A Proof of the MV Matching Algorithm

Vijay V. Vazirani[*]

May 13, 2014

Abstract

This paper gives the first complete proof of correctness of the Micali-Vazirani [MV80] general graph maximum cardinality matching algorithm. Our emphasis is on arriving at the simplest possible proof; graph-theoretic machinery developed for this purpose also helps simplify the description of this algorithm. For all practical purposes, this is still the most efficient known algorithm for the problem.

1 Introduction

The process of proving correctness of the Micali-Vazirani [MV80] general graph maximum cardinality matching algorithm was started in [Vaz94]; however, as detailed in Section 1.2, the paper had deficiencies. The purpose of this paper is to provide a complete proof of this algorithm in the simplest possible terms; graph-theoretic machinery developed for this purpose also helps simplify the description of this algorithm. Barring the case of very dense graphs, for which a slightly better running time is known (though only of theoretical importance), this is still the most efficient known algorithm for the problem; see Section 1.1 for precise details. This paper is fully self-contained so as to be suitable for pedagogical and archival purposes.

Matching occupies a central place in the theory of algorithms as detailed below. From the viewpoint of efficient algorithms, bipartite and non-bipartite matching problems are qualitatively different. First consider the process of finding a maximum matching by repeatedly finding augmenting paths. Whereas the in former case, the lengths of all alternating paths from an unmatched vertex f to a matched vertex v must have the same parity, even or odd, in the latter they can be of both parities. Edmonds defined the key notion of *blossoms* and finessed this difficulty in non-bipartite graphs by "shrinking" blossoms.

The most efficient known maximum matching algorithms in both bipartite and non-bipartite graphs resort to finding minimum length augmenting paths w.r.t. the current matching. However, from this perspective, the difference between the two classes of graphs becomes even more pronounced. Unlike the bipartite case, in non-bipartite graphs minimum length alternating paths do not possess an elementary property, called breadth first search honesty[1] in Section 2. Indeed, in the face of this debilitating shortcoming, the problem of finding minimum length alternating

[*]Supported by NSF Grants CCF-0914732 and CCF-1216019, and a Guggenheim Fellowship. College of Computing, Georgia Institute of Technology, Atlanta, GA 30332–0280, E-mail: vazirani@cc.gatech.edu.

[1]Intuitively, it states that in order to find shortest alternating paths from an unmatched vertex f to all other vertices, there is never a need to find a longer path to any vertex v.

paths appears to be intractable. It is a testament to the remarkable structural properties of matching that despite this, a near-linear time algorithm is possible.

Edmonds' blossoms are not adequate for the task of finding minimum length augmenting paths since by "shrinking" these blossoms, length information is completely lost. What is needed is a definition of blossoms[2] from the perspective of minimum length alternating paths, as given in [Vaz94] and simplified in the current paper.

Matching has had a long and distinguished history spanning more than a century. The following quote from Lovasz and Plummer's classic book [LP86], pg. 409, is most revealing:

> Matching theory serves as an archetypal example of how a "well-solvable" problem can be studied. ... [It] is a central part of graph theory, not only because of its applications, but also because it is the source of important ideas developed during the rapid growth of combinatorics during the last several decades.

Interestingly enough, matching has played an equally central role in the development of the theory of algorithms: time and again, its study has not only yielded powerful tools that have benefited other problems but also quintessential paradigms for the entire field. Examples of the latter include the primal-dual paradigm [Kuh55], the definitions of the classes **P** [Edm65b] and # **P** [Val79], and the equivalence of random generation and approximate counting for self-reducible problems [JVV86]. Examples of the former include the notion of an augmenting path [Kon31, Ege31], a method for determining the defining inequalities of the convex hull of solutions to a combinatorial problem [Edm65a], the canonical paths argument for showing expansion of the underlying graph of a Markov chain [JS89], and the Isolating Lemma [MVV87]. And at the interface of algorithms and economics lies another highly influential matching algorithm: the stable matching algorithm of Gale and Shapley [GS62] which was mentioned prominently in Shapely's citation for the 2012 Nobel Prize in Economics.

