

# Chapter 34

## Matchings

By Sarel Har-Peled, November 14, 2024<sup>①</sup>

Version: 0.4

I've never touched the hard stuff, only smoked grass a few times with the boys to be polite, and that's all, though ten is the age when the big guys come around teaching you all sorts to things. But happiness doesn't mean much to me, I still think life is better. Happiness is a mean son of a bitch and needs to be put in his place. Him and me aren't on the same team, and I'm cutting him dead. I've never gone in for politics, because somebody always stand to gain by it, but happiness is an even crummier racket, and their ought to be laws to put it out of business.

---

Momo, Emile Ajar

### 34.1. Definitions and basic properties

#### 34.1.1. Definitions

Definition 34.1.1. 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. An edge  $e = uv$  in the matching *covers* the vertices  $u$  and  $v$ .

Definition 34.1.2. 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 34.1.3. 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.

Problem 34.1.4 (Maximum size matching). 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* (aka *maximum cardinality matching*).

Problem 34.1.5 (Maximum weight matching). Given a graph  $G$  and a weight function on the edges, compute the maximum weight matching in  $G$ .

Remark 34.1.6. 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.

#### 34.1.2. 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*.

---

<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.

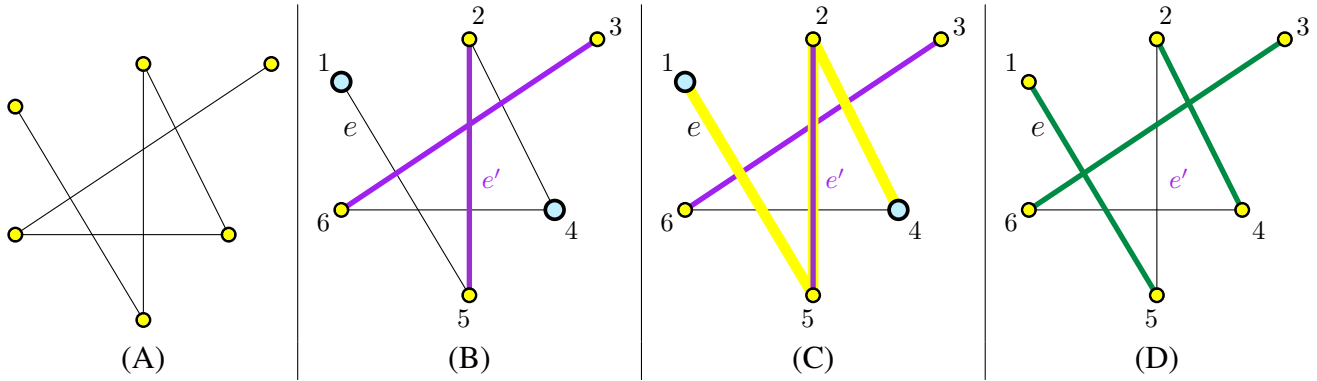


Figure 34.1: (A) The input graph. (B) A maximal matching in  $G$ . The edge  $e$  is free, and vertices 1 and 4 are free. (C) An alternating path. (D) The resulting matching from applying the augmenting path.

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

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

An augmenting path is an alternating path that starts and end with a free edge, and the two endpoints of the path are also free.

**Lemma 34.1.8.** 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 34.1.9.** Let  $M$  be a matching, and  $T$  be a maximum matching, and  $k = |T| - |M|$ . Then  $M$  has (at least)  $k$  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  $H = (V, E')$ , where  $V$  is the set of vertices used by the edges of  $E'$ , see **Figure 34.2**. 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 *disjoint* 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  $H$ .

