A Theory of Alternating Paths and Blossoms, from the Perspective of Minimum Length

Vijay V. Vazirani

University of California, Irvine, vazirani@ics.uci.edu, https://www.ics.uci.edu/~vazirani/

The Micali-Vazirani (MV) algorithm for finding a maximum cardinality matching in general graphs, which was published in 1980, remains to this day the most efficient known algorithm for the problem. The current paper gives the first complete and correct proof of this algorithm.

The MV algorithm resorts to finding minimum length augmenting paths. However, such paths fail to satisfy an elementary property, called breadth first search honesty in this paper. In the absence of this property, an exponential time algorithm appears to be called for — just for finding one such path. On the other hand, the MV algorithm accomplishes this, and additional tasks, in linear time. The saving grace is the various “footholds” offered by the underlying structure, which the algorithm uses in order to perform its key tasks efficiently.

The theory expounded in this paper elucidates this rich structure and yields a proof of correctness for the algorithm. It may also be of independent interest, as a set of well-knit graph-theoretic facts.

Key words: Maximum matching problem, efficient algorithms, alternating paths, augmenting paths, blossoms, double depth first search

* Supported in part by NSF Grant CCF-2230414.
1. Introduction The following quote, from Lovasz and Plummer’s classic book [18], pg. 12, provides a nice backdrop for the work reported in this paper:

*The concept of an alternating path, although quite simple, is one of the most important in all of matching theory.*

For the significance of this notion in the design of efficient algorithms for matching, as well as the parallel development of the notion of an augmenting path for flow algorithms, we refer the reader to [18, 2]. The computational importance of minimum length augmenting paths was first recognized by Dinitz [1], in the context of flow theory, and this basic idea gave rise to several efficient maximum flow algorithms, see [2]. Independent and simultaneous works, by Hopcroft and Karp [12] and Karzanov [15], studied minimum length augmenting paths in the context of matching, and used this notion to give the most efficient algorithm of its time for maximum matching in bipartite graphs; see Section 10 for improvements obtained in recent years.

Edmonds [5] defined the notion of blossoms and used it to give the first polynomial time algorithm for finding a maximum matching in general graphs. His proof of correctness was built around graph-theoretic facts, which formalize the manner in which augmenting paths traverse blossoms and their complex nested structure.

The most efficient known algorithm for general graph matching is due to Micali and Vazirani [20]. It resorts to finding minimum length augmenting paths and uses the scheme proposed in [12, 15], see Section 1.2. The blossoms it finds are special and are defined in Section 8; in contrast, Edmonds’ blossoms do not take into consideration length information and are therefore inadequate for the purpose of finding minimum length augmenting paths. The description of this algorithm given in [20], via a pseudo-code, is complete and error-free; however, the paper did not attempt a proof of correctness.

A proof of correctness was attempted in [27]. That paper correctly recognized the fact that an elaborate, new theory of alternating paths and blossoms, from the perspective of minimum length paths, was called for to give such a proof. As detailed in Section 1.1, although that paper made some important contributions, it had serious shortcomings.

The current paper completes the task started in [27] by presenting the pertinent theory in full detail and using it to give the first complete and correct proof of the MV algorithm. In addition, it uses innovative expository techniques to render the algorithm easier to comprehend. Considering the special status of the maximum matching problem within the theory of algorithms (see below), it was not appropriate to leave its most efficient known algorithm in an essentially unproven state. Hence the investment of (substantial) effort to produce the current paper, notwithstanding the lapse of considerable time since the publication of the algorithm.

In the case of bipartite graphs, minimum length alternating paths from an unmatched vertex to a matched vertex can be of one parity only, either even or odd. Consequently, such paths possess an elementary property, called *breadth first search (BFS) honesty* in this paper: Let $p$ be a minimum alternating path from unmatched vertex $f$ to $v$ and let $u$ lie on $p$. Then the part of $p$ from $f$ to $u$ is a minimum alternating path from $f$ to $u$, and not any longer. In the presence of this property, a straightforward alternating BFS suffices for executing a phase\(^1\) in linear time.

In general graphs, the existence of minimum length alternating paths of both parities from an unmatched vertex to a matched vertex leads to a difficulty, whose origin lies in the fact that such paths are not BFS honest. In the situation described above, assume that $p$ is a minimum length alternating path from $f$ to $v$. Now, the part of $p$ from $f$ to $u$ can be arbitrarily longer than a minimum path from $f$ to $u$, of either parity. This happens because all minimum paths from $f$ to $u$ contain $v$ at an odd length, see Example 2 in Section 3.2.

As a result, the following fundamental difficulty arises: For finding a minimum augmenting path in the graph, we need to find arbitrarily long paths to intermediate vertices, even though the latter

\(^1\) See Section 1.2 for a definition.
do admit short paths, see Section 3. As such, this appears to call for an exponential time\textsuperscript{2} algorithm. How then does the MV algorithm accomplish this task within the same time as bipartite graphs, i.e., linear time for a phase.

The theory expounded in this paper answers the question raised above by showing how the underlying structure offers several different “footholds” for the key tasks which the algorithm needs to perform in order to home in on a solution quickly. The bottom line is that in non-bipartite graphs, although minimum length alternating paths are not BFS honest in the manner described above, they are not arbitrarily BFS dishonest. In fact, BFS honesty can be restored, if it is defined from a different — though profoundly interesting — perspective, see Theorem 6 and Remark 5.

This theory also yields a proof of correctness of the MV algorithm. Additionally, it lays bare a stark contrast: on the one hand, the extreme complexity of the problem being handled by the algorithm, and on the other, the simplicity of the algorithm itself.

Matching has had a long and distinguished history within graph theory and combinatorics, spanning more than a century and a half [18]. Its algorithmic history is equally long, dating back to the mid-nineteenth century work of Carl Gustav Jacobi on the bipartite case, as mentioned in [29]. Its exalted status in the theory of algorithms arises from the fact that its study has yielded quintessential paradigms and powerful algorithmic techniques, which form the foundation of the modern theory of algorithms, as we know it today. These include definitions of the classes $\mathcal{P}$ [6] and $\#\mathcal{P}$ [25], the primal-dual paradigm [16], the equivalence of random generation and approximate counting for self-reducible problems [14], characterizing facets of the convex hull of solutions to a combinatorial problem [5], the canonical paths argument in the Markov chain Monte Carlo method [13], and the Isolation Lemma [22, 28].

1.1. Overview and Contributions. Besides stating the contributions of this paper, at the end of this section, we will also state the contributions of [27] and point out the nature of its shortcomings due to which the current paper is called for.

A number of structural notions and definitions need to be given. Rather than stating them all up-front, we have spread them over sections in which they are put to use for the first time. The elementary ones, pertaining to minimum length alternating paths, are given in Section 3.1. Section 4 gives definitions required for stating the algorithm, primary among them being tenacity of vertices and edges, and the classification of edges into props and bridges.

The algorithm itself involves two main ideas: the new search procedure called double depth first search (DDFS) and the precise synchronization of events. The former is described in Section 2 in a completely self-contained manner, so it can be read without reading the rest of the paper. The latter is described in Section 9.1.

Section 8 gives the central notions of base of a vertex and blossom; these are essential to the proof of correctness of the algorithm and for clarifying the “footholds” mentioned in the Introduction. As described in Section 8.1, in order to define the base of a vertex $v$, we need to show that the set $B(v)$, defined in Definition 17, is a singleton. However, its proof requires the notion of a blossom and its associated properties. On the other hand, blossoms can be defined only after defining the base of a vertex. Therefore, we are faced a severe “chicken-and-egg” problem.

Our proof resolves this problem by carrying out an induction on tenacity. This is done in the central structural theorem, Theorem 3. The induction basis, for the lowest tenacity vertices in the graph, is given in Section 8.2. It shows that for such vertices $v$, $B(v)$ is a singleton; this unique vertex is defined to be base($v$). The proof of this fact is not straightforward and is accomplished by using the power of DDFS. Indeed, one of the main innovations of the current paper is the use of this procedure, not only in the algorithm, but also for its proof.

\textsuperscript{2} Recall that the problem of finding long paths, such as a Hamiltonian path, is NP-hard.
Once the base of the lowest tenacity vertices is well defined, their blossoms can be defined and properties of these blossoms and properties of paths traversing through them can be established. In the induction step, which is carried out in Section 8.3, these facts are crucially used to establish analogous facts for higher tenacity vertices.

The graph in which the induction basis is carried out is far simpler than the one in which the induction step is carried out, because the latter contains blossoms defined in the previous iterations of the induction. Because of this simplicity, the induction basis plays the role of a crucial in which proof techniques can be developed with relative ease for the various claims and, as detailed below, the “correct” order in which these proofs need to be carried out can be determined. In the proof of the induction step, we first apply a transformation on previously defined blossoms, see Definition 24, after which the structure of the proofs becomes similar to those in the induction basis and these proofs are omitted, unless they contain a significant new insight.

The following steps were taken to render the algorithm easier to comprehend:

1. Wherever possible, procedures are described in plain English. For example, DDFS is described in English in Section 2; readers who prefer to understand it via a pseudocode can find it in [27].

2. For ease of comprehension, DDFS has been described in the simpler setting of a directed, layered graph $H$. In the algorithm, DDFS is run on the original graph $G$. However, describing DDFS on $G$ is too cumbersome; this was done in [27]. Instead, we provide a mapping from $G$ to $H$ in Section 5.3 via which the reader can easily trace the steps DDFS executes in $G$. To further help the reader, in the many illustrative examples given, the distance of vertices from the unmatched vertex is proportional to their minlevels, as would be the case in the corresponding layered graph $H$.

3. The structural definitions given in this paper are of two types: purely graph-theoretic ones and algorithmic ones; the latter depend on the manner in which ties are resolved in a run of the algorithm. Whereas the former include base and blossom, the latter include petal and bud, see Section 5.2. We have demarcated these two types of definitions and in Section 9.2 we have pointed out relationships between them.

At a high level, [27] also accurately identified the interplay between the notions of tenacity, base, blossom and bridge. However, the actual definitions given for some of these notions were incorrect. For example, in [27], the central notion of base of a vertex was defined for any vertex of finite tenacity. However, it turns out that there may be vertices of finite tenacity which have no base, e.g., see Example 14. In the current paper, base has been defined only for vertices of eligible tenacity, see Definition 15 for this notion. Theorem 3 in [27] “proves” that every vertex of finite tenacity has a well-defined base – it is obviously incorrect. Furthermore, Theorem 3 in [27] is incorrect even for vertices of eligible tenacity. We provide three high-level reason due to which the proofs of this and other theorems given in [27] are incorrect:

1. [27] failed to identify the “chicken-and-egg” problem stated above and tried to “prove” the existence of base in a stand-alone manner, instead of relating it to other fundamental notions, as is done in Theorem 3 in the current paper.

2. The idea of using the power of DDFS to carry out proofs was not obtained at that time. This idea qualitatively changed the nature of the proofs: instead of dealing with individual alternating paths, which can get arbitrarily complicated, leading to incorrect proofs, we could deal with the high level structures whose existence is guaranteed by the DDFS Certificate, proved in Theorem 1.

3. The mindset in [27] was the following: The culminating fact which needed to be proven was the existence of a bridge of the right tenacity on a maxlevel path\(^3\). A variety of other structural facts needed to be established prior to that.

\(^3\) This fact is indeed a central one, as detailed in Section 9.4.
In retrospect, this ordering was incorrect. As mentioned above, in the current paper, the order is dictated by the proof of the induction basis, since its simple setting makes transparent the “right” order of implications, see Section 8.2. Furthermore, the same order is retained in the proof of the induction step as well, see Section 8.3. In particular, the existence of a bridge is the first, and not the last, fact we established in the induction basis, as well as in the step.

1.2. Running Time and Related Papers The simplest scheme for finding a maximum matching is to start with an empty matching, and iteratively find augmenting paths and augment the current matching. As shown in [3], when there is no such path, the matching must be maximum. In order to improve the running time, [12, 15] proposed finding multiple augmenting paths in each iteration.

Definition 1. (Phase) In a graph $G = (V, E)$ with matching $M$, a phase consists of finding a maximal set of disjoint minimum length augmenting paths and augmenting $M$ along all paths found.

As shown in [12, 15], $O(\sqrt{n})$ phases suffice$^4$ for finding a maximum matching. These papers also show how to implement a phase in $O(m)$ time in a bipartite graph, thereby getting a total running time of $O(m\sqrt{n})$. The MV algorithm executes a phase in almost linear time. Its precise running time is $O(m\sqrt{n} \cdot \alpha(m, n))$ in the pointer model, and $O(m\sqrt{n})$ in the RAM model (see Theorem 8 for details).

We note that small theoretical improvements to the running time, for the case of very dense graphs, have been given in recent years: $O(m/\sqrt{n} \log(n^2/m)/\log n)$ [11] and $O(n^w)$ [21] where $w$ is the best exponent of $n$ for multiplication of two $n \times n$ matrices. The former improves on MV for $m = n^{2-o(1)}$ and the latter for $m = \omega(n^{1.85})$. However, the latter algorithm involves a large multiplicative constant in its running time which comes from the use of fast matrix multiplication as a subroutine, thereby making the small improvement in the exponent not very meaningful.

Prior to [20], Even and Kariv [7] had used the idea of finding augmenting paths in phases to obtain an $O(n^{2.5})$ maximum matching algorithm. However, their algorithm is extremely complicated and its correctness is hard to ascertain, in particular because there is no journal version of this result.

Subsequent to [20], [9] gives an efficient scaling algorithm for finding a minimum weight matching in a general graph with integral edge weights and at the end of the paper, it claims that the unit weight version of their algorithm achieves the same running time as MV. A much better version of the weighted approach to cardinality matching was given recently in [10], again achieving the same running time. The rest of the history of matching algorithms is very well documented and will not be repeated here, e.g., see [18, 27].

2. Double Depth First Search (DDFS) This section is fully self-contained and describes the procedure of double depth first search (DDFS). For ease of comprehension, we have presented DDFS in the simplified setting of a directed, layered graph $H$. In the MV algorithm, DDFS is run on the original graph $G$, which is far more complex. In $G$, DDFS terminates with either a new blossom — more precisely, a new petal — or the existence of a new augmenting path. These correspond to Case 1 and Case 2, detailed below. We will provide an explicit mapping between the two settings in Section 5.3.

The input to DDFS is a directed, layered graph $H = (V, E)$. $V$ is partitioned into $h+1$ layers, for some $h > 0$. The layers are numbered from 0 to $h$ and are named $l_0, \ldots, l_h$, with $l_0$ being the lowest layer and $l_h$ the highest layer. The layer number of a vertex $v$ is denoted by $l(v)$. We will assume that for each $v \in V$, $l(v)$ is easily available; in fact, it can be obtained in unit time. If $l(v) < l(u)$, then we will say that $v$ is deeper than $u$. Each directed edge $(u,v) \in E$ runs from a higher to a
lower layer, not necessarily consecutive, i.e., \( l(u) > l(v) \). \( V \) contains two special vertices, \( r \) and \( g \), for red and green, not necessarily in the same layer, and neither of them in \( l_0 \). See Figure 1 for a layered graph with \( h = 7 \).

At a high level, the objective of DDFS is to grow two DFS trees, \( T_r \) and \( T_g \), rooted at \( r \) and \( g \), respectively, in such a way that \( T_r \) and \( T_g \) share at most one vertex and the deepest vertex (vertices) in the two trees is (are) “as deep as possible”. Furthermore, this needs to be done in time that is linear in the sum of the sizes of the two trees. Because of the DDFS Requirement, stated below, it is trivial to grow any one tree very deep, all the way to the lowest layer, \( l_0 \). However, this will not achieve the more interesting and useful objective stated above. For that, we grow each tree in such a way that it does not “block off” the other tree, by growing them in a highly coordinated manner; the latter is the main point of DDFS.

![Layered graph](image)

**Figure 1.** Layered graph \( H \) with \( r \) and \( g \) in layer \( l_7 \).

We require that \( H \) satisfies:

**DDFS Requirement:** Starting from every vertex \( v \in V \), there is a path to a vertex in layer \( l_0 \).

Vertex \( v \) will be called a bottleneck if every path from \( r \) to \( l_0 \) and every path from \( g \) to \( l_0 \) contains \( v \); \( v \) is allowed to be \( r \) or \( g \) or a vertex in layer \( l_0 \). Let \( p \) be a path from \( r \) or \( g \) to layer \( l_0 \). Since layer numbers on \( p \) are monotonically decreasing, if there is a bottleneck, the one having highest level must be unique. It will be called the highest bottleneck and we will denote it by \( b \). If \( H \) has a bottleneck, we will say that we are in Case 1. Otherwise, there must be distinct vertices \( r_0 \) and \( g_0 \) in layer \( l_0 \) such that there are vertex-disjoint paths from \( r \) to \( r_0 \) and \( g \) to \( g_0 \); this will be called Case 2.
As stated above, in the MV algorithm, these two cases correspond to the creation of a new petal and the discovery of a new augmenting path, respectively. In the graph of Figure 1, DDFS will terminate in Case 1, with bottleneck $b$, as shown in Figure 2. In the graph of Figure 3, which differs from the graph of Figure 1 only in the two edges going from $c$ to $g_0$, DDFS terminates in Case 2, with disjoint paths from $r$ to $r_0$ and $g$ to $g_0$.

