Maximum Matching in General Graphs

We have already discussed the Maximum Matching problem for bipartite graphs, by reducing it to the Maximum Flow problem. Here, we discuss the maximum matching problem on general graphs. In particular, we will discuss the **Edmonds' Blossom Algorithm** which finds the maximum matching in a general graph in polynomial time.

The Problem

Recall, that a **matching** in a graph G is a subset of edges, such that no two edges share any common vertex.

The **maximum matching problem**: Given a general graph G, find a maximum-cardinality matching M in G.

A **perfect matching** in a graph G is a matching which covers every vertex of the graph. If G has n nodes, a perfect matching in G will have $\frac{n}{2}$ edges (if it exists). If a perfect matching exists in G, then it is a maximum matching of G.

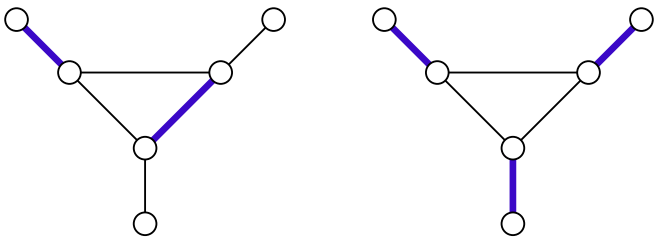

Figure 1: Two matchings M_1 and M_2 in a graph G. M_2 is a perfect matching. (Note that G is not bipartite.)

Consider the graph G in Figure 1. Observe that G is not bipartite, since it has a cycle of odd length (i.e. a 3-cycle). Due to this, we cannot use simple algorithms that find matching in bipartite graphs (that we have discussed earlier). The presence of odd cycles is what makes it not very easy to find the maximum matching. To find a maximum matching in such a graph, we will need to specially handle these odd cycles, and do so in polynomial time.

Goal: Given a graph G, in polynomial time, find a maximum matching M in G. As a simpler version, we will try to answer this question: **Does G have a perfect matching?**

Improvement

In our algorithm, we will start with the empty matching, and iteratively improve M by finding a matching of larger size, till we find a maximum matching.

1. How to know if M is a maximum matching?
2. If M is not a maximum matching, how to find a matching M' of larger value ($|M'| > |M|$)?

We define some terminologies and state a theorem for answering these questions:

Let M be any matching in graph $G = (V, E)$.

A vertex $v \in V$ is said to be M**-matched**, if there exists $u \in V$, such that $(u, v) \in M$ (i.e. v exists in the matching M). Otherwise, v is said to be M**-exposed**.

The **defect** of a matching M is the number of unmatched vertices in M, i.e. the number of vertices that are M-exposed. A maximum matching has minimum defect, and a perfect matching has zero defect. (In Figure 1, M_1 has a defect of 2.)

Define a M**-augmenting path** as follows: Any path $p = \langle v_0, v_1, ..., v_k \rangle$ $(k \geq 1)$ is an M-augmenting path, if
1. v_0 and v_k are M-exposed.
2. Alternating edges in the path are in the matching, i.e. $(v_1, v_2), (v_3, v_4), ..., (v_{k-2}, v_{k-1}) \in M$.

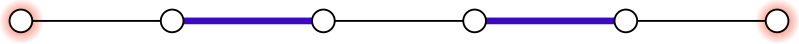

Figure 2: An M-augmenting path p (in a larger graph G). Glowing vertices are M-exposed.

Observe, that given an M-augmenting path, we can **augment along path** p: exchange the matched edges to be unmatched and the unmatched edges to be matched in the path p. Doing so necessarily gives us a larger matching M' (such that $|M'| = |M| + 1$). (Mathematically, we may write $M' = M \Delta p$ where p denotes the set of edges in path p.)

Figure 3: Matching M' obtained by augmenting along M-augmenting path p, M' has one more matched edge as compared to M.

For example, in Figure 1, matching M_1 is not a maximum matching. There exists a M_1-augmenting path (the path from the top-right corner vertex to the bottom vertex, passing through the right side of the triangle which is matched). If we augment along this path, we get M_2 which is a matching of larger size (and also happens to be the maximum matching.)

Clearly, if M is a maximum matching, there will not exist any M-augmenting paths. (Otherwise we can find a matching of larger size). It turns out, that this condition is sufficient as well.

> **Theorem 1 (Berge's Theorem)**
>
> A matching M is a maximum cardinality matching in a graph G if and only if there is no M-augmenting path in G.

