Maximum Matching in Arbitrary Graphs

CSCI 4113/6101
Instructor: Norbert Zeh
October 31, 2025

In this topic, we return to the problem of computing a maximum matching. So far, however, our focus for all the matching problems we have studied has been on how to solve them on bipartite graphs. We learned that the general strategy for finding a maximum matching is to find augmenting paths. We proved that for such a path P, $M \oplus P$ is a matching that is bigger than M, and that a matching is maximum if it does not have an augmenting path. These results were independent of the graph being bipartite. We also showed that augmenting paths are the main tool for finding a minimum-weight perfect matching, only we need to be careful about the edges we allow to be included in an augmenting path.

The part of the algorithms we have discussed so far that *does* rely on the graph being bipartite is how we find augmenting paths. We proved that, in a bipartite graph, alternating BFS finds an augmenting path if such a path exists, and we mentioned already when discussing maximum matching in bipartite graphs that alternating BFS may fail to find an augmenting path even if one exists if the graph is not bipartite. Thus, our focus in this topic is to discuss how to find augmenting paths in non-bipartite graphs. The algorithm we discuss, called Edmonds's algorithm is far from the fastest algorithm for this problem. It highlights the basic ideas for dealing with non-bipartite graphs though and forms the basis of numerous significantly faster maximum matching algorithms described in the literature.

1 Why is Finding Augmenting Paths Hard?

So what makes finding an augmenting path in non-bipartite graphs hard? Clearly, there is no subdivision of the vertices into two subsets U and W such that every edge has one endpoint in U and the other in W. Thus, we cannot start alternating BFS from the unmatched vertices in U and hope to discover an unmatched vertex in W. This simply does not make sense when the graph isn't bipartite. This is not the real problem though because, as we will see, we can simply run alternating BFS from *all* unmatched vertices, and then, if we are lucky, we can take two vertices v and w connected by an edge not in M and obtain an augmenting path from P_v, P_w, and this edge $\{v, w\}$.

The real problem is that non-bipartite graphs contain odd cycles. Indeed, this is the one property that sets bipartite graphs apart from non-bipartite graphs: all cycles in bipartite graphs have even length. As the example in Fig. 1 shows, it is odd cycles that may prevent alternating BFS from finding an augmenting path even if one exists.

2 Edmonds's Algorithm

Edmonds's algorithm finds a maximum matching in any graph G. Just as the basic matching algorithm for bipartite graphs, it does so by starting with any matching M, repeatedly looking for an augmenting

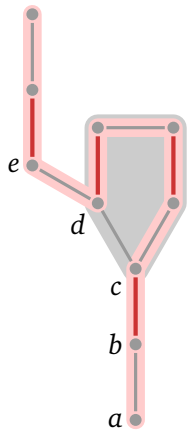

Figure 1: A search that follows the path $\langle a, b, c, d\rangle$ (as a BFS does and a DFS might) turns d into an odd vertex. Since alternating BFS does not explore edges not in M that are incident to odd vertices, the edge $\{d, e\}$ is never explored. Thus, alternating BFS from a fails to find the red augmenting path. The problem is the odd cycle shaded in grey. In a sense, alternating BFS happens to walk around this cycle the wrong way. If we use alternating DFS and we are lucky, we may traverse this cycle in the counter-clockwise direction. This makes d an even vertex and allows us to find the augmenting path. Still, we can't just rely on luck when designing algorithms.

path to grow the current matching, and declaring the current matching to be a maximum matching if no such path can be found.

Since we just argued that alternating BFS may fail to find an augmenting path even if the current matching M is not maximum, Edmonds's algorithm must employ a more complicated, and more costly, strategy to find an augmenting path. This strategy starts by computing a maximal alternating forest F again, this time with all unmatched vertices in G as its roots. The following lemma provides a sufficient (but not necessary) condition for the current matching to be maximum:

Lemma 1. *Let G be a graph, let M be a matching of G, and let F be a maximal alternating forest with respect to M. Then M is a maximum matching if there exists no edge $\{u, v\}$ in G such that u and v are both even vertices in F.*

Proof. We prove the contrapositive. Assume that M is not a maximum matching. Then, as shown in our discussion of maximum matching in bipartite graphs, there must exist an augmenting path $P = \langle v_0, \ldots, v_k\rangle$ for M. We will prove that one of the edges in P must have two even endpoints.

