Maximum Matching in Bipartite Graphs

CSCI 4113/6101
Instructor: Norbert Zeh
October 27, 2025

In the previous topic, we have shown that a maximum matching in a bipartite graph can be found in $O\left(n^2m\right)$ time via a reduction from maximum matching to maximum flow. In this topic, we show that finding a maximum matching is in fact much easier than finding a maximum flow, both conceptually and in terms of the amount of time required to find a maximum matching. We will start with a simple $O(nm)$-time algorithm to find a maximum matching. The discussion of this algorithm introduces the core concepts used by all matching algorithms, also those for minimum-weight perfect matching and maximum-weight matching. One of these concepts is that of an **augmenting path**. Just as with augmenting path algorithms for computing maximum flows, the name "augmenting path" suggests that we use these paths to augment (i.e., grow) the matching, but an augmenting path for a matching is defined differently from an augmenting path for a feasible flow. So make sure you keep these two concepts separate in your head, in spite of the common name. Having introduced the key concepts needed to find a maximum matching, we will look at a clever algorithm, due to Hopcroft and Karp, that finds multiple augmenting paths simultaneously. As we will show, this allows the algorithm to finish in $O\left(m\sqrt{n}\right)$ time.

1 A Framework for Finding Maximum Matchings

All maximum matching algorithms we will discuss in this course use the same high-level strategy to find a maximum matching. They start with an arbitrary matching M and then grow this matching iteratively. Each iteration tests whether M is a maximum matching. If this is the case, the algorithm stops and returns M. Otherwise, it constructs a larger matching from M and uses this larger matching as the input to the next iteration.

To turn this outline into a complete algorithm, we need to discuss

- How to find the initial matching M,
- How to decide whether M is a maximum matching, and
- How to construct a larger matching from M if M is not maximum.

Since any matching is good enough as the initial matching, and $M = \emptyset$ is clearly a matching, this is a valid starting point. In practice, we may want to start with a maximal matching instead because, as we discussed in the previous topic, such a matching can be found in $O(n + m)$ time and starting with a hopefully large matching reduces the number of iterations needed before we obtain a maximum matching. In theory, this reduces the running time of the algorithm by only a constant factor because a maximal matching may be as small as half the size of a maximum matching, that is, starting with a maximal matching instead of the empty matching cuts the number of iterations in half (see Exer. 1).

We answer the last two questions together. In the following section, we define the concept of an augmenting path P for a matching M and prove that (a) M is maximum if it has no augmenting path and (b) for any augmenting path P of M, $M \oplus P$ is matching of size $|M \oplus P| = |M| + 1$. We use the notation $A \oplus B$ to denote the symmetric difference of two sets A and B: $A \oplus B = (A \setminus B) \cup (B \setminus A)$. In words, $A \oplus B$ contains all the elements that are in exactly one of the two sets A and B. When M is a matching and P is (the set of edges in) a path, then you can think about the construction of $M \oplus P$ from M as "flipping" the edges in P: Every edge in P that is not in M is in $M \oplus P$, and every edge in P that is in M is not in $M \oplus P$.

2 A Characterization of Maximum Matchings via Augmenting Paths

An **alternating path** in G with respect to a matching M is a path in G whose edges alternate between being in M and not being in M. We call an alternating path **even** or **odd** if it contains an even or odd number of edges. An **alternating cycle** is a cycle of even length whose edges alternate between being in M and not being in M. We call an alternating path P an **augmenting path** if both endpoints of P are unmatched. This implies in particular that P must be odd and must start and end with unmatched edges. These definitions are illustrated in Fig. 1. As this figure illustrates though, it is possible that an alternating path is odd, starts and ends with unmatched edges, but is not an augmenting path. It is crucial that the endpoints of the path are unmatched for an alternating path to be an augmenting path.

In this section, we prove that a matching M is maximum if and only if it has no augmenting path. The "only if" direction is easy to prove:

Proposition 1. *If $G = (V, E)$ is a graph, $M \subseteq E$ is a matching, and P is an augmenting path for M, then $M \oplus P$ is a matching of size $|M \oplus P| = |M| + 1$.*

Proof. Since P is an augmenting path, it is odd, starts with an unmatched edge, and ends with an

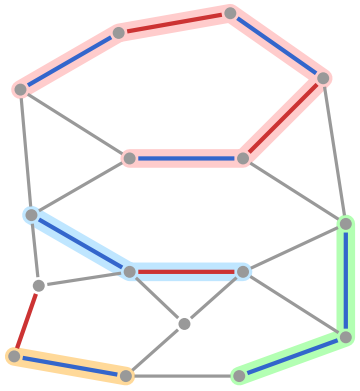


Figure 1: Illustration of alternating and augmenting paths for a matching M formed by the red edges. The red, blue, and yellow paths are alternating. The green path is not, because it contains two consecutive unmatched edges. The red path R is augmenting because its endpoints are both unmatched. In particular, $M \oplus R$ is a matching of size $|M \oplus R| = |M| + 1$. The blue path B is not an augmenting path, because it has even length. However, since B has even length and its vertices have no incident edges in $M \setminus B$, $M \oplus B$ is a matching of the same size as M. Even though it starts and ends with unmatched edges (the same edge in this case), the yellow path Y is not an augmenting path because its left endpoint has an incident edge in $M \setminus Y$, which implies that $M \oplus Y$ is not a matching.

unmatched edge. Since P is alternating, this implies that P contains one more edge not in M than edges in M. Therefore,

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

It remains to prove that $M \oplus P$ is a matching.

So consider an arbitrary vertex $v \in V$. If $v \notin P$, then it has the same incident edges in $M \oplus P$ as in M. Since M is a matching, v has at most one incident edge in M and, thus, at most one incident edge in $M \oplus P$.

If v is an endpoint of P, then v has no incident edges in M because P is an augmenting path for M. Thus, it has no incident edges in $M \setminus P$ either, that is, all incident edges of v in $M \oplus P$ belong to $P \setminus M$ and, therefore, to P. Since v is an endpoint of P, it has only one incident edge in P and, therefore, at most one incident edge in $M \oplus P$.