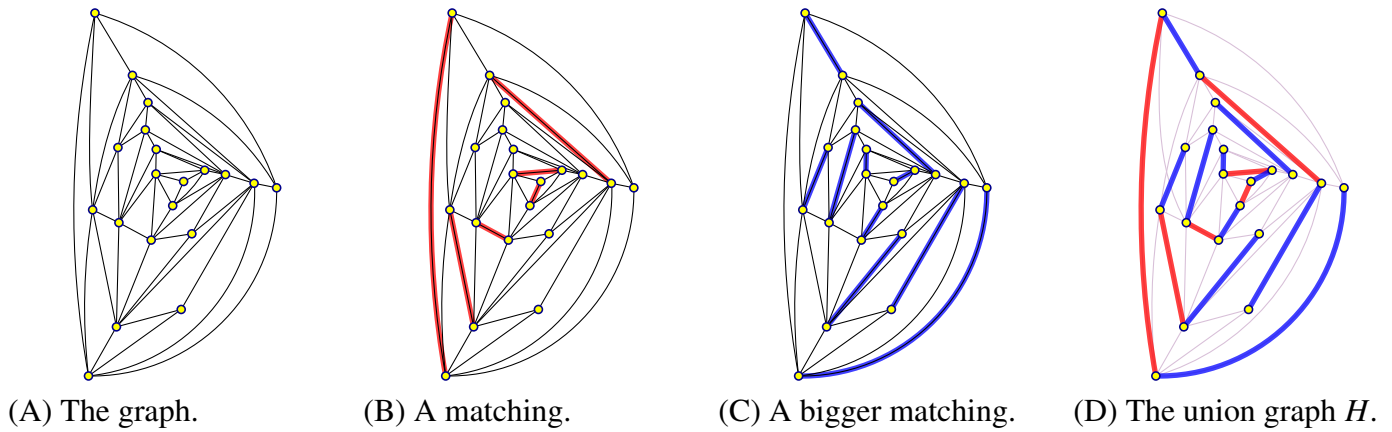


Figure 34.2: The graph formed by the union of matchings.

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  $k$  (vertex disjoint) augmenting paths were of length  $\geq u/k$  then the total number of vertices in  $H$  would be at least  $(u/k + 1)k > u$ , since a path of length  $\ell$  has  $\ell + 1$  vertices. A contradiction. ■

The lemma readily implies:

**Corollary 34.1.10.** *A matching  $M$  is maximum  $\iff$  there is no augmenting path for  $M$ .*

## 34.2. Unweighted matching in bipartite graph

### 34.2.1. 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 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  $r \in R$ , induces 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 outputs the current matching (i.e., it 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 ends 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 34.1.10](#), 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, each iteration of the algorithm can be implemented in linear time. We thus have the following result:

**Lemma 34.2.1.** 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.

### 34.3. Hall's theorem

Consider a bipartite graph  $G = (\mathcal{L} \cup \mathcal{R}, E)$ . An  $\mathcal{L}$ -*matching* is a matching  $M$  that covers all the vertices of  $\mathcal{L}$  – that is, for all vertices  $v \in L$ , there is an edge  $e \in M$  that covers  $v$ . The *neighborhood* of a set  $L \subseteq \mathcal{L}$  is the set

$$N(L) = \{r \in \mathcal{R} \mid \exists \ell \in L \text{ such that } \ell r \in E\}.$$

If  $|\mathcal{L}| = |\mathcal{R}|$  then an  $\mathcal{L}$ -matching is simply a perfect matching.

**Theorem 34.3.1 (Hall's theorem).** For a bipartite graph  $G = (\mathcal{L} \cup \mathcal{R}, E)$ , has an  $\mathcal{L}$ -matching  $M \iff$  for all  $L \subseteq \mathcal{L}$ , we have  $|L| \leq |N(L)|$ .

*Proof:*  $\mathcal{L}$ -*matching*  $\implies$  **Hall's condition.** One direction is easy, if there is an  $\mathcal{L}$ -matching  $M$  in  $G$ , then for any set  $L$ , we have that  $M$  contains  $|L|$  edges in the matching  $M$  that covers the vertices of  $L$ . The endpoints of these edges are distinct, and they are all contained in  $\mathcal{R}$ . To this end, let

$$R = R(M, L) = \{r \in \mathcal{R} \mid \ell r \in M \text{ and } \ell \in L\},$$

and observe that  $|R| = |L|$ . Clearly,  $R \subseteq N(L)$ . Implying that  $|L| = |R| \leq |N(L)|$ .

$\overline{\mathcal{L}\text{-matching}} \implies \overline{\text{Hall's condition}}$ . Assume there is no  $\mathcal{L}$ -matching  $M$  in  $G$ . Then, consider a maximum matching  $M$  in  $G$ . there must be a free vertex  $\ell \in L$  that is not in  $M$ . Let  $L$  be all the vertices on  $\mathcal{L}$  reachable by an alternating path in  $G$  starting in  $L$ . Similarly, let  $R$  be the set of all vertices in  $\mathcal{R}$  reachable by an alternating path starting at  $\ell$ . Observe that:

