Outline

In this lecture, we learn the so-called augmenting path algorithm for computing a maximum matching in a bipartite graph. The algorithm is based on the notion of augmenting paths, and we will also cover how to find an augmenting path using the alternating tree procedure.

1 Augmenting path algorithm

Previously, we learned the greedy algorithm that finds a maximal matching in a bipartite graph whose size is always at least half of the maximum size of a matching. In this section, we will introduce an algorithm that is guaranteed to compute a maximum matching of a bipartite graph. The central idea of the algorithm lies in the concept of **augmenting paths**, so the algorithm is referred to as the **augmenting path algorithm**. Let us elaborate on them as follows.

Let $G = (V, E)$ be a bipartite graph, and let M be a matching of G. We say that a vertex $v \in V$ is **M-exposed** if v is not connected to an edge in M. We say that a sequence of edges $e_1, \dots, e_k$ is **M-alternating** if for every two consecutive edges e_i and e_{i+1}, either $e_i \in M, e_{i+1} \notin M$ or $e_i \notin M, e_{i+1} \in M$ holds. An **M-augmenting path** is an M-alternating path if the first and last vertices are M-exposed. Figure 2.1 shows an M-alternating path and an M-augmenting path where the edges in M are colored red. By definition, the first and last edges of an M-augmenting path are

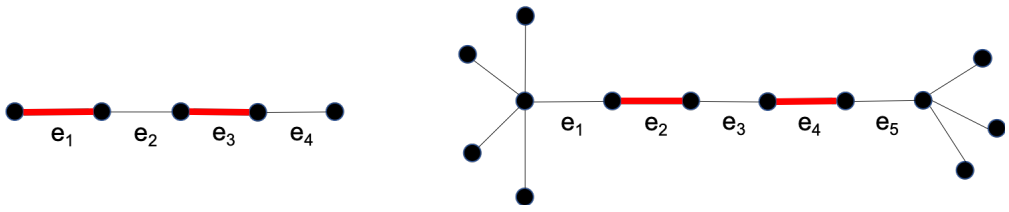

Figure 2.1: an M-alternating path and an M-augmenting path

not contained in M. Moreover, as an M-augmenting path is M-alternating, it has an odd number of edges.

The key idea is that if there is an M-augmenting path, we can improve the matching. In Figure 2.2, we have a bipartite graph and an M-augmenting path $u_1, v_1, \dots, u_4, v_4$. On the path, we switch the

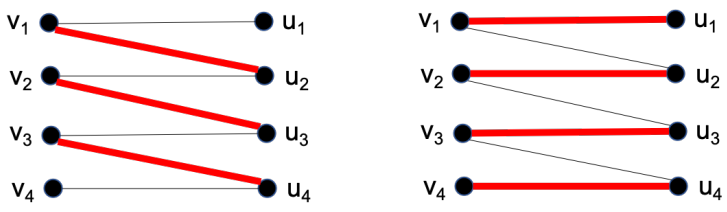

Figure 2.2: improving the matching by an augmenting path

role of the matching edges and that of the edges not in the matching. In other words, we remove every edge $e \in M$ from M and add every edge $e \notin M$ to M. By doing so, we obtain a larger matching as shown in the second graph of Figure 2.2. To formalize it, we take an M-augmenting path P. We define the **symmetric difference** of M and P, denoted $M \oplus P$, as

$$M \oplus P = (M \setminus P) \cup (P \setminus M).$$

Hence, an edge $e \in E$ belongs to $M \oplus P$ if and only if e is contained in precisely one of M and P. Taking the symmetric difference of the matching M and an M-augmenting path P, a change is made only on the edges of P. To be more precise, if P is given by a sequence of edges $e_1, \ldots, e_{2\ell-1}$ for some $\ell \geq 1$ with $e_1, e_3 \ldots, e_{2\ell-1} \notin M$ and $e_2, e_4, \ldots, e_{2\ell-2} \in M$, we get $e_1, e_3 \ldots, e_{2\ell-1} \in M$ and $e_2, e_4, \ldots, e_{2\ell-2} \notin M$ after taking the synmetric difference. Then P becomes an alternating path with one more matching edge. We refer to this procedure of taking the symmetric difference of M and P as **augmenting** the edges of an M-augmenting path P. In Figure 2.3, the graph is obtained

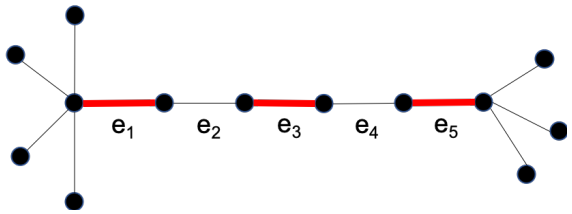

Figure 2.3: an alternating path by an M-augmenting path

from augmenting the edges of the M-augmenting path given in Figure 2.1.