1.1 Running time and related papers

The MV algorithm finds minimum length augmenting paths in phases; each phase finds a maximal set of disjoint such paths and augments the matching along all paths. $O(\sqrt{n})$ such phases suffice for finding a maximum matching [HK73, Kar73]. 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 8.5 for details). As is standard, n denotes the number of vertices and m the number of edges in the given graph.

We note that small theoretical improvements to the running time, for the case of very dense graphs, have been given in recent years: $O(m\sqrt{n}\log(n^2/m)/\log n)$ [GK04] and $O(n^w)$ [MS04], 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 $m = \omega(n^{1.85})$; additionally, the latter algorithm involves a large multiplicative constant in its running time which comes from the use of fast matrix multiplication as a subroutine in this algorithm.

Prior to [MV80], Even and Kariv [EK75] had used the idea of finding augmenting paths in phases to obtain an $O(n^{2.5})$ maximum matching algorithm. However, the algorithm is very extensive, there is no journal version of the result, and its correctness is hard to ascertain.

[2]To immediately get a feel for the difference in definitions, see Figure 18 and its explanation.

Subsequent to [MV80], [Blu90] and [GT91] have claimed algorithms having the same running time as MV, and we need to clarify the status of these works. It is easy to reduce the problem of finding an augmenting path in a bipartite graph to the problem of finding an $s - t$ path in a certain directed graph. [Blu90] attempts an analogous approach for general graphs. Given G, he defines a certain directed graph which has two vertices $[v, A]$ and $[v, B]$ corresponding to each vertex v in G . He shows that the problem of finding an augmenting path in G is equivalent to finding a "strongly simple" path in the directed graph; such a path is not allowed to contain $[v, B]$ if is contains $[v, A]$. He then claims that a certain procedure, called MBFS finds the latter path in linear time. A proof is not offered in [Blu90] and there is no journal version. Furthermore, the requirements on the directed path are such that even a quadratic time implementation is not clear.

The second paper [GT91] gives an efficient scaling algorithm for finding a minimum weight matching in a general graph with integral edge weights and it claims that the unit weight version of their algorithm achieves the same running time as MV. However is loss of simplicity and crucial structural insights, which are of independent interest, in solving cardinality matching by reducing it to the harder problem of weighted matching. The rest of the history of matching algorithms is very well documented and will not be repeated here, e.g., see [LP86, Vaz94].

1.2 Overview and contributions of this paper

The MV algorithm involves two main ideas: a new search procedure called double depth first search (DDFS) and the precise synchronization of events. The former is described in Section 4, and the latter in Section 3 and via Figures 23 and 24. The potential of finding other applications for DDFS, as well as exploring variants and generalizations, remains unexplored so far. To facilitate wide dissemination, we have made Section 4 fully self-contained.

To point out the contributions of this paper, we compare it to [MV80] and [Vaz94]. [MV80] stated the matching algorithm in pseudocode; in retrospect, this description is complete and error-free. However, the paper did not provide a proof of correctness and running time.

The main contribution of [Vaz94] was to identify purely graph-theoretic notions that are critically needed for proving correctness of the algorithm. Additionally, these notions are also crucial for giving a conceptual description of the algorithm. These notions include BFS-honesty, base of a vertex, a definition of blossoms from the perspective of minimum length alternating paths, and a classification of edges into props and bridges. Structural facts about these notions were stated in seven theorems. Although the statements of these theorems were largely correct, their proofs, which involved low level arguments about individual paths and their complicated intersections with other paths and other structures, were, in retrospect, incorrect. Additionally, the paper failed to take full advantage of these structural notions to arrive at the simplest and cleanest description of the algorithm.

