Chapter 32

Matchings

By Sariel Har-Peled, August 31, 2023[1] Version: 0.3

> 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

32.1. Definitions and basic properties

32.1.1. Definitions

Definition 32.1.1. For a graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ a set $M \subseteq \mathsf{E}$ of edges is a ***matching*** if no pair of edges of M has a common vertex.

Definition 32.1.2. 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 32.1.3. 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 32.1.4 (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 32.1.5 (Maximum weight matching). Given a graph G and a weight function on the edges, compute the maximum weight matching in G.

Remark 32.1.6. 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.

32.1.2. Matchings and alternating paths

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

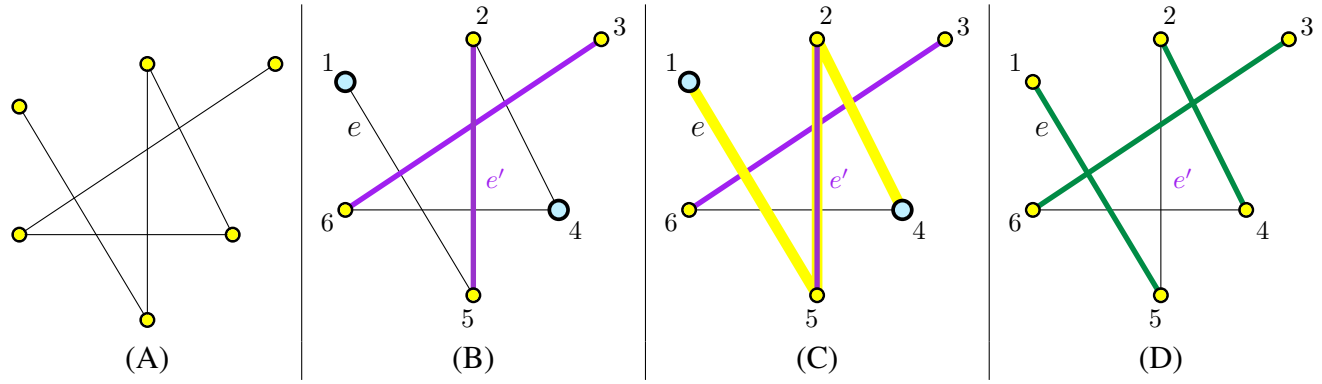


Figure 32.1: (A) The input graph. (B) A maximal matching in G. The edge e is free, and vertices 1 and 4 are free. (C) An alternating path. (D) The resulting matching from applying the augmenting path.

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 32.1.7. A path $\pi = v_1 v_2, \ldots, v_{2k+2}$ is an ***augmenting*** path for a matching M in a graph G if

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

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 32.1.8. *If M is a matching and π is an augmenting path relative to M, then*

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

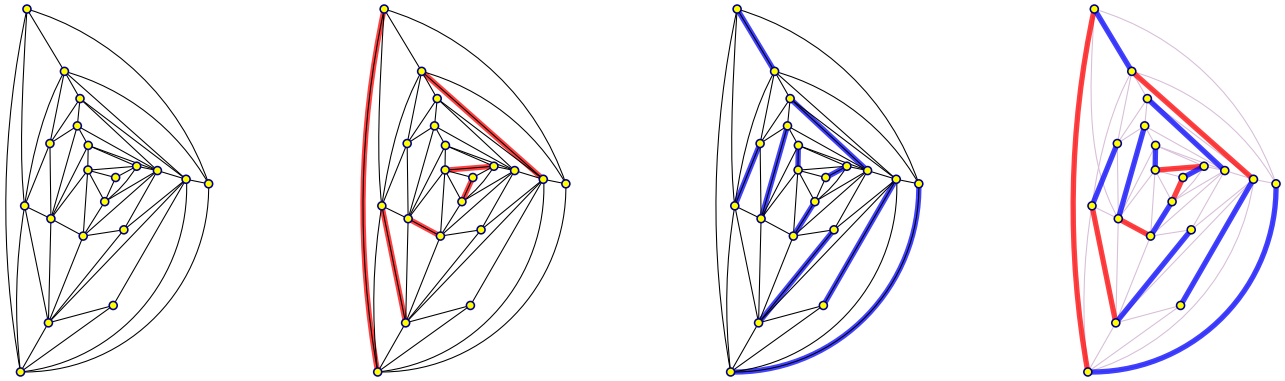
is a matching of size $|M| + 1$.