- (I)  $R$  can not contain any free vertex, as then there would be an alternating path  $\pi$  from a free vertex to a free vertex, which implies that  $M \oplus \pi$  is a bigger matching. A contradiction to the assumption that  $M$  is maximum matching.
- (II)  $L$  contains  $\ell$  (duh).
- (III)  $L$  contains no other free vertex. Indeed, all the vertices in  $L$  alternately reachable from  $\ell$ , have a matching edge as their last edge (duh<sup>2</sup> [but really, all of math is a sequence of obvious observations]).
- (IV)  $|L| = |R| + 1$ . Indeed, any vertex in  $R$  is attached to a unique vertex in  $L - \ell$  by the matching  $M$  — otherwise it would be free, and that is not legal, because of (I).
- (V)  $N(L) = R$ . Clearly,  $R \subseteq N(L)$ . As for the other direction, Consider any vertex  $v \in L$ . There is an alternating path from  $\ell$  to  $v$ , denote it by  $\pi$ . The last edge  $vu$  in this path is a matching edge, and  $u \in R$ . All other edges  $vx$ , that are not in the matching, can be used to define a longer alternating path  $\pi \mid vx$ , implying that  $x \in R$ . Namely,  $N(v) \subseteq R$ , and thus  $N(L) \subseteq R$ .

We are done, as  $|L| = |R| + 1 = |N(L)| + 1$ . Namely, Hall's condition fails for  $L$ , as  $|L| > |N(L)|$ . ■

#### 34.3.1. Some exercises

**Question 34.3.2.** Consider a standard deck of cards – there are 13 ranks (1, ..., 10, Princess, Queen and King. There are 4 suits: ♠, ♣, ♥, ♦, (thus 52 cards overall). Consider dividing the cards into piles of 4 cards, where no pile contains the same number twice. Show, that can select exactly one card from each pile, such that overall we get all 13 possible values.

**Question 34.3.3.** Consider a bipartite graph  $G = (\mathcal{L} \cup \mathcal{R}, E)$  that is  $k$ -regular (i.e., all vertices have the same degree  $k$ ). Prove that there is a perfect matching in  $G$ .

**Question 34.3.4.** Prove that given a  $k$ -regular bipartite graph, describe an algorithm that color the edges with  $k$  colors, such that no two edges with the same color share a vertex.

**Question 34.3.5.** Let  $R$  and  $B$  be two sets of  $n$  points in the plane. Consider the natural bipartite graph  $G = (R \cup B, E)$ , where the length of an edge is the distance between the two vertices its connect. Describe a polynomial time algorithm that computes a perfect matching  $M$  between  $R$  and  $B$  that minimizes the longest edge in  $M$ .

## 34.4. Bipartite unweighted matchings: The Hopcroft-Karp algorithm

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

### 34.4.1. Some more structural observations

We need three basic observations:

- (A) If we augmenting along a shortest path, then the next augmenting path must be longer (or at least not shorter). See [Lemma 34.4.1](#) below.
- (B) As such, if we always augment along shortest paths, then the augmenting paths get longer as the algorithm progress, see [Corollary 34.4.2](#) below.
- (C) Furthermore, all the augmenting paths of the same length used by the algorithm are vertex-disjoint (!). See [Lemma 34.4.3](#) 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 34.4.1.** 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  $\sigma_2$  – importantly, both  $\sigma_1$  and  $\sigma_2$  are augmenting paths of the original matching  $M$ .

Observe that for any sets  $B, C, D$ , we have  $B \oplus (C \oplus D) = (B \oplus C) \oplus D$ . This implies that

$$M \oplus N = M \oplus (M \oplus \pi \oplus \pi') = \pi \oplus \pi'.$$

As such, we have

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

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

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

By definition, we have that  $|\pi \oplus \pi'| = |\pi| + |\pi'| - 2|\pi \cap \pi'|$ . To see why the factor 2 is there, observe that for  $e \in \pi \cap \pi'$  we have  $e \notin \pi \oplus \pi'$ . 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 34.4.2.** *Let  $\pi_1, \pi_2, \dots, \pi_t$  be the sequence of augmenting paths used by the algorithm of [Section 34.2.1](#) (which always augments the matching along the shortest augmenting path). We have that  $|\pi_1| \leq |\pi_2| \leq \dots \leq |\pi_t|$ .*

**Lemma 34.4.3.** *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 the sake of contradiction, 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 that they share 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 34.4.1](#), we conclude that

$$|\pi_j| \geq |\pi_i| + 2|\pi_i \cap \pi_j| > |\pi_i|.$$

A contradiction. ■

### 34.4.2. 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 extract all the 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 34.4.4](#) 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.