Lemma 2.1. *Let $G = (V, E)$ be a bipartite graph. Let M be a matching, and let P be an M-augmenting path. Then $M \oplus P$ is a matching of G with $|M \oplus P| = |M| + 1$.*

Proof. Let us first argue that $M \oplus P$ is a matching. Let e_1, e_2 be two distinct edges in $M \oplus P = (M \setminus P) \cup (P \setminus M)$If $e_1, e_2 \in M \setminus P \subseteq M$, then as M is a matching, e_1 and e_2 do not intersect. If $e_1, e_2 \in P \setminus M$, then as P is an M-alternating path, e_1 and e_2 are not consecutive on p and thus do not intersect. Then we may focus on the case where one edge is part of $M \setminus P$ and the other is in $P \setminus M$. Note that the edges in $M \setminus P$ do not touch the internal vertices of P as they are covered by the edges in $M \cap P$. Furthermore, they do not touch the two end vertices of P because they are M-exposed by definition. Therefore, any two distinct edges in $M \oplus P$ do not intersect, so it is a matching.

For the second part, note that

$$|M \oplus P| = |M \setminus P| + |P| - |M \cap P|.$$

If P has length $2\ell - 1$ for some $\ell \geq 1$, then P has $\ell - 1$ edges in M, so $|M \cap P| = \ell - 1$ and $|P| - |M \cap P| = \ell$. Therefore, it follows that

$$|M \oplus P| = |M \setminus P| + |M \cap P| + 1 = |M| + 1,$$

as required. □

Lemma 2.1 leads to a natural algorithm that iteratively improves the given matching for a bipartite graph by finding an augmenting path. To be more precise, we consider the following algorithm. Algorithm 1, called the augmenting path algorithm, is proven to find a maximum matching of a bipartite graph.

Algorithm 1 Augmenting path algorithm for maximum bipartite matching

Initialize $M = \emptyset$.
while there is an M-augmenting path **do**
  Find an M-augmenting path P
  Update M as $M = M \oplus P$
end while
Return M

Theorem 2.2. *Let $G = (V, E)$ be a bipartite graph, and let M be a matching. Then M is a maximum matching if and only if there is no M-augmenting path in G.*

Proof. If G has an M-augmenting path, then by Lemma 2.1, $M \oplus P$ is a matching that contains one more edge than M, which means that M is not a maximum matching.

For the converse direction, let us assume that M is not a maximum matching. Let M^* be a maximum matching of G. Then we consider the symmetric diffence of M and M^* given by

$$M \oplus M^* = (M \setminus M^*) \cup (M^* \setminus M).$$

Then consider the subgraph H of G on the vertex set V whose edge set is $M \oplus M^*$. Since M and M^* are matchings of G, a vertex is icident to at most one edge of M and at most one edge of M^*, so every vertex of H has degree at most 2. This in turn implies that H is a union of vertex-disjoint paths and cycles. Moreover, the paths and cycles have edges that alternate between M and M^*. In other words, for any two consecutive edges e_1, e_2 on one of the paths and cycles, it holds that $e_1 \in M, e_2 \in M^*$ or $e_1 \in M^*, e_2 \in M$. As a result, the number of edges in M and the number of edges in M^* equal in each cycle of H. At the same time, since M^* contains more edges than M, there is a path P that has more edges in M^* than M. This means that P is an $(M \setminus M^*)$-augmenting path in H. Consider the two end points of P. They are covered by M^*. If they were not M-exposed, P could be extended with an edge in $M \setminus M^*$, contradicting our choice of P. This implies that P is an M-augmenting path in G. □

Theorem 2.2 establishes the correctness of Algorithm 1 that it computes a maximum matching of a bipartite graph. In addition to its correctness, we also care about its **computational complexity**, which essentially measures the amount of computational costs to terminate. For Algorithm 1, we will analyze its **time complexity** or **iteration complexity**. Recall that each augmenting path increases the size of a matching by 1 while the maximum size of a matching is at most $|V|/2$. Therefore, the number times the while loop is incurred is at most $|V|/2$. Then, what remains is to analyze the computational complexity of finding an M-augmenting path. In the next section, we will show that an M-augmenting path can be found in $O(|E|)$[1] time.

2 Alternating tree procedure

Before explaining the algorithm to find an M-augmenting path, let us establish the notion of **trees** and **forests**. First, we say that two distinct vertices u and v are connected if there is an uv-path and that a graph is connected if any of its two distinct vertices are connected. A connected graph is a tree if any two distinct vertices are connected by exactly one path. The terminology comes from the fact that a tree can be depicted in a hierarchical fashion. In Figure 2.4, we have a tree

[1]This is the big-O notation: $O(|E|)$ means that for any graph with $|E|$ edges, the algorithm terminates in $C \times |E|$ iterations where C is a constant that does not depend on the input graph.