Proof: Think about removing π from the graph all together. What is left of M, is a matching of size $|M| - |M \cap \pi|$. Now, add back π and alternate the edges of the matching M with the free edges of π. Clearly, the new set of edges is a matching, since π 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 32.1.9. *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 32.2. 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.



(A) The graph. (B) A matching. (C) A bigger matching. (D) The union graph H.

Figure 32.2: The graph formed by the union of matchings.

As for the claim on the length of the shortest augmenting path. Let $u = |\mathsf{V}(H)| \leq 2(|T| + |M|)$. 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 ℓ has $\ell + 1$ vertices. A contradiction. ■

The lemma readily implies:

Corollary 32.1.10. *A matching M is maximum $\iff$ there is no augmenting path for M.*

32.2. Unweighted matching in bipartite graph

32.2.1. The slow algorithm; algSlowMatch

The algorithm. Let $\mathsf{G} = (L \cup R, \mathsf{E})$ be a bipartite graph. Let $M_0 = \emptyset$ be an empty matching. In the ith 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 $\mathsf{E} \setminus M_{i-1}$ from left to the right. Formally, an edge $lr \in \mathsf{E} \setminus M_{i-1}$, 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* π_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 32.1.10, 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 32.2.1. *Given a bipartite undirected graph $\mathsf{G} = (L \cup R, \mathsf{E})$, with n vertices and m edges, one can compute the maximum matching in G in $O(nm)$ time.*

32.2.2. The Hopcroft-Karp algorithm

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

32.2.2.1. 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 32.2.2 below.

(B) As such, if we always augment along shortest paths, then the augmenting paths get longer as the algorithm progress, see Corollary 32.2.3 below.

(C) Furthermore, all the augmenting paths of the same length used by the algorithm are vertex-disjoint (!). See Lemma 32.2.4 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 32.2.2. *Let M be a matching, and π be the shortest augmenting path for M, and let π' 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 σ_1 and σ_2 – importantly, both σ_1 and σ_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 π 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

$$|\pi| + |\pi'| - 2\,|\pi \cap \pi'| \geq 2\,|\pi| \qquad \Longrightarrow \qquad |\pi'| \geq |\pi| + 2\,|\pi \cap \pi'|\,. \qquad \blacksquare$$

The above lemma immediately implies the following.

Corollary 32.2.3. *Let $\pi_1, \pi_2, \ldots, \pi_t$ be the sequence of augmenting paths used by the algorithm of Section 32.2.1 (which always augments the matching along the shortest augmenting path). We have that $|\pi_1| \leq |\pi_2| \leq \ldots \leq |\pi_t|$.*

Lemma 32.2.4. *For all i and j, such that $|\pi_i| = \cdots = |\pi_j|$, we have that the paths π_i and π_j are vertex disjoint.*

Proof: Assume for the sake of contradiction, that $|\pi_i| = |\pi_j|$, $i < j$, and π_i and π_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 π_k is disjoint from π_i and π_j.

Now, let M_i be the matching after π_i was applied. We have that π_j is not using any of the edges of $\pi_{i+1}, \ldots, \pi_{j-1}$. As such, π_j is an augmenting path for M_i. Now, π_j and π_i share vertices. It definitely can not be that they share the two endpoints of π_j (since they are free) - so it must be some interval vertex of π_j. But then, π_i and π_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 π_j - since it contains the only matching edge adjacent to v in M_i. Namely, $\left|\pi_i \cap \pi_j\right| \geq 1$. Now, by Lemma 32.2.2, we conclude that

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

A contradiction. ∎

32.2.2.2. 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 extract all the augmenting paths of length k for M in G in $O(m)$ time, for $k = 1, 3, 5, \ldots$. Specifically, we apply this extraction algorithm, till $k = 1 + 2\left\lceil \sqrt{n} \right\rceil$. This would take $O(km) = O(\sqrt{n}m)$ 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 32.2.5 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.