In Case 1, let $V_b (E_b)$ be the set of all vertices (edges) that lie on all paths from $r$ or $g$ to $b$. In Case 2, let $E_p$ be the set of all edges that lie on all paths starting from $r$ or $g$ and ending at $r_0$ or $g_0$.

The objective of DDFS: The first objective of DDFS is to determine which of these two cases holds. Additional objectives of DDFS in the two cases are:

**Case 1:** DDFS needs to find the highest bottleneck, $b$, and partition the vertices in $V_b \setminus \{b\}$ into two sets $S_R$ and $S_G$, called the red set and green set respectively, with $r \in S_R$ and $g \in S_G$. These sets should satisfy:
1. There is a path from $r$ to $b$ in $S_R \cup \{b\}$ and a path from $g$ to $b$ in $S_G \cup \{b\}$.
2. There are two spanning trees, $T_r$ and $T_g$, in $S_R \cup \{b\}$ and $S_G \cup \{b\}$, and rooted at $r$ and $g$, respectively. Furthermore, DDFS needs to find such a pair of trees.

**Case 2:** DDFS needs to find distinct vertices $r_0$ and $g_0$ in layer $l_0$, and vertex disjoint paths from $r$ to $r_0$ and $g$ to $g_0$. In this case, as soon as DDFS finds these two paths, it halts, even if it has not traversed all edges of $E_p$, since a new augmenting path has already been found. Let $E'_p \subseteq E_p$ denote the edges which DDFS has actually traversed.

The two DFS trees: DDFS involves the coordinated growth of two DFS trees, the red tree $T_r$ and the green tree $T_g$, rooted at $r$ and $g$, respectively. At each point in the algorithm, each tree has a well-defined center of activity, i.e., the vertex it is currently exploring. These are denoted by $C_r$ and $C_g$ and are initialized to the two roots, $r$ and $g$, respectively. When a center of activity is at a vertex $u$ and is ready to move, it must be the case that the color of $u$ is the same as that of the center of activity. If the center moves to a vertex $v$, the edge $(u, v)$ is assigned to the corresponding tree and given the color of the center.

Therefore, each edge $(u, v)$ has the same color as that of $u$, i.e., all edges out of $u$ will get the color of $u$. Note however that the color of $v$ may be different from that of $u$. Figure 2 shows $T_r$ and $T_g$ after DDFS has been performed on the graph of Figure 1. $T_r$ consists of broken edges and $T_g$ consists of solid edges; the edge from $b$ to $l_0$ is in neither tree. Note that $b$ is in both trees and gets neither color.

At termination, DDFS provides the following certificate.

**DDFS Certificate:** In Case 1, for every vertex $v \in V_b \setminus \{b\}$, if $v$ is red, there a path from $r$ to $v$ in $T_r$ and a disjoint path from $g$ to $b$ in $T_g$. And if $v$ is green, there exists a path from $g$ to $v$ in $T_g$ and a disjoint path from $r$ to $b$ in $T_r$. In Case 2, there are vertex disjoint paths from $r$ to $r_0$ and $g$ to $g_0$ having colors red and green, respectively.

**Running time:** The running time of DDFS needs to be $O(|E_b|)$ in Case 1 and $O(|E'_p|)$ in Case 2.

**Coordinated growth of the two trees:** We will first describe aspects of DDFS in which the two trees function as “normal” DFS trees in a directed graph, and then we will describe their coordination; in particular, the coordination determines, at each step, which tree grows. Initially, all vertices, other than $r$ and $g$, are marked “unvisited” and all edges are marked “unexplored”.

Every vertex in $T_r \cup T_g$, other than $r$, $g$, and $b$, has a unique parent; $r$ and $g$ have no parent and $b$ has two parents, one of each color. Assume that $C_r = u$ and it is $T_r$’s turn to grow. If so, $T_r$
picks an unexplored edge, say \((u, v)\), out of \(u\). If \(v\) is already marked “visited” and \(C_g \neq v\), then \(T_r\) picks another unexplored edge out of \(u\). If \(v\) is marked “unvisited”, then \(v\) is marked “visited”, \(u\) is designated the parent of \(v\), and \(C_r\) moves to \(v\). The last case, i.e., \(v\) is already marked “visited” and \(C_g = v\) is dealt with below. When all outgoing edges from \(u\) have been explored, \(C_r\) backtracks from \(u\) to its parent if \(u \neq r\); the case \(u = r\) is dealt with below. The growth of \(T_g\) is analogous. Since \(H\) is acyclic, the trees have no back edges.

We next describe the coordination between the two trees. We will adopt the (arbitrary) convention that \(C_r\) will “try to keep ahead of” \(C_g\), and \(C_g\) will “try to catch up”. Following this convention, the moves of \(C_r\) are as follows: If \(l(C_r) > l(C_g)\), then \(C_r\) keeps moving until the first time that \(l(C_r) \leq l(C_g)\). If \(l(C_r) = l(C_g)\) and \(C_r \neq C_g\), then \(C_r\) moves one step and stops; at this point, \(l(C_r) < l(C_g)\). If \(C_r = C_g\), the two centers of activity have met and this case is described below. The moves of \(C_g\) are as follows: If \(l(C_r) < l(C_g)\), then \(C_g\) keeps moving until the first time that \(l(C_g) \leq l(C_r)\) and then it stops.

**When the two centers of activity meet:** As stated above, when one of the centers of activity traverses an edge, the edge is assigned to the corresponding tree and is assigned its color.

However, the assigning of color to a vertex is not so straightforward and is not done in a greedy manner. Indeed, a vertex \(v\) may first be added to one tree and later this decision may be reverted.
this happens if \( C_r \) and \( C_g \) meet at \( v \). We will adopt the convention that when this happens, first \( C_g \) backtracks and tries to find an alternative path that is as deep as \( v \). If it fails, then \( C_g \) occupies \( v \) and \( C_r \) tries to find an alternative path that is as deep as \( v \). If \( C_r \) also, fails, then it must have backtracked all the way to the root \( r \) and DDFS terminates.

We will explain these moves in detail via Figure 2. In this figure, the numbers on the edges indicate the order in which they are added to the two trees. Observe that the two centers of activity meet for the first time at \( a \). At this point, DDFS needs to determine if \( a \) is the highest bottleneck, and if not, then which of the trees can find an alternative path at least as deep as \( a \), so search may resume. By the convention established above, \( C_g \) tries first. After it backtracks all the way to \( g \), it traverses edges number 5 and 6 and arrives at a vertex that is as deep as \( a \), and DDFS resumes.

The two centers of activity meet for a second time — at vertex \( c \). This time, \( C_g \) backtracks all the way to \( g \), without finding an alternative path. As per our convention, \( C_g \) now occupies \( c \) and \( C_r \) tries to find an alternative path as deep as \( c \).

However, at this stage, we need to introduce an important notion, namely the pointer \( \text{Barrier} \). Its purpose is to prevent \( C_g \) from backtracking from a vertex more than once. At the start of DDFS, the Barrier is initialized to \( g \). At this stage, since \( C_g \) has backtracked from \( c \) all the way to \( g \), i.e., the current position of the Barrier, it is now moved to \( c \).

Next, the two centers of activity meet at \( b \). By our convention, \( b \) is first given to \( T_r \) and \( C_g \) attempts to find an alternative path. However, it backtracks all the way to the Barrier, which is currently at \( c \), without success. At this point, the Barrier is moved to \( b \), \( b \) is given to \( T_g \) and \( C_r \) attempts to find an alternative path. However, it backtracks all the way to \( r \) without finding an alternative path. At this point, we conclude that \( b \) is the bottleneck. In general, when \( C_r \) backtracks all the way to \( r \), DDFS terminates in Case 1 and the current meeting point is declared the bottleneck.

In Figure 3, after backtracking from \( b \), \( C_g \) does manage to find an alternative path as deep as \( b \), when it explores edge number 15. At this point, DDFS resumes and \( C_r \) reaches \( r_0 \) in layer \( l_0 \) and \( C_g \) reaches \( g_0 \) in that layer, hence terminating in Case 2.

**Theorem 1.** DDFS accomplishes the objectives stated above in the required time.

**Proof:** In Case 1, tree \( T_r \) contains paths consisting of red colored vertices from \( r \) to \( b \) and from \( r \) to each red vertex. A similar claim holds about tree \( T_g \). In Case 2, there is a path consisting of red colored vertices from \( r \) to \( r_0 \) in tree \( T_r \) and there is a path consisting of green colored vertices from \( g \) to \( g_0 \) in tree \( T_g \). Therefore, the DDFS Certificate holds.

Finally, it is easy to see that each edge of \( H \) is explored by at most one tree and if so, only once. Clearly, \( T_r \) backtracks from each vertex at most once and because of the Barrier, the same holds for \( T_g \) as well. The theorem follows.

3. **Elementary Definitions and a Fundamental Notion** In Section 3.1, we will present some elementary definitions pertaining to minimum length augmenting paths. Using these definitions, in Section 3.2 we present the property of breadth first search honesty, due to which an alternating BFS works in bipartite graphs, yielding a linear time algorithm for a phase. The lack of this property in non-bipartite graphs necessitate a much more complex algorithm.

---

5 Observe that either of the trees could have arrived at \( v \) first. This happens despite our convention that \( C_r \) keeps ahead of \( C_g \) — the reason is that \( C_g \) may have used a long edge to arrive at \( v \) before \( C_r \). In Figure 2, this happens when the two trees meet at vertex \( c \).
3.1. Elementary Definitions  A matching $M$ in an undirected graph $G = (V, E)$ is a set of edges no two of which meet at a vertex. Our problem is to find a matching of maximum cardinality in the given graph. *Henceforth all definitions will be w.r.t. a fixed matching $M$ in $G$. Edges in $M$ will be said to be matched and those in $E - M$ will be said to be unmatched.* Vertex $v$ will be said to be matched if there is a matched edge incident at it and unmatched otherwise.

An *alternating path* is a simple path whose edges alternate between $M$ and $E - M$, i.e., matched and unmatched. An alternating path that starts and ends at unmatched vertices is called an *augmenting path*. Clearly the number of unmatched edges on such a path exceeds the number of matched edges on it by one\(^6\). Its significance lies in that flipping matched and unmatched edges on such a path leads to a valid matching of one higher cardinality. Edmonds’ matching algorithm operates by iteratively finding an augmenting path w.r.t. the current matching, which initially is assumed to be empty, and augmenting the matching. When there are no more augmenting paths w.r.t. the current matching, it can be shown to be maximum.

The MV algorithm finds augmenting paths in phases as proposed in [12, 15]. In each *phase*, it finds a maximal set of disjoint minimum length augmenting paths w.r.t. the current matching and it augments along all paths. [12, 15] show that only $O(\sqrt{n})$ such phases suffice for finding a maximum matching in general graphs. The remaining task is designing an efficient algorithm for a phase.

**Definition 2.** (Length of Minimum Length Augmenting Path) Throughout, $l_m$ will denote the length of a minimum length augmenting path in $G$; if $G$ has no augmenting paths, we will assume that $l_m = \infty$.

\(^6\) Observe that if $M = \emptyset$, then any edge is an augmenting path, of length one.
DEFINITION 3. (Evenlevel and oddlevel of vertices) The evenlevel (oddlevel) of a vertex \( v \), denoted evenlevel(\( v \)) (oddlevel(\( v \))), is defined to be the length of a minimum even (odd) length alternating path from an unmatched vertex to \( v \); moreover, each such path will be called an evenlevel(\( v \)) (oddlevel(\( v \))) path. If there is no such path, evenlevel(\( v \)) (oddlevel(\( v \))) is defined to be \( \infty \).

We will typically denote an unmatched vertex by \( f \). Its evenlevel is zero and its oddlevel is the length of the shortest augmenting path starting at \( f \); if no augmenting path starts at \( f \), oddlevel(\( f \)) = \( \infty \). The length of a minimum length augmenting path w.r.t. \( M \) is the smallest oddlevel of an unmatched vertex. In all the figures, matched edges are drawn broken and unmatched edges solid; unmatched vertices are drawn with a small circle.

![Figure 4](image)

Figure 4. The evenlevels and oddlevels of vertices are indicated; missing levels are \( \infty \).

DEFINITION 4. (Maxlevel and minlevel of vertices) For a vertex \( v \) such that at least one of evenlevel(\( v \)) and oddlevel(\( v \)) is finite, maxlevel(\( v \)) (minlevel(\( v \))) is defined to be the bigger (smaller) of the two.

DEFINITION 5. (Outer and inner vertices) A vertex \( v \) with finite minlevel is said to be outer if evenlevel(\( v \)) < oddlevel(\( v \)) and inner otherwise.

DEFINITION 6. (Odd and even w.r.t. \( p \)) Let \( p \) be an alternating path from unmatched vertex \( f \) to \( v \) and let \( u \) lie on \( p \). The length of path \( p \), denoted \( |p| \), is the number of edges on \( p \). The part of \( p \) from \( f \) to \( u \) will be denoted by \( p[f \ to \ u] \) and \( p[f \ to \ u] \) will denote the part of \( p \) from \( f \) to the vertex just before \( u \). Other combinations are self-explanatory. We will say that vertex \( u \) is even w.r.t. \( p \) if \( |p[f \ to \ u]| \) is even and it is odd w.r.t. \( p \) if \( |p[f \ to \ u]| \) is odd.

EXAMPLE 1. In the figures hereafter, matched and unmatched edges are drawn with broken and solid lines, respectively. Additionally, unmatched vertices are drawn with a small circle, e.g., vertex \( f \) in Figure 4. The numbers in this figure indicate the evenlevels and oddlevels of vertices, with missing numbers being infinity.

3.2. The Notion of BFS Honesty Let \( p \) be an alternating path from unmatched vertex \( f \) to \( v \). We will say that \( p \) is a minimum alternating path if \( |p[f \ to \ u]| = \text{evenlevel}(\( v \)) \) (\( |p[f \ to \ u]| = \text{oddlevel}(\( v \)) \) if \( |p[f \ to \ u]| \) is even (odd).
Breadth first search (BFS) honesty is the following property: Let $p$ be a minimum alternating path from unmatched vertex $f$ to $v$ and let $u$ lie on $p$. Then $p[f \to u]$ is a minimum alternating path from $f$ to $u$. Bipartite graphs satisfy this property, and as a consequence, a straightforward alternating BFS suffices for finding minimum augmenting paths, see Section 4, or for a complete description, see Section 2.1 in [27].

Surprisingly enough, this elementary property does not hold in non-bipartite graphs, as illustrated in Example 2. This basic difference arises because in bipartite graphs, minimum length alternating paths to a vertex $v$ can be of one parity only, either even or odd, but in non-bipartite graphs, they can be of both parities. As a result, the following may happen: Let $p$ be an evenlevel($v$) path from $f$ to $v$ and let $u$ lie on it with $|p[f \to u]|$ being odd, say. Then $p[f \to u]$ can be arbitrarily longer than an oddlevel($u$) path. The reason is that every oddlevel($u$) path, say $q$, contains $v$ at an odd length, and extending $q$ to $v$ to get an even length path will result in a self-intersecting path, see Example 2. Consequently, for the graph in Figure 5, we would need to find longer and longer odd-length alternating paths from $f$ to $w$ in order to find minimum length alternating paths from $f$ to other vertices, e.g., $u$ and $v$.

In summary, the following fundamental difficulty arises: For finding a minimum length augmenting path, we need to find arbitrarily long paths to intermediate vertices, even though the latter do admit short paths, see Section 3. As such, this appears to call for an exponential time algorithm. Recall that finding short paths is easy and long paths is hard, e.g., Hamiltonian path is NP-hard.

Example 2. In Figure 5, oddlevel($w$) = 7. An evenlevel($v$) path is shown in this figure. Observe that $w$ occurs at a length of 11 on this path. Also observe that $v$ occurs at an odd length on the oddlevel($w$) path. It will be instructive for the reader to find an evenlevel($u$) path; observe that $w$ occurs at a length of 9 on it.

4. Some Essential Definitions As mentioned in the Introduction, in order to implement a phase in linear time in non-bipartite graphs, we need to exploit the elaborate structure offered by minimum length alternating paths. In this section, we present some facts that are absolutely necessary to describe the MV algorithm. The proof of correctness of the algorithm requires additional structural properties, presented in Section 8.

Definition 7. (Tenacity of vertices and edges) Define the tenacity of vertex $v$, tenacity($v$) = evenlevel($v$) + oddlevel($v$). If $(u, v)$ is an unmatched edge, then tenacity($u, v$) = evenlevel($u$) + evenlevel($v$) + 1, and if it is matched, tenacity($u, v$) = oddlevel($u$) + oddlevel($v$) + 1.

The notion of tenacity is central to the structural facts that follow. Clearly tenacity($f$) $\geq l_m$ for an unmatched vertex $f$, see Definition 2 for the notion of $l_m$. Furthermore, tenacity($f$) = $l_m$ if and only if $f$ participates in a minimum length augmenting path.

Definition 8. (Minimum tenacity of a vertex in $G$) Throughout, $t_m$ will denote the tenacity of a minimum tenacity vertex in $G$.

Clearly $t_m \leq l_m$. If $t_m = l_m$, the situation is particularly simple, since there are no blossoms and essentially the bipartite graph algorithm works for executing a phase. Henceforth we will assume that $t_m < l_m$.

Example 3.