The following ideas are central to our new proof:

- A key property of blossoms – that they contain all minimum length alternating paths from the base to all vertices in the blossom – is exploited to simplify the picture considerably. An elaborate induction, carried out on the parameter "tenacity" in Theorem 6.12, is used to not only prove this property and other structural facts but also to define some key notions, including base of a vertex and blossom. Because of this property, the complicated intersections of lower tenacity paths get encapsulated inside a blossom, which offers, to the rest of the graph, a surprisingly simple interface.

- A simpler definition of blossoms is given. This definition is recursive and it fits naturally into the proof by induction described above. The definition is equivalent to the one in [Vaz94], and their equivalence is established in Section 9.
- The work-horse in Theorem 6.12 turns out to be the procedure of DDFS; as described above, this procedure is central to the algorithm as well. The information output by this procedure, as summarized in DDFS Certificate in Section 4, is the reason for its use in Theorem 6.12.

It is difficult to overemphasize the importance of well-chosen examples for understanding this result; indeed, most of the intuition lies in them and we have included several. Furthermore, they have been drawn in such a way that they easily reveal their structural properties (this involves drawing vertices in layers, according to their minlevel).

An implementation of this algorithm, due to Bruno Loff, is available on the web [Lof14]. This promises to be a valuable pedagogical tool since it provides a complete post-mortem of the run of the algorithm on the graph input by the user.

2 Basic 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. All definitions henceforth are 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 it has 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[3]. 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 [HK73, Kar73]. 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. [HK73, Kar73] 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. 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 $l_m = \infty$.

Definition 2.1 (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 ∞.

[3]Observe that if $M = \emptyset$, then any edge is an augmenting path, of length one.

In this paper, we will typically denote an unmatched vertex by f. Clearly, the evenlevel of each unmatched vertex, say f, will be zero and its oddlevel will be the length of the shortest augmenting path starting at f; if f is in no augmenting path, oddlevel$(f) = \infty$. The length of a minimum length augmenting paths w.r.t. M is clearly the smallest oddlevel of an unmatched vertex. In all the figures, matched edges are drawn dotted, unmatched edges solid, and unmatched vertices are drawn with a small circle. In Figure 1, evenlevels and oddlevels of vertices are indicated; missing levels are ∞.

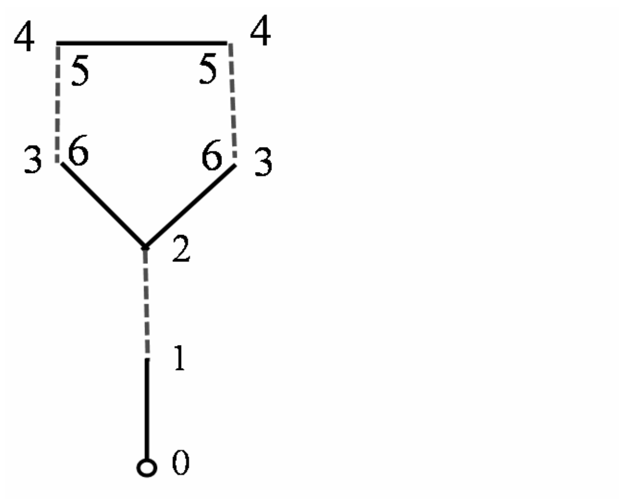

Figure 1: Evenlevels and oddlevels of vertices are indicated; missing levels are ∞.