Lemma 32.2.5. *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 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 32.1.9, 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 $n \geq 4$. ∎

32.2.2.3. 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 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

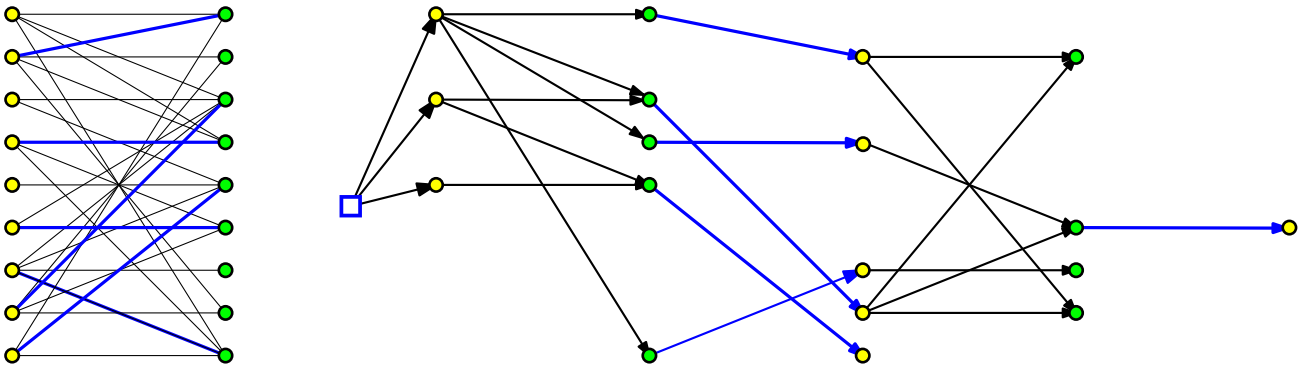


Figure 32.3: (A) A bipartite graph and its matching. (B) Its layered graph.

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 $L_1, R_1, L_2, R_2, L_3, \ldots$ be the layers of the BFS. By assumption, the first free vertex below L_1 encountered in the tree is of level R_τ, 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^{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^{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_τ, till we visit all such vertices.

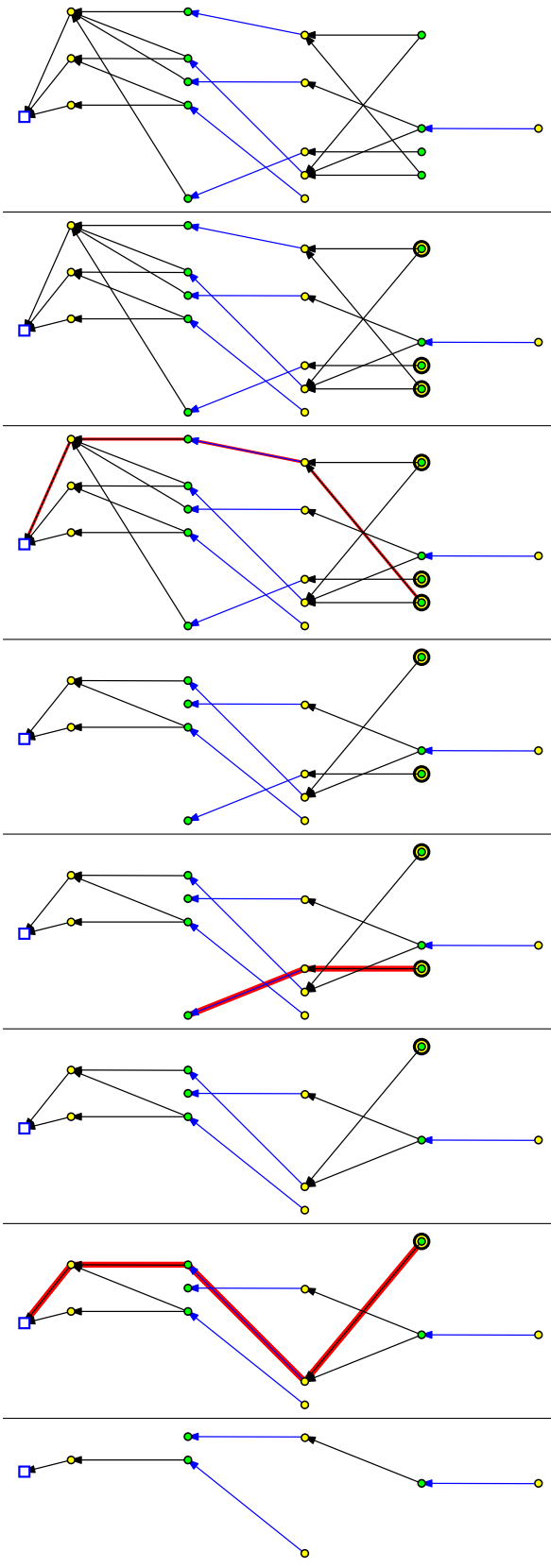
This algorithm results in a collection of augmenting paths P_τ, which are vertex disjoint. We claim that P_τ is the desired set maximal cardinality disjoint set of augmenting paths of length k.