In Figure 6, the tenacities of vertices are marked. They are $\alpha = 13, \beta = 15, \gamma = 17$ and $\delta = \infty$. In Figure 7, the tenacities of edges are marked. They are $\alpha = 13, \beta = 15$ and $\gamma = 17$.

Definition 9. (Predecessor, prop and bridge) Consider a minlevel($v$) path and let $(u, v)$ be the last edge on it; clearly, $(u, v)$ is matched if $v$ is outer and unmatched otherwise. In either case, we will say that $u$ is a predecessor of $v$ and that edge $(u, v)$ is a prop. An edge that is not a prop will be defined to be a bridge.

Definition 10. (The relations pred and pred*) Let $v$ be a vertex such that minlevel($v$) is finite. If $v$ is an outer vertex, it will have a unique predecessor, namely its matched neighbor; otherwise,
it will have one or more predecessors. The relation pred is defined as follows: we will say that \( u \) is pred \( v \) if \( u \) is a predecessor of \( v \); we will also write is as \( u = \text{pred} \ v \). The relation \( \text{pred}^* \) is the reflexive, transitive closure of the relation pred. If \( u \) is \( \text{pred}^* v \), we will also write as \( u = \text{pred}^* v \).

**Example 4.** In Figure 5, the two horizontal edges and the oblique unmatched edge at the top are bridges and in Figure 9, \((w, w')\) and \((u, v)\) are bridges; the rest of the edges in these two graphs are props. In Figure 8, edge \((u, v)\) is a bridge. This bridge is unusual because \( u \) is \( \text{pred}^* v \), even though \( u \) is not \( \text{pred} v \).

**Definition 11.** (*The support of a bridge*) Let \((u, v)\) be a bridge of tenacity \( t \leq l_m \). Then, its support is defined to be

\[
\text{support}(u, v) = \{ w \mid \text{tenacity}(w) = t \text{ and } \exists \text{ a maxlevel}(w) \text{ path containing } (u, v) \}.
\]

**Example 5.** In the graph of Figure 6, the supports of the bridges of tenacity \( \alpha, \beta \) and \( \gamma \) are the sets of vertices of tenacity \( \alpha, \beta \) and \( \gamma \), respectively. In the graph of Figure 8, the tenacity of bridge \((u, v)\) is 13 and its support consists of two vertices, \( v \) and its matched neighbor. In Figure 9, the supports of the bridges \((w, w')\) and \((u, v)\) are all vertices of tenacity 11 and 13, respectively. Observe that in Figure 9, \( f \) is not in the support of any bridge and \( \text{tenacity}(f) = \infty \).

5. **A Description of the MV Algorithm** The MV algorithm executes phases as defined in Section 1.2. Each phase starts with the matching, say \( M \), computed in the last phase. Its most basic
Figure 6. The tenacity of vertices is indicated; here $\alpha = 13$, $\beta = 15$, $\gamma = 17$ and $\delta = \infty$.

Figure 7. The tenacity of each edge is indicated; here $\alpha = 13$, $\beta = 15$ and $\gamma = 17$.

Figure 8. Edge $(u, v)$ is a bridge.

The task is to find the minlevel and maxlevel of all vertices reachable from the unmatched vertices. For this purpose, the algorithm calls the procedures MIN and MAX iteratively as described below.

In this section, we have described the MV algorithm using only the definitions stated in Section 4. However, in some places, more clarity results from using notions that are defined later in the paper; if so, we have referred to the appropriate definitions. The reader is advised to get a broad
idea of the algorithm on first reading and occasionally return to this section while reading the rest of the paper.

5.1. Procedures MIN and MAX  At the beginning of a phase, all unmatched vertices are assigned a minlevel of 0, the rest are assigned a temporary minlevel of ∞. No vertices are assigned maxlevels at this stage. The algorithm for a phase is organized in search levels, denoted by $i$, starting at 0. At each search level, MIN executes one step of alternating BFS and is followed by MAX, which executes DDFS, if needed. See Algorithm 1 for a summary of the main steps.

If $i$ is even (odd), MIN searches from all vertices, $u$, having an evenlevel (oddllevel) of $i$ along incident unmatched (matched) edges, say $(u, v)$. If edge $(u, v)$ has not been scanned before, MIN will determine if it is a prop or a bridge as follows. If $v$ has already been assigned a minlevel of at most $i$, then $(u, v)$ is a bridge. Otherwise, $v$ is assigned a minlevel of $i + 1$, $u$ is declared a predecessor of $v$ and edge $(u, v)$ is declared a prop. Note that if $i$ is odd, $v$ will have only one predecessor – its matched neighbor, and if $i$ is even, $v$ will have one or more predecessors.

Once an edge is identified as a bridge, if MIN is able to ascertain its tenacity, say $t$, then the edge is inserted in the list of bridges of tenacity $t$, $Br(t)$. MIN is able to ascertain the tenacity of a bridge as long as it is not an anomalous bridge, as defined below; in the latter case, MAX finds the tenacity of this bridge. Task 2 in Theorem 7 proves that by the end of execution of procedure MIN at search level $i$, the algorithm would have identified every bridge of tenacity $2i + 1$. 

Figure 9. Edges $(w, w')$ and $(u, v)$ are bridges, with the latter being an anomaly bridge.
Algorithm 1 At search level $i$:

1. MIN:
For each level $i$ vertex, $u$, search along appropriate parity edges incident at $u$.
For each such edge $(u, v)$, if $(u, v)$ has not been scanned before then
   If minlevel($v$) $\geq i + 1$ then
     minlevel($v$) $\leftarrow$ $i + 1$
     Insert $u$ in the list of predecessors of $v$.
     Declare edge $(u, v)$ a prop.
   Else declare $(u, v)$ a bridge,
     and if tenacity$(u, v)$ is known, insert $(u, v)$ in $Br$(tenacity$(u, v)$).
End
End

2. MAX:
For each edge in $Br(2i + 1)$:
   Find its support using DDFS.
   For each vertex $v$ in the support:
     maxlevel($v$) $\leftarrow$ $2i + 1$ $-$ minlevel($v$)
     If $v$ is an inner vertex, then
       For each edge $e$ incident at $v$ which is not prop,
         if its tenacity is known, insert $e$ in $Br$(tenacity$(e)$).
     End
End

After MIN is done, procedure MAX calls DDFS with each bridge of tenacity $2i + 1$ and finds the support of this bridge. In the process, DDFS finds all vertices, $v$, having tenacity$(v)$ $= 2i + 1$. Since their minlevels are at most $i$, they are already known, and hence maxlevel$(v)$ $= 2i + 1$ $-$ minlevel$(v)$ can be easily computed. Clearly, if minlevel$(v)$ is evenlevel$(v)$ then maxlevel$(v)$ will be oddlevel$(v)$ and if minlevel$(v)$ is oddlevel$(v)$ then maxlevel$(v)$ will be evenlevel$(v)$.

EXAMPLE 6. For each bridge in the first five figures, its tenacity gets ascertained by MIN (including the bridge $(u, v)$ in Figure 8). We next explain the notion of an anomalous bridge via the graph in Figure 9. At search level 4, MIN searches from vertex $u$ along edge $(u, v)$ and realizes that $v$ already has a minlevel of 3 assigned to it. Moreover, $u$ got its minlevel from its matched neighbor. Therefore, MIN correctly identifies edge $(u, v)$ to be a bridge. However, it is not able to ascertain tenacity$(u, v)$ since evenlevel$(v)$ is not known at this time. At search level 5, after conducting DDFS on bridge $(w, w')$ (of tenacity 11), MAX will assign maxlevel$(v)$ $= 8$, which is also evenlevel$(v)$. Therefore, at that time, tenacity$(u, v)$ will be ascertained to be 13 by MAX and edge $(u, v)$ is inserted in $Br(13)$. Thus $(u, v)$ is an anomalous bridge.

Let us explain this notion in more general terms. Let $(u, v)$ be an unmatched bridge such that the evenlevel of one of the endpoints, say $v$, has not been determined at the point when MIN realizes that $(u, v)$ is a bridge; if so $v$ must be an inner vertex. The evenlevel of $v$ will be determined by MAX at search level $\text{tenacity}(v) - 1)/2$ and at this point, tenacity$(u, v)$ is ascertained and the edge is inserted in $Br$(tenacity$(u, v)$). An important point to note in Figure 9, is that tenacity$(v) < \text{tenacity}(u, v)$. This ensures that maxlevel$(v)$ is known at search level $(\text{tenacity}(v) - 1)/2$, i.e., before the search level at which bridge $(u, v)$ needs to be processed by MAX, namely search level $(\text{tenacity}(u, v) - 1)/2$. 


Assume that DDFS is processing a bridge of tenacity $2i + 1$ and vertex $v$ is in its support. If $v$ is inner and has an incident unmatched edge $(u, v)$ which is not a prop, then it must be an anomalous bridge. MAX will ascertain its tenacity and insert it in $Br(\text{tenacity}(u, v))$. Note that $\text{tenacity}(u, v) > 2i + 1$ and bridge $(u, v)$ will need to be processed in a higher search level.

Let $l_m$ be the length of a minimum length augmenting path in a phase. Then during search level $j_m$, where $l_m = 2j_m + 1$, a maximal set of such paths is found. This is described in Section 5.4.

5.2. The Notions of Petal and Bud  
Assume that DDFS is called with a bridge $(u, v)$ of tenacity $t$ and it terminates in Case 1. Then the bottleneck $b$ found is called a bud. Note that $b$ will always be an outer vertex. The set of vertices of tenacity $t$ encountered by DDFS, which must lie in the support of $(u, v)$, form a new petal. Formally, the petal consists of all vertices in the support of $(u, v)$ minus the supports of all bridges processed thus far in this search level (which will all be of tenacity $t$). Clearly a vertex is included in at most one petal.

Example 7. In the graph of Figure 10, MAX will call DDFS with the bridge $(r_1, r_2)$, which is of tenacity 9, at search level 4. The two DFSs will be rooted at $r_1$ and $r_2$, and DDFS will terminate in Case 1 with $b$ as the highest bottleneck. The four vertices which constitute the support of bridge $(r_1, r_2)$ form the new petal and $b$ is the bud of this petal. Observe that $b$ does not belong to this petal.

When a new petal is found, the algorithm executes the following steps: It creates a new node, called petal-node; this has the shape of a doughnut in Figure 10. All vertices of the new petal point\textsuperscript{7} to the petal-node; $b$ is not in the petal and does not point to the petal-node. The new petal-node

\textsuperscript{7} To avoid cluttering up Figure 10, only one vertex is pointing to the petal-node.
points to the two endpoints of its bridge, \( r_1 \) and \( r_2 \), and to its bud, \( b \). These pointers will enable the algorithm to:

1. Skip over this petal in future DDFSs.
2. Efficiently find an alternating path through the petal.

![Diagram](image)

**Figure 11.** The bud formed when DDFS is performed on bridge \((l_1,l_2)\); its bud is \( b \).

**Figure 12.** The bud and blossom formed when DDFS is performed on bridge \((r_1,r_2)\); their bud and base is \( f \).

**Definition 12.** (The bud of a vertex) Define a function \( \text{bud} : V \to V \) as follows. If vertex \( v \) is in a petal then \( \text{bud}(v) = b \), where the bud of this petal is \( b \), and if \( v \) is not in a petal, then \( \text{bud}(v) = v \).

At any point in the execution of the algorithm, the function \( \text{bud}^*(v) \) is defined recursively as follows: If \( \text{bud}(v) = v \) then \( \text{bud}^*(v) = v \), else \( \text{bud}^*(v) = \text{bud}^*(\text{bud}(v)) \). Clearly, \( \text{bud}^*(v) \) will keep changing as the algorithm proceeds.

The notions of petal and bud are intimately related to the notions of blossom and base. Whereas the first pair is algorithmic — the exact petals and buds found depend on the manner in which the algorithm resolves ties — the second pair is purely graph-theoretic. The relationship between these notions is established in Section 9.2. Here we simply note that a blossom is a union of petals and the base of a vertex \( v \) will be \( \text{bud}^*(v) \) at the end of MAX in search level \((t - 1)/2\) where \( \text{tenacity}(v) = t \).

**Example 8.** In the graph of Figure 11, the two bridges \((l_1,l_2)\) and \((r_1,r_2)\) are of the same tenacity. The algorithm will break this tie arbitrarily and perform DDFS on these bridges in one of the two orders. In Figures 11 and 12, the order is \((l_1,l_2)\) first and \((r_1,r_2)\) second; these figures show the petals and buds found after DDFS is performed on the first and second bridge, respectively. Observe that \( b \) does not belong to the first petal but it does belong to the second petal. The blossom is the union of both petals and its base is \( f \). The reader is encouraged to work out the petals if DDFS is performed on these bridges in the reverse order.

**5.3. The Mapping from Graph \( G \) to \( H \)** We will give a succinct description of this mapping here; for a more in-depth treatment, see Sections 8.2.1 and 8.3.1. Each time DDFS is called, a new directed graph \( H \) is defined. It is a function of the bridge that triggers the current DDFS and the
petals which have been found so far. A well-chosen subset of the vertices of \( G \) will form the vertices of \( H \). For each vertex \( v \) of \( G \) that is chosen, its name in \( H \) will be \( v_H \) and we will define its level, \( l(v_H) = \text{minlevel}(v) \).

Assume that DDFS is called with bridge \((r, g)\). Then \( H \) must have the two vertices \( \text{bud}*(r) \) and \( \text{bud}*(g) \). The rest of \( H \) is recursively defined as follows. If \( \text{minlevel}(v) > 0 \) then corresponding to each predecessor \( u \) of \( v \) in \( G \), \( H \) has the vertex \( \text{bud}*(u) \) and the directed edge \((v, \text{bud}*(u))\). If \( \text{minlevel}(v) = 0 \) then \( l(v_H) = 0 \) and \( v_H \) has no outgoing edges. It is easy to confirm that \( H \) satisfies the DDFS Requirement. For details see Sections 8.2.1 and 8.3.1.

**Example 9.** In the graph of Figure 13, DDFS called with bridge \((u, v)\) ends in Case 2: the two centers of activity terminate at distinct unmatched vertices, \( f_1 \) and \( f_2 \). This indicates the presence of a minimum length augmenting path between \( f_1 \) and \( f_2 \). The next task is to find such a path.

![Figure 13](image.png)

**Figure 13.** DDFS on the bridge \((u, v)\) terminates with two unmatched vertices, \( f_1 \) and \( f_2 \), leading to an augmentation.

![Figure 14](image.png)

**Figure 14.** The tenacities of all four bridges is indicated. Bridge \((a, b)\) has empty support.

**Example 10.** All bridges considered so far had non-empty support. However, this will not be the case in a typical graph, e.g., consider bridge \((a, b)\) of tenacity 17, in Figure 14. Clearly the support of this bridge is \( \emptyset \). DDFS will discover this right away since the two endpoints of this bridge have the same \( \text{bud}^* \), i.e. \( \text{bud}^*(a) = \text{bud}^*(b) \). Whether a bridge has empty support is not known a priori — it will become clear only after DDFS is performed on this bridge. Therefore, DDFS needs to be performed on every one of the bridges.

\(^8\) It is possible that \( \text{bud}^*(r) = \text{bud}^*(g) \). This happens if the bridge \((r, g)\) has empty support and therefore a new petal is not formed, e.g., see Example 10.
5.4. Finding Augmenting Paths  

MAX will find augmenting paths during search level \( j_m \), where \( l_m = 2j_m + 1 \) and \( l_m \) is the length of a minimum length augmenting path in the current phase. However, not every bridge of tenacity \( l_m \) leads to an augmentation — DDFS on a bridge of tenacity \( l_m \) can end in Case 1, i.e., a bottleneck, as well.

![Diagram](image)

**Figure 15.** Constructing a minimum length augmenting path between unmatched vertices \( f_1 \) and \( f_2 \).

5.4.1. Finding one augmenting path  

In Figure 15, \( \text{minlevel}(u) > \text{minlevel}(v) \) and therefore edge \((u, v)\) is an anomalous bridge. When DDFS is performed on this bridge, assume that the red DFS trees has root \( u \). Since \( v \) is already in a petal with \( \text{bud}^*(v) = b \), the green DFS tree will have root \( b \). The two trees will simply follow predecessors and will terminate at \( f_1 \) and \( f_2 \), respectively.

A DFS from \( u \) in the red tree will yield a path from \( u \) to \( f_1 \), say \( p_1 \). Since \( v \) is in a petal with \( \text{bud}^*(v) = b \), the algorithm needs to find an alternating path from \( v \) to \( b \), say \( p_2 \), starting with a matched edge, i.e., of even length. The construction of this path is described below\(^9\). Additionally, the algorithm needs to find a path, say \( p_3 \), from \( b \) to \( f_2 \) in the green tree of the DDFS performed on bridge \((u, v)\). Then, the complete augmenting path from \( f_1 \) to \( f_2 \) will be \( p_1^{-1} \circ (u, v) \circ p_2 \circ p_3 \). Clearly, \( p_1 \) and \( p_3 \) are easy to find.

\(^9\) In the language of Definition 20, \( p_2^{-1} \) is an evenlevel(b; v) path.
We next describe how to find \( p_2 \). The algorithm observes that evenlevel(\( v \)) = maxlevel(\( v \)) and therefore \( p_2 \) must use the bridge of the petal containing \( v \). Using the petal node, the algorithm finds the endpoints of this bridge, namely \( c \) and \( d \). It notices that \( c \) and \( v \) have the same color, say red. Therefore, it looks for a path from \( c \) to \( v \) in the red tree and a path from \( d \) to \( b \) in the green tree. For finding the latter path, it jumps from \( w \) to \( \text{bud}^*(w) = a \) and then follows predecessors in the green tree till it reached \( b \).