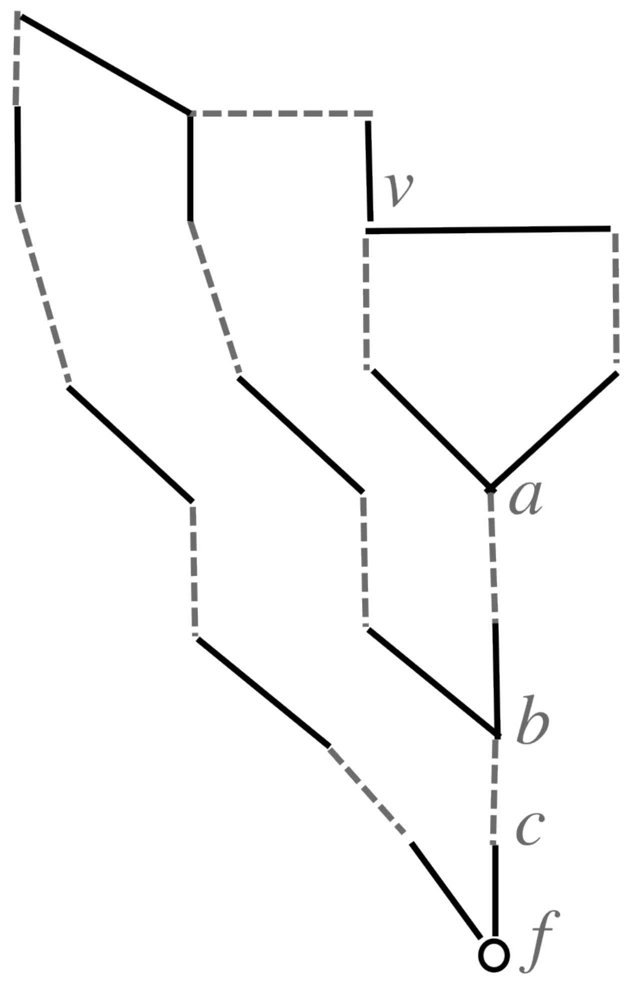

Figure 2: Vertex v is not BFS-honest on oddlevel(a) and evenlevel(c) paths.

Definition 2.2 (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 2.3 (Outer and inner vertices) A vertex v is said to be *outer* if evenlevel$(v) <$ oddlevel(v) and *inner* otherwise.

Let p is an alternating path from unmatched vertex f to v and let u lie on p. Then $p[f$ to u will denote the part of p from f to u. Similarly $p[f$ to $u)$ denotes the part of p from f to the vertex just before u, etc.

Breadth first search (BFS) is a natural way of finding minimum length paths in unweighted graphs. For finding minimum length augmenting paths in bipartite graphs, a slight extension to an alternating breadth first search suffices, e.g., see Section 3 (for a complete description, see Section 2.1 in [Vaz94]). The property of minimum length alternating paths that enables this simple search scheme to work will be called *breadth first search honesty*: If p is a minimum length alternating path from f to v and u lies on p then $p[f \text{ to } u]$ is a minimum length alternating path from f to u. As a consequence of this property, having obtained a minimum length alternating path from f to u, one just has to extend it in a minimal way to v, respecting its alternating nature, to find a minimum length alternating path from f to v.

This elementary property does not hold in non-bipartite graphs, e.g., if v lies on $p[f \text{ to } u]$ but with the opposite parity. In this case, extending p to v will lead to a non-simple path[4] This is precisely the reason that the task of finding minimum length augmenting paths is considerably more difficult in non-bipartite graphs than in bipartite graphs.

In Figure 2, v is not BFS-honest w.r.t. the oddlevel(a), oddlevel(b) and evenlevel(c) paths; it occurs at length 9 on the first and at length 11 on the other two, even though oddlevel(v) = 7. Thus shortest paths to a, b and c, of appropriate parity, involve odd length alternating paths from f to v which are longer than oddlevel(v). Furthermore, this is not just an academic exercise: Suppose this graph had another edge (c, d), where d is a new unmatched vertex. Then the only augmenting path uses the evenlevel(c) path.

Intuitively, whereas short paths are easy to find in a graph, finding long paths is intractable, e.g. Hamiltonian path. Hence, as stated in the Introduction, the problem of finding minimum length augmenting paths in general graphs, which may involve paths to certain vertices that are much longer than the shortest path, is so efficiently solvable.