Proof: We need to show that, if M is not a matching of maximum cardinality, then there must exist a M-augmenting path. Suppose M is a non-maximum matching in G. Let M' be another matching, with $|M'| > |M|$.

Consider the subgraph $G_{M \Delta M'} = (V, E = M \Delta M')$. This subgraph only has edges in M or in M' but not both. For a given vertex $v \in V$, it can only be a part of at most one edge in M and at most one edge in M', thus its degree must be 0, 1 or 2. So, $G_{M \Delta M'}$ will be a collection of isolated vertices, and vertex-disjoint cycles and paths. In any cycle/path, edges must alternate being in M and in M'.

In the cycles, the number of M' edges and the number of M edges are equal. In a path, either there is one more M' edge, or one more M edge. There must necessarily be at least one path out of these, in which the number of M' edges is more (why?). This is a M-augmenting path:
1. The end vertices are not connected to any M edges, so they are M-exposed.

2. Every alternating edge in this path is M.

Thus, there must exist a M-augmenting path, if M is not a maximum matching. ■

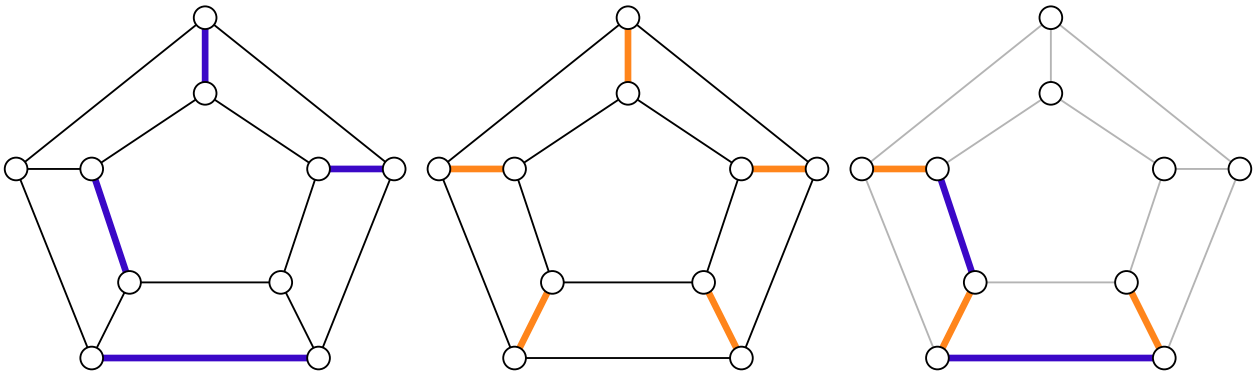

Figure 4: (a) A matching M in G. (b) A matching M', $|M'| > |M|$. (c) The graph $G_{M\Delta M'}$. We see that there is a M-augmenting path from (c).

This answers our first question. From an algorithmic perspective, we can start with the empty matching, and keep improving it until we get the maximum matching. The only missing step is that we need a procedure to **find an M-augmenting path** for some matching M in G.

For finding a maximum matching:
- $M \leftarrow \{\}$
- **while** M is not a maximum matching
  - Find an M-augmenting path p
  - $M \leftarrow$ augment M along p

Bounding the size of the maximum matching

We will try to characterise, when does a perfect matching exist, or what is the size of the maximum matching. Let $G = (V, E)$ be a graph. Let $R \subseteq V$ be an arbitrary set of vertices in G. Let $G \setminus R$ denote the induced subgraph of G with the vertex set $V \setminus R$.

Let $o(G \setminus R)$ denote the number of components of odd cardinality in $G \setminus R$. Any such component cannot be perfectly matched within itself, i.e. in any perfect matching M of G, at least one vertex v in such an odd component must be matched with some vertex outside this component. This other vertex must be in R (why?).

So, for each odd component, at least one corresponding vertex must exist in R, for the possibility of a perfect matching to exist. Thus, we can state:

- If $|o(G \setminus R)| > |R|$, then no perfect matching exists.