If v is an internal vertex of P, then let $\{u, v\}$ and $\{v, w\}$ be the two edges in P incident to v. Since P is an alternating path, w.l.o.g. $\{u, v\} \in M$ and $\{v, w\} \notin M$. Since M is a matching and $\{u, v\} \in P$, this implies that v has no incident edges in $M \setminus P$. Thus, the only edge in $M \oplus P = (M \setminus P) \cup (P \setminus M)$ incident to v is $\{v, w\}$.

Since we have shown that every vertex in G has at most one incident edge in $M \oplus P$, $M \oplus P$ is a matching. □

Prop. 1 immediately implies that M is not a maximum matching if there exists an augmenting path P for M, because $M \oplus P$ is a bigger matching. To prove the converse, we need the following lemma, illustrated in Fig. 2.

Lemma 2. *For two matchings M_1 and M_2, $M_1 \oplus M_2$ is a collection of disjoint paths $P_1, \ldots, P_k$ and cycles $C_1, \ldots, C_\ell$. If a_1 is the number of paths among $P_1, \ldots, P_k$ that are augmenting for M_1, and a_2 is the number of paths among $P_1, \ldots, P_k$ that are augmenting for M_2, then $|M_1| - |M_2| = a_2 - a_1$.*

Proof. Every vertex has at most one incident edge in M_1 and at most one incident edge in M_2. Thus, it has at most two incident edges in $M_1 \oplus M_2$. Since this is true for every vertex, $M_1 \oplus M_2$ is a collection of disjoint paths and cycles.

We have $|M_1| = |M_1 \cap M_2| + |M_1 \setminus M_2|$ and $|M_2| = |M_1 \cap M_2| + |M_2 \setminus M_1|$. Thus,

$$\begin{aligned}
|M_1| - |M_2| &= |M_1 \setminus M_2| - |M_2 \setminus M_1| \\
&= |M_1 \cap (M_1 \oplus M_2)| - |M_2 \cap (M_1 \oplus M_2)| \\
&= \sum_{i=1}^{k} \big(|M_1 \cap P_i| - |M_2 \cap P_i|\big) + \sum_{i=1}^{\ell} \big(|M_1 \cap C_i| - |M_2 \cap C_i|\big).
\end{aligned}$$

Since M_1 and M_2 are matchings, the edges in every path P_i and every cycle C_i alternate between being in M_1 and M_2. Thus, every cycle C_i has even length and contains the same number of edges from M_1 and M_2, that is, $|M_1 \cap C_i| - |M_2 \cap C_i| = 0$, for all $1 \le i \le \ell$. Similarly, every even-length path P_i contains the same number of edges from M_1 and M_2, that is, $|M_1 \cap P_i| - |M_2 \cap P_i| = 0$, for any such path.

Every odd-length path either contains one more edge from M_1 than from M_2 or one more edge from M_2 than from M_1. Let $\mathcal{P}_1$ be the set of paths that contain one more edge from M_2 than from M_1, and let

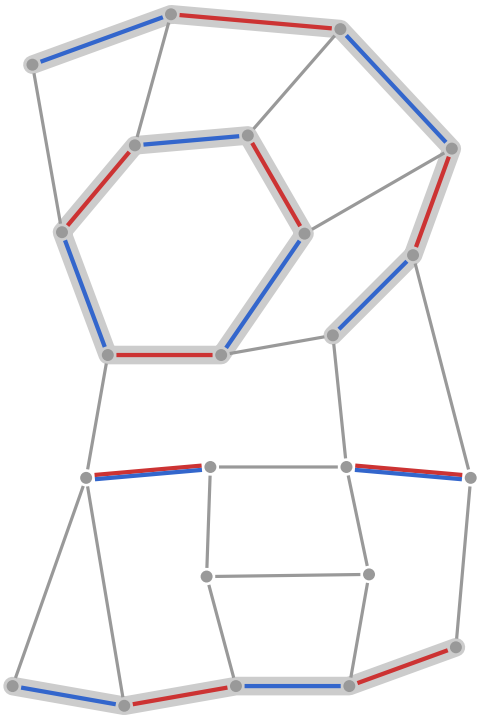


Figure 2: The symmetric difference $M_1 \oplus M_2$ (shaded) of two matchings M_1 (red edges) and M_2 (blue edges) is composed of a collection of alternating paths and cycles. In this example, one of the paths is an augmenting path for M_1 and there are no augmenting paths for M_2. Thus, $|M_2| = |M_1| + 1$.

$\mathcal{P}_2$ be the set of paths that contain one more edge from M_1 than from M_2. Then

$$|M_1| - |M_2| = |\mathcal{P}_2| - |\mathcal{P}_1|.$$

Next we prove that a path among $P_1, \ldots, P_k$ belongs to $\mathcal{P}_1$ if and only if it is an augmenting path for M_1. By symmetry, a path among $P_1, \ldots, P_k$ belongs to $\mathcal{P}_2$ if and only if it is an augmenting path for M_2. Thus, $|\mathcal{P}_1| = a_1$, $|\mathcal{P}_2| = a_2$, and $|M_1| - |M_2| = a_2 - a_2$, as claimed.

So consider a path $P_i \in \mathcal{P}_1$. Since P_i is an alternating path with respect to M_1, to prove that P_i is an augmenting path for M_1, we need to prove that its endpoints u and v are unmatched by M_1. Consider u. The proof that v is unmatched is analogous. The edges in P_i alternate between M_1 and M_2, and P_i contains one more edge from M_2 than from M_1. Thus, the edge e in P_i incident to u is in M_2. Since $e \in P_i \subseteq M_1 \oplus M_2$, this implies that $e \notin M_1$. Thus, if u were matched by M_1, the edge f in M_1 incident to u would have to be a different edge, $f \neq e$. Since $e \in M_2$ and M_2 is a matching, we have $f \notin M_2$, so $f \in M_1 \oplus M_2$. This shows that u would have two incident edges in $M_1 \oplus M_2$, but being the endpoint of P_i, it has degree 1 in $M_1 \oplus M_2$. This is a contradiction, so u must be unmatched. This shows that P_i is augmenting for M_1.