To find the complete path from \( d \) to \( b \), the algorithm must find a path from \( a \) to \( w \) in the smaller petal\(^10\). This time, it observes that evenlevel(\( w \)) = minlevel(\( w \)) and therefore the required path does not use the bridge of the smaller petal. Instead it is found by doing a DFS in the green tree, assuming that the color of \( w \) was green in the DDFS conducted on bridge \( (w, w') \). Then \( p_2 \) is obtained by concatenating the path from \( v \) to \( c \) with \( (c, d) \) with the path from \( d \) to \( b \). The latter consists of \( (d, w) \) concatenated with the path from \( w \) to \( a \) concatenated with the path from \( a \) to \( b \).

![Figure 16](image_url)  
**Figure 16.** The evenlevel(\( v \)) path, which starts at \( f \), followed by edge \((v, f')\) is an augmenting path.

**Example 11.** In Figure 16, we have added one edge to the graph of Figure 5, namely \((v, f')\) and the unmatched vertex \( f' \). The evenlevel(\( v \)) path, which starts at \( f \), followed by edge \((v, f')\) yields the unique augmenting path in this figure. Therefore, finding evenlevel(\( v \)) in Figure 5 was not just an academic matter. However, the MV algorithm will not find this path by following the

\(^{10}\) Again, in the language of Definition 20, this is an evenlevel(\( a; w \)) path.
arrows in the figure; instead it will use the bridges and blossoms as detailed above. In particular, the bridge of tenacity \( \gamma = 17 \), as shown in Figure 7, triggers a DDFS which will find this path by skipping over the blossom of tenacity \( \beta = 15 \).

5.4.2. Finding a maximal set of disjoint paths After the first path, say \( p \), is found, its vertices are removed. As a result, there may be other vertices that cannot be on minimum length augmenting paths that are disjoint from \( p \). These vertices are recursively removed using the procedure \textsc{RECURSIVE \text{\textsc{REMOVE}}} which works as follows: A vertex \( v \) needs to be removed by procedure \textsc{RECURSIVE \text{\textsc{REMOVE}}} if it is matched and it has no predecessors in the current graph or if \( v \) is unmatched and is isolated.

At this point \textsc{MAX} will process the next bridge of tenacity \( l_m \). When it encounters another bridge which makes DDFS terminate in Case 2, it finds another augmenting path. This continues until all bridges of tenacity \( l_m \) are processed. Lemma 18 shows that this will result in a maximal set of paths of length \( l_m \).

6. Relationship between Tenacity of an Edge and Tenacities of its Endpoints The relationship depends on whether the edge is matched or unmatched, and in the latter case, whether it is a prop or a bridge. The answer in each case is significant and will determine how to carry out the proofs of the main theorem, Theorem 3, in Sections 8.2 and 8.3, and to establish facts showing correct synchronization of events, in Section 9.1.

\textbf{Lemma 1.} Let \( (u, v) \) be a matched edge of finite tenacity. Then \( \text{evenlevel}(v) = \text{oddlevel}(u) + 1 \) and \( \text{tenacity}(v) = \text{tenacity}(u, v) \). Furthermore, \( (u, v) \) is a bridge if and only if \( \text{oddlevel}(u) = \text{oddlevel}(v) = i \), where \( \text{tenacity}(u, v) = 2i + 1 \).

\textbf{Proof :} Let \( p \) be an oddlevel(\( u \)) path. If \( v \) lies on \( p \) then \( (u, v) \) must also lie on \( p \), since \( p \) is an alternating path. If \( p \) is not a simple odd length alternating path to \( u \), giving a contradiction. Therefore, \( v \) does not lie on \( p \) and hence \( p \circ (u, v) \) is a minimum even length alternating path to \( v \). Therefore, \( \text{evenlevel}(v) = \text{oddlevel}(u) + 1 \) and similarly \( \text{evenlevel}(u) = \text{oddlevel}(v) + 1 \). Therefore \( \text{tenacity}(v) = \text{evenlevel}(v) + \text{oddlevel}(v) = \text{oddlevel}(u) + 1 + \text{oddlevel}(v) = \text{tenacity}(u, v) = \text{evenlevel}(u) + \text{oddlevel}(u) = \text{tenacity}(u) \).

Now, there are two cases, either one of \( u \) or \( v \) has an oddlevel of \( < i \) or oddlevel(\( u \)) = oddlevel(\( v \)) = \( i \). In the first case, assume oddlevel(\( u \)) = \( i \). If so, \( u \) is a predecessor of \( v \) and \( (u, v) \) is a prop. In the second case, neither endpoint is a predecessor of the other and \( (u, v) \) is a bridge. Finally, if \( (u, v) \) is a bridge, the first case cannot apply. Therefore oddlevel(\( u \)) = oddlevel(\( v \)) = \( i \).

As a result of Lemma 1, in several proofs given in this paper, it will suffice to restrict attention to only one of the end points of a matched edge \( (u, v) \), since given evenlevel(\( v \)) and oddlevel(\( v \)), the levels of \( u \) are fully determined.

\textbf{Lemma 2.} Let \( (u, v) \) be an unmatched edge of finite tenacity and assume that \( u \) is a predecessor of \( v \). Then \( \text{tenacity}(v) = \text{tenacity}(u, v) \).

\textbf{Proof :} We will show that oddlevel(\( v \)) = evenlevel(\( u \)) + 1. If so, tenacity(\( v \)) = evenlevel(\( v \)) + oddlevel(\( v \)) = evenlevel(\( v \)) + evenlevel(\( u \)) + 1 = tenacity(\( u, v \)), thereby proving the lemma. Since \( v \) is getting its minlevel from the unmatched edge \( (u, v) \), minlevel(\( v \)) = oddlevel(\( v \)) and therefore \( v \) is an inner vertex.

Since \( u \) is a predecessor of \( v \), there is an oddlevel(\( v \)) path, say \( q \), that ends with the edge \( (u, v) \). Now, oddlevel(\( v \)) = |\( q \)| and evenlevel(\( v \)) ≤ \( q \) − 1. Therefore, oddlevel(\( v \)) > evenlevel(\( u \)).

Let \( p \) be an evenlevel(\( u \)) path. First assume that \( v \) lies on \( p \). If \( v \) is odd w.r.t. \( p \), then oddlevel(\( v \)) < evenlevel(\( u \)), a contradiction. Therefore, \( v \) is even w.r.t. \( p \). But then evenlevel(\( v \)) <

\footnote{For this notion, see Definition 6.}
evenlevel(u) implying that evenlevel(v) < oddlevel(v) and that v is an outer vertex, another contradiction. Therefore v does not lie on p. Therefore p ◦ (u, v) is an oddlevel(v) path and therefore oddlevel(v) = evenlevel(u) + 1 holds, giving the lemma. □

**Lemma 3.** Let (u, v) be an unmatched bridge. Then:
1. tenacity(v) ≤ tenacity(u, v).
2. If tenacity(v) = tenacity(u, v) then v is an outer vertex.

**Proof:** Let p be an evenlevel(u) path starting at unmatched vertex f, say. If v lies on p, there are two cases. If v is even w.r.t. p, then p[f to u] ◦ (u, v) is an odd length alternating path from f to v of length at most evenlevel(u). Therefore, oddlevel(v) < evenlevel(u). If v is odd w.r.t. p, then p[f to v] is an odd length alternating path from f to v, again leading to the same conclusion. Next, assume that v does not lie on p. Then, p ◦ (u, v) is an odd length alternating path from f to v. Therefore oddlevel(v) ≤ evenlevel(u) + 1 in all cases. Adding evenlevel(v) to both sides of this inequality we get tenacity(v) ≤ tenacity(u, v). As a consequence, we get that minlevel(v) ≤ i, where tenacity(u, v) = 2i + 1.

Next assume that tenacity(v) = tenacity(u, v). Then oddlevel(v) = evenlevel(u) + 1. If v were an inner vertex, then oddlevel(v) = minlevel(v) and the previous equality implies that (u, v) is a prop, leading to a contradiction. Therefore v is an outer vertex and evenlevel(v) = minlevel(v); clearly, evenlevel(v) ≤ i. □

**Remark 1.** Let (u, v) be an unmatched edge and let u be a predecessor of v. Then tenacity(u) can be smaller than, equal to, or bigger than tenacity(u, v). The first case is illustrated by prop (r_1, l_2) and the third case by the props out of f in Figure 10. Let (u, v) be an unmatched bridge and assume that tenacity(v) < tenacity(u, v). Then v can be an inner or an outer vertex. The bridge (a, b) in Figure 14 illustrates both possibilities; although (a, b) has empty support, it is easy to show this for bridges with non-empty support as well.

**7. Limited BFS-Honesty** Consider a minimum length alternating path, p, from unmatched vertex f to a vertex v; p is allowed to be of either parity. The notion of tenacity enables us to characterize a subset of vertices of p that are BFS honest on p, namely all vertices on p whose tenacity is at least as large as that of v. This BFS-honesty will be critically exploited later.

**Definition 13.** (even/odd w.r.t. p) Let p be an evenlevel(v) or oddlevel(v) path starting at unmatched vertex f and let u lie on p. Then |p[f to u]| will denote the length of this path from f to u, and if it is even (odd) we will say that u is even (odd) w.r.t. p.

**Definition 14.** (BFS honesty on p) Let p be an evenlevel(v) or oddlevel(v) path starting at unmatched vertex f and let u lie on p. We will say that u is BFS honest on p if |p[f to u]| = evenlevel(u) (oddlevel(u)) if u is even (odd) w.r.t. p.

**Example 12.** Observe that the graphs of Figures 5 and 6 are identical, with vertex names given in the former and vertex tenacities in the latter. The vertices u and v are BFS honest on all evenlevel and oddlevel paths to the vertices of tenacity α. However, the two vertices of tenacity α that lie on the evenlevel(u) and evenlevel(v) paths are not BFS honest on these paths.

**Theorem 2.** Let p be an evenlevel(v) or oddlevel(v) path starting at unmatched vertex f and let vertex u ∈ p with tenacity(u) ≥ tenacity(v). Then u is BFS honest on p. Furthermore, if tenacity(u) > tenacity(v) then |p[f to u]| = minlevel(u).

**Proof:** Assume w.l.o.g. that p is an evenlevel(v) path and that u is even w.r.t. p (by Lemma 1). Suppose u is not BFS honest on p, and let q be an evenlevel(u) path, i.e., |q| < |p[f to u]|. First consider the case that evenlevel(v) = maxlevel(v), and let r be a minlevel(v) path. Let u' be the matched neighbor of u. Consider the first vertex of r that lies on p[u' to v]. If this vertex
is even w.r.t. \( p \) then oddlevel(\( u \)) \( \leq |r| + |p[u \to v]| \). Additionally, evenlevel(\( u \)) \( < |p[f \to u]| \), hence tenacity(\( u \)) \( < \) tenacity(\( v \)), leading to a contradiction. On the other hand, if this vertex is odd w.r.t. \( p \) then minlevel(\( v \)) \( = |r| > \) evenlevel(\( u \)), because otherwise there is a shorter even path from \( f \) to \( v \) than evenlevel(\( v \)). We combine the remaining argument along with the case that evenlevel(\( v \)) \( = \) minlevel(\( v \)) below.

Consider the first vertex, say \( w \), of \( q \) that lies on \( p(u \to v) \) – there must be such a vertex because otherwise there is a shorter even path from \( f \) to \( v \) than evenlevel(\( v \)). If \( w \) is odd w.r.t. \( p \) then we get an even path to \( v \) that is shorter than evenlevel(\( v \)). Hence \( w \) must be even w.r.t. \( p \). Then, \( q[f \to w] \circ p[w \to u] \) is an odd path to \( u \) with length less than evenlevel(\( v \)), where \( \circ \) denotes the concatenation operator. Again we get tenacity(\( u \)) \( < \) tenacity(\( v \)), leading to a contradiction.

We next prove the second claim of the theorem. First consider the case that evenlevel(\( v \)) \( = \) minlevel(\( v \)), and assume for contradiction that \( |p[f \to u]| = \) maxlevel(\( u \)). Then tenacity(\( u \)) \( < 2 \cdot \) maxlevel(\( u \)) \( < 2 \cdot |p[f \to v]| = 2 \cdot \) minlevel(\( v \)) \( < \) tenacity(\( v \)), a contradiction.

Therefore, evenlevel(\( v \)) \( = \) maxlevel(\( v \)). As before, let \( r \) be a minlevel(\( v \)) path, and consider the first vertex of \( r \) that lies on \( p[u \to v] \). If this vertex is even w.r.t. \( p \) then oddlevel(\( u \)) \( \leq |r| + |p[u \to v]| \). Hence tenacity(\( u \)) \( \leq \) tenacity(\( v \)), which leads to a contradiction. On the other hand, if this vertex is odd w.r.t. \( p \) then minlevel(\( v \)) \( = |r| > \) evenlevel(\( u \)), because otherwise there is a shorter even path from \( f \) to \( v \) than evenlevel(\( v \)). Now the claim follows because otherwise tenacity(\( u \)) \( < \) tenacity(\( v \)).

\[ \square \]

**Corollary 1.** Let \( p \) be an evenlevel(\( v \)) or oddlevel(\( v \)) path and let \( u \) lie on \( p \). If \( u \) is not BFS honest on \( p \) then tenacity(\( u \)) \( < \) tenacity(\( v \)).

### 8. Base, Blossom and Bridge

Recall Definitions 2 and 8 which defined \( l_m \) and \( t_m \) as the length of a minimum length augmenting path and the tenacity of a minimum tenacity vertex, respectively. As noted earlier, \( t_m \leq l_m \) and the case \( t_m = l_m \) is trivial. For the rest of the paper, we will deal with the main case, namely \( t_m < l_m \); Example 14 explains the importance of this assumption. In Definition 15, we introduce the notion of eligible tenacity. Theorem 3 helps establish the central notions of base and blossom for vertices of eligible tenacity.

**Definition 15.** (Eligible tenacity) An odd number \( t \), with \( t_m \leq t < l_m \), will be said to be an eligible tenacity.

**Definition 16.** (Higher and lower on a path) Let \( v \) be a vertex and \( p \) be an evenlevel(\( v \)) or oddlevel(\( v \)) path; assume it starts at unmatched vertex \( f \). If \( u \) and \( w \) are two vertices on \( p \) and if \( u \) is further away from \( f \) on \( p \) than \( w \), then we will say that \( u \) is higher than \( w \) and \( w \) is lower than \( u \) on \( p \).

**Definition 17.** (The set \( B(v) \)) Let \( v \) be a vertex of eligible tenacity and \( p \) be an evenlevel(\( v \)) or oddlevel(\( v \)) path starting at unmatched vertex \( f \). Let \( t = \) tenacity(\( v \)) and consider all vertices of tenacity \( > t \) on \( p \); clearly, this set contains \( f \). Among these vertices, define the highest one to be the base of \( v \) w.r.t. \( p \), denoted \( F(p,v) \). Clearly, \( F(p,v) \) is even w.r.t. \( p \), and by Theorem 2, it is an outer vertex. Finally define:

\[
B(v) = \{ F(p,v) \mid p \text{ is an evenlevel(} v \text{) or oddlevel(} v \text{) path}\}.
\]

#### 8.1. Central Structural Facts

As mentioned in Section 1.1, the central structural fact needed to prove correctness of the algorithm is that for every vertex \( v \) such that tenacity(\( v \)) \( = t \) and \( t_m \leq t \leq l_m \), every maxlevel(\( v \)) path contains a bridge of tenacity \( t \). The proof of this fact requires several other structural facts. Among them, the most important one is that for a vertex \( v \) of eligible tenacity, the set \( B(v) \) is a singleton. Once this fact is proven, we can define the base of \( v \) to be the vertex in \( B(v) \) and move on to defining the notion of a blossom and proving its properties.
However, the proof of this fact is not straightforward because of the following “chicken and egg problem”. On the one hand, the proof of this fact requires the notion of blossom and its associated properties. On the other hand, blossoms can be defined only after defining the base of a vertex, i.e., after proving this fact.

We will break this deadlock by proving this fact via an induction on tenacity: for each value of tenacity, say \( t \). Once this fact is proven for vertices of tenacity \( \leq t \), the base of vertices of tenacity \( \leq t \) can be defined. Following this, blossoms of tenacity \( t \) can be defined and properties of these blossoms and properties of paths traversing through these blossoms can be established. These properties are then used in the next step of the induction to prove this fact for the next higher value of tenacity.

### 8.2. The Induction Basis

This section is devoted to proving the induction basis for Theorem 3. This involves proving all statements mentioned in Theorem 3 for the case \( t = t_m \); each statement is proven in a separate lemma. For this purpose we will define a subgraph of \( G \), namely \( H_m' \). Its structure is fairly simple, thereby making the proofs easy. In contrast, the analogous graph to be handled for the induction step is considerably more complex. The saving grace is that the proofs for the base case provide much insight on how to proceed with the induction step.

**LEMMA 4.** Let \((u, v)\) be an unmatched edge of tenacity \( t_m \). Then \((u, v)\) is a bridge if and only if \( \text{minlevel}(u) = \text{minlevel}(v) = i \), where \( t_m = 2i + 1 \).

