

# Chapter 30

## Matchings

By Sarel Har-Peled, November 1, 2015<sup>①</sup>

Version: 0.2

### 30.1. Definitions and basic properties

#### 30.1.1. Definitions

Definition 30.1.1. For a graph  $G = (V, E)$  a set  $M \subseteq E$  of edges is a *matching* if no pair of edges of  $M$  has a common vertex.

Definition 30.1.2. A matching is *perfect* if it covers all the vertices of  $G$ . For a weight function  $w$ , which assigns real weight to the edges of  $G$ , a matching  $M$  is a *maximum weight matching*, if  $M$  is a matching and  $w(M) = \sum_{e \in M} w(e)$  is maximum.

Definition 30.1.3. A matching  $M$  is a *maximal*, if  $M$  is a matching and it can not be made bigger by adding any edge.

Thus, a maximal matching is locally optimal, while a maximum matching is the global largest/heaviest possible matching.

Definition 30.1.4. If there is no weight on the edges, we consider the weight of every edge to be one, and in this case, we are trying to compute a *maximum size matching*.

Problem 30.1.5. Given a graph  $G$  and a weight function on the edges, compute the maximum weight matching in  $G$ .

Remark 30.1.6. There is a simple way to compute a maximum size matching in a bipartite graph using network flow. Here we present an alternative algorithm that does not use network flow.

#### 30.1.2. Matchings and alternating paths

Consider a matching  $M$ . An edge  $e \in M$  is a *matching edge*. Naturally, Any edge  $e' \in E(G) \setminus M$  is *free*. In particular, a vertex  $v \in V(G)$  is *matched* if it is adjacent to an edge in  $M$ . Naturally, a vertex  $v'$  which is not matched is *free*.

An *alternating path* is a simple path that its edges are alternately matched and free. An *alternating cycle* is defined similarly. The *length* of a path/cycle is the number of edges in it.

Definition 30.1.7. A path  $\pi = v_1v_2, \dots, v_{2k+2}$  is an *augmenting* path for a matching  $M$  in a graph  $G$ :

---

<sup>①</sup>This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

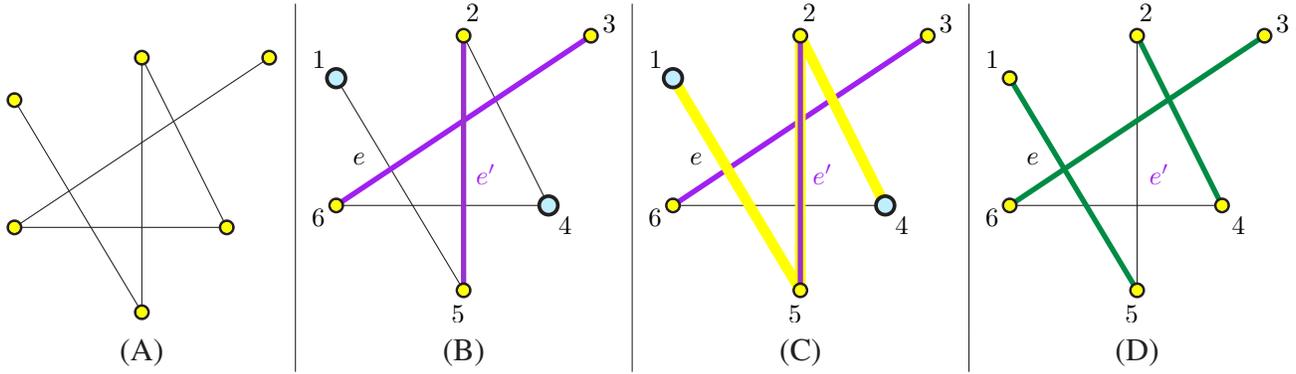


Figure 30.1: (A) The input graph. (B) A maximal matching in  $G$ . The edge  $e$  is free, and vertices 1 and 4 are free. (C) An alternating path. (D) The resulting matching from applying the augmenting path.