Now consider a path P_i that is an augmenting path for M_1. By the definition of an augmenting path, P_i is an alternating path with respect to M_1 whose endpoints are unmatched by M_1. Thus, the first and last edges of P_i do not belong to M_1. This implies that P_i has one more edge from M_2 than from M_1, that is, $P_i \in \mathcal{P}_1$. □

PROPOSITION 3. *If $G = (V, E)$ is a graph, and $M \subseteq E$ is a matching, then M is maximum if and only if there is no augmenting path in G with respect to M.*

Proof. The "only if" direction follows from Prop. 1. For the "if" direction, assume that M is not maximum, and let M' be a maximum matching. Since $|M'| > |M|$, Lem. 2 shows that one of the paths in $M \oplus M'$ must be an augmenting path for M. □

3 How to Find an Augmenting Path in a Bipartite Graph

Everything we have discussed so far applies to arbitrary graphs. Even in non-bipartite graphs, the strategy to find a maximum matching is to start with an arbitrary matching and then repeatedly look for augmenting paths to grow the matching. By Prop. 3, once we fail to find an augmenting path, the matching is maximum. To complete the algorithm, we need to discuss how to test whether there exists an augmenting path and, if so, find one. This is where assuming that the graph is bipartite makes our life a lot easier. In this section, we show that in a bipartite graph, we can find an augmenting path, or verify that none exists, in $O(n + m)$ time.

To decide whether there exists an augmenting path with respect to M, we need a structure called an **alternating forest**. This is a rooted forest $F \subseteq G$ (i.e., a collection of rooted trees) whose roots are unmatched and with the property that every path in F from a root of F to one of its descendants is an alternating path with respect to M. For every node v, we refer to the tree in F that contains v as T_v, to the root of T_v as r_v, and to the path from r_v to v in T_v as P_v. We call a vertex **odd** or **even** depending on whether P_v is a path with an odd or even number of edges. See Fig. 3 for an illustration. We call an alternating forest F **maximal** if there is no alternating forest $F' \supset F$ with the same roots.

LEMMA 4. *Given a graph $G = (V, E)$, a matching $M \subseteq E$, and a set $R \subseteq V$ of vertices unmatched by M, a maximal alternating forest with R as its set of roots can be found in $O(n + m)$ time.*

Proof. We use an adaptation of BFS, called **alternating BFS**. We initialize the BFS queue to contain all vertices in R and mark these vertices as explored; all other vertices are marked as unexplored. Note that this implies that the queue contains only even vertices initially. The algorithm maintains the invariant that the queue contains only even vertices at all times.

When removing an even vertex v from the queue, we inspect all edges not in M incident to v. For each such edge (v, w) whose other endpoint w is unexplored, we mark w as explored and make it a child of v in F. This turns w into an odd vertex. If w is matched by an edge (w, x), we mark x as explored and make x a child of w in F. This makes x an even vertex, and we add x to the queue. (Note that we do not need to check whether x is explored; since w was unexplored and we always add the two endpoints of a matching edge to F together, x is explored if and only if w is explored.)

The same analysis as for standard BFS shows that this procedure takes $O(n + m)$ time and it is easily verified that it outputs an alternating forest F. We need to prove that the alternating forest is maximal.

Let $F' \supseteq F$ be a largest alternating forest with R as its set of roots. For each vertex $v \in F'$, let T'_v be the tree in F' that contains v, let r'_v be the root of T'_v, and let P'_v be the path from r'_v to v in T'_v.

If $F' = F$, then F is maximal. So assume that $F' \supset F$. Since all roots of F' belong to R, this implies that there exists a vertex $v \in F'$ that is not a vertex of F and whose parent u in F' is in F.

If v is an odd vertex of F', then P'_v has odd length. Since r'_v is unmatched, this implies that neither