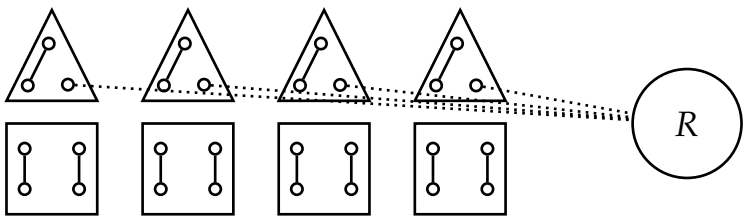

Figure 5: Triangles depict odd components and squares depict even components of $G \setminus R$. For a perfect matching to exist, at least one vertex in each odd component must be matched with a vertex in R.

We can also bound the defect of a matching. For some $R \subseteq V$, at least $o(G \setminus R) - |R|$ vertices must be unmatched. So,

$$\text{min defect} \geq \max_{R \subseteq V} o(G \setminus R) - |R|$$

If d vertices in a matching are unmatched, then $n - d$ vertices are matched, which means the size of the matching is $\frac{1}{2}(n - d)$. So, we have

$$\text{size of a max matching} \leq \min_{R \subseteq V} \frac{1}{2}(n + |R| - |o(G \setminus R)|) \tag{1}$$

We will use this fact extensively while describing the algorithm. The celebrated Tutte-Berge theorem states that the above inequality is actually tight.

Theorem 2 (Tutte Berge Theorem)

For any maximum matching M of an undirected graph G, we have

$$|M| = \min_{R \subseteq V} \frac{1}{2}(|V| + |R| - o(G \setminus R))$$

We will arrive at a proof of this theorem by describing the blossom algorithm.

Alternating Trees

An M**-alternating tree** is a subgraph of G with the following properties: the vertices of T will be alternately colored blue and red.

- The root is colored blue.
- All leaves are colored blue.
- Every red vertex r has exactly one child s, and the edge (r, s) is in the matching.

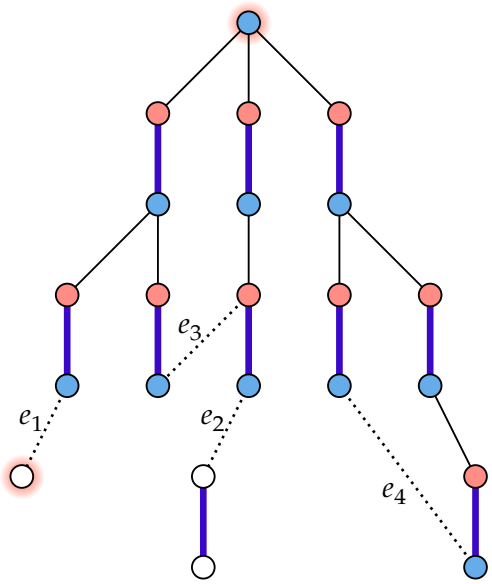

Figure 6: An M alternating tree, which is a subgraph of G. There maybe other vertices and edges (matched or unmatched) present in G, not shown here. Dotted edges are not in the tree.

For a M-alternating tree T, let us define some terms:

- $V(T) \subseteq V$: The set of vertices in the tree.
- $R(T)$: The set of red vertices in the tree; $B(T)$: the set of blue vertices in the tree. $R(T) \cup B(T) = V(T) \subseteq V$.
- Note that, by the structure of the tree, we always have: $|B(T)| = |R(T)| + 1$.
- For some vertex $v \in V$, $N(v)$ denotes the neighbour set of vertex v, i.e. the set of vertices having an edge with v. (This considers all edges in the graph G, not just in the subgraph T.)

To proceed to find a M-augmenting path, suppose we have some M-alternating tree T. We will try to expand T. There are the following cases:

1. **Case 1:** There exists some neighbor of a blue node which is out of the tree: For some node $b \in B(T)$, there exists $c \in N(b)$ such that $c \notin V(T)$. Consider any such (b, c) edge.
   a) **Case 1a:** c is M-exposed. This means, that the edge (b, c) is similar to the edge e_1 in Figure 6. This means, we have found an M-augmenting path, which consists of the path from the root node to vertex b in T, and the edge (b, c). Clearly, both end points are M-exposed, and the path is alternating by virtue of being a tree path. So, we are done here (remember, our goal was to just find a M-augmenting path).
   b) **Case 1b:** c is M-matched. Let (c, d) be the edge which is in the matching M. So, the (b, c) edge is similar to e_2 in Figure 6. Note that the vertex d cannot be in the tree T (why?). In this case, we can expand the tree to include the nodes c and d. We color c red, and d blue. This maintains all our invariants for the tree T.
2. **Case 2:** All neighbors of blue nodes are in the tree: For all $b \in B(T)$, for all $c \in N(B)$, $c \in V(T)$. We have already explored all edges from blue nodes. We call such a tree T **stuck** (there are no more blue node edges to outside-the-tree nodes).
   a) **Case 2a:** All neighbors of blue nodes are red nodes: For all $b \in B(T)$, $N(b) \subseteq R(T)$. This means, all edges connected to blue nodes are tree edges, or non tree edges that connect to other red nodes in the tree, like edge e_3 in Figure 6. We call this tree T **frustrated**. Observe, that this implies that the subgraph T is locally 2-colorable (bipartite). In this case, we can claim that **no perfect matching exists**, i.e. the current subgraph is maximally matched.