**Proof:**  Since \((u, v)\) is unmatched, \( \text{tenacity}(u, v) = \text{evenlevel}(u) + \text{evenlevel}(v) + 1 \). First assume \((u, v)\) is a bridge. Since the tenacity of a vertex cannot be less than \( t_m \), by Lemma 3, \( \text{tenacity}(u) = \text{tenacity}(v) = \text{tenacity}(u, v) \), and \( u \) and \( v \) are both outer vertices. Therefore, \( \text{minlevel}(u) = \text{evenlevel}(u) \) and \( \text{minlevel}(v) = \text{evenlevel}(v) \). Therefore, \( \text{oddelevel}(v) = \text{evenlevel}(u) + 1 \) and \( \text{oddelevel}(u) = \text{evenlevel}(v) + 1 \).

Since \( t_m = 2i + 1 \), \( \text{minlevel}(u) = \text{evenlevel}(u) \leq i \) and \( \text{minlevel}(v) = \text{evenlevel}(v) \leq i \). Assume one of them has \( \text{minlevel} < i \), say \( u \). Then, \( \text{oddelevel}(v) = \text{evenlevel}(u) + 1 \leq i \). This implies that \( \text{evenlevel}(v) < i \), since \( v \) is outer, giving \( \text{tenacity}(u) < 2i \), a contradiction. Therefore, \( \text{minlevel}(u) = \text{minlevel}(v) = i \).

Next, assume \( \text{minlevel}(u) = \text{minlevel}(v) = i \). Since \((u, v)\) is an unmatched edge of tenacity \( t_m \), \( \text{evenlevel}(u) + \text{evenlevel}(v) + 1 = t_m \). Therefore \( \text{evenlevel}(u) = \text{evenlevel}(v) = i \). Now, since \((u, v)\) unmatched, \( u \) is not a predecessor of \( v \) and \( v \) is not a predecessor of \( u \). Therefore \((u, v)\) is a bridge. \( \square \)

If \((u, v)\) is a matched edge of tenacity \( t_m \), an analogous statement to Lemma 4 holds by Lemma 1.

**LEMMA 5.** Let \( v \) be a vertex of tenacity \( t_m \) and \( p \) be a maxlevel\((v)\) path. Then \( p \) contains a unique bridge of tenacity \( t_m \).

**Proof:** Assume that \( p \) starts at unmatched vertex \( f \). Let \( q \) be a minlevel\((v)\) path and assume it starts at unmatched vertex \( f' \), which may or may not be the same as \( f \). If \( p \) and \( q \) meet only at \( v \) then \( f \neq f' \) and \( p \circ q \) is an augmenting path of length \( t_m \), leading to a contradiction. Therefore the intersection of \( p \) and \( q \) contains vertices in addition to \( v \). Let \( u \) be the highest vertex of \( p[f \text{ to } v] \) that is also on \( q \). Since \( q \) must contain the matched edge incident at \( u \), the vertex \( u \) must be even w.r.t. \( p \). By definition, \( \text{tenacity}(u) \geq t_m \), and therefore by Theorem 2, \( u \) is BFS honest on \( p \) as well as \( q \). Therefore, \( \text{evenlevel}(u) = |p[f \text{ to } u]| \). If \( u \) is odd w.r.t. \( q \), then \( \text{oddelevel}(u) = |q[f' \text{ to } u]| \) thereby implying that \( \text{tenacity}(u) < |p \circ q| = t_m \), a contradiction. Therefore \( u \) is even w.r.t. \( q \) as well and \( |q[f' \text{ to } u]| = \text{evenlevel}(u) \).
Now, $p[u \to v] \circ q[v \to u]$ is an odd length cycle having two unmatched edges incident at $u$ and is fully matched otherwise. By concatenating $p[1 \to u]$ to an appropriate subpath of this cycle, we can obtain even and odd alternating paths to all vertices of this cycle, other than $u$. For any vertex $w \neq u$ on this cycle, the sum of the lengths of the even and odd paths is $t_m$. Therefore, these must be minimum length alternating paths and tenacity($w$) = $t_m$. Furthermore, since this cycle has length at least 3, evenlevel($u$) = $|p[1 \to u]| < i$.

Assume that this odd cycle has length $2k + 1$ and number its edges consecutively, starting at $u$. Let $(w, w')$ be the $k + 1$st edge, i.e., the middle edge. Then clearly, minlevel($w$) = minlevel($w'$) = i. Therefore, by Lemma 4, $(w, w')$ is a bridge of tenacity $t_m$. Clearly, besides $w$ and $w'$, no other vertices on $p$ can have minlevel of i. Therefore there are no other bridges of tenacity $t_m$ on it. □

Note that in the proof given above, $v \in \text{support}(w, w')$. In general, $v$ may lie in the support of several bridges of tenacity $t_m$.

8.2.1. The Graphs $H_m$ and $H'_m$ Proving that the set $B(v)$ is a singleton is not straightforward even for vertices of tenacity $t_m$. A major simplification is achieved by using the power of DDFS — in particular, the DDFS Certificate provided on its termination. DDFS is carried out on a special directed, layered graph $H_m$, which satisfies the DDFS Requirement. A closely related graph, $H'_m$, will also be defined; it is a subgraph of $G$. We will use the information obtained from DDFS on $H_m$ to find minlevel and maxlevel paths in $H'_m$. The mapping between the vertices of $H_m$ and $H'_m$ will be obvious from the definitions given below.

Let $U_m = \{v \in V \mid \text{tenacity}(v) = t_m\}$. Consider all vertices $v \in U_m$ such that minlevel($v$) = $i$, where $t_m = 2i + 1$, and let $p$ denote an arbitrary minlevel($v$) path. Denote by $V_m$ the union of all vertices on all such paths $p$ and denote by $P_m$ the union of all edges on all such paths $p$. Observe that all edges in $P_m$ are props in the original graph $G$. For each edge $(u, v) \in P_m$, direct it from $v$ to $u$ if $u$ is a predecessor of $v$; clearly, $u$ is pred $v$. If as a result, there is a directed path from $v$ to $w$, then $w$ is pred* $v$; these terms are defined in Definition 10. To keep the notation simple, we will denote this set of directed edges by $P_m$ as well; the context will easily clarify which graph an edge belongs to.

We will partition $V_m$ into $i + 1$ layers numbered $0, \ldots, i$, where the layer number of $v \in V_m$ is $l(v) = \text{minlevel}(v)$. Let $H_m$ denote the directed layered graph $H_m = (V_m, P_m)$. It is easy to check that $H_m$ satisfies the DDFS Requirement.

Let $B_m$ denote the set of all bridges of tenacity $t_m$ and define undirected graph $H'_m = (V_m, (P_m \cup B_m))$. Define a matching in $H'_m$ as follows: edge $e \in (P_m \cup B_m)$ is matched if and only if $e$ is matched in $G$. Clearly, $H'_m$ is a subgraph of $G$. The relations pred and pred* given in Definition 10 carry over to graph $H'_m$ as well.

**Lemma 6.** For every $v$ in $U_m$, $H'_m$ contains all possible evenlevel($v$) and oddlevel($v$) paths that are present in $G$.

**Proof:** By definition, $H'_m$ contains all possible minlevel($u$) paths for all vertices $u$ such that minlevel($u$) = $i$. Since it contains all edges of $B_m$, it contains all possible maxlevel($u$) paths as well, for these vertices.

Let $v \in U_m$ with minlevel($v$) < $i$ and let $p$ be a minlevel($v$) path in $G$. We will show that path $p$ is present in $H'_m$. Pick any maxlevel($v$) path in $G$, say $q$. By Lemma 4, there is a unique bridge of tenacity $t_m$ on $q$, say $(u', v')$. Assume that $u'$ is higher than $v'$ on $q$; for definition of “higher on a path” see Definition 16. Since $p \circ q[v \to u']$ is a minlevel($u'$) path, it is present in $H'_m$. Therefore, $H'_m$ contains $p$. For similar reasons, $H'_m$ contains all possible maxlevel($v$) paths that are present in $G$. □

Corresponding to each bridge $(u, v) \in B_m$, conduct DDFS in $H_m$ starting at $u$ and $v$, and denote by $k_{u,v}$ the bottleneck found. Clearly, each bottleneck is an outer vertex. Note that two different
bridges may have the same bottleneck. Using the DDFS Certificate and the mapping between $H_m$ and $H'_m$ we get:

**Lemma 7.** Let bridge $(u,v) \in B_m$ and let $w \in \text{support}(u,v)$. Then:
1. Consider minlevel($w$) and maxlevel($w$) paths which consist of an evenlevel($k_{(u,v)}$) path followed by an alternating path using vertices in support($u,v$). Then $H'_m$ contains all such paths.
2. $w = \text{pred}^*(u)$ or $w = \text{pred}^*(v)$ or both.
3. $k_{(u,v)} = \text{pred}^*(w)$.

**Procedure Bottleneck:** The input to this procedure is an edge $(u,v) \in B_m$.

Let $k_{(u,v)}$ be the bottleneck found when DDFS is conducted in $H_m$, starting at $u$ and $v$. If tenacity($k_{(u,v)}) > t_m$, HALT.

Otherwise, there is a bridge of tenacity $t_m$, say $(u',v')$, such that $k_{(u,v)} \in \text{support}(u',v')$. Conduct DDFS on bridge $(u',v')$ to find its bottleneck, $k_{(u',v')}$. Clearly, minlevel($k_{(u',v')} < \text{minlevel}(k_{(u,v)})$, and $k_{(u',v')} = \text{pred}^*(k_{(u,v)})$. If tenacity($k_{(u',v')} = t_m$ repeat this process until a bottleneck, say $b$, is encountered which has tenacity $> t_m$. This is bound to happen because the minlevels of the bottlenecks are decreasing and eventually the bottleneck will turn out to be an unmatched vertex, say $f$, and $f$ satisfies tenacity($f) \geq l_m > t_m$.

**Example 13.** Let us illustrate Procedure Bottleneck on the graph of Figure 18. When called with bridge $(u,u')$, the bottleneck found it $b'$. However, tenacity($b') = t_m = 15$ and $b'$ is in the support of $(v,v')$. DDFS on this bridge will result in the bottleneck of $b$. Since tenacity($b) > 15$, the procedure halts and $b$ is the base of the endpoints of both bridges.

The vertex $b$ identified by Procedure Bottleneck is very special, as will be established next. It is called a base; a formal definition is given below. Clearly, $b$ is an outer vertex. In general, $H'_m$ will have a number of bases.

For each base $b$ in $H'_m$, define the set

$S_{b,t_m} = \{v \in U_m \mid b \text{ is pred}^* \text{ of } v\}$.

**Lemma 8.** Let $v \in S_{b,t_m}$. Then every evenlevel($v$) and oddlevel($v$) path in $H'_m$ consists of an evenlevel($b$) path followed by an alternating path using vertices of $S_{b,t_m}$.

**Proof:** For each edge $(u,v) \in B_m$, let $k_{(u,v)}$ denote the bottleneck found by procedure DDFS when run on vertices $u$ and $v$. Define the set

$K_b = \{k_{(u,v)} \mid (u,v) \in B_m \text{ and } k_{(u,v)} \in S_{b,t_m}\}$.

Clearly, each vertex $y \in K_b$ is an outer vertex. By the definition of $S_{b,t_m}$, there is a path, in $H'_m$, from each vertex $y \in K_b$ to $b$ which follows down predecessors only; let us call this path $p_y$. Clearly, an evenlevel($b$) path concatenated with $p_y^{-1}$ is an evenlevel($y$) path. This fact combined with Lemma 7 completes the proof of the current lemma.

Finally, we prove that the statement of Lemma 8 is true in $G$ as well.

**Lemma 9.** Let $v \in S_{b,t_m}$. Then every evenlevel($v$) and oddlevel($v$) path in $G$ consists of an evenlevel($b$) path followed by an alternating path using vertices of $S_{b,t_m}$.

**Proof:** Suppose for some $v \in S_{b,t_m}$ there is an evenlevel($v$) or oddlevel($v$) path $p$ that does not contain $b$ on it. Consider the lowest vertex on $p$ that is in $S_{b,t_m}$ and let it be $u$. Now, using the same arguments as in Lemma 5, it is easy to show that if $u$ is an outer vertex, then tenacity($b) \leq t_m$, a contradiction, since tenacity($b) > t_m$. Next suppose that $u$ is an inner vertex. Since $u \in S_{b,t_m}$, there is a bridge $(u',v')$ of tenacity $t_m$ in whose support $u$ lies. Now by the DDFS Certificate, there are
disjoint paths from the two endpoints of this bridge to \( u \) and \( b \) yielding an odd alternating path from an unmatched vertex to \( b \), say \( q \), which is such that evenlevel\((b) + |q| = t_m \) implying that tenacity\((b) \leq t_m \), a contradiction. The lemma follows. \( \square \)

**Corollary 2.** For every \( v \in U_m \), the set \( B(v) \) is a singleton.

**Definition 18.** (The base of a vertex of tenacity \( t_m \) and basal vertices) For each \( v \in U_m \), define base\((v)\) to be the unique vertex, say \( b \), in the set \( B(v) \). We will say that the base of \( v \) is \( b \). Each such vertex \( b \) will be called a basal vertex. Clearly, \( b \) is an outer vertex and tenacity\((b) > t_m \).

Let \( b \) and \( b' \) be two distinct basal vertices. Since tenacity\((b') > t_m \), \( b \neq \text{pred}'(b') \). Now that the base of vertices of tenacity \( t_m \) is well-defined, we can define blossoms of tenacity \( t_m \). Note that the definition of blossoms of higher tenacity will be given later and will be more involved.

**Example 14.** In the graph of Figure 17, vertices \( u \) and \( v \) do not have a base, even though they are of finite tenacity. Observe that tenacity\((u) = \text{tenacity}(v) = 3 \) and the evenlevel and oddlevel paths to these vertices do not contain any vertex of tenacity greater than 3. Since \( l_m = 3 \), \( u \) and \( v \) are not of eligible tenacity. If edge \((f_2, v)\) is removed, then one may argue that \( f_1 \) can be regarded as a basal vertex. We have added this edge in order to show that even such a claim cannot be made.

![Figure 17](image)

**Figure 17.** Vertices \( u \) and \( v \) have no base. The tenacity of bridge \((u, v)\) is indicated and is the same as the tenacities of \( u \) and \( v \).

**Definition 19.** (Blossom of tenacity \( t_m \) and base \( b \)) Let \( b \) be a basal vertex as defined in Definition 18. Then the blossom of tenacity \( t_m \) and base \( b \) is the set \( B_{b, t_m} = \{ v \in U_m \mid \text{base}(v) = b \} \).

Clearly, \( B_{b, t_m} \neq \emptyset \), \( B_{b, t_m} = S_{b, t_m} \) and \( b \notin B_{b, t_m} \).

**Example 15.** Blossoms of tenacity \( t_m \) can be quite complex, as illustrated in Figure 18, even though they do not contain nested blossoms. This blossom will have two petals, see Section 5.2.
for this notion. The exact petals will depend on the order in which DDFS is conducted on bridges $(u, u')$ and $(v, v')$.

**Definition 20.** (Shortest path from a base to a vertex of tenacity $t_m$) Let $v \in U_m$ and $b = \text{base}(v)$. Then by an evenlevel$(b; v)$ (oddlevel$(b; v)$) path we mean a minimum even (odd) length alternating path in $G$ from $b$ to $v$ that starts with an unmatched edge.

![Diagram of a blossom with vertices $u, u', v, v'$ and edges connecting them in a tree-like structure.](image)

**Figure 18.** The set of vertices having base $b$ form a blossom of tenacity 15 and $t_m = 15$. Observe that $b'$ is not basal. This blossom contains the endpoints of bridges $(u, u')$ and $(v, v')$.

**Lemma 10.** Let $v \in U_m$ and $b = \text{base}(v)$. Let $p$ be an evenlevel$(b)$ path and $q$ be an evenlevel$(b; v)$ or oddlevel$(b; v)$ path. Then $q$ meets $p$ at $b$ only.

**Proof:** By Lemma 1, it suffices to prove the lemma for an oddlevel$(b; v)$ path $q$. For contradiction, assume that $q$ meets $p$ at vertices besides $b$. By Lemma 9, oddlevel$(v) \geq |p| + |q|$. We will define certain subpaths of $q$ as segments as follows. Follow along $q$ from $b$ until it meets $p$, at $w$ say. Then $q[b \text{ to } w]$ will be called a segment. Subsequent to this, each time $q$ leaves $p$, at vertex $r$, say, and meets up $p$ again, at $s$, say, then $q[r \text{ to } s]$ is called a segment. Eventually, $q$ leaves $p$ at $y$, say, and ends up at $v$. Then, $q[y \text{ to } v]$ is the last segment.

Now, there are two cases: either there is a segment, say $q[u \text{ to } w]$ such that $u$ and $w$ are both outer vertices or there is no such segment. In the first case, consider the odd length cycle $q[u \text{ to } w] \circ p[w \text{ to } u]$ and assume that $u$ is higher than $w$ on $p$. Then, all vertices on this cycle, other than $w$ have tenacity $< t_m$, a contradiction.