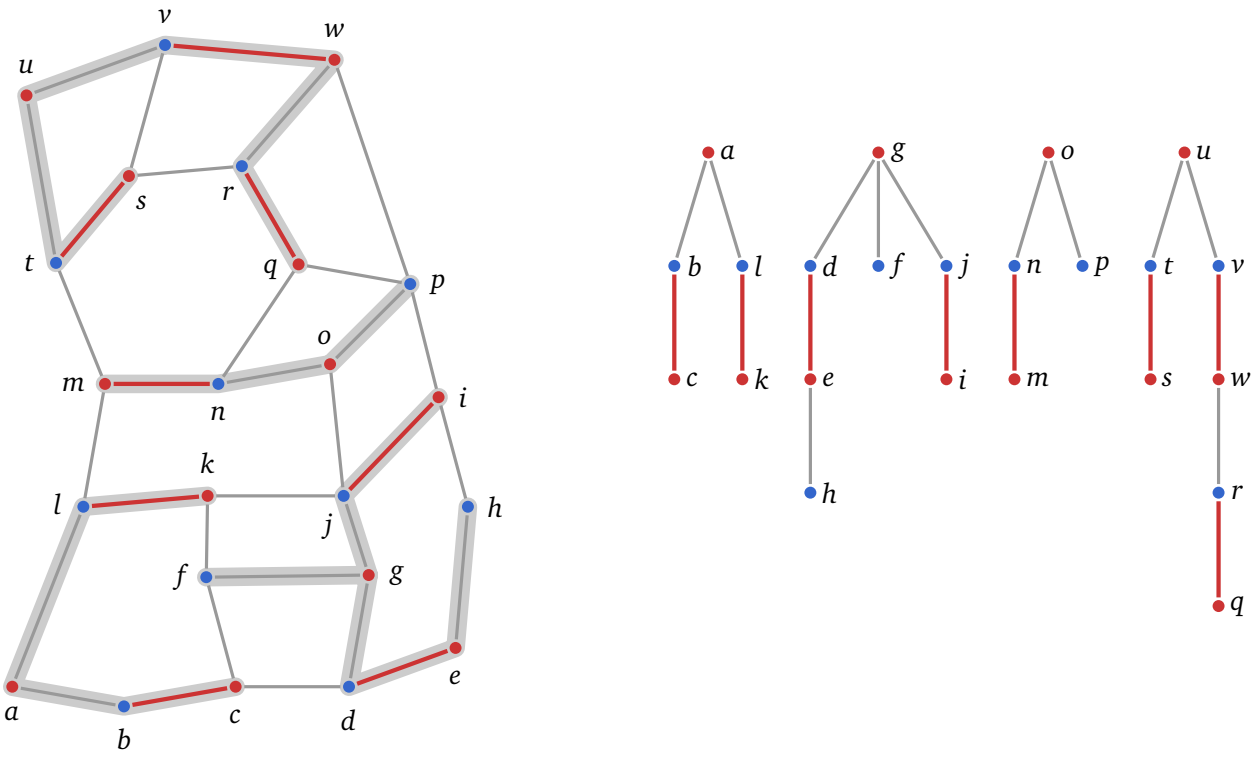


Figure 3: A bipartite graph (vertices in U are red, vertices in W are blue) and a matching (red edges). An alternating forest whose roots are the unmatched vertices in U is shaded and, to highlight its rooting, also drawn separately on the right. Note that, given that we root the alternating forest in the unmatched vertices in U, and since G is bipartite, all even vertices are in U, and all odd vertices are in W.

the first nor the last edge of P'_v is in M. Since v is odd, u is even. By the choice of v, the path $P'_u \subset P'_v$ from $r'_u = r'_v$ to u in $T'_u = T'_v$ is a path in F. Since this path has even length, this shows that u is even. Since $u \in F$, this implies that alternating BFS adds u to its queue when discovering it. Therefore, it also removes u from the queue. At this time, it explores all edges not in M incident to u, including the edge $\{u, v\}$. When exploring this edge, v is added to F unless v is already in F. Thus, $v \in F$, a contradiction.

If v is an even vertex of F', then P'_v has even length. Since r'_v is unmatched, this implies that $\{u, v\} \in M$. For every edge $\{x, y\} \in M$, alternating BFS adds either both or none of x and y to F. Thus, since $u \in F$, v must also be in F, again a contradiction.

Since we arrive at a contradiction if $F' \supset F$, we must have $F' = F$, that is, F is a maximal alternating forest. □

Now consider some vertex v such that there exists an alternating path P from a vertex $u \in R$ to v. The next lemma states that, while alternating BFS may not find this particular path P, it always does find an alternating path from a vertex in R to v and, therefore, adds v to F. This statement is true only if G is bipartite. Since being able to discover all vertices reachable from unmatched vertices (in U) is the key to discovering an augmenting path via alternating BFS (see the proof of Lem. 6 below), this is the reason why finding augmenting paths in arbitrary graphs is harder: alternating BFS may fail to find such a path even if one exists.

Lemma 5. *Let $G = (U, W, E)$ be a bipartite graph, let $M \subseteq E$ be a matching, let $R \subseteq U$, and let F be a*

maximal alternating forest whose roots are the vertices in R. Then $v \in F$ if and only if there exists an alternating path from a vertex in R to v.

Proof. If $v \in F$, then P_v is an alternating path from r_v to v.

For the opposite direction, assume that there exists an alternating path A from some vertex $u \in R$ to v. If all vertices in A belong to F, then $v \in F$. Otherwise, there exists a vertex $z \in A$ such that $z \notin F$ but its predecessor y in A belongs to F. Let A' be the subpath of A from u to y.

If A' has odd length, then the last edge in A' is not in M because u is unmatched. Thus, $\{y,z\} \in M$. Since $u \in U$, A' has odd length, and G is bipartite, we have $y \in W$. Thus, since G is bipartite and $r_y \in U$, the path P_y must also have odd length. Since r_y is unmatched, this implies that P_y ends in an edge not in M. Therefore, since $z \notin F$, adding z and the edge $\{y,z\}$ to F produces an alternating forest $F' \supset F$ with R as its set of roots. This is a contradiction because F is a maximal alternating forest with R as its set of roots.

If A' has even length, then an analogous argument shows that, once again, there exists an alternating forest $F' \supset F$ with R as its set of roots, contradicting the maximality of F.

Since we obtain a contradiction from the assumption that there exists a vertex $z \in A$ that is not in F, this shows that every vertex in A belongs to F, which finishes the proof of the lemma. □

The following lemma demonstrates now why alternating forests are useful for finding augmenting paths:

Lemma 6. *Let $G = (U,W,E)$ be a bipartite graph, let $M \subseteq E$ be a matching, and let F be maximal alternating forest whose roots are the unmatched vertices in U. Then M is maximum if and only if F contains no unmatched vertex in W. If F does contain an unmatched vertex $w \in M$, then P_w is an augmenting path for M.*

Proof. First assume that F contains an unmatched vertex $w \in W$. Then P_w is an alternating path from r_w to w. Since r_w is unmatched, this makes P_w an augmenting path. In particular, by Prop. 3, M is not maximum.

Conversely, assume that M is not maximum. Then, by Prop. 3, there exists an augmenting path P for M. This path has odd length and has two unmatched endpoints. Since G is bipartite, this implies that one of the endpoints must be an unmatched vertex $u \in U$, and the other must be an unmatched vertex $w \in W$. Since P is an alternating path from u to w, Lem. 5 shows that $w \in F$. □

By Lem. 4, it takes $O(n+m)$ time to find an alternating forest F whose roots are the unmatched vertices in U. By Lem. 6, M is maximum if F does not contain an unmatched vertex $w \in W$ and, if it does, then P_w is an augmenting path. Thus, we immediately obtain the following corollary.

Corollary 7. *Given a bipartite graph $G = (U,W,E)$ and a matching M of G, it takes $O(n+m)$ time to decide whether there exists an augmenting path P for M and, if so, find such a path.*

4 An $O(nm)$-Time Maximum Matching Algorithm for Bipartite Graphs

We have all the pieces to build our first maximum matching algorithm for bipartite graphs now. The algorithm starts with an arbitrary matching M and repeatedly looks for an augmenting path to grow

the matching. Once it fails to find an augmenting path, it stops and returns the current matching. By Prop. 3, M is a maximum matching at this point, so the algorithm is correct.