Since all unmatched vertices of G are roots of F, and v_0 and v_k are unmatched, both v_0 and v_k are even. In particular, there exists a largest even index $i \in [k]_0$ such that v_i is even. If $i = k-1$, then both endpoints of the edge $\{v_{v-1}, v_k\}$ are even. So assume that $i < k-1$.

Since i is even, the edge $\{v_{i-1}, v_i\}$ is in M, so the edge $\{v_i, v_{i+1}\}$ is not in M, and the edge $\{v_{i+1}, v_{i+2}\}$ is in M. When alternating BFS dequeues v_i, it explores all edges not in M incident to v_i. Thus, it adds the edge $\{v_i, v_{i+1}\}$ to F unless v_{i+1} is already in F. This shows that $v_{i+1} \in F$. When alternating BFS discovers an endpoint of an edge in M, it adds both endpoints of the edge to M. Thus, $v_{i+1} \in F$ implies that $v_{i+2} \in F$, and one is the parent of the other. Therefore, one of v_{i+1} and v_{i+2} is even, and the other is odd. By the choice of v_i, v_{i+2} is odd. Thus, v_{i+1} is even, that is, the edge $\{v_i, v_{i+1}\}$ has two even endpoints. □

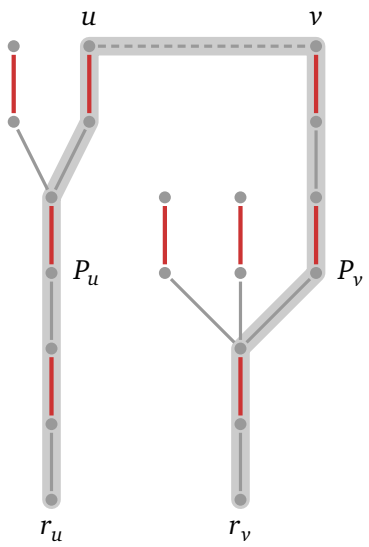

Figure 2: An augmenting path found by Edmonds's algorithm. The edge (u, v) that joins the two even paths from u to r_u and from v to r_v is drawn dashed.

The next thing after computing an alternating forest F that Edmonds's algorithm does in its search for an augmenting path for M is to inspect all edges of G. If it does not find any edge with two even endpoints, then Lem. 1 shows that M is a maximum matching, so the algorithm exits and reports the current matching as a maximum matching. If it does find an edge $\{u, v\}$ with two even endpoints, then it may be able to identify an augmenting path for M, based on the following lemma, illustrated in Fig. 2:

Lemma 2. *Let G be a graph, let M be a matching of G, let F be a maximal alternating forest with respect to M, and let $\{u, v\}$ be an edge with two even endpoints. If $r_u \neq r_v$, then $P_u \circ P_v^r$ is an augmenting path for M, where P_v^r denotes the reversal of P_v.*

Proof. If $r_u \neq r_v$, then P_u and P_v are disjoint, because they belong to different trees T_u and T_v in F. P_u is a path from r_u to u. P_v is a path from r_v to v. Thus, P_v^r is a path from v ro r_v. Since $\{u, v\}$ is an edge in G, the path $P = P_u \circ P_v$ is a path in G. Both P_u and P_v are alternating paths. Since u and v are both even, the last edges in P_u and P_v are both in M. Thus, the edge $\{u, v\}$ is not in M. This makes P an alternating path. Since r_u and r_v, being roots of F, are both unmatched, P is an augmenting path for M. □

After Edmonds's algorithm finds an edge $\{u, v\}$ with two even endpoints, it can check whether $r_u \neq r_v$ by following parent pointers in F from u and v to trace P_u and P_v and find r_u and r_v. This takes linear time. Thus, if $r_u \neq r_v$, Edmonds's algorithm succeeds in finding an augmenting path for M in $O(n + m)$ time.

This leaves the case when the algorithm finds an edge $\{u, v\}$ with two even endpoints but $r_u = r_v$. In this case, Lem. 1 cannot be used to declare the current matching to be a maximum matching, nor can Lem. 2 be used to construct an augmenting path for M. This is when Edmonds's algorithm needs to try harder to decide whether M is a maximum matching and, if not, find an augmenting path for M.