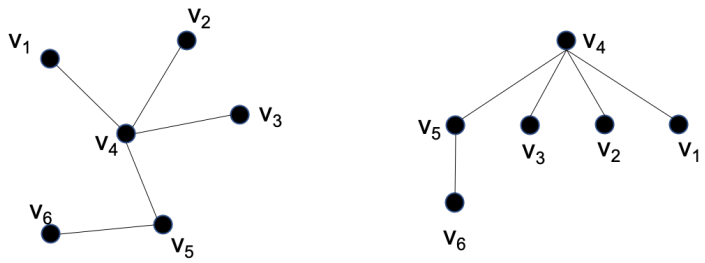

Figure 2.4: a tree and its hierarchical representation

and its hierarchical representation. The way it works is that we take any vertex v as a **root** and expand the tree with its neighbors. We say that a vertex of degree 1 in a tree is a **leaf** vertex. Then we take all leaf vertices and add their neighbors to the tree on the next level. The **connected components** of a graph are its maximal connected subgraphs. A forest is a graph all of whose connected components are trees.

Exercise 2.3. A graph is a forest if and only if it has no cycle as a subgraph.

We consider the procedure given by Algorithm 2. As illustrated in Figure 2.5, the algorithm

Algorithm 2 Alternating tree algorithm to find an M-augmenting path

Input: a bipartite graph $G = (V, E)$ and a matching M
while there is an M-exposed vertex in G **do**
  Take an M-exposed vertex r and set it as the root.
  Initialize $T = \{r\}$ and $L = \{r\}$
  while $L \neq \emptyset$ **do**
    Take a vertex $u \in L$
    if u has a neighbor that is M-exposed **then**
      Return the path from the root to the M-exposed neighbor of u on T.
    else
      for every neighbor v **do**
        Take the vertex w such that $vw \in M$
        Update $T = T \cup \{v, w\}$ and $L = (L \setminus \{u\}) \cup \{w\}$
      end for
    end if
  end while
  Delete all vertices in T from G
end while

builds a tree structure starting from an M-exposed vertex as its root. We call such a tree an **M-alternating tree**. Let us go through the algorithm step by step. First at level 0, where the root vertex r is located, we consider the neighbors of r. If r has an M-exposed neighbor w, then rw is an M-augmenting path. Otherwise, all neighbors of r are covered by M, in which case for every neighbor v of r, we take $vw \in M$ and add it to the tree. After this update, v is on level 1 while w is on level 2. Moreover, w is a leaf vertex. As done for the root vertex, we repeat the procedure for every leaf vertex of the tree. Once the tree expansion is completed and no augmenting path is

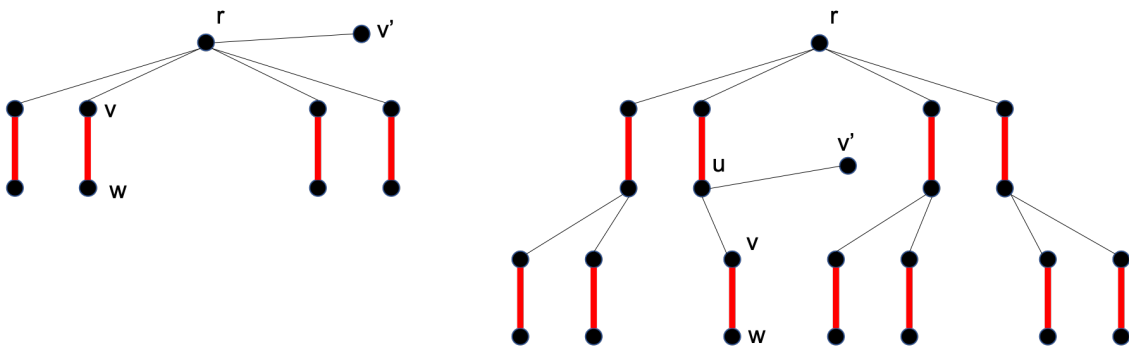

Figure 2.5: an M-alternating tree

found, we delete the vertices in T by taking the subgraph induced by the vertices not in T. Then we start a new search with another M-exposed vertex. If no augmenting path is found until we delete all vertices in G, then we conclude that there is no M-augmenting path in G and that M is a maximum matching.

Let us prove that Algorithm 2 correctly decides whether G contains an M-augmenting path as a subgraph and that if one exists, Algorithm 2 finds an M-augmenting path.