      We can show this from Equation 1: Let $R = R(T)$. If we remove the set of vertices R from G, then all blue vertices become isolated (there are no blue-blue edges in G). So, all these isolated blue vertices form odd-sized components in $G \setminus R$. So, the $o(G \setminus R) \geq |B(T)|$. Then, the minimum defect is $\geq o(G \setminus R) - |R| \geq |B(T)| - |R| = 1$. As the defect is at least 1, there cannot exist a perfect matching.
   b) **Case 2b:** The only other case, is that there exists a blue-blue edge: There exists $b, c \in V$, $c \in N(b)$. Edge e_4 in Figure 6 is an example of such an edge.

      Observe, that this implies we have found a cycle of odd length in T. We call this cycle a **blossom**.

      In this case, we **contract the blossom** C to obtain the **contracted graph** G/C. We remove all nodes in C and replace it with a pseudovertex v_C. We retain all edges, while not keeping any parallel edges, i.e. add an edge (v_C, u) if there exists an edge $(c, u), c \in C$ in G. We color this pseudovertex blue.

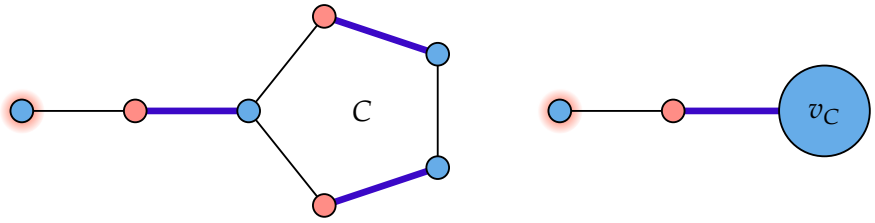

Figure 7: (a) A *blossom* C (b) The graph G/C after contracting C.

In the contracted graph G/C, define the **contracted matching** M' as the matching of G/C corresponding to M. Let T' be the **contracted tree** corresponding to T. Then we continue to expand the tree T' in G/C.

Let's just recap what we have till now:

- We wish to find the maximum matching in a graph G, or determine whether G has a perfect matching.
- We have the notion of M-augmenting paths, so we can start with the empty matching and keep augmenting by M-augmenting paths to reach the maximum matching. So our goal reduces to **just finding an M-augmenting path in a matching M.**
- To find an M-augmenting path, we traverse G while building an M-alternating tree. While the tree is not stuck, we can explore blue-node edges which go outside the tree, if they go to an unmatched (M-exposed) vertex, we have found a M-augmenting path and we are done. If they go to a matched vertex, we simply grow the tree and continue our search.
- If the tree is stuck, then there are two possibilities, one is that all blue edges have been explored and no odd cycle has been found. Then, we claim that the root node cannot be matched, so no perfect matching can exist. The other possibility is that we find a **blossom** C (an odd cycle), which we contract to v_C, and continue our search in the resultant graph.

We need to now justify this **contraction** step, and the validity of the other cases in the resultant graph as well. We do so via a series of Lemmas:

Lemma 1: After a blossom contraction, M' is a matching of G' and T' is a M'-alternating tree.

Proof: M' is a matching in G' as there are no two edges sharing the same vertex. The pseudo vertex can be matched to at most one other vertex (which must be in the tree). T' follows the properties of an M-alternating tree, as the root is blue (either the old root or the psuedovertex); the leaves are blue, and the pseudovertex must be at an even distance from the root (since the LCA of the blue-blue edge that was found in T must have been blue), so it is colored blue. Clearly any red vertex in T' will have exactly one blue child, as it did in T.

Lemma 2: Let T be the tree after a series of blossom contractions. Every pseudovertex v_C represents a connected subgraph containing odd number of original vertices in V.