Since $r_u = r_v$, we have $P_u = \langle z_0, \ldots, z_h, u_1, \ldots, u_k = u \rangle$ and $P_v = \langle z_0, \ldots, z_h, v_1, \ldots, v_\ell = v \rangle$, where $\{z_0, \ldots, z_h\}$, $\{u_1, \ldots, u_k\}$, and $\{v_1, \ldots, v_k\}$ are disjoint sets of vertices. Since both $\{z_h, u_1\}$ and $\{z_h, v_1\}$ are edges in F but every odd vertex in F has only its mate as a child, z_h is even, $\{z_{h-1}, z_h\} \in M$, and

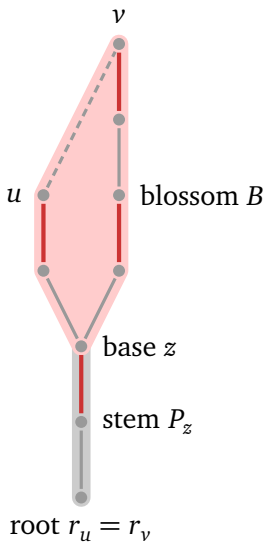

Figure 3: A flower composed of blossom and stem. The edge $\{u, v\}$ that closes the blossom is drawn dashed. The blossom is shaded red. Its stem is shaded grey.

$\{z_h, u_1\}, \{z_h, v_1\} \notin M$. Since u and v are both even, we have $\{u_{k-1}, u_k\}, \{v_{\ell-1}, v_\ell\} \in M$. This makes $B = \langle z_h, u_1, \ldots, u_k, v_\ell, \ldots, v_1, z_h \rangle$ and odd-length cycle whose edges alternate between being in M and not being in M, except that the two edges incident to z_h are not in M. Moreover, $S = \langle z_0, \ldots, z_h \rangle$ is an even alternating path from z_0, which is unmatched, to z_h, and this path shares only z_h with B. We call $B \cup S$ a **flower** with **blossom** B and **stem** S, as one can squint at $B \cup S$ and see a tulip with blossom B and stem S. We call z_h the **base** of B. This is illustrated in Fig. 3.

The following lemma demonstrates the significance of blossoms for finding an augmenting path. In this lemma and in the remainder of this topic, we use the notation G/B to denote the graph obtained from G by contracting the blossom B. This graph is obtained from G by deleting all vertices in B and their incident edges and adding a new vertex v_B. There exists an edge $\{u, v_B\}$ in G/B if and only if there exists an edge $\{u, v\}$ in G with $u \notin B$ and $v \in B$. Similarly, let M/B be the edge set obtained from M by deleting all those edges from M that have both endpoints in B. Note that only the base z of B may have an incident edge $\{u, z\} \in M$ with $u \notin B$. If there exists such an edge in M, then this edge is replaced by the edge $\{u, v_B\}$ in M/B. This is illustrated in Fig. 4.

Lemma 3. *Let G be a graph, let M be a matching of G, and let $B \cup S$ be a flower in G with respect to M, with blossom B and stem S. Then M is a maximum matching in G if and only if M/B is a maximum matching in G/B. Moreover, given an augmenting path P' for M/B in G/B, an augmenting path P for M in G can be found in $O(n)$ time.*

Proof. First assume that there exists an augmenting path P' for M/B in G/B. If P' does not contain v_B, then P' is also an augmenting path for M in G, as all edges of P' are edges of G in this case, and each edge of P' is in M/B if and only if it is in M. This is illustrated in Fig. 5a.

The case when If P' contains v_B is illustrated in Fig. 5b. Let u and v be the endpoints of P'. Since P' has odd length, we can assume w.l.o.g., that the subpath P'_u of P' from u to v_B has even length, and the subpath P'_v of P' from v_B to v has odd length. Since u and v are unmatched, this implies that either