- (i)  $\pi$  is simple,
- (ii) for all  $i$ ,  $e_i = v_i v_{i+1} \in E(G)$ ,
- (iii)  $v_1$  and  $v_{2k+1}$  are free vertices for  $M$ ,
- (iv)  $e_1, e_3, \dots, e_{2k+1} \notin M$ , and
- (v)  $e_2, e_4, \dots, e_{2k} \in M$ .

**Lemma 30.1.8.** *If  $M$  is a matching and  $\pi$  is an augmenting path relative to  $M$ , then*

$$M' = M \oplus \pi = \{e \in E \mid e \in (M \setminus \pi) \cup (\pi \setminus M)\}$$

*is a matching of size  $|M| + 1$ .*

*Proof:* Think about removing  $\pi$  from the graph all together. What is left of  $M$ , is a matching of size  $|M| - |M \cap \pi|$ . Now, add back  $\pi$  and alternate the edges of the matching  $M$  with the free edges of  $\pi$ . Clearly, the new set of edges is a matching, since  $\pi$  is disjoint from the rest of the matching, this alternation results in a valid matching, and its size is  $|M'| = |M| - |M \cap \pi| + |\pi \setminus M| = |M| + 1$ . ■

**Lemma 30.1.9.** *Let  $M$  be a matching, and  $T$  be a maximum matching, and  $k = |T| - |M|$ . Then  $M$  has  $k$  vertex disjoint augmenting paths. At least one of length  $\leq u/k - 1$ , where  $u = 2(|T| + |M|)$ .*

*Proof:* Let  $E' = M \oplus T$ , and let  $H = (V, E')$ . Clearly, every vertex in  $H$  has at most degree 2 because every vertex is adjacent to at most one edge of  $M$  and one edge of  $T$ . Thus,  $H$  is a collection of paths and (even length) cycles. The cycles are of even length since the edges of the cycle are alternating between two matchings (i.e., you can think about the cycle edges as being 2-colorable).

Now, there are  $k$  more edges of  $T$  in  $M \oplus T$  than of  $M$ . Every cycle have the same number of edges of  $M$  and  $T$ . Thus, a path in  $H$  can have at most one more edge of  $T$  than of  $M$ . In such a case, this path is an augmenting path for  $M$ . It follows that there are at least  $k$  augmenting paths for  $M$  in  $H$ .

As for the claim on the length of the shortest augmenting path. Let  $u = |V(H)| \leq 2|T| + |M|$ . Observe that if all these (vertex disjoint) augmenting paths were of length  $\geq u/k$  then the total number of vertices in  $H$  would be at least  $(u/k + 1)u > u$ , since a path of length  $\ell$  has  $\ell + 1$  vertices. A contradiction. ■

The lemma readily implies:

**Corollary 30.1.10.** *A matching  $M$  is maximum  $\iff$  there is no augmenting path for  $M$ .*

## 30.2. Unweighted matching in bipartite graph

### 30.2.1. The slow algorithm; **algSlowMatch**

**The algorithm.** Let  $G = (L \cup R, E)$  be a bipartite graph. Let  $M_0 = \emptyset$  be an empty matching. In the  $i$ th iteration of **algSlowMatch**, let  $L_i$  and  $R_i$  be the free vertices in  $L$  and  $R$ , relative to the matching  $M_{i-1}$ . If there is an edge in  $G$  between a vertex of  $L_i$  and  $R_i$ , we just add this edge to the matching, and go on to the next iteration.

Otherwise, we build a new graph  $H_i$ . We orient all the edges of  $E \setminus M_{i-1}$  from left to the right. Formally, an edge  $lr \in E \setminus M_{i-1}$ , with  $l \in L$  and  $r \in R$ , is going to induce the directed edge  $(l, r)$  in  $H_i$ . Similarly, the matching edges  $lr \in M_{i-1}$  are oriented from the right to left, as the new directed edge  $(r, l)$ .

Now, using **BFS**, compute the *shortest path*  $\pi_i$  from a vertex of  $L_i$  to a vertex of  $R_i$ . If there is no such path, the algorithm stops and reports that the current matching is a maximum matching. Otherwise, the algorithm updates  $M_i = M_{i-1} \oplus \pi_i$ , and continues to the next iteration.

