vertices V is as follows.

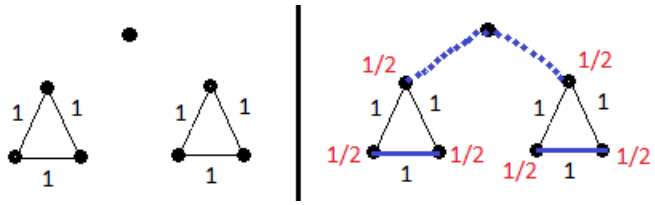

Figure 4.6: LP algorithm fails

$$\min \sum_{e \in G} x_e w_e \tag{4.18}$$

$$\text{s.t.} \sum_{e \in \delta(v)} x_e = 1 \; ; \forall v \in V \tag{4.19}$$

$$0 \leq x_e \; ; \forall e \in G \tag{4.20}$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \; ; \forall S \subseteq V, |S| odd \geq 3 \tag{4.21}$$

The dual for this L.P is as follows.

$$\max \sum_{w \in V} y_w + \max \sum_{S \subseteq V, |S| odd \geq 3} y_S \tag{4.22}$$

$$\text{s.t. } y_u + y_v + \sum_{e \in \delta(S)} y_S \leq w_e \; ; \forall e \equiv (u,v) \; ; \forall u, v \in V \; ; \forall S \subseteq V, |S| odd \geq 3 \tag{4.23}$$

$$y_S \geq 0 \; ; \forall S \subseteq V, |S| odd \geq 3 \tag{4.24}$$

4.1.4 Further reading

Read lecture Chandra Chekuri's notes at `http://courses.engr.illinois.edu/cs598csc/sp2010/Lectures/Lecture10.pdf`.