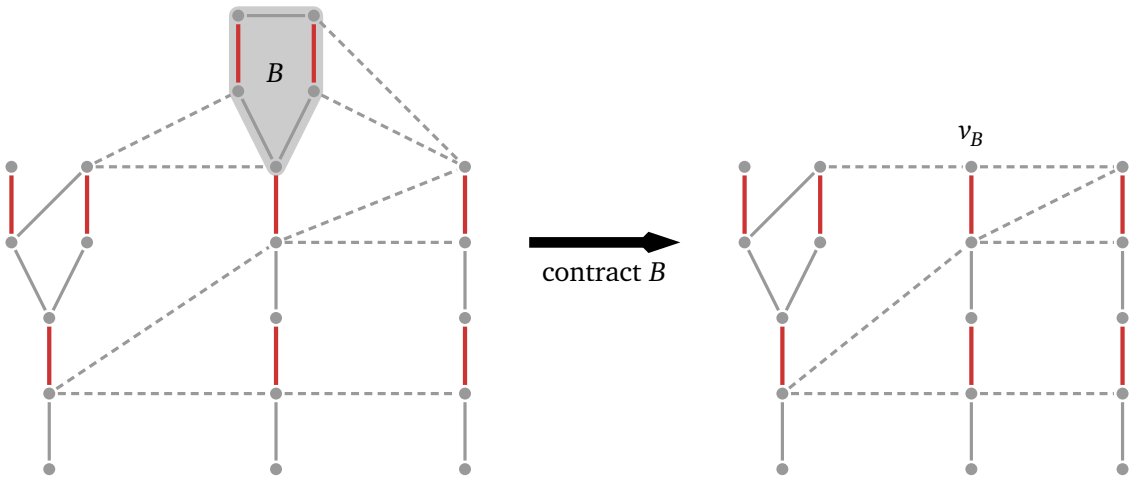

Figure 4: Contracting the blossom B creates a new vertex v_B. Every edge with exactly one endpoint in B is now incident to v_B. Duplicate edges are removed. Each edge in G/B is matched if its corresponding edge in G is matched. Edges in F and in flowers are drawn solid. All other edges are drawn dashed.

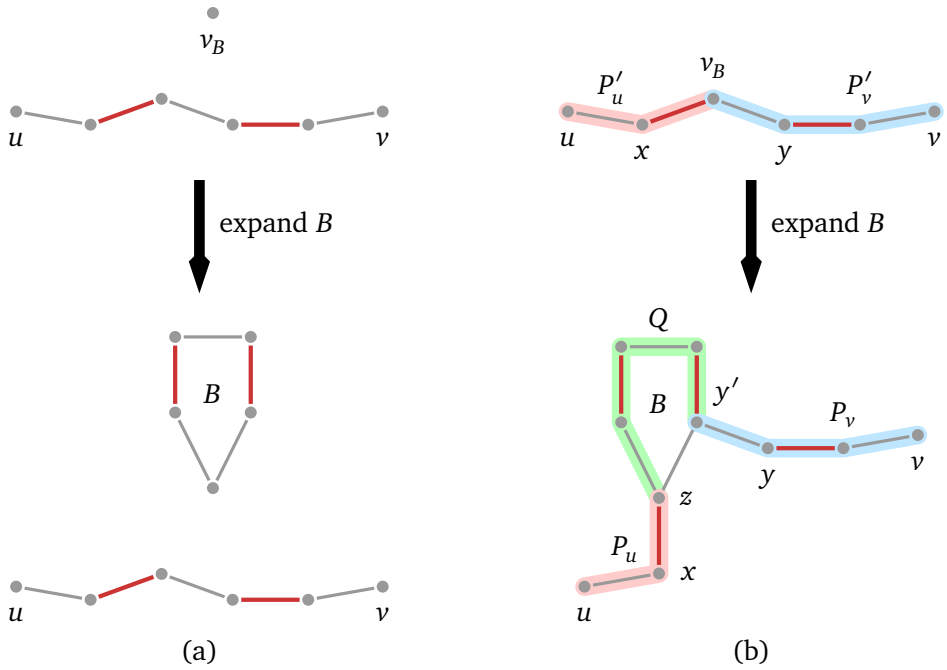

Figure 5: (a) The augmenting path remains unchanged after expanding the blossom B if v_B is not on the path. (b) Construction of an augmenting path in G from the subpaths P'_u (red) and P'_v (blue) of P' before and after v_B and an even path Q (green) from the base z of the blossom to a neighbour $y' \in B$ of y.

$u = v_B$ (P'_u is empty) or the last edge in P'_u is in M/B.

If $u = v_B$, S must be empty (contain no edges) because otherwise, the last edge $\{x, z\} \in S$ would be in M, that is, M/B would contain the edge $\{x, v_B\}$. Since S is empty, the base z of B is unmatched. In this case, we define $P_u = \langle z \rangle$, which makes P_u an even alternating path from an unmatched vertex to z in G.