**Analysis.** An augmenting path has an odd number of edges. As such, if it starts in a free vertex on the left side, then it must end in a free vertex on the right side. As such, such an augmenting path, corresponds to a path between a vertex of  $L_i$  to a vertex of  $R_i$  in  $H_i$ . By **Corollary 30.1.10**, as long as the algorithm has not computed the maximum matching, there is an augmenting path, and this path increases the size of the matching by one.

Observe, that any shortest path found in  $H_i$  between  $L_i$  and  $R_i$  is an augmenting path. Namely, if there is an augmenting path for  $M_{i-1}$ , then there is a path from a vertex of  $L_i$  to a vertex of  $R_i$  in  $H_i$ , and the algorithm computes the shortest such path.

We conclude, that after at most  $n$  iterations, the algorithm would be done. Clearly, the algorithm can be easily implemented in linear time. We thus have the following result:

**Lemma 30.2.1.** *Given a bipartite undirected graph  $G = (L \cup R, E)$ , with  $n$  vertices and  $m$  edges, one can compute the maximum matching in  $G$  in  $O(nm)$  time.*

### 30.2.2. The Hopcroft-Karp algorithm

We next improve the running time – this requires quite a bit of work, but hopefully exposes some interesting properties of matching in bipartite graphs.

#### 30.2.2.1. Some more structural observations

We need three basic observations:

- (A) If we augment along a shortest path, then the next augmenting path must be longer (or at least not shorter). See **Lemma 30.2.2** below.
- (B) As such, if we always augment along shortest paths, then the augmenting paths get longer as the algorithm progresses, see **Corollary 30.2.3** below.
- (C) Furthermore, all the augmenting paths of the same length used by the algorithm are vertex-disjoint (!). See **Lemma 30.2.4** below. (The main idea of the faster algorithm is to compute this block of vertex-disjoint paths of the same length in one go, thus getting the improved running time.)

**Lemma 30.2.2.** *Let  $M$  be a matching, and  $\pi$  be the shortest augmenting path for  $M$ , and let  $\pi'$  be any augmenting path for  $M' = M \oplus \pi$ . Then  $|\pi'| \geq |\pi|$ . Specifically, we have  $|\pi'| \geq |\pi| + 2|\pi \cap \pi'|$ .*

*Proof:* Consider the matching  $N = M \oplus \pi \oplus \pi'$ . Observe that  $|N| = |M| + 2$ . As such, ignoring cycles and balanced paths,  $M \oplus N$  contains two augmenting paths, say  $\sigma_1$  and  $\sigma_2$  (relative to  $M$ ). Observe that  $M \oplus N = \pi \oplus \pi'$ , and as such

$$|\pi \oplus \pi'| = |M \oplus N| \geq |\sigma_1| + |\sigma_2|.$$

Since  $\pi$  was the shortest augmenting path for  $M$ , it follows that  $|\sigma_1| \geq |\pi|$  and  $|\sigma_2| \geq |\pi|$ . We such,

$$|\pi \oplus \pi'| \geq |\sigma_1| + |\sigma_2| \geq |\pi| + |\pi| = 2|\pi|.$$

By definition, we have that  $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$ . Combining with the above, we have

$$|\pi| + |\pi'| - 2|\pi \cap \pi'| \geq 2|\pi| \quad \implies \quad |\pi'| \geq |\pi| + 2|\pi \cap \pi'|. \quad \blacksquare$$

The above lemma immediately implies the following.

**Corollary 30.2.3.** *For the sequence of augmenting paths used by the algorithm of Section 30.2.1 (which we remind the reader always augment the matching along the shortest augmenting path), we have that  $|\pi_1| \leq |\pi_2| \leq \dots \leq |\pi_t|$ , where  $t$  is the number of augmenting paths computed by the algorithm, and  $\pi_1, \pi_2, \dots, \pi_t$  are the sequence of augmenting paths used by the algorithm.*

**Lemma 30.2.4.** *For all  $i$  and  $j$ , such that  $|\pi_i| = \dots = |\pi_j|$ , we have that the paths  $\pi_i$  and  $\pi_j$  are vertex disjoint.*

*Proof:* Assume for contradiction, that that  $|\pi_i| = |\pi_j|$ ,  $i < j$ , and  $\pi_i$  and  $\pi_j$  are not vertex disjoint, and assume that  $j - i$  is minimal. As such, for any  $k$ , such that  $i < k < j$ , we have that  $\pi_k$  is disjoint from  $\pi_i$  and  $\pi_j$ .