Definition 2.4 (Tenacity of vertices and edges) Define the tenacity of vertex v, tenacity(v) = evenlevel(v)+oddlevel(v). If (u, v) is an unmatched edge, its tenacity, tenacity(u, v) = evenlevel(u)+ evenlevel(v) + 1, and if it is matched, tenacity(u, v) = oddlevel(u) + oddlevel(v) + 1. Throughout, t_m will denote the tenacity of a minimum tenacity vertex in G.

In Figure 1, the tenacity of each edge in the 5-cycle is 9 and the tenacity of the rest of the edges is ∞. Figures 3 and 4 give the tenacity of vertices and edges, respectively, in a more interesting graph. The notion of tenacity of vertices plays a crucial role in Section 5, which uses it to limit the extent of BFS-dishonesty in minimum length alternating paths. The notion of tenacity of edges plays a crucial role in Theorem 6.12, Statement 2, which uses it to pinpoint the unique edge on a maxlevel(v) path that leads to finding the maxlevel of v, namely a bridge (see below for definition) whose tenacity is the same as tenacity(v),

Definition 2.5 (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.

In Figure 2, the two horizontal edges and the oblique unmatched edge at the top are bridges and the rest of the edges of this graph are props. In Figure 5, (w, w') and (v, v') are bridges, and in Figure 6, (w, w') and (u, v) are bridges; the rest of the edges in these two graphs are props.

Definition 2.6 (The support of a bridge) Let (u, v) be a bridge of tenacity $t \leq l_m$. Then,

[4]Recall that in a bipartite graph, all alternating paths from f to v must have the same parity, either even or odd. In general graphs, paths of both parity may exist.

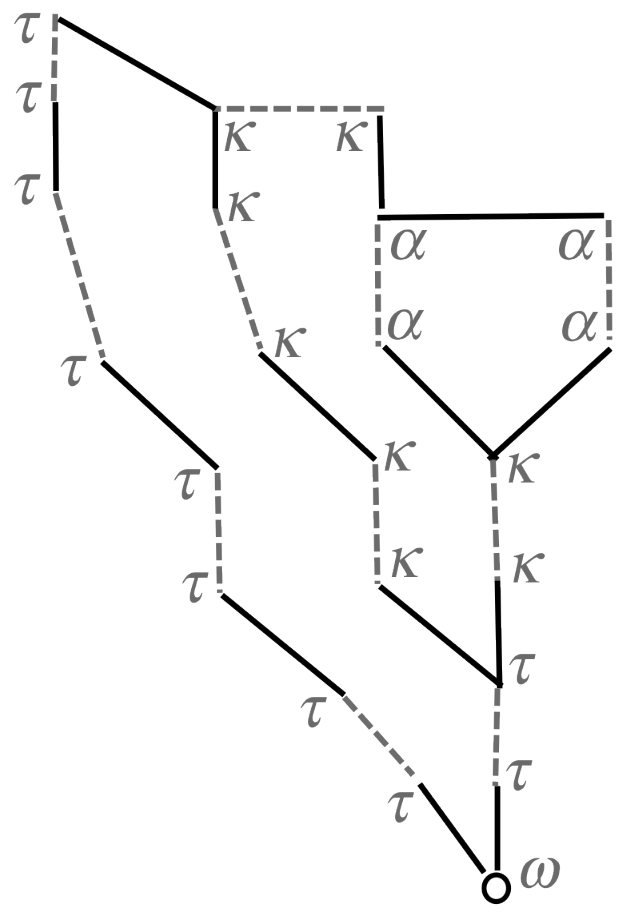
Figure 3: The tenacity of vertices is indicated; here $\alpha = 13$, $\kappa = 15$, $\tau = 17$ and $\omega = \infty$.