**Lemma 34.4.4.** *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  $|T| \leq |M| + O(\sqrt{n})$ .*

*Proof:* At this point, the shortest augmenting path for the current matching  $M$  is of length at  $\geq \sqrt{n}$ . By [Lemma 34.1.9](#), 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  $n \geq 4$ . ■

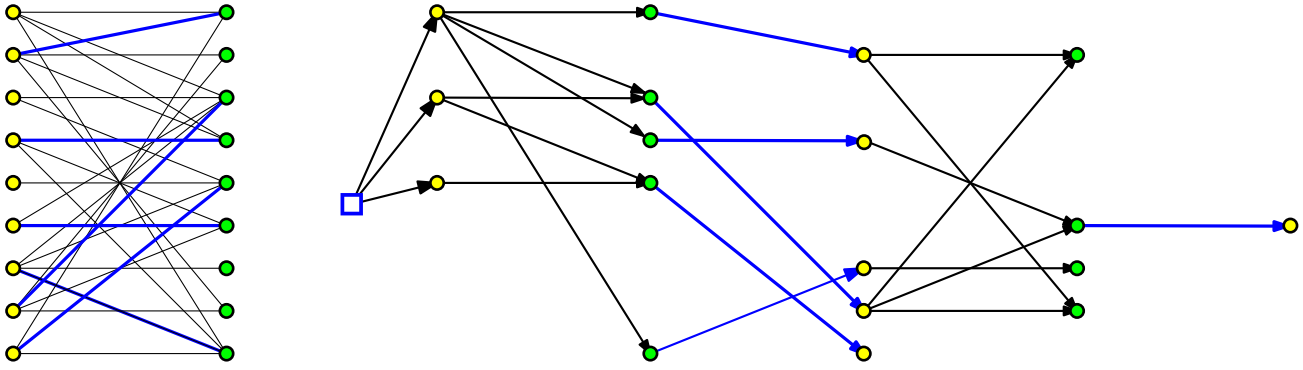


Figure 34.3: (A) A bipartite graph and its matching. (B) Its layered graph.

### 34.4.3. 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 paths simultaneously. 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  $L_1, R_1, L_2, R_2, L_3, \dots$  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**, which is impossible for bipartite graphs, so there are no such edges here). 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 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

