1 Definitions and Notations

We use $G = (V, E)$ to denote a undirected unweighted graph, where V and E are the set of vertices and the set of edges respectively. We use M to denote a matching in G.

A **blossom** is a set of nodes and edges starting at an open vertex, with a "stem" of even number of edges (matched and unmatched edges alternating), and a cycle of odd number of edges (again with alternating matched and unmatched edges, but with the two edges adjacent to the stem unmatched). A illustration is shown in Figure 8.1(a).

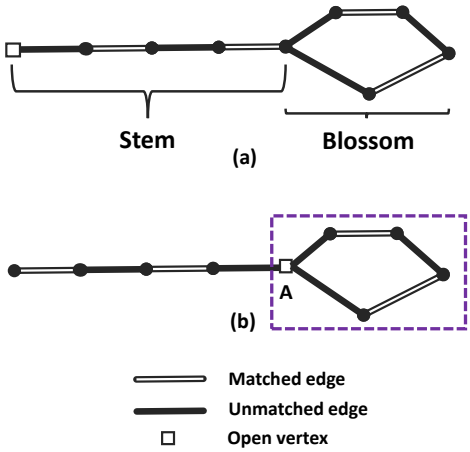

Figure 8.1: An example of blossom and the toggling of the stem.

2 Edmonds Blossom Algorithm Overview

The idea of Edmonds Blossom Algorithm comes from the Berge's Theorem, which we proved in the previous lecture.

Theorem 8.1. *Given a graph G and a matching M, M is a maximum matching if and only if there exists no M-augmenting path.*

For a blossom, we can always toggle the stem part, such that the open vertex moves to the blossom. As an illustration, after toggling the stem from Figure 8.1(a), we get the blossom in (b), where the vertex A is now the open vertex. After this toggling, the length of stem is zero, and the blossom is now the part in the purple rectangle.

Here we introduce an algorithm `FindAugPath` to explore augmenting paths in a graph given a matching M. There are three possible results of this algorithm:

1. **It returns "No M-augmenting path".** In this case, M is the maximum matching.

2. **An augmenting path P is found.** We can now augment along P, by setting $M \leftarrow M \Delta P$. Recall that $M \Delta P$ is the symmetric difference of M and P, and is a matching of size $|M| + 1$.

3. **A blossom is found.** Denote the blossom as B. Toggle the stem and shrink the blossom, getting the graph G/B. Since the blossom contains an open vertex at the base (after the toggle), the new vertex v_B obtained by shrinking is an open vertex in the new graph G/B. (This process was explained in the last lecture, but we illustrate it in Figure **??**). Call `FindAugPath` on G/B and M/B and get an M/B-augmenting path P' in G/B. Finally, extend P' to M-augmenting path P in G. (See Figure **??**.)

The correctness of `FindAugPath` are captured in the following theorem:

Theorem 8.2. *The algorithm* ***`FindAugPath`*** *runs in $O(m)$ time, and given graph G and matching M, if there exists an M-augmenting path in G, it returns either (a) a blossom B, or (b) an M-augmenting path.*

And to ensure that if there were an M-augmenting path in G, we will indeed find an M/B-augmenting path in G/B, and can lift this back to an M-augmenting path in the original graph, we need the second theorem.

Theorem 8.3. *Given graph G and matching M, let B be a blossom in G. There exists an M-augmenting path in G if and only if there exists an M/B augmenting path in G/B.*

Moreover, given P', an M/B augmenting path in G/B, we can get back an M-augmenting path P in G in $O(m)$ time.

We will prove these theorems in the following sections. But before that, let us show why this is an $O(mn^2)$ time algorithm.

2.1 Runtime Analysis

Observe that if we get back an M-augmenting path, we can increase the size of the matching; this takes only $O(mn)$ time. Else, if we get back a blossom B, we recurse on a smaller graph. This may happen repeatedly, but after at most $O(n)$ recursions we either return an augmenting path P on some smaller graph or get back "No augmenting path" for an answer. In the former case, we can lift this augmenting path back to an M-augmenting path for G; in the latter case we can use Theorem 8.3 to claim that there was no M-augmenting in the original graph. This takes a total of $O(mn)$ time. And we would need to do this at most $n/2$ times, since each time we augment the size of the matching by 1. The total runtime is $O(mn^2)$.