If P'_u is non-empty, then let P_u be the path obtained from P'_u by replacing v_B with z. Let $\{x, v_b\}$ be the last edge in P'_u, which is in M because u is unmatched and P'_u has even length. Thus, $\{x, z\} \in M$. Since all other edges of P_u are edges of both G/B and G and are in M if and only if they are in M/B, this implies that P_u is an even alternating path in G from an unmatched vertex to z also in this case.

Now let $\{v_B, y\}$ be the first edge of P'_v. Since this is an edge of G/B, G contains an edge $\{y', y\}$, for some $y' \in B$. Since P'_v has odd length and v is unmatched, we have $\{v_B, y\} \notin M/B$. Thus, $\{y', y\} \notin M$. This makes the path P_v obtained from P'_v by replacing v_B with y' an odd alternating path from y' to v, because all vertics of P'_v, except v_B, are vertices of G, and every edge of P'_v, except $\{v_B, y\}$, is an edge of G and belongs to M/B if and only if it belongs to M.

Now observe that there exists an even-length alternating path $Q \subseteq B$ from z to y' because B has odd length and is alternating, with z the only vertex with two incident edges in B that are not in M. The first edge in this path Q is not in M. The last edge is.

Finally, note that all vertices in Q are in B, z is the only vertex in P_u that belongs to B, y' is the only vertex in P_v that belongs to B, and P_u and P_v share only vertices in B because P'_u and P'_v share only v_B. Thus, $P = P_u \circ Q \circ P_v$ is a path in G. Its endpoints are the unmatched endpoints of P_u and P_v. The edges in P alternate between being in M and not being in M because P_u and P'' are alternating paths of even length, and P_v is an alternating path of odd length. This makes P an augmenting path for M.

This proves that M is not maximum if M/B is not maximum. Moreover, the construction of an augmenting path P for M from an augmenting path P' for M/B can clearly be implemented in $O(n)$ time, given that P' and B both have size $O(n)$.

To finish the proof, assume that M is not a maximum matching. We need to prove that M/B is not a maximum matching. Observe that $M' = M \oplus S$ is a matching of G of size $|M'| = |M|$ because S is an alternating path of even length and with an unmatched endpoint. See Fig. 6. Thus, since M is not maximum, neither is M'. Also, since M and M' contain the same edges from B, M'/B is a matching of G/B of size $|M'/B| = |M/B|$. Thus, to prove that M/B is not a maximum matching of G/B, it suffices to prove that M'/B is not a maximum matching of G/B.[1]

If M' is not maximum, then there exists an augmenting path P for M'. If this path does not contain any vertex in B, then $P' = P$ is an augmenting path for M'/B, that is, M'/B is not maximum either. If P contains a vertex from B (see Fig. 7), then note that P has two unmatched endpoints but B contains only one unmatched vertex, z. Thus, one of the endpoints of P is not in B. Call this endpoint u. Let x be the vertex closest to u that belongs to B, let P' be the subpath of P from u to x, and let y be the neighbour of x in P'. Then the path P'' obtained by replacing x with v_B in P' is a path from u to v_B in G/B. Since P' is an alternating path with respect to M', P'' is an alternating path with respect to M'/B. Since $u \notin B$ and u is unmatched by M', it as also unmatched by M'/B. Since z is unmatched by M', the only edges in M' incident to vertices in B are edges of B. Thus, v_B is unmatched by M'/B. This makes

[1]The reason why we reason about M' and M'/B instead of M and M/B is that B is a blossom also with respect to M', but M' leaves the base z of B unmatched, that is, the flower with blossom B with respect to M' has an empty stem. This ensures that no augmenting path for M' can contain any vertices of this stem other than z, something that is not necessarily true for M unless S is empty, in which case $M' = M$.