In the second case, since the first segment starts at an outer vertex, namely $b$, it ends at an inner vertex. Therefore, the next segment again starts at an outer vertex and so on. Finally, for the last segment, $q[y \text{ to } v]$, $y$ must be an outer vertex. Assume that $p$ starts at unmatched vertex $f$. If so, $p[f \text{ to } y] \circ q[y \text{ to } v]$ is a shorter odd alternating path from $f$ to $v$ than $|p| + |q|$, leading to another contradiction. The lemma follows. □

Lemmas 9 and 10 give:
**Corollary 3.** Let \( v \in U_m \) and \( b = \text{base}(v) \). Then every evenlevel(\( v \)) (oddlevel(\( v \)) path consists of an evenlevel(\( b \)) path followed by an evenlevel(\( b \); \( v \)) (oddlevel(\( b \); \( v \)) path, where the latter lies in \( \{b\} \cup B_{b,t_m} \).

Lemma 9 and Corollary 3 give:

**Lemma 11.** Let \( B_{b,t_m} \) and \( B_{b',t_m} \) be two blossoms with bases \( b \neq b' \). Then \( B_{b,t_m} \cap B_{b',t_m} = \emptyset \).

Finally, we need to prove one more fact, which specifies the manner in which a minimum length alternating path to vertex \( v \), with tenacity(\( v \)) > \( t_m \), uses the vertices of a blossom of tenacity \( t_m \). This fact will be used in the induction step. Note that in Lemma 12, we are allowing \( b \) to appear either before or after \( u \) on path \( p \).

**Lemma 12.** Let \( p \) be an evenlevel(\( v \)) path from unmatched vertex \( f \) to an arbitrary vertex \( v \) such that there is a blossom \( B_{b,t_m} \) with \( p \cap B_{b,t_m} \neq \emptyset \). Then the base of this blossom, \( b \), also lies on \( p \) and there is a vertex \( u \in (p \cap B_{b,t_m}) \) such that \( p[b \text{ to } u] \) contains all the vertices of \( p \cap B_{b,t_m} \) and \( p[b \text{ to } u] \) is an evenlevel(\( b \); \( u \)) path. Furthermore, \( u \) is BFS honest on \( p \) if and only if \( b \) occurs before \( u \) on \( p \).

**Proof:** For the sake of contradiction, assume that \( p \) does not intersect \( B_{b,t_m} \cup \{b\} \) in the manner described above. If so, one of two cases holds:

1. \( b \) does not occur on \( p \).
2. \( b \) occurs on \( p \), but \( p \) enters and exits the set \( B_{b,t_m} \cup \{b\} \) more than once.
   
   For both cases, assume that \( q \) is an evenlevel(\( b \)) path starting at \( f \).
   
   **Case 1.** Assume that \( x \) and \( y \) are the first and last vertices of \( p \) in \( B_{b,t_m} \). There are three cases. If \( p[y \text{ to } v] \) does not intersect \( q \), then an evenlevel(\( y \)) path concatenated with \( p[y \text{ to } v] \) is shorter than \( p \) by Corollary 3. If \( p[y \text{ to } v] \) does intersect \( q \) and the last vertex of \( p \) on \( q \) is even w.r.t. \( q \), say \( w \), then an evenlevel(\( w \)) path concatenated with \( p[w \text{ to } v] \) is shorter than \( p \). Finally, if \( p[y \text{ to } v] \) does intersect \( q \) and the last vertex of \( p \) on \( q \) is odd w.r.t. \( q \), say \( w \), then \( p[f \text{ to } x] \circ r^{-1} \circ q[b \text{ to } w] \circ p[w \text{ to } v] \) is shorter than \( p \), where \( r \) is an evenlevel(\( b \); \( x \)) path.

**Case 2.** There are two cases: after visiting \( b \), \( p \) either enters or exits \( B_{b,t_m} \). In the first case, assume that \( y \) is the last vertex of \( p \) in \( B_{b,t_m} \). Then \( q \circ r \circ p[y \text{ to } v] \) is shorter than \( p \), where \( r \) is an evenlevel(\( b \); \( y \)) path. In the second case, assume that \( x \) is the first vertex of \( p \) in \( B_{b,t_m} \). Then \( p[f \text{ to } x] \circ r^{-1} \circ p[b \text{ to } v] \) is shorter than \( p \), where \( r \) is an evenlevel(\( b \); \( x \)) path.

**Definition 21.** (BFS honesty on \( p \) with respect to the base) Let \( p \) be an evenlevel(\( v \)) or oddlevel(\( v \)) path starting at unmatched vertex \( f \) to an arbitrary vertex \( v \), let \( u \) lie on \( p \) with tenacity(\( u \)) = \( t_m \) and let \( b = \text{base}(u) \). Then we will say that \( u \) is BFS honest on \( p \) w.r.t. \( b \) if \( p[b \text{ to } u] \) is an evenlevel(\( b \); \( u \)) (oddlevel(\( b \); \( u \)) path assuming \( |p[b \text{ to } u]| \) is even (odd)). Note that we are allowing \( b \) to appear either before or after \( u \) on path \( p \).

**Lemma 12** and Theorem 2 yield:

**Lemma 13.** Let \( p \) be an evenlevel(\( v \)) path from unmatched vertex \( f \) to an arbitrary vertex \( v \). Let \( w \in p \) with tenacity(\( w \)) = \( t_m \) and let \( b = \text{base}(w) \). Then \( b \) also lies on \( p \) and \( w \) is BFS honest on \( p \) w.r.t. \( b \). Furthermore, \( w \) is BFS honest on \( p \) if and only if \( b \) occurs before \( w \) on \( p \).

**Proof:** The crux of the matter is to identify the vertex \( u \) on \( p \) which plays the role of vertex \( u \) in Lemma 12, i.e., \( p[b \text{ to } u] \) contains all vertices in \( p \cap B_{b,t_m} \). By that lemma, \( p[b \text{ to } u] \) is an evenlevel(\( b \); \( u \)) path. By Lemma 9, there is an evenlevel(\( u \)) path that ends with the subpath \( p[b \text{ to } u] \). Since tenacity(\( w \)) ≥ tenacity(\( u \)), by Theorem 2 we get that the part of this path from \( f \) to \( w \) is

\[ \text{tenacity}(w) = \text{tenacity}(u); \] however, in the induction step, the analogous fact will involve an inequality and hence we have used an inequality here.\[12\]
a minimum alternating path to \( w \). Hence, by Corollary 3, \( w \) is BFS honest on \( p \) w.r.t. \( b \). The last statement also follows as in Lemma 12.

**Example 16.** In the graphs of Figures 20 and 21, \( u \) and \( u' \) are BFS honest w.r.t. \( b \) on the evenlevel\((v)\) path. Note that \( u \) and \( u' \) are BFS honest on the evenlevel\((v)\) path as well. A different kind of example is provided by the graph of Figure 23, in which \( u \) is not BFS honest on the evenlevel\((v)\) path; however it is BFS honest w.r.t. \( b \) on that path. These two types of examples also illustrate the last statement of Lemma 12, i.e., \( u \) is BFS honest on \( p \) if and only if \( b \) occurs before \( u \) on \( p \).

**Remark 2.** As a result of the induction basis, we have established that every vertex \( v \) of tenacity \( t_m \) has a unique base, base\((v)\). As stated in Section 8.1, the induction steps in Theorem 3, will enable us to establish an analogous fact about higher and higher tenacity vertices, eventually establishing it for every vertex of eligible tenacity. As a result, every such vertex \( w \) will have a unique base, base\((w)\), which is an outer vertex and satisfies tenacity\((\text{base}(w)) \geq \) tenacity\((w)\). For ease of exposition, Definition 22 assumes that base\((w)\) is well defined for each vertex \( w \) of eligible tenacity, in order to define the iterated bases of \( v \), where tenacity\((v) = t_m \). A more “correct”, though more cumbersome, way of doing this would be to define higher and higher iterated bases of \( v \) after each induction step.

**Definition 22.** (Iterated bases of a vertex of tenacity \( t_m \)) For \( v \in U_m \), let base\((v) = b \). Define the first iterated base of \( v \) to be \( b \), denoted as follows: base\(^1\)(\( v \)) = b. When base\((b)\), base\((\text{base}(b))\) etc. get defined in the induction step, we can define higher iterated bases of \( v \). Thus, for \( k \geq 1 \), we will say that base\(^{k+1}\)(\( v \)) = base\((\text{base}^k(v))\), assuming that base\(^k\)(\( v \)) and base\((\text{base}^k(v))\) exist in the graph.

### 8.3. The Induction Step

In this section, we will prove the induction step for Theorem 3; the basis of the induction was proved in Section 8.2.

**Induction Hypothesis:** Each of the statements in Theorem 3 holds for tenacities in the range \([t_m, t - 2]\), where \( t \) is odd and \( t_m + 2 \leq t < l_m \).

After proving Statement 2, certain key notions will become well-defined for the case of tenacity \( t \). These definitions are formally stated after the proof of Statement 2 and they will be used for formally stating and proving the rest of the statements.

These include the base of vertices of tenacity \( t \), blossoms of tenacity \( t \) and iterated bases of a vertex. Whereas the first two notions were defined in the induction basis as well, the third was not. The reason is the following. For a vertex \( v \) such that tenacity\((v) = t_m \), let base\((v) = b \). Since tenacity\((b) > t_m \), base\((b)\) was not yet defined and therefore it was not possible to talk about the iterated base, base\((\text{base}(v))\). In the induction step, we will need the following definition regarding the iterated bases of a vertex which have already been defined in the previous iterations of the induction.

**Definition 23.** Let \( t \) be an eligible tenacity\(^{13} \) and vertex \( v \) be such that let tenacity\((v) < t \). Define \( k(t, v) \) such that base\(^{k(t,v)}\)(\( v \)) is the first iterated base of \( v \) having tenacity at least \( t \), i.e.,

\[
k(t, v) = \arg \min \{l \mid \text{tenacity} (\text{base}^l(v)) \geq t \}.
\]

**Theorem 3.** Let \( t \) be an eligible tenacity, \( U_t = \{v \in V \mid \text{tenacity}(v) = t\} \) and \( v \in U_t \). The following hold:

**Statement 1:** Every maxlevel\((v)\) path contains a bridge of tenacity \( t \).

**Statement 2:** The set \( B(v) \) is a singleton.

\(^{13}\) See Definition 15.
**Statement 3:** Let \( v \in U_t \) and \( b = \text{base}(v) \). Then every evenlevel\((v)\) (oddlevel\((v)\)) path consists of an evenlevel\((b)\) path followed by an evenlevel\((b;v)\) (oddlevel\((b;v)\)) path, where the latter lies in \( \{b\} \cup B_{b,t} \).

**Statement 4:** The blossoms of tenacity \( t \) are disjoint and the set of blossoms of tenacity at most \( t \) form a laminar family.

**Statement 5:** Let \( p \) be an evenlevel\((v)\) path from unmatched vertex \( f \) to an arbitrary vertex \( v \). Let \( w \in p \) with tenacity\((w) = t \) and let \( b = \text{base}(w) \). Then \( b \) also lies on \( p \) and \( w \) is BFS honest on \( p \) w.r.t. \( b \). Furthermore, \( w \) is BFS honest on \( p \) if and only if \( b \) occurs before \( w \) on \( p \).

**Proof of Statement 1:** The proof is given in Lemma 14 and is very different from the proof of the analogous fact, given in Lemma 5, in the induction basis. The reason is that the former needs to account for blossoms defined in the previous iterations of the induction.

**Lemma 14.** Let \( v \) be a vertex of tenacity \( t \) and \( p \) be a maxlevel\((v)\) path. Then \( p \) contains a bridge of tenacity \( t \).

**Proof:** Assume that \( p \) starts at unmatched vertex \( f \). Define sets \( S_1, S_2 \) and \( S \) as follows:

\[
S_1 = \{w \mid w \text{ is on } p, \text{tenacity}(w) = t \text{ and } |p[f \text{ to } w]| = \text{maxlevel}(w)\}
\]

\[
S_2 = \{w \mid w \text{ is on } p, \text{tenacity}(w) < t \text{ and } \text{base}^{k(t,w)}(w) \text{ is higher than } w \text{ on } p\}
\]

\[
S = S_1 \cup S_2
\]

For a definition of “higher/lower on a path” see Definition 16. Let \( w \) be the vertex in \( S \) that is lowest on \( p \). Observe that if \( w \in S_2 \), then applying Statement 5 for the previous induction step to \( p \), we get that \( w \) must be odd with respect to \( p \). There are two cases:

**Case 1.** \( w \) is even with respect to \( p \). By the argument given above, \( w \notin S_2 \) and therefore \( w \in S_1 \) and \( \text{tenacity}(w) = t \). By Theorem 2, \(|p[f \text{ to } w]| = \text{maxlevel}(w)\). Let \( w' \) be the matched neighbor of \( w \). By Lemma 1, \( \text{tenacity}(w') = \text{tenacity}(w,w') = t \). By Theorem 2 and since \( w' \notin S \), \(|p[f \text{ to } w']| = \text{minlevel}(w')\). Now, since \( w' \) is odd with respect to \( p \) and \( w \) is even with respect to \( p \), \( w \) and \( w' \) are both inner vertices and their predecessors are given by unmatched edges incident at them. Therefore neither is \( w \) a predecessor of \( w' \) nor is \( w' \) a predecessor of \( w \). Hence \((w, w')\) is a bridge of tenacity \( t \).

**Case 2.** \( w \) is odd with respect to \( p \). Let \( w' \) be the matched neighbor of \( w \) and let \((u, w)\) be the unmatched edge on \( p \) incident at \( w \); \( u \) is lower than \( w \) by 1 on \( p \). Clearly \(|p[f \text{ to } u]|\) is even. We will show that \((u, w)\) is a bridge of tenacity \( t \).

First let us ascertain that \( w \) is not a predecessor of \( u \). Suppose \( \text{tenacity}(u) \geq t \). By Theorem 2, \(|p[f \text{ to } u]| = \text{minlevel}(u) = \text{evenlevel}(u)\). Therefore, the predecessor of \( u \) is its matched neighbor. If \( \text{tenacity}(u) < t \), then by Statement 5 of the previous induction step, \(|p[f \text{ to } u]| = \text{evenlevel}(u)\) and the predecessor of \( u \) lies in the blossom of tenacity \( t - 2 \) containing \( u \). Since \( u \notin S_2 \), the base of this blossom is lower than \( u \) on \( p \) and hence \( w \) does not lie in this blossom. Therefore, in all cases, \( w \) is not a predecessor of \( u \). Furthermore, we have shown that in all cases, \(|p[f \text{ to } u]| = \text{evenlevel}(u)\); we will use this fact below.

Next, let us ascertain that \( u \) is not a predecessor of \( w \). If \( \text{tenacity}(w) = t \), \( w \in S_1 \), and by Theorem 2, \(|p[f \text{ to } w]| = \text{maxlevel}(w)\). If \( \text{tenacity}(w) < t \), \( w \in S_2 \), and \( \text{base}^{k(t,w)}(w) \) is higher than \( w \) on \( p \). Now, the predecessor of \( w \) lies in the blossom whose base is \( \text{base}^{k(t,w)}(w) \). Therefore, in both cases, \( u \) is not a predecessor of \( w \).

\[14\text{ Note that } B_{b,t}, \text{ namely a blossom of tenacity } t \text{ and base } b, \text{ can be defined only after proving Statement 2; this is done in Definition 26.} \]
The two facts imply that \((u, w)\) is a bridge. Finally, we will show that tenacity\((u, w) = t\), thereby completing the proof. We will consider two cases. First assume that tenacity\((w) = t\). By Theorem 2, \(|p[f to w]| = \text{oddlevel}(w) = \text{evenlevel}(u) + 1\); the last equality follows from the fact that \(|p[f to u]| = \text{evenlevel}(u)\). Now,
\[
\text{tenacity}(u, w) = \text{evenlevel}(u) + \text{evenlevel}(w) + 1 = \text{evenlevel}(u) + (t - \text{oddlevel}(w)) + 1 = t.
\]

Next, assume that tenacity\((w) < t\); if so, as argued above, base\(^{k(t,w)}\)(\(w\)) is higher than \(w\) on \(p\). Let \(q\) be a \text{minlevel}(\(v\)) path from \(f\) to \(v\). Since tenacity\((v) = t\), \(|p| + |q| = t\). Let \(b' = \text{base}\(^{k(t,w)}\)(\(w\)). Then, \(q \circ p[v to b']\) is an \text{evenlevel}(\(b'\)) path and by Statement 5, \(q \circ p[v to w]\) is an \text{evenlevel}(\(w\)) path. Finally we get
\[
\text{tenacity}(u, w) = \text{evenlevel}(u) + \text{evenlevel}(w) + 1 = |p[f to u]| + |q \circ p[v to w]| + 1 = |p| + |q| = t.
\]

\[\square\]

\textbf{Remark 3.} As was done in Lemma 5, Lemma 14 can be strengthened to show uniqueness of the bridge as well. Since we will not need this fact, we will not prove it.

\textbf{8.3.1. The Graphs} \(H_t\) and \(H'_t\) For proving the induction step, we will define graphs \(H_t\) and \(H'_t\), which are analogous to \(H_m\) and \(H'_m\) defined in Section 8.2.1. Recall that \(U_t = \{v \in V \mid \text{tenacity}(v) = t\}\). Let \(B_t\) denote the set of all bridges of tenacity \(t\) and let \(W_t\) denote the end points of all bridges in \(B_t\). For \(v \in W_t\), define
\[
v^* = \begin{cases} 
v & \text{if tenacity}(v) = t, \\
\text{base}\(^{k(t,v)}\)(\(v\)) & \text{if tenacity}(v) < t
\end{cases}
\]

We now explain the second case of the definition of \(v^*\), i.e., if tenacity\((v) < t\). In this case, \(v\) is in a blossom of tenacity \(t - 2\) which must have been defined in the previous induction step. Now, \(v^*\) is meant to be the base of this blossom; it is given by \(\text{base}\(^{k(t,v)}\)(\(v\)).