Each iteration of the algorithm takes linear time, by Cor. 7. Thus, to bound the running time of the algorithm, we need to bound the number of iterations the algorithm executes before M is maximum. Since G has n vertices, every edge has two endpoints, and no two edges in a matching share an endpoint, we have the following observation:

OBSERVATION 8. *Any matching M in a graph with n vertices has size $|M| \leq n/2$.*

Since each iteration of the algorithm grows the matching by one edge, Obs. 8 implies that the algorithm runs for at most $n/2$ iterations. Since each iteration takes $O(n+m)$ time, we conclude that the algorithm runs in $O(n^2+mn)$ time. As we show next, we can do a little better:

THEOREM 9. *A maximum-cardinality matching in a bipartite graph can be found in $O(nm)$ time.*

Proof. We already argued that a maximum matching in a bipartite graph can be found in $O(n^2+nm)$ time. We usually compute matchings only in graphs with at least as many edges as vertices, in which case $O(n^2+nm) = O(nm)$. In theory though, $n^2+nm \notin O(nm)$ if $m \in o(n)$. In this case, we can use the following trick: In constant time, we can test whether the graph as no edges at all. In this case, we output the empty matching. If G has at least one edge, then we can remove all vertices without incident edges. This takes $O(n) = O(mn)$ time. These vertices are clearly irrelevant for finding a maximum matching. In the resulting subgraph $G' \subseteq G$, every vertex has at least one incident edge. Since every edge has 2 endpoints, this immediately implies that G' has at most $2m$ vertices, and $O(n^2+nm) = O(nm)$. Together with the preprocessing step to test whether G has any edges at all and to eliminate vertices without incident edges, the total running time of the algorithm is thus $O(1+nm) = O(nm)$. □

5 THE HOPCROFT-KARP ALGORITHM

If we want to improve on the algorithm in the previous section, then we need to reduce the cost per iteration or reduce the number of iterations. Without the use of fancy data structures, improving on the linear cost per iteration is a tall order. As we show next, it *is* possible to ensure that the algorithm terminates after at most $2\sqrt{n}$ iterations. Thus, its running time reduces from $O(nm)$ to $O\left(m\sqrt{n}\right)$. This improvement is due to Hopcroft and Karp.

The basic idea is this: In each iteration, we don't find only one augmenting path but multiple *disjoint* augmenting paths $P_1, \ldots, P_k$. As the next lemma shows, $M \oplus (P_1 \cup \cdots \cup P_k)$ is a matching of size $|M \oplus (P_1 \cup \cdots \cup P_k)| = |M| + k$. Thus, an iteration may grow the matching by more than one edge, which clearly reduces the number of iterations needed to obtain a maximum matching. As before, the algorithm terminates once it fails to find even a single augmenting path. By Prop. 3, the matching is maximum at this point.

LEMMA 10. *If $G = (V,E)$ is a graph, $M \subseteq E$ is a matching, and $P_1, \ldots, P_k$ are disjoint augmenting paths for M, then $M \oplus (P_1 \cup \cdots \cup P_k)$ is a matching of size $|M \oplus (P_1 \cup \cdots \cup P_k)| = |M| + k$.*

Proof. For $i \in [k]_0$, let $M_i = M \oplus (P_1 \cup \cdots \cup P_i)$, and let $V_i = \bigcup_{j=1}^{i} V(P_i)$ be the vertex set of the paths $P_1, \ldots, P_i$. We use induction on i to show that M_i is a matching of size $|M_i| = |M| + i$ and that every

vertex in $V \setminus V_i$ has the same incident edges in M_i as in M.

For $i = 0$, we have $M_0 = M$, so M_0 is a matching of size $|M| + 0$ and every vertex trivially has the same incident edges in M_0 as in M.

For $i > 0$, the induction hypothesis shows that M_{i-1} is a matching of size $|M| + i - 1$, and all vertices in $V \setminus V_{i-1}$ have the same incident edges in M_{i-1} as in M. Since the paths $P_1, \ldots, P_k$ are disjoint, we have $V(P_i) \subseteq V \setminus V_{i-1}$. Thus, all vertices in P_i have the same incident edges in M_{i-1} as in M. In particular, P_i, being an augmenting path for M, is an augmenting path for M_{i-1}. Thus, by Prop. 1, $M_i = M_{i-1} \oplus P_i$ is a matching of size $|M_{i-1}| + 1 = |M| + i$. Every vertex in $V \setminus V(P_i)$ has the same incident edges in M_i as in M_{i-1}. Since every vertex in $V \setminus V_{i-1}$ has the same incident edges in M_{i-1} as in M, this implies, that every vertex in $(V \setminus V_{i-1}) \cap (V \setminus V(P_i)) = V \setminus (V_{i-1} \cup V(P_i)) = V \setminus V_i$ has the same incident edges in M_i as in M. □

To bound the number of iterations of the Hopcroft-Karp algorithm by $2\sqrt{n}$, we need to choose the paths $P_1, \ldots, P_k$ we find in each iteration carefully. Let us call an augmenting path P for M a **shortest augmenting path** for M if there is no shorter augmenting path than P for M (a bit obvious really). We call a set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of shortest augmenting paths for M **inclusion-maximal** if the paths in $\mathcal{P}$ are disjoint and every shortest augmenting path for M not in $\mathcal{P}$ shares at least one vertex with some path in $\mathcal{P}$. The Hopcroft-Karp algorithm finds an inclusion-maximal set of shortest augmenting paths in each iteration. As Lem. 17 below shows, this can be done in $O(n + m)$ time.

Before proving this, we show why finding an inclusion-maximal set of augmenting paths in each iteration reduces the number of iterations the algorithm executes. We start with the following lemma:

Lemma 11. *Each iteration of the Hopcroft-Karp algorithm increases the length of the shortest augmenting path with respect to the current matching M.*

Proof. Let M be the matching at the beginning of the current iteration, let M' be the matching at the end of the current iteration, let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be the inclusion-maximal set of shortest augmenting paths we use to construct M' from M, and let ℓ be the length of the paths in $\mathcal{P}$.

Consider any augmenting path $P = \langle v_0, \ldots, v_{\ell'} \rangle$ with respect to M'. We need to show that $\ell' > \ell$.

If P is disjoint from all paths in $\mathcal{P}$, then P is also an augmenting path for M, and $\mathcal{P} \cup \{P\}$ is a collection of disjoint augmenting paths for M. Since $\mathcal{P}$ is an inclusion-maximal set of shortest augmenting paths for M, P is not a shortest augmenting path for M, that is, $\ell' > \ell$.