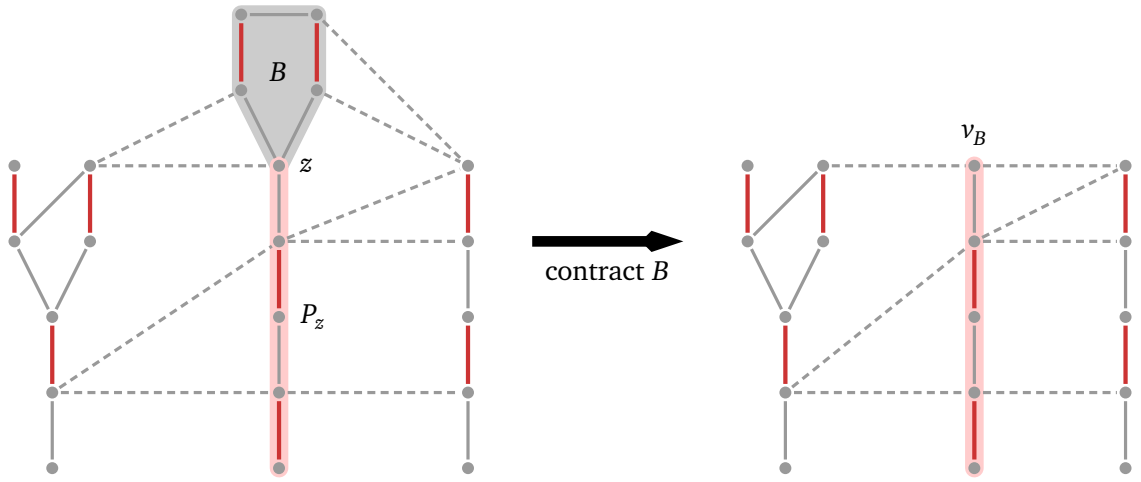

Figure 6: The matching $M' = M \oplus P_z$ obtained from the matching M in Fig. 4. The blossom B is shaded grey. Its stem P_z is shaded pink. The matching on the right is the matching M'/B obtained after contracting the blossom B. It has the same size as the matching M/B in Fig. 4.

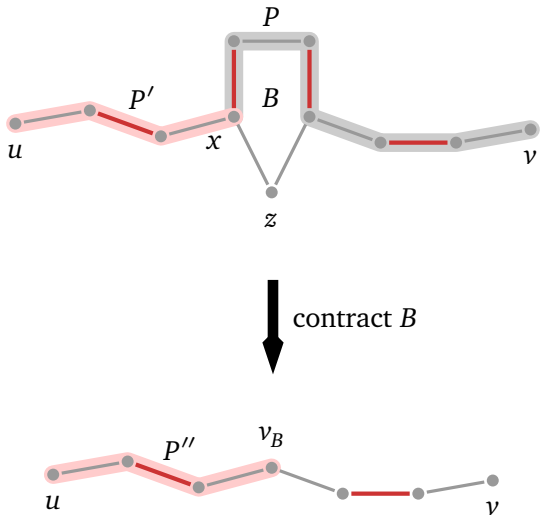

Figure 7: Contraction of a blossom in an augmenting path P in the proof of Lem. 3. The path P is shaded. The subpath P' and its corresponding path P'' in G/B are shaded pink.

P'' an alternating path with respect to M' with two unmatched endpoints, that is, P'' is an augmenting path for M'/B, and M'/B is not maximum. □

We are ready to discuss Edmond's algorithm in its entirety. It starts with an arbitrary matching M, just like any other maximum matching algorithm. Then it calls a procedure AUGMENT(G, M). This procedure either returns an augmenting path P for M or reports that M is a maximum matching. In the former case, the algorithm replaces M with $M \oplus P$ and then continues trying to grow this matching by calling AUGMENT(G, M) again. This continues until AUGMENT(G, M) reports that M is a maximum matching. When this happens, the algorithm exits and outputs M. Since the initial matching has size at least 0 and any matching of G has at most $n/2$ edges, the algorithm terminates after calling AUGMENT(G, M) at

most $n/2+1$ times. Next, we show that AUGMENT(G,M) correctly reports an augmenting path for M or reports that M is maximum, and that it takes $O(nm)$ time to do so. Thus, Edmonds's algorithm finds a maximum matching in $O\left(n^2m\right)$ time.

Consider an invocation AUGMENT(G,M). This invocation starts by computing an alternating forest F with respect to M and with all unmatched vertices as its roots. If there is no edge $\{u,v\}$ with two even endpoints, then Lem. 1 shows that M is maximum, so the algorithm exits and reports M as a maximum matching. If there is such an edge, then the algorithm follows parent pointers from u and v to find P_u and P_v. If $r_u \neq r_v$, then Lem. 2 shows that $P = P_u \circ P_v^r$ is an augmenting path for M. AUGMENT(G,M) constructs this path from P_u and P_v and returns it. If $r_u = r_v$, then AUGMENT(G,M) computes the blossom B of $P_u \cup P_v \cup \{\{u,v\}\}$ and constructs G/B and M/B. It recursively calls AUGMENT$(G/B, M/B)$. If this recursive call reports that M/B is a maximum matching for G/B, then Lem. 3 shows that M is a maximum matching for G, so AUGMENT(G,M) exits and reports that M is a maximum matching. If AUGMENT$(G/B, M/B)$ returns an augmenting path P' for M/B, then AUGMENT(G,M) uses the second part of Lem. 2 to construct an augmenting path P for M from P', and returns this path P. This is illustrated in Figs. 8 and 9.