Proof: Use induction. The base case is that C only consists of simple original vertices. Then the size of C is odd, so v_C corresponds to an odd number of vertices in V. Inductively, some C may consist of psuedovertices as well as normal vertices. Each of these represents an odd number of vertices in V. Since C is an odd cycle, and the sum of an odd number of odd numbers must be odd, so v_C corresponds to an odd number of original vertices in V.

Lemma 3: (**Lifting the augmenting path**) An M-augmenting path exists in G iff an M'-augmenting path exists in G/C.

Proof: Suppose we find a M'-augmenting path p. If p does not pass through any pseudo vertex, then it is a M-augmenting path. Otherwise, suppose it passes through v_C. We replace v_C with some path $u \to \ldots \to v$, internal to the blossom C, where u and v are the appropriate vertices which join with the rest of the path, and we take the even length subpath from u to v in the blossom C. This ensures, the resulting path is an M-augmenting path.

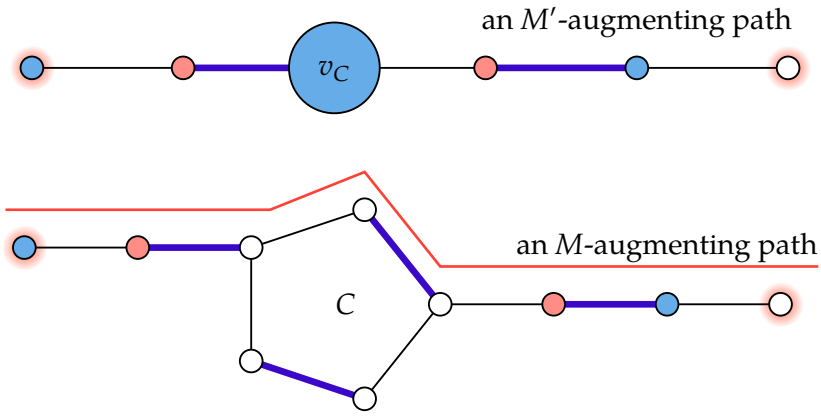

Figure 8: Lifting an M'-augmenting path to an M-augmenting path.

Lemma 4: After a series of blossom contractions, if an alternating tree T becomes frustrated (Case 2a), it contains exactly one unmatchable vertex, and G does not have a perfect matching.

We have already shown this for the case that T does not contain any pseudo vertices. We just extend the proof: Take $R = R(T)$. Applying Equation 1 on G, the graph $G \setminus R$ will have all blue vertices isolated. Each blue vertex in G represents either a single original vertex, or an odd number of original vertices (Lemma 2). Thus, each blue vertex forms a odd component in $G \setminus R$. The minimum defect is therefore $o(G \setminus R) - |R| \geq |B| - |R| = 1$. Since the minimum defect is at least 1, G does not have any perfect matching. ■