Let \(W_t^* = \{v^* \mid v \in W_t\}\). For a vertex \(v^* \in W_t^*\), let \(p\) denote an arbitrary \text{minlevel}(\(v^*\)) path. Let \(V_t\) denote the set of all vertices of tenacity at least \(t\) on all \text{minlevel}(\(v^*\)) paths \(p\), for all vertices \(v^* \in W_t^*\). Observe that \(V_t\) contains the base of each blossom of tenacity \(t - 2\) contained in the following set:
\[
\{B_0 v^*, t - 2 \mid v \in W_t \text{ and tenacity}(v) < t\} \cup \{B_0 v^*, t - 2 \mid v \in V_t \text{ and tenacity}(v) = t\}.
\]

The next definition is related to Definition 10 and is motivated by Statement 5 of the induction hypothesis.

\textbf{Definition 24.} (The relations \text{pred}_t and \text{pred}^*_t) Let \(v \in V_t\) with tenacity\((v) = t\) and let \(u\) be a predecessor of \(v\); clearly, \(u\) may be unmatched. Define:
\[
\text{pred}_t(v; u) = \begin{cases} 
u & \text{if tenacity}(u) \geq t, \\
\text{base}\(^{k(t,u)}\)(\(u\)) & \text{if tenacity}(u) < t
\end{cases}
\]

We will say that a vertex \(w\) is \text{pred}_t of \(v\), denoted \(w = \text{pred}_t(v)\), if tenacity\((v) = t\) and there is a predecessor \(u\) of \(v\) such that \(\text{pred}_t(v; u) = w\). The relation \text{pred}^*_t is recursively defined as follows: given vertices \(u, v \in V_t\), with tenacity\((v) = t\), we will say that \(u\) is \text{pred}^*_t of \(v\), denoted \(u = \text{pred}^*_t(v)\), if either \(u = v\) or \(u = \text{pred}^*_t(\text{pred}_t(v))\).

Next, we define directed graph \(H_t\) and undirected graph \(H'_t\). Analogous to the definitions of graphs \(H_m\) an \(H'_m\) given in Section 8.2.1, the vertex sets of both \(H_t\) and \(H'_t\) is the same and is \(V_t\).
The edge set of $H_t$, $E_t$, is defined as follows: For vertices $w, v \in V_t$, if $w$ is pred$_t$ of $v$, then there is a directed edge $(v, w) \in E_t$. Define directed graph $H_t = (V_t, E_t)$.

Corresponding to the set of bridges of tenacity $t$, $B_t$, define

$$B'_t = \{(u^*, v^*) \mid (u, v) \in B_t\}.$$ 

Define undirected graph $H'_t = (V_t, (B'_t \cup E'_t))$, where the edge set, $E'_t$, is obtained by undirecting each edge in $E_t$. An edge $e \in E'_t$ is matched in $H'_t$ if and only if the corresponding edge is present in $G$ and is matched in $G$. Figure 19 gives graph $H'_t$ for $t = 19$, corresponding to the graph of figure 14.

![Diagram](image)

**Figure 19.** Graph $H'_t$ for $t = 19$, corresponding to the graph of figure 14.

The extra complexity that arises in the proof of the induction step is captured in the various definitions given above in this section. This complexity does not change the basic ideas needed for proving statements analogous to those given in Section 8.2 for the induction basis — the only difference in the formal statements is that subscript “m” is replaced by “t” throughout and $t_m$ is replaced by $t$. Below, we will summarize this development. The proof of Statement 2 is followed by some key definitions which are used by the rest of the statements of Theorem 3.

Lemmas 6 and 7 carry over and so does Procedure Bottleneck. The output of a run of this procedure on input $(u, v) \in B_t$ is a vertex $b$. As in the induction basis, this vertex is very special, and is called a base; a formal definition is given below. Clearly, $b$ is an outer vertex. In general, $H'_t$ will have a number of bases.

For each base $b$ in $H'_t$, define the set

$$S_{b,t} = \{v \in U_t \mid b \text{ is pred}_t^t \text{ of } v\}.$$


Proof of Statement 2: Lemmas 8 and 9 also carry over and so does Corollary 2, thereby proving this statement.

Next, we formally define the base of vertices in $U_t$ and blossoms of tenacity $t$.

Definition 25. (The base of a vertex of tenacity $t$ and basal vertices) For each $v \in U_t$, define $\text{base}(v)$ to be the unique vertex, say $b$, in the set $B(v)$. We will say that the base of $v$ is $b$. Each such vertex $b$ will be called a basal vertex. Clearly, $b$ is an outer vertex and tenacity($b$) > $t$.

Definition 26. (Blossom of tenacity $t$ and base $b$) Let $b$ be a basal vertex with tenacity($b$) > $t$. Let $T_{b,t} = \{v \in U_t \mid \text{base}(v) = b\}$. Then the blossom of tenacity $t$ and base $b$ is the set

$$B_{b,t} = T_{b,t} \cup \bigcup_{v \in (T_{b,t} \cup \{b\}), \text{v is outer}} B_{v,t-2}.$$

Let $b$ be a basal vertex with tenacity($b$) > $t$ and $B_{b,t-2}$ be a blossom of tenacity $t - 2$. If $T_{b,t} = \emptyset$, then $B_{b,t} = B_{b,t-2}$.

Example 17. Figures 20 and 21 show two graphs with nested blossoms. Although the two “look different”, in both graphs, vertex $b$ is the base of vertices $u,u',v$ and $v'$; the tenacities of bridges $(u,u')$ and $(v,v')$ are 7 and 11, respectively; the set $T_{b,11} = \{v,v'\}$; and the blossoms are $B_{b,7} = \{u,u'\}$ and $B_{b,11} = \{u,u',v,v'\}$.

Definition 27. (Shortest path from a base to a vertex of tenacity $t$) Let $v \in U_t$ and $b = \text{base}(v)$. Then by an evenlevel($b; v$) (oddlevel($b; v$)) path we mean a minimum even (odd) length alternating path in $G$ from $b$ to $v$ that starts with an unmatched edge.

Proof of Statement 3: Claims analogous to Lemmas 9 and 10 hold thereby proving this statement. It is analogous to Corollary 3 in the induction basis.

Lemma 11 also holds; however, we state the latter below so as to state an important fact in Corollary 4.
Figure 22. The iterated bases of vertex $w$ are: base$(w) = b$, base$^2(w) = b'$ and base$^3(w) = f$, and those of vertex $u$ are: base$(u) = b'$ and base$^2(u) = f$.

**Lemma 15.** Let $B_{b,t}$ and $B_{b',t'}$ be two blossoms with bases $b \neq b'$. Then $B_{b,t} \cap B_{b',t'} = \emptyset$.

By Definition 26, if $B_{b,t}$ and $B_{b',t'}$ are two blossoms with $t < t'$, then either they are disjoint or the latter is contained in the former; note that we are allowing $b = b'$. This gives:

**Corollary 4.** The set of blossoms of tenacity at most $t$ form a laminar family.

**Proof of Statement 4:** Lemma 15 and Corollary 4 prove this statement.

**Remark 4.** Note that after the induction step dealing with tenacity $t$ vertices, we can define the iterated bases of a vertex of tenacity at most $t$; this is done in Definition 28. However, after the entire induction, this definition holds for all vertices of eligible tenacity.

**Definition 28.** (Iterated bases of a vertex of tenacity at most $t$) For $v \in U_t$, let base$(v) = b$. Define the first iterated base of $v$ to be $b$, denoted as follows: base$^3(v) = b$. Next, consider a vertex $u$ with tenacity$(u) < t$ and such that the induction hypothesis has established that base$^k(u) = v$, for $k \in \mathbb{Z}_+$. Then base$^{k+1}(u) = b$, i.e., base$^{k+1}(u) = base^{k}(u)$.

Once the iterated bases of vertices are defined, we can extend Definition 27 to define a shortest path from an iterated base, say $b$, to a vertex $v$, again denoted evenlevel$(b; v)$ or oddlevel$(b; v)$, depending on whether the path is even or odd in length.

**Example 18.** In the graph of Figure 22, the iterated bases of vertex $w$ are base$(w) = b$, base$^2(w) = b'$ and base$^3(w) = f$. Clearly, tenacity$(w) <$ tenacity$(b) <$ tenacity$(b') <$ tenacity$(f)$.

**Definition 29.** (BFS honesty on $p$ with respect to an iterated base) Let $p$ be an evenlevel$(v)$ path starting from unmatched vertex $f$ to an arbitrary vertex $v$ and let $u$ lie on $p$ with tenacity$(u) \leq t$. Let $b = base^k(u)$ be an iterated base of $u$ that is well-defined at this stage of the induction. Then we will say that $u$ is **BFS honest on $p$ w.r.t. $b$** if $p[b$ to $u]$ is an evenlevel$(b; u)$ (oddlevel$(b; u)$) path assuming $|p[b$ to $u]|$ is even (odd). Note that we are allowing $b$ to come either before or after $u$ on $p$. 
Proof of Statement 5: We will first prove a fact that is analogous to Lemma 12; the proof is also analogous and is omitted.

Lemma 16. Let $p$ be an evenlevel($v$) path from unmatched vertex $f$ to an arbitrary vertex $v$ such that there is a blossom $B_{b,t}$ with $p \cap B_{b,t} \neq \emptyset$. Then the base of this blossom, $b$, also lies on $p$ and the following hold:
1. There is a vertex $u \in (p \cap B_{b,t})$ such that $p[b \text{ to } u]$ contains all vertices in $p \cap B_{b,t}$ and $p[b \text{ to } u]$ is an evenlevel($b;u$) path.
2. $u$ is BFS honest on $p$ if and only if $b$ occurs before $u$ on $p$.

Then Statement 5 follows along on the lines of Lemma 13. This completes the proof of Theorem 3.

Example 19. Consider the evenlevel($v$) path\textsuperscript{15} in the graph of Figure 22. Vertex $u$ is BFS honest on this path w.r.t. $b'$. The reader is encouraged to determine an oddlevel($b$) path, say $p$, and observe that $w$ is BFS honest on $p$ w.r.t. $b$. Similarly, for the graph of Figure 23, let $q$ denote the evenlevel($v$) path. Then vertex $u$ is BFS honest on $q$ w.r.t. $b$ and $b'$.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure23}
\caption{Let $p$ be the evenlevel($v$) path. Vertex $u$ is BFS honest on $p$ w.r.t. $b'$; however, $u$ is not BFS honest on $p$ w.r.t. $f$.}
\end{figure}

As observed in Example 14, vertices of tenacity $l_m$ may have no base and as a result, Statements 2 to 5 of Theorem 3 do not hold for them. However, Statement 1 does hold and needs to be proven; in particular, DDFS on the corresponding bridge will reveal an augmenting path if one exists. This case is singled out in the next theorem. Its proof is identical to that of Statement 1.

Theorem 4. For every vertex $v$ of tenacity $l_m$, every maxlevel($v$) path contains a bridge of tenacity $l_m$.

\textsuperscript{15}This path is made explicit in Figure 5.
8.4. BFS Honesty and Iterated Bases  In this section, we prove structural properties which show the sense in which BFS honesty can be restored, if viewed from the perspective of iterated bases, see Remark 5. These properties also go to proving that the MV algorithm will find a maximal set of disjoint minimum length augmenting paths. In contrast, the properties established in Sections 8.2 and 8.3 help prove that the algorithm correctly finds one minimum length augmenting path.

Definition 30. (Number of iterated bases of a vertex of eligible tenacity) Let \( v \) be a vertex of eligible tenacity. As noted earlier, the tenacities of its iterated bases keeps increasing, i.e., tenacity(base\( ^1 \)(\( v \))) < tenacity(base\( ^2 \)(\( v \))) < tenacity(base\( ^3 \)(\( v \)))... Clearly there is an \( l \) such that base\( ^l \)(\( v \)) is not a vertex of eligible tenacity, since tenacity(base\( ^l \)(\( v \))) > \( l_m \). Now base\( ^l \)(\( v \)) does not have a base and \( v \) has exactly \( l \) iterated bases.

Theorem 5. Let \( v \) be a vertex of eligible tenacity, and let \( p \) be an evenlevel(v) or oddlevel(v) path, starting at unmatched vertex \( f \), say. Assume that \( v \) has exactly \( l \) iterated bases. Then the following hold:
1. All \( l \) iterated bases of \( v \) lie on \( p \); moreover, they occur in the order base\( ^l \)(\( v \)), base\( ^{l-1} \)(\( v \)), ..., base\( ^1 \)(\( v \)) on \( p \).
2. Each iterated base is BFS honest on \( p \).
3. \( v \) is BFS honest on \( p \) w.r.t. each iterated base.

Proof: We will prove by an induction on \( k \), for \( 1 \leq k \leq l \), the following statement: base\( ^k \)(\( v \)) lies on \( p \) and \( p[f \text{ to } \text{ base}^k(v)] \) is an evenlevel(base\( ^k \)(\( v \))) path. This will establish the first two statements of the theorem and the third will then follow by Theorem 3.

Let base\( ^1 \)(\( v \)) = \( b \). By Statement 3 of Theorem 3, every evenlevel(v) (oddlevel(v)) path consists of an evenlevel(b) path concatenated with an evenlevel(b;v) (oddlevel(b;v)) path. Therefore \( p[f \text{ to } \text{ b}] \) is an evenlevel(b) path, hence establishing the basis of the induction.

Assume that the claim is true for \( k \), where \( 1 \leq k < l \), and let base\( ^k \)(\( v \)) = \( u \). Then \( p[f \text{ to } u] \) is an evenlevel(u) path. Let base\( ^l \)(\( v \)) = \( w \). Again by Statement 3 of Theorem 3, \( w \) lies on \( p[f \text{ to } u] \) and \( p[f \text{ to } w] \) is an evenlevel(w) path. The claim follows.

Corollary 5. Let \( v \) be an arbitrary vertex and \( p \) be an evenlevel(v) or oddlevel(v) path, starting at unmatched vertex \( f \), say. Let \( u \) be a vertex of eligible tenacity which is BFS honest on \( p \), and assume that \( u \) has exactly \( l \) iterated bases. Then the \( l \) iterated bases of \( u \) satisfy the three conditions stated in Theorem 5.

Then since \( p[f \text{ to } u] \) is a minimum alternating path to \( u \), Theorem 5 applies to it, leading to Corollary 5. The contrasting situation, when \( u \) is not BFS honest on \( p \), is studied next.

Theorem 6. Let \( v \) be a vertex with tenacity(v) \( \leq l_m \) and \( p \) be an evenlevel(v) or oddlevel(v) path, starting at unmatched vertex \( f \), say. Let \( u \) be a vertex of eligible tenacity which occurs on \( p \) but is not BFS honest on \( p \). Assume that \( u \) has exactly \( l \) iterated bases. Then the following hold:
1. All \( l \) iterated bases of \( u \) lie on \( p \).
2. There is a number \( k \) with \( 1 \leq k < l \) such that the \( l \) iterated bases and \( u \) occur in the order base\( ^l \)(\( u \)), ..., base\( ^{k+1} \)(\( u \)), followed by \( u \), followed by base\( ^{k+1} \)(\( u \)), ..., base\( ^1 \)(\( u \)) on \( p \).
3. Each of the iterated bases base\( ^l \)(\( u \)), ..., base\( ^{k+1} \)(\( u \)), base\( ^k \)(\( u \)) is BFS honest on \( p \).
4. \( u \) is BFS honest on \( p \) w.r.t. each of the iterated bases base\( ^{k+1} \)(\( u \)), ..., base\( ^1 \)(\( u \)).

Proof: Since \( u \) is not BFS honest on \( p \), base\( ^{k+1} \)(\( u \)) occurs after \( u \) on \( p \). If base\( ^l \)(\( u \)) is BFS honest on \( p \), i.e., if maxlevel(base\( ^l \)(\( u \))) = oddlevel(base\( ^l \)(\( u \))) = p[f to base\( ^l \)(\( u \))], then by Statement 5 of Theorem 3, base\( ^2 \)(\( u \)) occurs before base\( ^l \)(\( u \)) on \( p \); moreover, evenlevel(base\( ^2 \)(\( u \))) = \|p[f to base\( ^2 \)(\( u \))]\|. If so, the \( l \) bases and \( u \) occur in the order base\( ^l \)(\( u \)), ..., base\( ^2 \)(\( u \)), followed by \( u \), followed by base\( ^{k+1} \)(\( u \)) on \( p \). By
Theorems 2 and 3 we get that each of the bases base\(^t\)(u), \ldots, base(u) is BFS honest on p, and u is BFS honest on p w.r.t. base(u).