The correctness of this procedure follows immediately from Lems. 1–3. To prove that Edmonds's algorithm runs in $O\left(n^2m\right)$ time, we need to prove that AUGMENT runs in $O(nm)$ time. Excluding the recursive call AUGMENT$(G/B, M/B)$ possibly made by the invocation AUGMENT(G,M), the invocation AUGMENT(G,M) constructs an alternating forest F, inspects all edges to check whether one of them has two even endpoints, traces the paths P_u and P_v in F, and, if $r_u = r_v$, constructs and contracts the blossom B to obtain G/B and M/B. It is easily verified that, if we represent graphs using an adjacency list representation, all of these steps can be implemented in $O(n+m)$ time. Since a blossom is a cycle of odd length, it has length at least 3. Contracting the blossom replaces the vertices in the blossom with a single vertex, so G/B has at least 2 vertices fewer than G. Thus, if $T(n,m)$ is the running time of AUGMENT(G,M) on a graph G with n vertices and m edges, then we have the following recurrence:

$$T(n,m) \leq O(n+m) + T(n-2,m)$$

A graph with at most 2 vertices does not contain any odd-length cycles. Thus, we have $T(n,m) = O(1)$, for $n \leq 2$. This means that the recursion tree of this recurrence has size at most $n/2$, with the cost of each invocation bounded by $O(n+m)$. Thus, AUGMENT(G,M) takes $O\left(n^2+mn\right)$ time. We can reduce this running time to $O(nm)$ by performing a one-time preprocessing step at the beginning of the algorithm that removes all vertices without incident edges. This takes $O(n+m)$ time and ensures that AUGMENT(G,M) takes $O(nm)$ time because the input graph now has at most twice as many vertices as edges. The total running time of Edmond's algorithm thus is $O\left(n+n^2m\right)$, which is bounded by $O\left(n^2m\right)$ if the input graph has at least one edge. If the graph has no edges at all, it takes constant time to test this and output the empty matching. Thus, we obtain the following theorem:

THEOREM 4. *A maximum-cardinality matching in an arbitrary graph G can be found in $O\left(n^2m\right)$ time.*

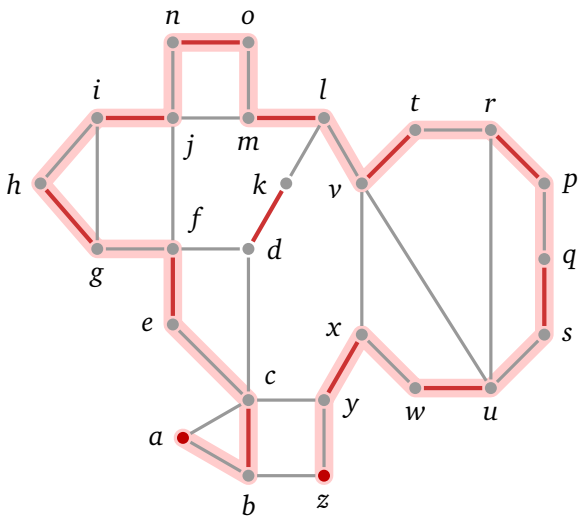

Figure 8: A matching in a graph that leaves the two red vertices unmatched. This matching is not a maximum matching, as the shaded augmenting path shows. Figure 9 shows how Edmonds's algorithm finds this augmenting path.