So assume that P shares a vertex v with at least one path $P_i \in \mathcal{P}$. Let $S = (P_1 \cup \cdots \cup P_k) \oplus P$. Since $M' = M \oplus (P_1 \cup \cdots \cup P_k)$, we have $P_1 \cup \cdots \cup P_k = M \oplus M'$, that is, $S = (M \oplus M') \oplus P = M \oplus (M' \oplus P)$. $M' \oplus P$ is a matching of size $|M'| + 1 = |M| + k + 1$ because P is an augmenting path for M'. By Lem. 2, S forms a collection of paths and cycles that includes at least $|M' \oplus P| - |M| = k + 1$ augmenting paths for M. Since any augmenting path for M has length at least ℓ, the set S thus contains at least $\ell(k + 1)$ edges. Since every vertex in P_i has an incident edge in M' and the endpoints of P are unmatched by M', v must be an internal vertex of P. Its incident edge in M' belongs to both P_i and P because v has an incident edge in M' that belongs to P and an incident in M' that belongs to P_i but M' is a matching and thus contains at most one edge incident to v. Thus, P and P_i share at least one edge. This implies that $|S| \leq |P_1 \cup \cdots \cup P_t \cup P| \leq k\ell + \ell' - 1$. Since $|S| \geq \ell(k + 1)$, this gives $k\ell + \ell' - 1 \geq \ell(k + 1)$, that is, $\ell' \geq \ell + 1$. □

Using Lem. 11, we can prove that

LEMMA 12. *The Hopcroft-Karp algorithm terminates after at most $2\sqrt{n}$ iterations.*

Proof. If the algorithm terminates after fewer than $\sqrt{n}$ iterations, the lemma holds. So assume that the algorithm runs for at least $\sqrt{n}$ iterations, and let M be the matching obtained after the $(\sqrt{n})$th iteration. By Lem. 11, any augmenting path for M has length at least $\sqrt{n}$.

Now let M' be the maximum matching of G produced by the algorithm, and let $\delta = |M'| - |M|$. Since every iteration of the Hopcroft-Karp algorithm finds at least one augmenting path (as long as the current matching is not maximum yet), the algorithm takes at most δ iterations to produce M' from M, that is, the algorithm exits after at most $\sqrt{n} + \delta$ iterations.

By Lem. 2, $M \oplus M'$ is a set of disjoint cycles and paths that includes δ augmenting paths for M. Any such path has length at least $\sqrt{n}$ (because every augmenting path for M does). Since these paths are disjoint and G has n vertices, we must have $\delta \leq \sqrt{n}$. Thus, the algorithm exits after at most $\sqrt{n} + \delta \leq 2\sqrt{n}$ iterations. □

To finish the discussion of the Hopcroft-Karp algorthm, we need to show how to find an inclusion-maximal set of shortest augmenting paths for a given matching M. To do so, we need the concept of the **level graph** L of G with respect to M (see Fig. 4). Consider the alternating forest F of G with respect to M and with roots all the unmatched vertices in U, constructed using the algorithm in Lem. 4. Recall the definitions of the tree T_v, the root r_v of T_v, and the path P_v from r_v to v in T_v, for every vertex $v \in F$ (see § 3). Let the **level** ℓ_v of a vertex $v \in F$ be the number of edges in P_v, and let ℓ be the minimum level of all unmatched vertices in W ($\ell = \infty$ if F contains no unmatched vertex in W). Then L is a directed graph containing exactly those vertices $v \in V(F)$ with $\ell_v \leq \ell$ and containing a directed edge (u, v) if and only if it satisfies the following conditions:

- $u, v \in F$,
- $\{u, v\} \in E(G)$,
- $\ell_v = \ell_u + 1$,
- $\{u, v\} \in M$ if u is odd,
- $\{u, v\} \notin M$ if u is even.

Clearly, this graph can be constructed from F in $O(n + m)$ time. Since F can also be constructed in $O(n + m)$ time (by Lem. 4), constructing L from G and M takes $O(n + m)$ time.

Lem. 16 below shows that the level graph contains all shortest augmenting paths in G with respect to M. To prove this lemma, we need the following two lemmas first.

LEMMA 13. *Let $G = (U, W, E)$ be a bipartite graph, let $M \subseteq E$ be a matching, and let F be a maximal alternating forest of G with respect to M and with roots some set of unmatched vertices $R \subseteq U$. If F is constructed by running alternating BFS from the vertices in R, then every edge $\{u, v\}$ in G with $u, v \in F$ and $\ell_v > \ell_u + 1$ satisfies $u \in W$, $v \in U$, and $\{u, v\} \notin M$.*

Proof. Consider any edge $\{u, v\}$ in G with $u, v \in F$ and $\ell_v > \ell_u + 1$. Then $\{u, v\} \notin M$ because otherwise, u would be v's parent or vice versa, which would imply that $\ell_v \in \{\ell_u + 1, \ell_u - 1\}$,

Now assume for the sake of contradiction that $u \in U$ and $v \in W$. Let z be v's parent in F. Then $\ell_z = \ell_v - 1 > \ell_u$ and both u and z are even. It is easy to verify that alternating BFS enqueues even vertices by increasing values ℓ_v. Thus, it also dequeues them by increacing values ℓ_v. Since $\ell_u < \ell_z$, this implies that u is dequeued before z. Since z is v's parent in F, v is unexplored at the time when z is dequeued.

Thus, v is unexplored also when u is dequeued. When dequeuing u, alternating BFS would explore the edge $\{u, v\}$, would find v to be unexplored, and would make v a child of u, a contradiction. □

Using Lem. 13, we can prove that no alternating path makes "big steps forward" in terms of the levels of its vertices.