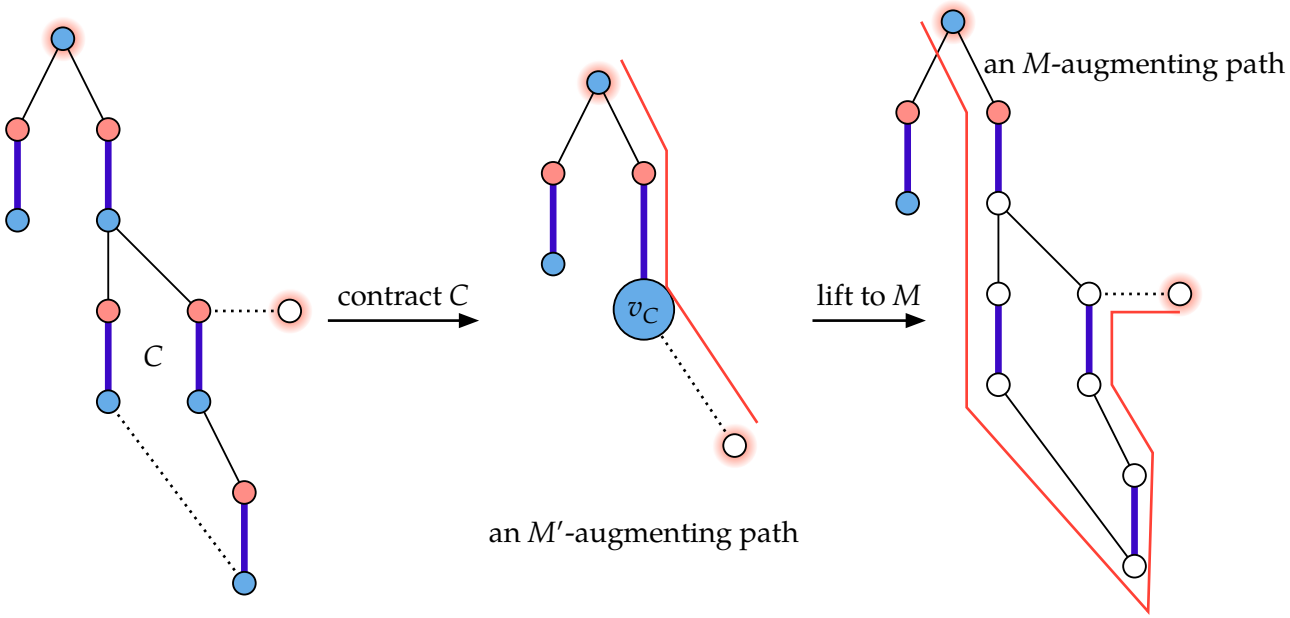

Figure 9: (a) We find a blossom C and contract it. (b) In the contracted graph we find an M' -augmenting path. (c) We lift it to get an M-augmenting path.

Thus, we have the following:

- If we get to case 1a, the augmenting path found can be lifted to an M augmenting path, and we are done.
- If we get to case 1b, we simply grow the tree by adding two new vertices.
- If we get to case 2a, then we report that no perfect matching exists (Lemma 4).
- If we get to case 2b, we found a blossom C, then we contract the blossom C, and proceed working on the contracted tree T'.

Thus, we can formulate a traversal algorithm that finds M-augmenting paths, or reports that no perfect matching exists in G.

What about maximum matching?

This solves the problem of determining whether a perfect matching exists or not, but our original goal was to find the maximum matching, even if no perfect matching exists. Then we need to do something else in Case 2a instead of stopping.

When we encounter Case 2a (a frustrated tree), we know from Lemma 4 that this specific subgraph is locally maximally matched; exactly one vertex (the root) cannot be matched. We have **accumulated one defect** (one unmatched vertex) in T.

If T is frustrated, we leave it as it is, keep its root unmatched, and restart our search from another M-exposed vertex, to build a new alternating tree. We continue this until we find an M-augmenting path (of course), or till every M-exposed vertex has become the root of a frustrated tree.

If we did not find any M-augmenting path (in our search), then we have a forest F of frustrated trees, and a total of $d = |F|$ vertices unmatched in M. We claim that at this point, M is a maximum matching, or in other words, d is the minimum defect attainable on graph G.

Let R_F denote the set of all red vertices in F, i.e. $R_F = \bigcup_{T \in F} R(T)$, similarly $B_F = \bigcup_{T \in F} B(T)$.

Consider the graph $G \setminus R_F$. Every $b \in B_F$ will become isolated, and by lemma 2, each of these represent odd-connected components in G. So,

$$o(G \setminus R_F) \geq |B_F|$$

From our tree invariants, we know that for each tree, $B(T) = R(T) + 1$. Since there are d frustrated trees, we can write

$$|B_F| = |R_F| + d$$

Substituting this in the inequality, we get

$$o(G \setminus R_F) \geq |R_F| + d$$
$$\Rightarrow d \leq o(G \setminus R_F) - |R_F|$$

Recall, that from Equation 1, we have, the minimum defect of any matching must be at least $o(G \setminus R) - |R|$ for some $R \subseteq V$. (If we achive some defect d which is equal to some $o(G \setminus R) - |R|$, then we can not decrease it any further, that defect must be the minimal possible.) We know that our matching has a defect of d. But we just showd that d can be at most the expression $o(G \setminus R_F) - |R_F|$. Thus, the inequality must be tight:

$$d = o(G \setminus R_F) - |R_F|$$

Thus, d must be the minimum defect attainable on G, hence M is a maximum matching of G. This also means R_F is the set that attains the maximum obstruction to get a perfect matching, i.e. $R = R_F$ maximises the term $o(G \setminus R) - |R|$.