Theorem 2.4. *Let $G = (V, E)$ be a bipartite graph, and let M be a matching. If Algorithm 2 does not return an M-augmenting path, then G contains no M-augmenting path as a subgraph.*

Proof. Let r_1 denote the first M-exposed vertex chosen as a root vertex, and let T_1 be the tree expanded from r_1 by Algorithm 2. We take W_1 as the set of vertices on an even level, including level 0 and take V_1 as the set of vertices on an odd level. Note that the vertices in $W_1 \setminus \{r_1\}$ and the vertices in V_1 are in one-to-one correspondence by the matching edges in T_1. As a result, we

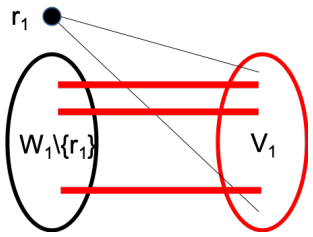

Figure 2.6: the first M-alternating tree

have $|W_1| = |V_1| + 1$. Repeating this procedure, Algorithm 2 gives rise to a partition of the vertex set V with k M-alternating trees for some $k \geq 1$, as illustrated in Figure 2.7. For $i \in \{1, \ldots, k\}$, let the root vertex of the ith alternating tree T_i be r_i, and let W_i and V_i denote the set of vertices on an even level and the set of vertices on an odd level, repsectively. Then we have

$$|W_i| = |V_i| + 1$$

for all $i \in \{1, \ldots, k\}$.

Next, let us observe that no vertex in W_1 is adjacent to a vertex in

$$V \setminus (V_1 \cup W_1) = (V_2 \cup W_2) \cup \cdots \cup (V_k \cup W_k).$$

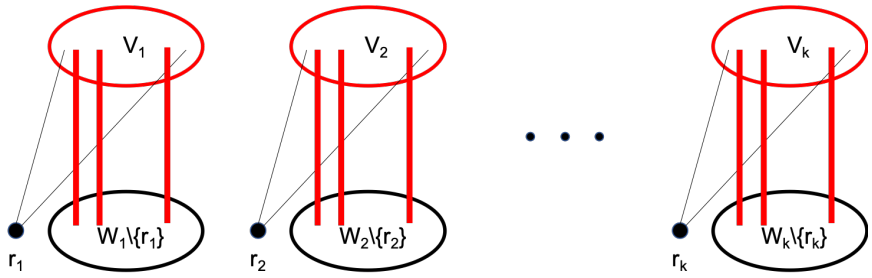

Figure 2.7: a partition of V with M-alternating trees

This holds because if a vertex $u \in W_1$ were adjacent to some vertex $v \in V \setminus (V_1 \cup W_1)$, Algorithm 2 would add it as a child of v to T_1. Similarly, no vertex in W_2 is adjacent to

$$V \setminus ((V_1 \cup W_1) \cup (V_2 \cup W_2)) = (V_3 \cup W_3) \cup \cdots \cup (V_k \cup W_k).$$

Repeating this argument, we observe that no two distinct vertices in $W_1 \cup \cdots \cup W_k$ are adjacent, as depicted in Figure 2.8.

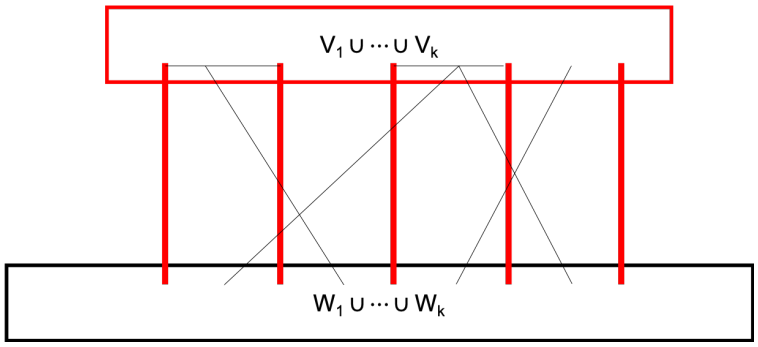

Figure 2.8: another illustration of the partition

Now we are ready to argue that M is a maximum matching. Let M' be another matching. Then every edge touches a vertex in $V_1 \cup \cdots \cup V_k$. At the same time, a vertex in $V_1 \cup \cdots \cup V_k$ is incident to at most one edge in the matching M'. This implies that

$$|M'| \leq |V_1 \cup \cdots \cup V_k| = \sum_{i=1}^{k} |V_i|$$

for any matching M'. As M already has $\sum_{i=1}^{k} |V_i|$ edges, M is a matching matching. Then, by Theorem 2.2, G has no M-augmenting path, as required. □

Now that we know the correctness of Algorithm 2, what remains is to analyze its computational complexity. Note that an edge is enumerated when one of its endpoints is part of an alternating tree. Hence, an edge is considered at most twice while running Algorithm 2. Therefore, the number of iterations required is $O(|E|)$.

Recall that the number of while loops incurred for Algorithm 1 is $O(|V|)$. As a result, the computational complexity of Algorithm 1 is $O(|V||E|)$.