LEMMA 14. *Let $G = (U, W, E)$ be a bipartite graph, let $M \subseteq E$ be a matching, let F be a maximal alternating forest of G with respect to M and with roots some set of unmatched vertices $R \subseteq U$, and let $P = \langle v_0, \ldots, v_k \rangle$ be an arbitrary alternating path with $v_0 \in R$. If F is constructed by running alternating BFS from the vertices in R, then $\ell_{v_i} \leq \ell_{v_{i-1}} + 1$, for all $i \in [k]$.*

Proof. For each edge $\{v_{i-1}, v_i\} \in P$, if $v_{i-1} \in W$ and $v_i \in U$, then $\{v_{i-1}, v_i\} \in M$ because $v_0 \in R \subseteq U$ and, since every vertex in R is unmatched, the first edge in P is not in M. Therefore, by Lem. 13, $\ell_{v_i} \leq \ell_{v_{i-1}} + 1$. If $v_{i-1} \in U$ and $v_i \in W$, then, by Lem. 13 once again, $\ell_{v_i} \leq \ell_{v_{i-1}} + 1$. □

Throughout the remainder of this topic, we call an alternating path an alternating path from $R \subseteq V$ to $v \notin R$ if its endpoints are v and a vertex $r \in R$. A *shortest* alternating path from R to v is a path of minimum length among all alternating paths from R to v.

LEMMA 15. *Let $G = (U, W, E)$ be a bipartite graph, let $M \subseteq E$ be a matching, and let F be a maximal alternating forest of G with respect to M and with roots some set of unmatched vertices $R \subseteq U$. If F is constructed by running alternating BFS from the vertices in R, then every path P_v with $v \in F$ is a shortest alternating path from R to v.*

Proof. Consider an arbitrary vertex $v \in F$, and let A_v be a shortest alternating path from R to v. Since P_v is an alternating path of length ℓ_v from R to v, we have $|A_v| \leq \ell_v$. We have to prove that $|A_v| \geq \ell_v$.

Let $A_v = \langle v_0, \ldots, v_k = v \rangle$. By Lem. 14, we have $\ell_{v_i} - \ell_{v_{i-1}} \leq 1$, for all $i \in [k]$. This shows that

$$\ell_v = \ell_{v_k} = \ell_{v_0} + \sum_{i=1}^{k} (\ell_{v_i} - \ell_{v_{i-1}}) \leq \ell_{v_0} + k.$$

Since $v_0 \in R$, we have $\ell_{v_0} = 0$. Thus, $\ell_v \leq k = |A_v|$. □

LEMMA 16. *A path P is a shortest augmenting path with respect to M if and only if it is a directed path in L from an unmatched vertex $u \in U$ to an unmatched vertex $w \in W$.*

Proof. Let R be the set of all unmatched vertices in U. By Lem. 15, for every vertex $v \in F$, P_v is a shortest alternating path from U to v. Since this is true for every vertex in F and, by Lem. 5, every vertex reachable from R via an alternating path is in F, ℓ is the length of a shortest augmenting path for M.

The "if" direction is easy now: If P is a directed path in L from an unmatched vertex $u \in U$ to an unmatched vertex $w \in W$, then it has length ℓ because $\ell_u = 0$, $\ell_w = \ell$, and every edge $(x, y) \in P$ satisfies $\ell_y = \ell_x + 1$, by the definition of L. Thus, P is a shortest augmenting path if it is alternating. This, however, also follows from the definition of L because if (x, y) and (y, z) are two consecutive edges in P, then either x and z are even, and y is odd, or x and z are odd, and y is even. In the former case, $\{x, y\} \notin M$ and $\{y, z\} \in M$. In the latter case, $\{x, y\} \in M$ and $\{y, z\} \notin M$. This shows that the edges in P alternate between being in M and not being in M, that is, P is an alternating path.

For the "only if" direction, consider a shortest augmenting path $P = \langle v_0, \ldots, v_k \rangle$ for M, with $v_0 \in U$ and $v_k \in W$. Then $\ell_{v_k} \geq \ell$. By Lem. 14, $\ell_{v_i} \leq \ell_{v_{i-1}} + 1$, for all $i \in [k]$. If there exists an index $i \in k$ with $\ell_{v_i} \leq \ell_{v_{i-1}}$, then

$$\ell \leq \ell_{v_k} = \ell_{v_0} + \sum_{i=1}^{k} (\ell_{v_i} - \ell_{v_{i-1}}) < \ell_{v_0} + k.$$

Since $v_0 \in R$, we have $\ell_{v_0} = 0$, so $\ell < k = |P|$. Since there exists an unmatched vertex $w \in W$ with $\ell_w = \ell$ and P_w is an augmenting path of lengh ℓ, this shows that P is not a shortest augmenting path in this case. Therefore, if P is a shortest augmenting path, we must have $\ell_{v_i} = \ell_{v_{i-1}} + 1$, for all $i \in [i]$. Since the edges in P alternate between being in M and not being in M, and the first edge in P is not in M, all edges in P are directed edges in L, that is, P is a directed path in L. □

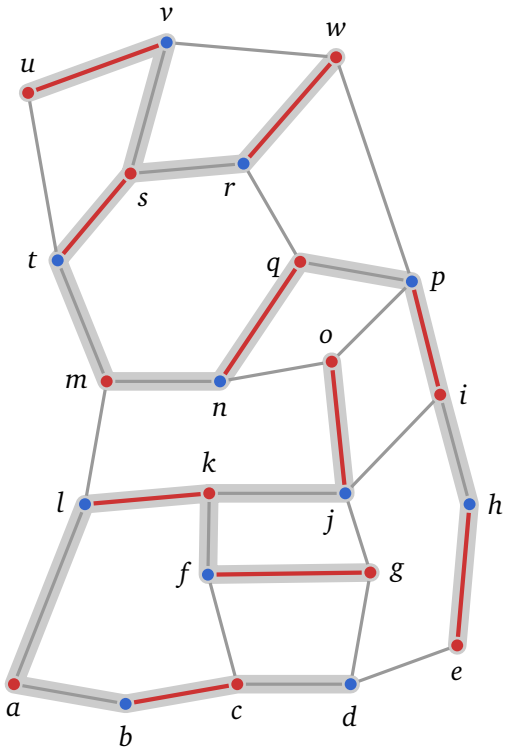
Lem. 16 now allows us to prove that

LEMMA 17. *An inclusion-maximal set of shortest augmenting paths can be found in $O(n+m)$ time.*