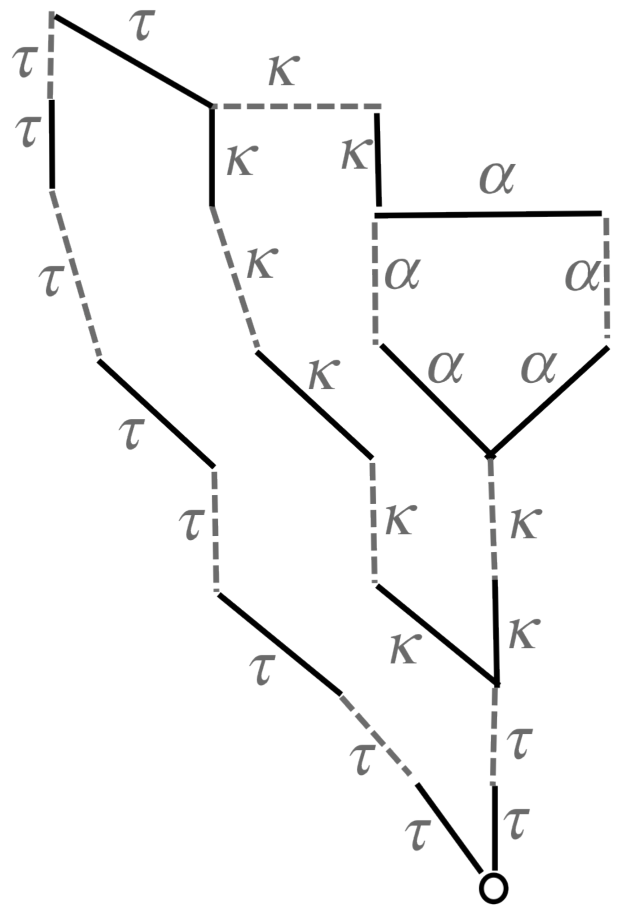
Figure 4: The tenacity of each edge is indicated; here $\alpha = 13$, $\kappa = 15$ and $\tau = 17$.

its support is defined to be $\{w \mid \text{tenacity}(w) = t \text{ and } \exists \text{ a maxlevel}(w) \text{ path containing } (u, v)\}$.

In the graph of Figures 2, 3 and 4, the supports of the bridges of tenacity α, κ and τ are the set of vertices of tenacity α, κ and τ, respectively. In the graph of Figure 5, edges (w, w') and (v, v') are bridges of tenacity 7 and 13, respectively. Unmatched vertex f is not in the support of either bridge. The support of bridge (v, v') is $\{u, v\}$ and the support of bridge (w, w') is all the remaining vertices other than f.

3 The algorithm

The first part of the MV algorithm for a phase finds the evenlevels and oddlevels of vertices. This part is organized in search levels, the first one being search level 0. Let $j_m = (l_m - 1)/2$, where l_m is the length of a minimum length augmenting path in G. Then during search level j_m, a maximal set of augmenting paths of length l_m is found and the current phase terminates. If $l_m = \infty$, i.e., there are no augmenting paths, the algorithm will reach a search level at which it has found the minlevels and maxlevels of all vertices reachable via alternating paths from the unmatched vertices. At this point it will halt and output the current matching, which will be maximum.