Now, let  $M_i$  be the matching after  $\pi_i$  was applied. We have that  $\pi_j$  is not using any of the edges of  $\pi_{i+1}, \dots, \pi_{j-1}$ . As such,  $\pi_j$  is an augmenting path for  $M_i$ . Now,  $\pi_j$  and  $\pi_i$  share vertices. It definitely can not be the two endpoints of  $\pi_j$  (since they are free) - so it must be some interval vertex of  $\pi_j$ . But then,  $\pi_i$  and  $\pi_j$  must share an edge - indeed, assume the shared vertex is  $v$  -  $\pi_j$  uses a matching edge of  $M_i$  adjacent to  $v$ , but this must belong to  $\pi_j$  - since it contains the only matching edge adjacent to  $v$  in  $M_i$ . Namely,  $|\pi_i \cap \pi_j| \geq 1$ . Now, by Lemma 30.2.2, we conclude that  $|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|$ . A contradiction.  $\blacksquare$

### 30.2.2.2. Improved algorithm

The idea is going to extract all possible augmenting shortest paths of a certain length in one iteration. Indeed, assume for the time being, that given a matching we can exact all augmenting paths of length  $k$  for  $M$  in  $\mathbb{G}$  in  $O(m)$  time, for  $k = 1, 3, 5, \dots$ . Specifically, we apply this extraction algorithm, till  $k = 1 + 2 \lceil \sqrt{n} \rceil$ . This would take  $O(km) = O(\sqrt{nm})$  time.

The key observation is that the matching  $M_k$ , at the end of this process, is of size  $|T| - \Omega(\sqrt{n})$ , see Lemma 30.2.5 below, where  $T$  is the maximum matching. As such, we resume the regular algorithm that augments one augmenting path at a time. After  $O(\sqrt{n})$  regular iterations we would be done.

#### Analysis.

**Lemma 30.2.5.** *Consider the iterative algorithm that applies shortest path augmenting path to the current matching, and let  $M$  be the first matching such that the shortest path augmenting path for it is of length  $\geq \sqrt{n}$ , where  $n$  is the number of vertices in the input graph  $\mathbb{G}$ . Let  $T$  be the maximum matching. Then  $|T| \leq |M| + O(\sqrt{n})$ .*

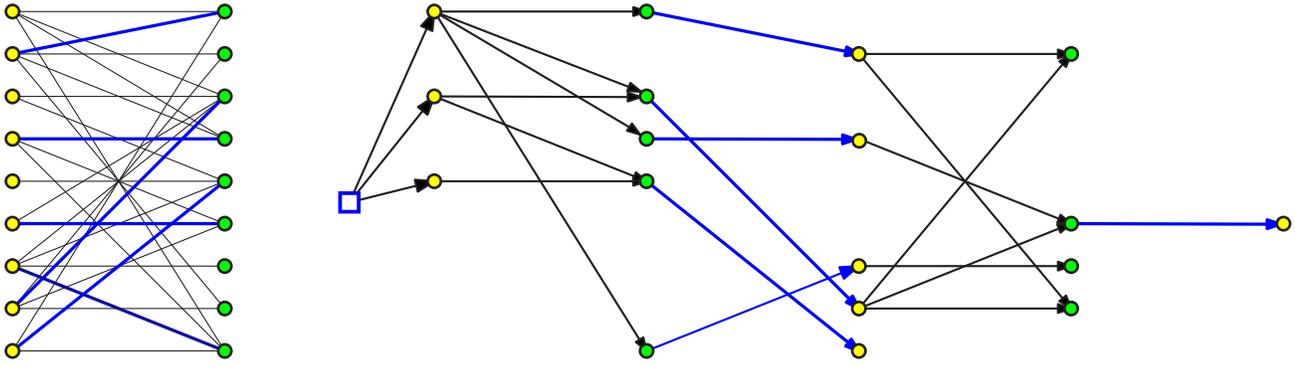


Figure 30.2: (A) A bipartite graph and its matching. (B) Its layered graph.