Note that this (constructively) proves Theorem 2 (Tutte-Berge Theorem), as we have found a matching that achieves the bound in Equation 1:

$$|M| = \frac{1}{2}(|V| - d) = \frac{1}{2}(|V| + |R_F| - o(G \setminus R_F)) = \min_{R \subseteq V} \frac{1}{2}(|V| + |R| - o(G \setminus R))$$

The algorithm

Finally, we bring everything together and specify the **Edmonds' Blossom Algorithm**:

We start with the empty matching M, and in each iteration, we try to find an M-augmenting path. If it exists, then we augment by the path, and continue. If no augmenting path exists, then M is a maximum matching.

Given a matching M, we need to find an M-augmenting path. We find some M-exposed vertex s. (If none exist, M is already a perfect matching). We will perform a modified BFS on G to find an M-augmenting path. We will maintain a M-alternating tree T (actually, a forest of such BFS trees). We first color s blue, and enqueue s. Initially, only s exists in our tree. In our BFS we maintain the invariant that the queue always only contains blue vertices. So we only ever explore edges from the blue nodes.

In the BFS iteration, we dequeue a blue node u. For each edge (u, v) incident on u:

- **If v is an out-of-tree vertex**:
  - **If v is M-exposed**: We have found an augmenting path: from the root s to u, and the edge (u, v). Return this augmenting path (after lifting the path (Lemma 3) some number of times if required).
  - **If v is M-matched**: Let $(v, r) \in M$ be the matched edge, then add the edges $(u, v), (v, r)$ in the tree, color v red and r blue. Continue.
- Otherwise,
  - **If v is a red vertex** (in the same tree, or a different tree), (similar to edge e_3 in Figure 6), we don't do anything, and continue.
  - **If v is a blue vertex** (must be in the same tree, why?), then we have found an odd cycle C. Contract the cycle C and continue searching on the contracted tree T'.

If at some point, our queue becomes empty (there are no more nodes to dequeue), then T is a frustrated tree (as in Case 2a). Then, we keep T as it is, and restart our BFS from a different M -exposed vertex.

Suppose we have exhausted all M-exposed vertices (and still not found any M-augmenting path). Then, M is a maximum matching, and we are done. (We have shown this in the previous section.)

For completeness (and analysis), here is an informal pseudocode for the complete algorithm.

EDMONDS-BLOSSOM(G)

```
1  M ← φ
2  while true
3     P ← FIND-AUGMENTING-PATH(G, M)
4     if P is not null
5        M ← M augmented along P
6     else
7        return M
```

FIND-AUGMENTING-PATH(G, M)

```
1  F ← φ (the BFS forest)
2  for each M-exposed vertex s in G
3     if s belongs to a previously frustrated tree
4        continue
5     Q ← empty queue
6     color(s) ← blue; add to a new tree T
7     enqueue s into Q
8     while Q is not empty
9        u ← dequeue a blue node from Q
10       for each edge (u, v)
11          if v ∉ F
12             if v is M-exposed
13                P ← path from root s to u in T + edge (u, v)
14                P' ← lift P
15                return P'
16             else if v is M-matched
17                let (v, r) ∈ M be the matched edge
18                add edges (u, v) and (v, r) to T
19                color(v) ← red; color(r) ← blue
20                enqueue r into Q
21          else
22             if color(v) = red
23                continue
24             else if color(v) = blue
25                contract the odd cycle (blossom) into a pseudovertex
26                update G and T to the contracted graph and tree
27                continue
28    mark T as a frustrated tree
29 return null
```

Time Complexity

- Outer loop in EDMONDS-BLOSSOM: On each iteration, we improve the size of the matching by at least one. Since the size of the maximum matching is at most $\frac{V}{2}$, so this loop runs at most $\frac{V}{2}$ times.
- Inner BFS (FIND-AUGMENTING-PATH):
  - Other than blossoms, exploring the graph takes $O(E)$ time.
  - Each blossom contraction (line 25) reduces the total number of vertices in the graph by at least 2. So, there can be at most $O(V)$ contractions. Each contraction involves updating the graph, which can take $O(E)$ time. So in total, contracting vertices take $O(VE)$ time.
  - Path lifting (line 14) takes $O(V)$ time (expanding the pseudo vertices and routing the path through odd cycles). This happens at most once in one call of FIND-AUGMENTING-PATH
  - Every other operation unaccounted for in this function runs in constant time.

Therefore, in total the time complexity is bounded by $O(V) \times O(VE) = O\left(V^2E\right)$, which is polynomial time.