Analysis. Building the initial graphs J and J^{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^{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.

Repeated application of Lemma 32.2.2 implies the following.

Observation 32.2.6. *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 32.2.7. *The set P_k is a maximal set of vertex-disjoint augmenting paths of length k for M.*



The reverse graph.

The free vertices at layer L_2.

Doing DFS from a free vertex reveals an augmenting path.

We remove the path and all the vertices it uses (except the last one, naturally).

A DFS from a free vertex that fails to arrive to the source.

We delete (i.e., mark as visited) all the edges/vertices visited by the failed DFS.

Another augmenting path from a free vertex resulting in a new augmenting path.

The layered graph is empty of free vertices in the layer of interest. Time to move on to the next iteration.

Figure 32.4: Extracting augmenting paths from the reverse layered graph.

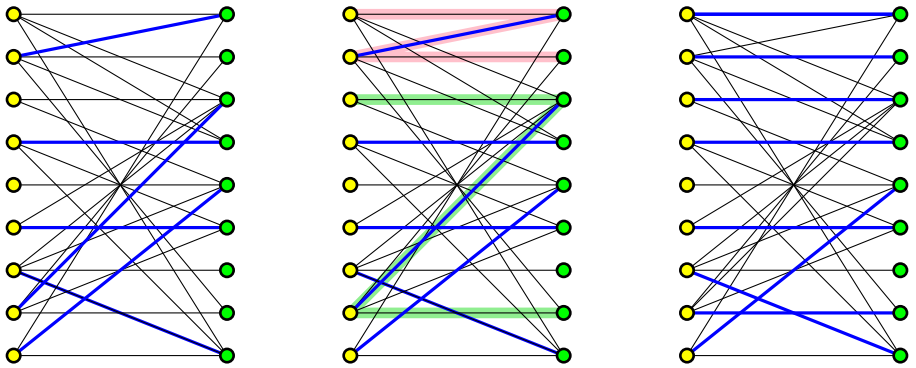


Figure 32.5: (A) A bipartite graph and its current matching. (B) Augmenting paths computed using the layered graph (see Figure 32.3. (C) The new matching after we apply the augmenting paths.

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 σ of that is disjoint from the vertices of the paths of P_k. By the above observation, the path σ is of length at least k.

The interesting case here is if σ is of length exactly k. Then, we could traverse σ in J, and this would go through unused vertices. Indeed, if any of the vertices of σ 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 σ is supposedly disjoint from the paths of P_k. ■

32.2.2.4. The result

Theorem 32.2.8. *Given a bipartite unweighted graph* G *with* n *vertices and* m *edges, one can compute maximum matching in* G *in* $O(\sqrt{n}m)$ *time.*

Proof: The algMatching_{HK} algorithm is described in Section 32.2.2.2, 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 32.2.1. Indeed, the algMatching_{HK} algorithm computes a sequence of sets of augmenting paths $P_1, P_3, P_5, \ldots$. 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 32.2.7 each set P_k contains a maximal set of such vertex-disjoint augmenting paths of length k. By Lemma 32.2.4, all augmenting paths of length k computed are vertex disjoint.

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

32.3. 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.