*Proof:* At this point, the shortest augmenting path for the current matching  $M$  is of length at  $\geq \sqrt{n}$ . By Lemma 30.1.9, this implies that if  $T$  is the maximum matching, then we have that there is an augmenting path of length  $\leq 2n/(|T| - |M|) + 1$ . Combining these two inequalities, we have that

$$\sqrt{n} \leq \frac{2n}{|T| - |M|} + 1,$$

which implies that  $|T| - |M| \leq 3\sqrt{n}$ , for  $n \geq 4$ . ■

### 30.2.2.3. Extracting many augmenting paths: **algExtManyPaths**

The basic idea is to build a data-structure that is similar to a **BFS** tree, but enable us to extract many augmenting path. So, assume we are given a graph  $G$ , as above, a matching  $M$ , and a parameter  $k$ , where  $k$  is an odd integer. Furthermore, assume that the shortest augmenting path for  $M$  in relation to  $G$  is of length  $k$ . Our purpose is to extract as many augmenting paths as possible that are vertex disjoint that are of length  $k$  ( $k = 1$  is exactly the greedy algorithm for maximal matching!).

To this end, let  $F$  be the set of free vertices in  $G$ . We build a directed graph, having a source vertex  $s$ , and that is connected to all the vertices of  $L_1 = L \cap F$  (all the free vertices in  $L$ ). Now, we direct the edges of  $G$ , as done above, and let  $H$  be the resulting graph (i.e., non-matching edges are directed from left to right, and matching edges are directed from right to left). Now, compute **BFS** on the graph  $H$  starting at  $s$ , and let  $\mathcal{T}$  be the resulting tree.

Let  $L_1, R_1, L_2, R_2, L_3, R_3, \dots$  be the layers of the **BFS**. By assumption, the first free vertex below  $L_1$  encountered in the tree is of level  $R_\tau$ , where  $\tau = \lceil k/2 \rceil$  (note, that no free vertex can be encountered on  $L_i$ , for  $i > 1$ , since all the free vertices of  $L$  are in  $L_1$ ).

Scan the edges of  $H$ . A back edge connects a vertex to a vertex that is in a higher level of the tree – we ignore such edges. The other possibilities, is an edge that is a forward edge – an edge between two vertices that belong to two consecutive levels of the **BFS** tree  $\mathcal{T}$ . Let  $J$  be the resulting graph of removing all backward and cross edges from  $H$  (a cross edge connects two vertices in the same layer of the **BFS**). All the remaining edges are either **BFS** edges or forward edges, and we direct them according to the **BFS** layers from the shallower layer to the deeper layer. The resulting graph is a DAG (which is an enrichment of the original tree  $\mathcal{T}$ ). Compute also the reverse graph  $J^{\text{rev}}$  (where, we just reverse the edges).

Now, let  $F_\tau = R_\tau \cap F$  be the free vertices of distance  $k$  from the free vertices of  $L_1$  (which are all free vertices). For every vertex  $v \in F_\tau$  do a **DFS** in  $J^{\text{rev}}$  till the **DFS** reaches a vertex of  $L_1$ . Mark all the vertices visited by the **DFS** as “used” – thus not allowing any future **DFS** to use these vertices (i.e., the **DFS** ignore