An aside: this was improved to $O(n^3)$ by Edmonds and Karp, and via a series of improvements, finally to a runtime of $O(m\sqrt{n})$ by Micali and V. Vazirani. The resulting algorithm is quite involved; see a recent paper of Vijay Vazirani giving a cleaner explanation.

2.2 Proof of Theorem 8.3

We first give the proof of Theorem 8.3 as follows.

Proof. Since we can toggle the stem as stated in Section 2, we assume that the stem has length zero, and the only one vertex in the cycle which links to two unmatched edges is open. Since all other nodes in the blossom are matched, any edges going to vertices outside B are non-matching edges.

($\Rightarrow$) Suppose there is an M-augmenting path in G, denoted as P. If P does not go through the blossom B, the path still exists in G/B. If P goes through B, because there is only one open vertex in B, there must exist at least open vertex v' in G which is an end of P but is not in B. Because v_B is open in G/B, the path from v' to v_B is an M/B-augmenting path in G/B.

($\Leftarrow$) Suppose there is an M/B-augmenting path P' in G/B. If P' does not go through v_B, P' still exists in G.

If P' goes through v_B, because v_B is open, it must be an end of P'; denote the other end of P' by s. Suppose the edge (denoted as e) which connects v_B in P' is originally connecting some vertex v in B. If v is the open vertex in B, then s to v is an augmenting path in G. Otherwise, there is a matched edge and an unmatched edge connecting v in B respectively. If e is matched, then go from s to v to the open vertex in B by the unmatched edge is an M-augmenting path in G. If e is unmatched, then go from s to v to the open vertex in B by the matched edge is an M-augmenting path in G.

Since the process to get from P' to the M-augmenting path in G be done algorithmically in $O(m)$ time, this proves the second part of the theorem. □

3 The Algorithm `FindAugPath`

The algorithm `FindAugPath` put all vertices in G in a layered structure to find an M-augmenting path or a blossom, or correctly report that there is no M-augmenting path. The construction is quite similar in spirit to the bipartite case, though with some differences since we may have odd cycles.

At a high level, here's the construction:

> Mark all open vertices as being in layer 0.
> For $i = 0, 1, 2, \ldots$:
> Mark all unmarked nodes with a *non-matching* edge to level $2i$ as being in layer $2i+1$.
> Mark all unmarked nodes with a *matching* edge to level $2i+1$ as being in layer $2i+2$.
>
> If there exists a "cross" edge between two nodes of the same layer, output a blossom or augmenting path, else say "No M-augmenting path".

An illustration of the algorithm is given in Figure 8.2. In order to argue correctness, it is worth going over the steps again in more detail. Let L_i denote the i-th layer vertices in the following discussion.

1. Put all open vertices in L_0 and mark them.
2. Given the even layer L_{2i}, construct L_{2i+1} as follows. For each vertex $u \in L_{2i}$ and edge $(u, v) \in E \backslash M$, do the following:
   (a) *v is unmarked.* Then put v in L_{2i+1}, and mark it.
   (b) v is marked. Let us look at the layer containing v.
      i. $v \in L_{2i}$. This means that there is an unmatched edge linking two vertices in the same layer. This must result in an augmenting path or a blossom! Indeed, the way we construct the layered graph, there are alternating paths P and Q from both u and v to open vertices in L_0. If P and Q do not intersect, then $P \circ (u, v) \circ Q$ gives

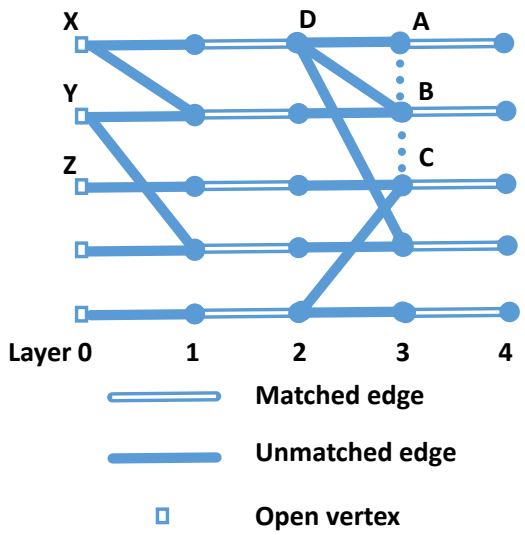

Figure 8.2: An illustration of `FindAugPath` algorihtm

an M-augmenting path. If P and Q intersect, they must first intersect some vertex w at an even layer, and the cycle containing u, v, w gives us the blossom, with the stem being one path from w back to an open vertex in L_0. (Success!)
As examples, in Figure 8.2, if there is an edge connecting A and B, we find a blossom cycle containing three vertices A, B, D, with the stem from X to D. On the other hand, an edge connecting C and B gives the augmenting path $Y \to B \to C \to Z$.

ii. v is in a previous even layer L_{2j}. If so, u should have been explored in the odd layer L_{2j+1}, which is impossible.

iii. v is in an odd layer. This is fine, we just observe that (u, v) is an *even-odd* edge.