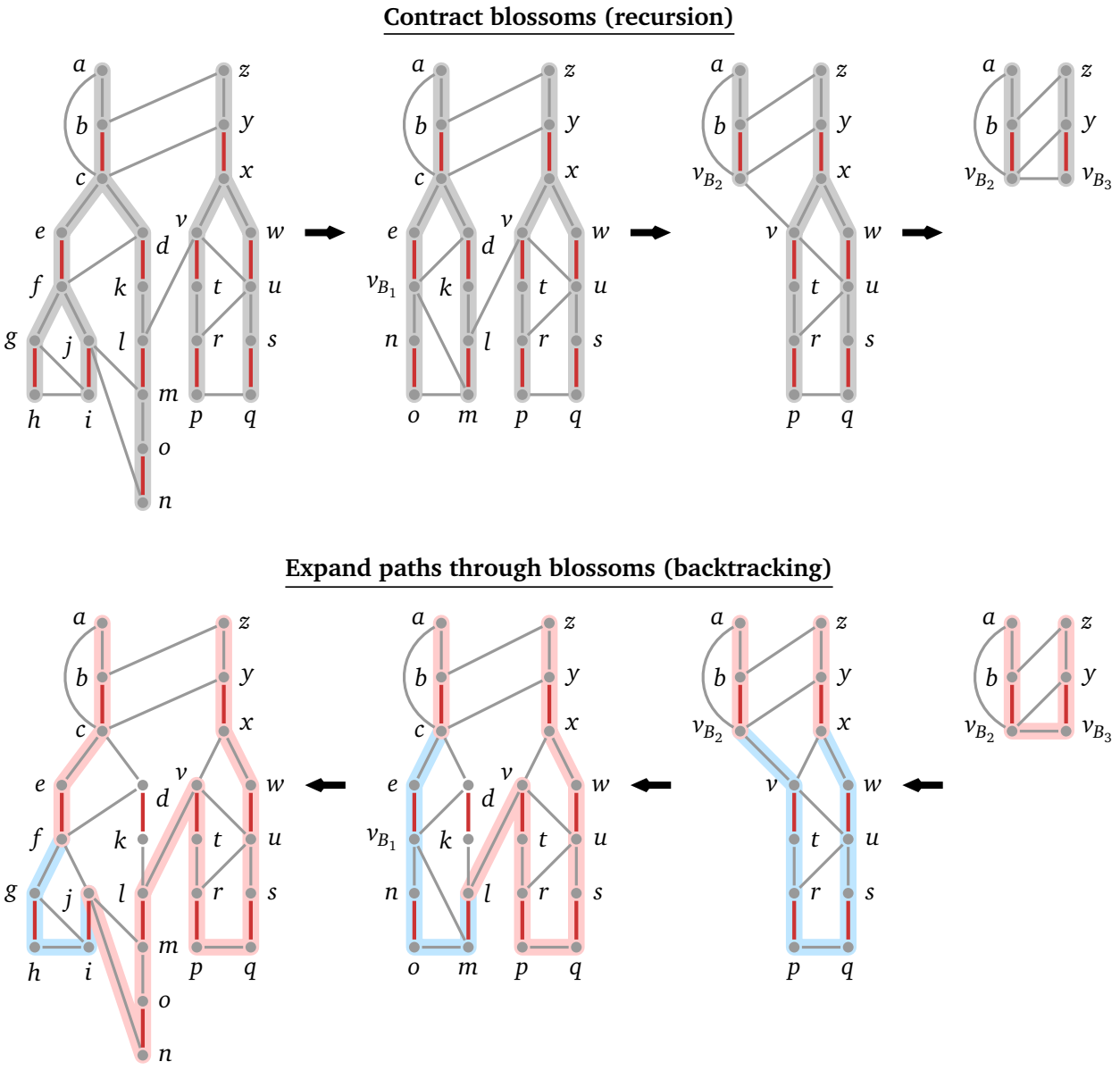

Figure 9: Illustration of Edmonds's algorithm. The top row shows the alternating forests (shaded) for G, G/B_1, $G/B_1/B_2$, and $G/B_1/B_2/B_3$, where $B_1 = \{f, g, h, i, j\}$ is a blossom in G, $B_2 = \{c, e, v_{B_1}, n, o, m, l, k, d\}$ is a blossom in G/B_1, and $B_3 = \{x, v, t, r, p, q, s, u, w\}$ is a blossom in $G/B_1/B_2$. The bottom row shows how the algorithm finds an alternating path $\langle a, b, v_{B_2}, v_{B_3}, y, z\rangle$ in $G/B_1/B_2/B_3$, then expands v_{B_3} to the path $\langle v_{B_2}, v, t, r, p, q, s, u, w, x\rangle$ through B_3 in $G/B_1/B_2$, expands v_{B_2} to the path $\langle c, e, v_{B_1}, n, o, m, l\rangle$ through B_2 in G/B_1, and finally expands v_{B_1} to the path $\langle f, g, h, i, j\rangle$ through B_1 in G.