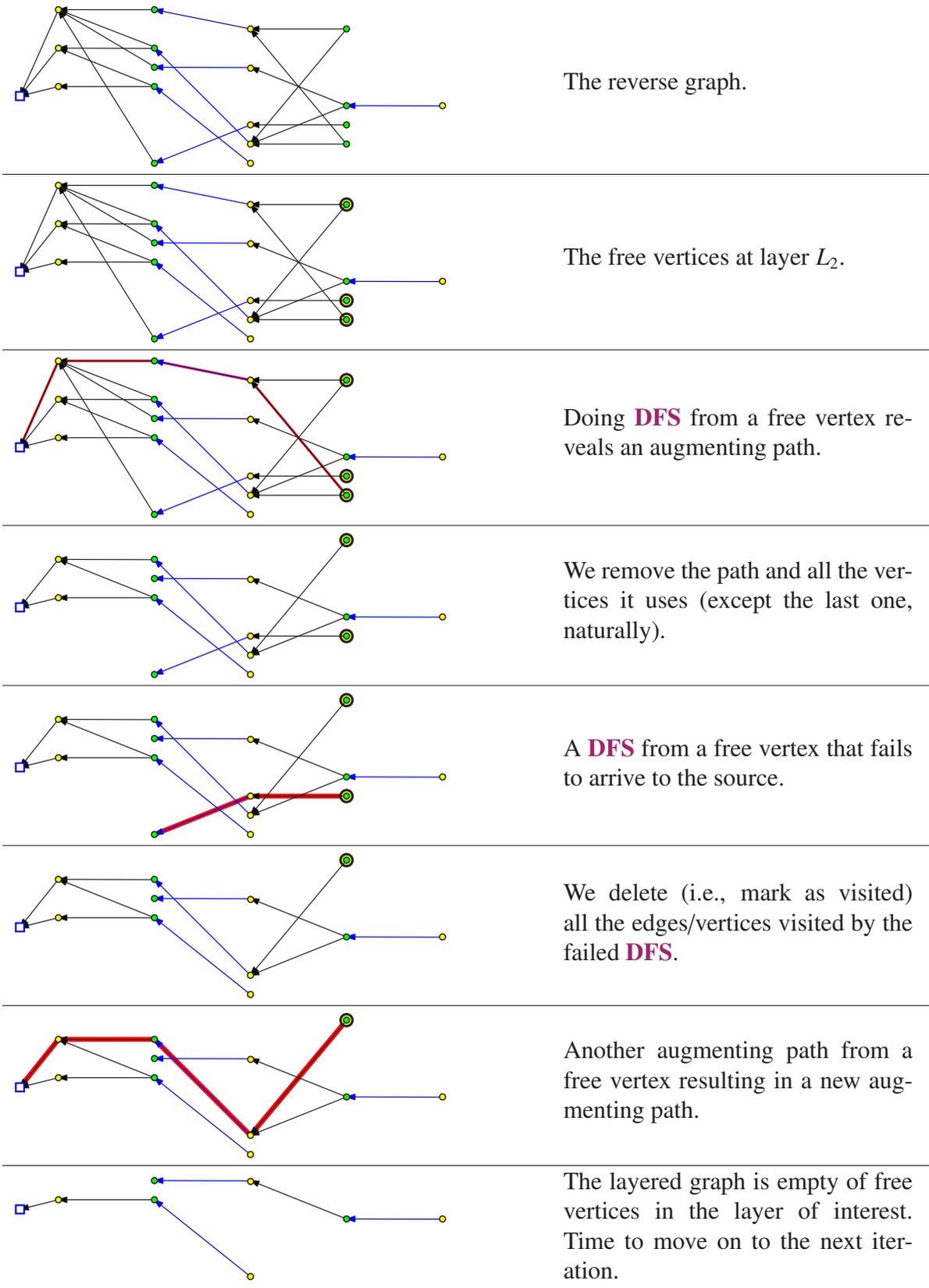


Figure 30.3: Extracting augmenting paths from the reverse layered graph.