3. Given the odd layer L_{2i+1}, construct L_{2i+2} as follows. For each vertex $u \in L_{2i+1}$ and *matching* edge $(u, v) \in M$, do the following:

   (a) v is unmarked. Then add v to L_{2i+2}, and mark v.

   (b) v is marked. Again, consider what layer v belongs to.

      i. $v \in L_{2i+1}$. This means that there is an matching edge linking two vertices in the same layer. This must result in an augmenting path or a blossom. The analysis is similar to Case b(i). (Success!)

      ii. v is in previous layers. This is impossible because all previous explored vertices are either open or matched using other edges. There cannot be a matching edge connecting v to u.

From this construction, we observe that if the algorithm does not succeed, all edges are even-odd edges. Now we can prove Theorem 8.2.

Proof. We now prove that if there is an M-augmenting path in G, the `FindAugPath` algorithm will either return an M-augmenting path or a blossom.

For sake of contradiction, suppose that there exists an M-augmenting path P, and the algorithm does not succeed. From the observation above, this means that when we constructed the layered graph, each edge in G was an even-odd edge. Consider the augmenting path P, the starting open

vertex is in L_0, which is an even layer. Thus the next vertex should be in an odd layer, and the next one should be in an even layer, and so forth. Because there are odd number of edges in P, there are an even number of vertices, and hence the last vertex must be in an odd layer. However, the last vertex is open, and hence in L_0, an even layer. We get a contradiction. □

4 The Tutte-Berge Formula

Finally, we can use the structure of Edmonds' Blossom Algorithm to give an algorithmic proof of the Tutte-Berge Formula. Recall this formula gave us a min-max relationship for the size of the maximum matching in general (even non-bipartite) graphs.

$$\mathrm{MM}(G) = \min_{U \subseteq V} \frac{n + |U| - \mathrm{odd}(G \backslash U)}{2} \tag{8.1}$$

Proof. We have already proved that $\mathrm{MM}(G) \leq \frac{n+|U|-\mathrm{odd}(G\backslash U)}{2}\ \forall U$.

Next we prove that the equality is possible based on the Edmonds Blossom Algorithm. Let a set U be all vertices in the odd levels in the `FindAugPath` with the maximum matching M. Since the algorithm will not success, all edges are even-odd. Thus $|\mathrm{odd(G\backslash U)}| = |\mathrm{V(G)\backslash V(U)}| = |\mathrm{V_{even}}|$. Also, we have some unexplored vertices and edges in the hierarchy. All the unexplored vertices must be matched (otherwise they must be in L_0) with another unexplored vertex (otherwise they should be explored). Hence the number of unexplored matching edges are half of the number of unexplored vertices, i.e., $|M_{\mathrm{unexplored}}| = \frac{1}{2}|V_{\mathrm{unexplored}}(G)|$.

Also, since all even vertices are matched to and only matched to one vertex, and the hierarchy must stop at some even layer, the number of even vertices equals to the number of explored matching edges.

To conclude:

$$\begin{aligned}
\frac{1}{2}\left(n - |U| - \mathrm{odd}(G\backslash U)\right) &= \frac{1}{2}\left(|V_{\mathrm{even}}| + |V_{\mathrm{odd}}| + |V_{\mathrm{unexplored}}| + |V_{\mathrm{odd}}| - |V_{\mathrm{even}}|\right) \\
&= \frac{1}{2}(2|V_{\mathrm{odd}}| + |V_{\mathrm{unexplored}}|) \\
&= |V_{\mathrm{odd}}| + \frac{1}{2}|V_{\mathrm{unexplored}}| \\
&= |M_{\mathrm{unexplored}}| + |M_{\mathrm{explored}}| \\
&= |M|
\end{aligned}$$

□