Next, assume that base(u) is not BFS honest on p. If so, by Statement 5 of Theorem 3, base\(^2\)(u) occurs after base(u) on p. Continuing in this manner, we will get the smallest k such that base\(^k\)(u) is BFS honest on p. If so, the iterated bases base\(^i\)(u), \ldots, base\(^k\)(u) are BFS honest on p, and u is BFS honest on p w.r.t. each of the iterated bases base(u), \ldots, base\(^k\)(u). Observe that base\(^i\)(v) occurs on both lists; in particular, it occurs on the first list because p[ for base\(^i\)(v)] is a maxlevel(base\(^k\)(v)) path. The theorem follows.

Example 20. The graph of Figure 23 illustrates Theorem 6. Let p be the evenlevel(v) path. Now vertex u is not BFS honest on p. The iterated bases of u are b, b′ and f. Of these, b′ and f are BFS honest on p and b is not BFS honest on p. To compensate for the latter, u is BFS honest on p w.r.t. b and b′. The iterated base b′ plays a dual role: b′ is BSF honest on p and u is BFS honest on p w.r.t. b′ as well.

Remark 5. Via Theorem 6, we can say in what sense it is possible to restore BFS honesty, even though vertices on p, such as u, are not BFS honest in the usual sense. The first goal of path p, before reaching v, is to arrive at base\(^k\)(u) with a maxlevel path, since that is the shortest way of arriving at v. This involves using the appropriate bridge, say (c, c′), in whose support base\(^k\)(u) lies. However, the path from c′ to base\(^k\)(u) involves going through the blossom B_{b′,t}, where b′ = base\(^k\)(u) and t = tenacity(base\(^k\)(u)).

Now path p enters this blossom at vertex u, which is the reason u is not BFS honest on p. Together with u, the bases base(u), \ldots, base\(^k\)(u) are also not BFS honest on p; however, base\(^k\)(u) is BFS honest on p, since p reaches it at its maxlevel. As proven in Theorem 6, this lack of BFS honesty is “compensated” by the fact that u is BFS honest on p w.r.t. the bases base(u), \ldots, base\(^k\)(u). The latter fact is crucially needed in reconstructing the path by the procedure FINDPATH. Observe that base\(^k\)(u) plays the role of a “link” between the two sets of iterated bases of u: it is BFS honest on p and u is BFS honest on p w.r.t. it.

Lastly, since tenacity(base\(^i\)(u)) \geq l_m \geq tenacity(v), by Theorem 2, base\(^i\)(u) is BSF honest on p. Therefore, the following situation cannot happen: all i iterated bases of u are not BSF honest on p.

9. Proof of Correctness and a Post-Mortem As stated in the Introduction, besides the procedure of DDFS, the other main idea behind the MV algorithm is the precise synchronization of events; this is described and proved in Section 9.1. Section 9.3 proves that the MV algorithm correctly executes a phase and also establishes the running time of the algorithm. Finally, Section 9.4 provides a post-mortem by raising and answering the question, “Why is it essential to formalize such an elaborate purely graph-theoretic structure for proving correctness of the MV algorithm?”

9.1. Synchronization of Events

Theorem 7. Let t be an odd number with t_m \leq t \leq l_m, i.e., t is either an eligible tenacity or t = l_m. The following hold:

1. Algorithm 1 finds Br(t), the set of bridges of tenacity t, by the end of execution of procedure MIN at search level i, where t = 2i + 1.

2. For each vertex v such that tenacity(v) = t, Algorithm 1 assigns minlevel(v) and maxlevel(v) correctly.

Proof: We will show, by strong induction on i, for i = 0 to \((l_m - 1)/2\), that at search level i, Algorithm 1 will accomplish:

Task 1: Procedure MIN assigns a minlevel of i + 1 to exactly the set of vertices having this minlevel. It also identifies all props that assign a minlevel of i + 1.
Task 2: By the end of execution of procedure MIN at this search level, \( Br(2i+1) \) is the set of all bridges of tenacity \( 2i+1 \).

Task 3: Procedure MAX assigns correct maxlevels to all vertices having tenacity \( 2i+1 \). This will establish both statements of the theorem.

The base case, \( i = 0 \), is obvious: MIN will assign an oddlevel of 1 to each neighbor of each unmatched vertex. Next we assume the induction hypothesis for all search levels less than \( i \), and prove that Algorithm 1 will accomplish the three tasks at search level \( i \).

Task 1: By the induction hypothesis, the minlevel assigned to vertex \( v \) at the beginning of execution of MIN at search level \( i \) is \( \infty \) if and only if \( \text{minlevel}(v) \geq i+1 \). Since MIN searches from all vertices having level \( i \) along the correct parity edges and assigns a minlevel to a vertex only if its currently assigned minlevel is \( \geq i+1 \), any vertex \( v \) that is assigned a minlevel in this search level must indeed satisfy \( \text{minlevel}(v) = i+1 \), and the edge that reaches \( v \) will be correctly classified as a prop.

We next prove that every vertex \( v \) with \( \text{minlevel}(v) = i+1 \) will be assigned its minlevel in this search level, and every prop that assigns a minlevel of \( i+1 \) will be classified as a prop. Let \( \text{minlevel}(v) = i+1 \), let \( p \) be a minlevel path, and let \( (u,v) \) be the last edge on \( p \). Clearly \( (u,v) \) is a prop, and every prop that assigns a minlevel of \( i+1 \) is of this type. Now, \( u \) must be BFS honest on \( p \): If not, then \( v \) must occur on a shorter path to \( u \), contradicting \( \text{minlevel}(v) < i+1 \). If \( |p| = i = \text{maxlevel}(u) \) then \( \text{tenacity}(u) < 2i+1 \). Otherwise, \( |p| = i = \text{minlevel}(u) \).

In either case, by the induction hypothesis, \( u \) has already been assigned a level of \( i \). Therefore, at search level \( i \), MIN will search from \( u \) along edge \( (u,v) \) and will find \( v \). By the induction hypothesis, at this point, either the minlevel of \( v \) is set to either \( \infty \) or \( i+1 \). In either case, \( v \) will be assigned a minlevel of \( i+1 \), \( u \) will be declared a predecessor of \( v \) and \( (u,v) \) will be declared a prop.

Task 2: Let \( (u,v) \) be a matched bridge with \( \text{tenacity}(u,v) = 2i+1 \). By Lemma 1, \( \text{tenacity}(u) = \text{tenacity}(v) = \text{tenacity}(u,v) \), and \( u \) and \( v \) are both inner. Therefore, \( \text{oddlevel}(u) = \text{oddlevel}(v) = i \). Hence during search level \( i \), MIN will determine that \( (u,v) \) is a bridge, that its tenacity is \( 2i+1 \), and will insert it in \( Br(2i+1) \).

Next assume that \( (u,v) \) is an unmatched bridge with \( \text{tenacity}(u,v) = 2i+1 \). By Lemma 3, if \( \text{tenacity}(v) = \text{tenacity}(u,v) = 2i+1 \), then \( v \) is an outer vertex and \( \text{evenlevel}(v) \leq i \). Therefore, the algorithm has already determined \( \text{evenlevel}(v) \). On the other hand, if \( \text{tenacity}(v) < 2i+1 \), then both its levels were determined by the end of the previous search level. By Lemma 3, these are the only two cases.

Therefore, in both cases, \( \text{tenacity}(u,v) \) will be ascertained by the end of execution of procedure MIN at search level \( i \) and \( Br(2i+1) \) will be the set of all bridges of tenacity \( 2i+1 \).

Task 3: Let \( \text{tenacity}(v) = t \). By Statement 1 of Theorem 3 or by Theorem 4, depending on whether \( t \) is an eligible tenacity or \( t = l_m \), \( v \) lies in the support of a bridge of tenacity \( 2i+1 \), and by Task 2, this bridge is in \( Br(2i+1) \) at the start of MAX in search level \( i \). Therefore, DDFS will assign maxlevel(\( v \)) in this search level.

Example 21. This example gives an insight into the second idea behind the MV algorithm, namely precise synchronizing of events. In Figure 24, the algorithm determines that \( (u,v) \) is a bridge of tenacity 15 at search level 6. However, according to Algorithm 1, DDFS should be performed on \( (u,v) \) at search level 7. The question arises, “Why wait till search level 7; why not perform DDFS on \( (u,v) \) when procedure MAX is run at search level 6?” In the graph of Figure 24 no mistakes will be made, provided the algorithm assigns tenacities of 15 vertices to \( a \) and \( b \), i.e., the same as the tenacity of bridge \( (u,v) \).

16 The latter case happens if evenlevel(\( u \)) = \( i \) and \( v \) has been reached earlier in this search level while searching along an edge \( (u',v) \) with evenlevel(\( u' \)) = \( i \).
Next consider the graph in Figure 25 in which, due to the bridge of tenacity 13, a and b have tenacities of 13. Assume that bridge (u, v) is processed before other bridges at search level 6; this is consistent with the arbitrary order in which bridges are processed\(^{17}\). This time around, a and b will be wrongly assigned tenacities of 15. This error is avoided by processing bridge (u, v) at search level 7. As a result, when the bridge of tenacity 13 is processed at search level 6, a and b will be assigned their correct tenacities, i.e., 13. When processing bridge (u, v), DDFS will skip over the blossom of tenacity 13 and not encounter a and b.

### 9.2. Relationship between Graph-Theoretic and Algorithmic Notions

As stated in Section 5.2, the notions of petal and bud are intimately related to the notions of blossom and base; whereas the former are algorithmic notions, the latter are graph-theoretic. This relationship is formally established in Lemma 17. At the end of search level \(i\), once MAX is done processing all bridges of tenacity \(t = 2i + 1\), all blossoms of tenacity \(t\) can be identified via this lemma; its proof is straightforward and is omitted.

**Lemma 17.** Let \(\text{tenacity}(v) = t\), and at the end of search level \(i = (t - 1)/2\), assume that \(\text{bud}^*(v)\) is b. Then \(\text{base}(v) = b\) and the set \(T_{b,t}\) defined in Definition 26 is:

\[
T_{b,t} = \{u \mid \text{tenacity}(u) = t \text{ and } \text{bud}^*(u) = b\}. 
\]

Furthermore, the blossom \(B_{b,t}\) consists of the union of all petals whose \(\text{bud}^*\) is b at the end of search level \(i = (t - 1)/2\), together with each blossom of tenacity \((t - 2)\) whose base is b or any of the vertices of these petals.

\(^{17}\) By making the example given in Figure 25 slightly bigger, one can easily ensure that there are no such ties.
Observe that if \( \text{bud}^*(v) \) is computed at the end of search level \( j > i \), then it may not be \( b \) anymore; however, it will be an iterated base of \( v \).

9.3. Execution of a Phase and Proof of Running Time The proofs given in Section 9.1 show that the MV algorithm correctly finds one minimum length augmenting path in the given graph with an initial matching. Lemma 18 shows that it correctly finds a maximal set of such paths as well. Finally, Theorem 8 concludes with a proof of the running time.

**Lemma 18.** The procedures given in Section 5.4 will find a maximal set of disjoint minimum length augmenting paths in \( G \).

**Proof:** Clearly, the first path, say \( p \), found by the algorithm will be of length \( l_m \), i.e., it will be a minimum length augmenting path. As argued earlier, the vertices removed by procedure RECURSIVE REMOVE of Section 5.4.2 cannot be part of a minimum length augmenting path that is disjoint from \( p \).

The crux of the matter is the following question: how do we guarantee that the remaining graph will “look like” a graph in which the first path is found, i.e., it has all the pointers and properties needed. The theorems of Section 8.4 guarantee that if a vertex \( v \in p \) then each of its iterated bases is on \( p \) and will be removed. It is easy to see that once the base of a blossom is removed, RECURSIVE REMOVE will remove all its remaining vertices as well.

This ensures that when DDFS is called with the next bridge of tenacity \( l_m \), the remaining graph will satisfy all the required properties so that the next path, if any, can be found by the same process. The lemma follows. \( \square \)

**Theorem 8.** The MV algorithm finds a maximum matching in general graphs in time \( O(m\sqrt{n}) \) on the RAM model and \( O(m\sqrt{n}\cdot\alpha(m, n)) \) on the pointer model, where \( \alpha \) is the inverse Ackerman function.

**Proof:** Each of the procedures of MIN, MAX, finding augmenting paths, and RECURSIVE REMOVE examine each edge a constant number of times in each phase. The only operation that remains is that of computing \( \text{bud}^* \) during DDFS. This can be implemented on the pointer model using the set union algorithm [24], which will take \( O(m\cdot\alpha(m, n)) \) time per phase. Alternatively, it can be implemented on the RAM model using the linear time algorithm for a special case of set union [8]; this will take \( O(m) \) time per phase. Since \( O(\sqrt{n}) \) phases suffice for finding a maximum matching [12, 15], the theorem follows. \( \square \)

A question arising from Theorem 8 is whether there is a linear time implementation of \( \text{bud}^* \) in the pointer model. [20] had claimed, without proof, that path compression by itself suffices to achieve this. They stated that because of the special structure of blossoms, a charging argument could be given that assigns a constant cost to each edge. This claim is left as an open problem.

9.4. The role of graph-theoretic structural properties in the MV algorithm Finally we address the following question, “Why was it essential to formalize such an elaborate purely graph-theoretic structure for proving correctness of the MV algorithm?” Now that the reader is familiar with the structural definitions and claims, this question can be answered.

Assume that \( \text{minlevel}(v) = i + 1 \) and \( \text{maxlevel}(v) = j + 1 \), so that \( \text{tenacity}(v) = i + j + 2 = t \), where \( t \) is an eligible tenacity or \( t = l_m \). It is easy to see that there must be a neighbor, say \( u \), of \( v \), such that \( \text{evenlevel}(u) = i \) or \( \text{oddlevel}(u) = i \), depending on the parity of \( i \). Therefore one step of breadth first search, while searching from \( u \), will lead to assigning \( v \) its correct minlevel. We will say that \( u \) is the *agent that assigns \( v \) its minlevel*. 
In contrast, none of the neighbors of \( v \) may have \( j \) as one of its levels. For instance, vertex \( b \) in the graph of Figure 5 has maxlevel\((b) = \text{oddlevel}(b) = 11 \). Observe that evenlevel\((a) = \text{evenlevel}(c) = 10 \) and evenlevel\((u) = 12 \), i.e., none of the neighbors of \( b \) has an evenlevel of 10. The following questions arise:

1. What is the agent that assigns \( v \) its maxlevel and does it exist for each vertex \( v \)?
2. Can this agent be found for each “relevant” vertex \( v \)?
3. How does this agent help assign \( v \) its maxlevel?

This paper provides very precise answers to all these questions. The agent that assigns \( v \) its maxlevel is a bridge whose tenacity equals tenacity\((v) \). Statement 1 of Theorem 3 and Theorem 4 prove that every maxlevel\((v) \) path contains such a bridge. For our purpose, relevant vertices are those whose tenacity is eligible or is \( l_m \). Theorem 7 proves that for each such vertex, a bridge will be found by the algorithm; moreover, it will be found “well in time”. Finally, maxlevels of vertices are found by executing a double depth first search on the endpoints of bridges.

These structural properties suffice for finding the first augmenting path. However, after its removal, can it be that the graph is left with “half-eaten blossoms” which simply do not support finding the next path via the same process as the first one, even though a path exists? Lemma 18, which is based on the structural properties established in Section 8.4, shows that subsequent paths can be found in the same way as the first one.

10. Discussion  Matching is one of the “big three” problems in combinatorial optimization, along with linear programming and flow. Spectacular progress in that field has improved the running times of those two and other fundamental problems in the last three decades. The running time of the maximum matching problem in bipartite graphs has also undergone several improvements: \( O(m^{10/7}) \) [19], \( O(m^{4/3}) \) [17], \( \tilde{O}(m + n^{1.5}) \) [26], and most recently to almost linear time, \( O(m^{1+o(1)}) \) [4]. A concerted effort has been made to improve the running time for general graph matching as well, but so far the MV algorithm has stood the test of time.

As is well known, general graph matching has numerous applications. We single out a particularly interesting and important one — to kidney exchange. This application leads to a matching market, which was mentioned in the Scientific Background for the 2012 Nobel Prize in Economics awarded to Alvin Roth and Lloyd Shapley [23].

Assume that agent \( A \) requires a kidney transplant and agent \( B \) has agreed to donate one of her kidneys to \( A \); however, their kidney types are not compatible. Assume further that \((A', B')\) is another pair of people with an incompatibility. If it turns out that \((A, B')\) and \((A', B)\) are both compatible pairs, then let us say that the two pairs are consistent; if so, both transplants can be performed.

Next assume that a number of incompatible pairs are specified, \((A_1, B_1), \ldots, (A_n, B_n)\) and for every two pairs, we know whether they are consistent. The problem is to find the maximum number of disjoint consistent pairs. This can be reduced to maximum matching as follows. Let \( G = (V, E) \) be a graph with \( V = \{v_1, \ldots, v_n\} \) where \( v_i \) represents the pair \((A_i, B_i)\), and \((v_i, v_j) \in E \) if and only if the two pairs \((A_i, B_i)\) and \((A_j, B_j)\) are consistent. Clearly a maximum matching in \( G \) will yield the answer.

11. Acknowledgements  I wish to thank Rohith Gangam and Ruta Mehta for diligently helping verify this proof, and Silvio Micali for embarking on a year-long journey, full of youthful exuberance, which led to the discovery of this algorithm. For the four-decade-long journey that led to the discovery of this proof, I must thank matching theory for its gloriously elegant combinatorial structure, which kept me going. Lastly, I wish thank the two referees for providing in-depth critiques of this paper, covering all aspects of the exposition and the proof.
References