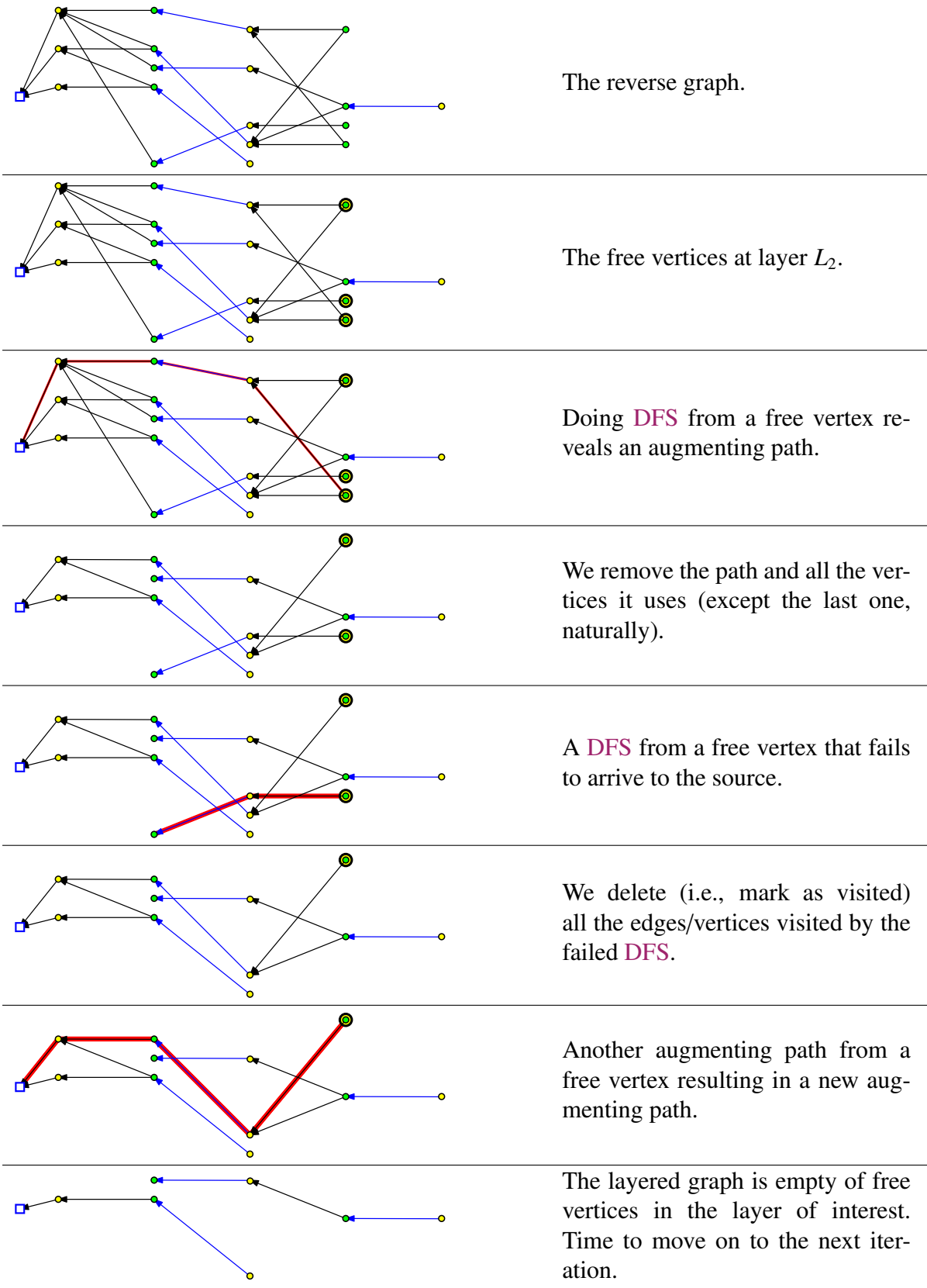


Figure 34.4: Extracting augmenting paths from the reverse layered graph.



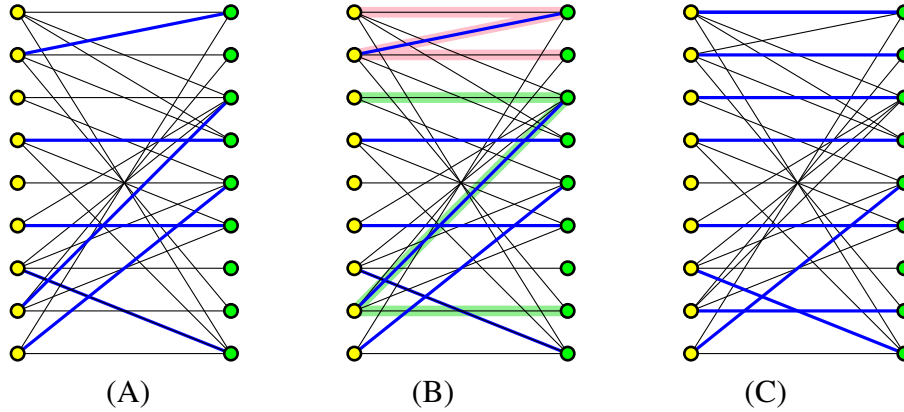


Figure 34.5: (A) A bipartite graph and its current matching. (B) Augmenting paths computed using the layered graph (see Figure 34.3). (C) The new matching after we apply the augmenting paths.

of the algorithm is  $O(n + m)$  as desired.

Repeated application of Lemma 34.4.1 implies the following.

**Observation 34.4.5.** *Assume  $M$  is a matching, such that the shortest augmenting path for it is of length  $k$ . Then, augmenting it with a sequence of paths of length  $k$ , results in matching  $M'$ , with its shortest augmenting path being of length at least  $k$ .*

**Lemma 34.4.6.** *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 that is disjoint from the vertices of the paths of  $P_k$ . By the above observation, the path  $\sigma$  is of length at least  $k$ .

The interesting case here is if  $\sigma$  is of length exactly  $k$ . 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  $P_k$ . ■

## The result

**Theorem 34.4.7.** *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 `algMatchingHK` algorithm is described in Section 34.4.2, 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 34.2.1. Indeed, the `algMatchingHK` 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 34.4.6 each set  $P_k$  contains a maximal set of such vertex-disjoint augmenting paths of length  $k$ . By Lemma 34.4.3, all augmenting paths of length  $k$  computed are vertex disjoint.

As such, now by induction, we can argue that if `algMatchingHK` 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. ■

## 34.5. 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.