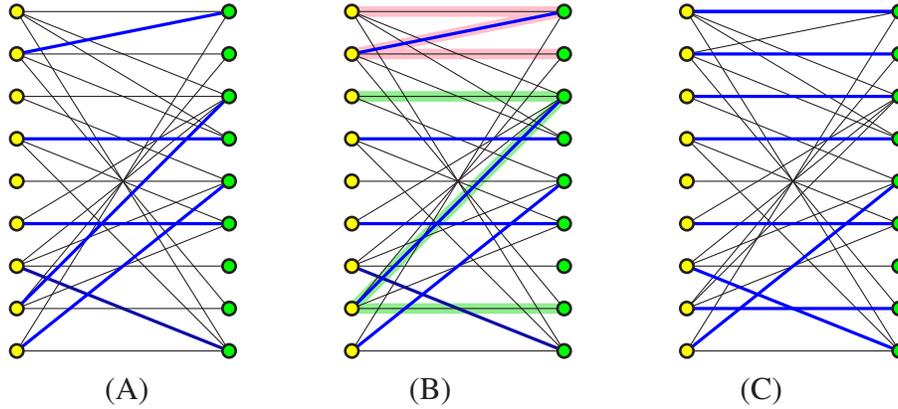


Figure 30.4: (A) A bipartite graph and its current matching. (B) Augmenting paths computed using the layered graph (see Figure ??). (C) The new matching after we apply the augmenting paths.

edges leading to used vertices). If the **DFS** succeeds, we extract the shortest path found, and add it to the collection of augmenting paths. Otherwise, we move on to the next vertex in  $F_\tau$ , till we visit all such vertices.

This algorithm results in a collection of augmenting paths  $P_\tau$ , which are vertex disjoint. We claim that  $P_\tau$  is the desired set maximal cardinality disjoint set of augmenting paths of length  $k$ .

**Analysis.** Building the initial graphs  $J$  and  $J^{\text{rev}}$  takes  $O(m)$  time. We charge the running time of the second stage to the edges and vertices visited. Since any vertex visited by any **DFS** is never going to be visited again, this imply that an edge of  $J^{\text{rev}}$  is going to be considered only once by the algorithm. As such, the running time of the algorithm is  $O(n + m)$  as desired.

**Lemma 30.2.6.** *The set  $P_k$  is a maximal set of vertex-disjoint augmenting paths of length  $k$  for  $M$ .*

*Proof:* Let  $M'$  be the result of augmenting  $M$  with the paths of  $P_k$ . And, assume for the sake of contradiction, that  $P_k$  is not maximal. Namely, there is an augmenting path  $\sigma$  of length  $k$  that is disjoint from the vertices of the paths of  $P_k$ . But then, we could traverse  $\sigma$  in  $J$ , and this would go through unused vertices. Indeed, if any of the vertices of  $\sigma$  were used by any of the **DFS**, then it would have resulted in a path that goes to a free vertex in  $L_1$ . But that is a contradiction, as  $\sigma$  is supposedly disjoint from the paths of  $P_k$ . ■

#### 30.2.2.4. The result

**Theorem 30.2.7.** *Given a bipartite unweighted graph  $G$  with  $n$  vertices and  $m$  edges, one can compute maximum matching in  $G$  in  $O(\sqrt{nm})$  time.*

*Proof:* The **algMatching<sub>HK</sub>** algorithm is described in Section 30.2.2.2, and the running time analysis is done above.

The main challenge is the correctness. The idea is to interpret the execution of this algorithm as simulating the slower the simpler algorithm of Section 30.2.1. Indeed, the **algMatching<sub>HK</sub>** algorithm computes a sequence of sets of augmenting paths  $P_1, P_3, P_5, \dots$ . We order these augmenting paths in an arbitrary order inside each such set. This results in a sequence of augmenting paths that are shortest augmenting paths for the current matching, and furthermore by Lemma 30.2.6 each set  $P_k$  contains a maximal set of such vertex-disjoint augmenting paths of length  $k$ . By Lemma 30.2.4, all augmenting paths of length  $k$  computed are vertex disjoint.

As such, now by induction, we can argue that if **algMatching<sub>HK</sub>** simulates correctly **algSlowMatch**, for the augmenting paths in  $P_1 \cup P_3 \cup \dots P_i$ , then it simulates it correctly for  $P_1 \cup P_3 \cup \dots P_i \cup P_{i+1}$ , and we are done. ■

### 30.3. Bibliographical notes

The description here follows the original paper of Hopcroft and Karp [HK73].

#### Bibliography

[HK73] J. E. Hopcroft and R. M. Karp. An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.