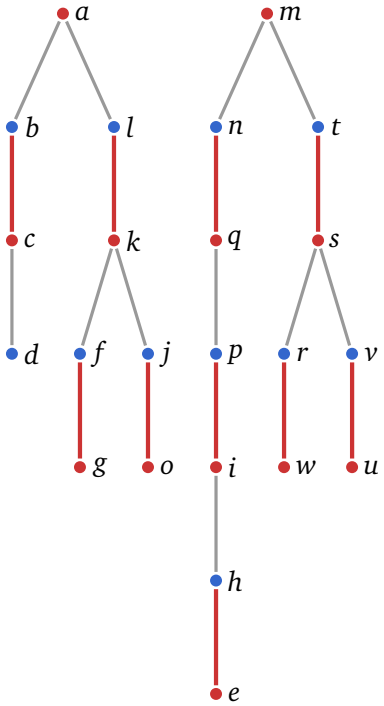
Proof. By Lem. 4 and the discussion before Lem. 16, the level graph L of G with respect to M can be found in $O(n+m)$ time.

By Lem. 16, a path P is a shortest augmenting path for M if and only if it is a directed path in L from an unmatched vertex in U to an unmatched vertex in W. From here on, we call such a path in L a *directed augmenting path* in L. Thus, to find an inclusion-maximal set $\mathcal{P}$ of shortest augmenting paths for M, we need to find an inclusion-maximal set of directed augmenting paths in L. To find such a set of paths, we initialize $\mathcal{P} = \emptyset$ and then iterate over the unmatched vertices in U. For each such vertex u, we run DFS in L from u. This search ends as soon as we have explored all vertices reachable from u or we have found an unmatched vertex in W, whichever comes first. If we find an unmatched vertex $w \in W$, then the path P from u to w explored by the search is a directed augmenting path from u to w in L. Thus, we add this path P to $\mathcal{P}$. No vertex in P can be part of another directed augmenting path in any inclusion-maximal set of directed augmenting paths in L that includes P. Any vertex not in P that is explored by the search cannot reach any unexplored vertex in W and thus is not part of any directed augmenting path in L either. Thus, all vertices explored by the current search can be ignored by subsequent searches for directed paths in L. By removing all vertices explored by the current search and their incident edges before advancing to the next unmatched vertex in U, we ensure that every vertex and every edge in L is explored by at most one search from an unmatched vertex in U, and the total cost of all searches is in $O(m)$.

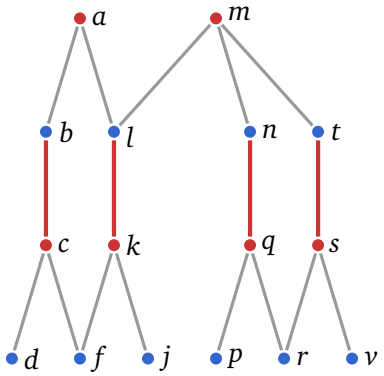
To see that the set $\mathcal{P}$ of paths obtained after the final search in L is an inclusion-maximal set of directed augmenting paths in L, observe first that removing the vertices in some path $P \in \mathcal{P}$ from L immediately after adding P to $\mathcal{P}$ explicitly ensures that the paths in $\mathcal{P}$ are disjoint. To see that no proper superset of $\mathcal{P}$ is a set of disjoint directed augmenting paths in L, consider an arbitrary directed augmenting path P in L. Then P is a path in L from an unmatched vertex $u \in U$ to an unmatched vertex in W. If P is disjoint from every path in $\mathcal{P}$, then none of the vertices in P is removed from L before the search from u starts. Indeed, we argued above that any previous search removes only vertices in augmenting paths found so far and vertices that cannot reach any unmatched vertex in W. Thus, when we start the DFS from u, P is still a directed augmenting path in L that starts at u, that is, the search from u finds such a path (not necessarily P) and adds it to $\mathcal{P}$. This is a contradiction because we assumed that P is disjoint from every path in $\mathcal{P}$, but it shares u with a path in $\mathcal{P}$. □



(a)



(b)



(c)

Figure 4: (a) A bipartite graph G (vertices in U are red, vertices in W are blue) and a matching (red edges). The alternating forest is shaded and, for clarity also drawn separately (b). (c) The level graph L of G corresponding to the alternating forest in (b). Edge directions are not shown; edges in L are directed downward.

Let us summarize the Hopcroft-Karp algorithm: Like the basic maximum matching algorithm from § 4, the Hopcroft-Karp algorithm starts with an arbitrary matching (e.g., the empty matching or a maximal matching) and then grows this matching in iterations. Each iteration constructs an alternating forest using the algorithm from Lem. 4. If this forest contains no unmatched vertex in W, then, by Lem. 6, there is no augmenting path for the current matching and, therefore, by Prop. 3, M is a maximum matching. The algorithm returns this matching in this case. Otherwise, the algorithm constructs the level graph L from F, uses Lem. 17 to find an inclusion-maximal set of shortest augmenting paths $P_1, \dots, P_k$, replaces the current matching M with $M \oplus (P_1 \cup \dots \cup P_k)$, and then starts another iteration. By Lem. 17, the cost per iteration is $O(n+m)$. By Lem. 12, the algorithm terminates after at most $2\sqrt{n}$ iterations. Thus, the algorithm runs in $O\left((n+m)\sqrt{n}\right)$ time. By using the same preprocessing step as in the proof of Thm. 9, we can test whether G has any edges at all and, if not, immediately return the empty matching. Otherwise, we can spend $O(n)$ time to remove all vertices without incident edges. This guarantees that $n \leq 2m$, so the running time including the linear cost of the preprocessing step becomes $O\left(1 + m\sqrt{n}\right) = O\left(m\sqrt{n}\right)$. We have shown the following theorem:

Theorem 18. *The Hopcroft-Karp algorithm computes a maximum matching in a bipartite graph in $O\left(+\sqrt{n}m\right)$ time.*

Exercises

Exercise 1. Provide an example of a graph G that has a maximal matching M and a maximum matching M' such that $|M'| = 2|M|$. Prove that this is the worst possible gap between the size of a maximal matching and the size of a maximum matching. Specifically, prove that, for every graph $G = (V, E)$, if M is a maximal matching of G and M' is a maximum matching of G, then $|M'| \leq 2|M|$.