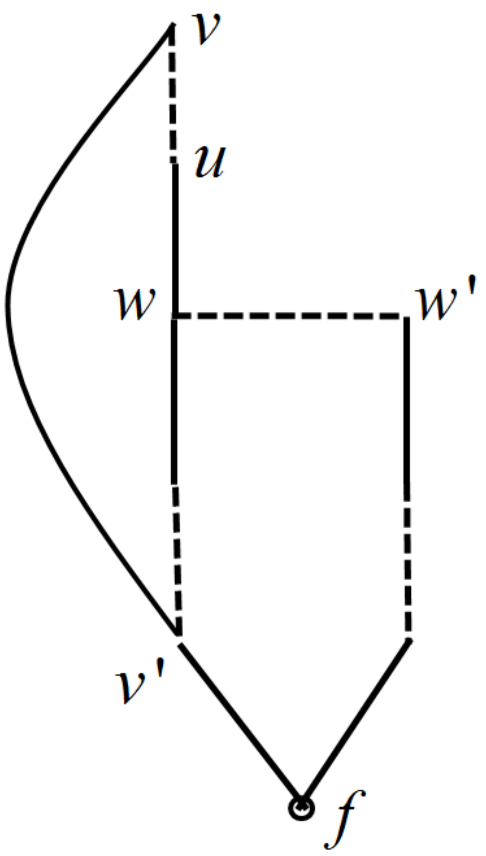

Figure 5: Edges (w, w') and (v, v') are bridges.

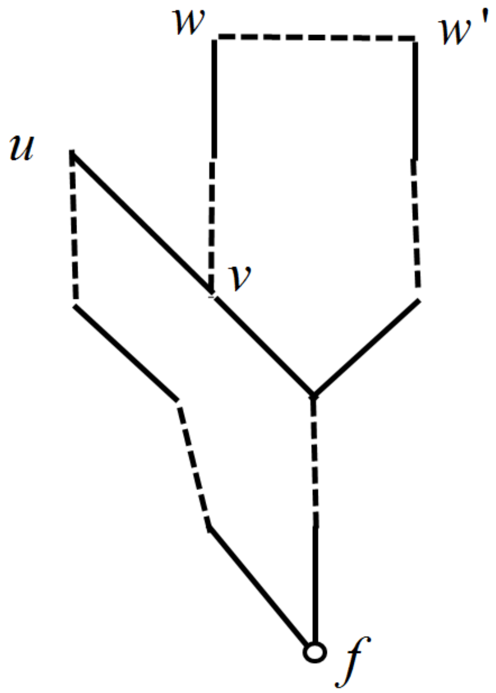

Figure 6: Edges (w, w') and (u, v) are bridges.

3.1 Procedures MIN and MAX

At the beginning of a phase, all unmatched vertices are assigned a minlevel of 0, and the rest are assigned a temporary minlevel of ∞; no maxlevels are assigned at this stage. During search level i procedure MIN finds all vertices, v, having minlevel$(v) = i + 1$ and assigns these vertices their minlevels. For each edge MIN scans, it also determines whether it is a prop or a bridge.

After MIN is done, procedure MAX uses the procedure DDFS, described in Section 4, to find all vertices, v, having tenacity$(v) = 2i + 1$ and assigns these vertices their maxlevels. Their minlevels are at most i and are already known and hence maxlevel$(v) = 2i + 1 -$ minlevel(v) can be computed. The precise synchronization of events stated above is an essential aspect of the algorithm and will be explained further below. See Algorithm 1 for a summary of the main steps; observe that the algorithm can be viewed as an intertwining of an alternating BFS (similar to the one used for bipartite graphs) with calls to the procedure DDFS.

Observe that in procedure MIN, if the predicate "minlevel$(v) \geq i + 1$" is true, then either the minlevel of v is still ∞ or it has already been found to be $i + 1$; even in the latter case, u is made a predecessor of v.

```
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 For

     End For

  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 For
         End For

     End For
     End For
```

In each search level, procedure MIN executes one step of alternating BFS as follows. If i is even (odd), it searches from all vertices, u, having an evenlevel (oddlevel) 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. If v 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. But if i is even, v may have one or more predecessors.

MIN is able to classify every edge as a prop or a bridge: this is natural, since a prop assigns minlevel to one of its endoints and a bridge doesn't. The algorithm also determines for each odd number t, the set of bridges of tenacity t; all these edges are inserted in the list $Br(t)$. 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 $Br(t)$. MIN is able to ascertain the tenacity of a bridge in all but one case.

This case is described next, together with an example.

Consider the edge (u, v) in Figure 6. 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 this time, tenacity(u, v) can be ascertained to be 13, and edge (u, v) is inserted in $Br(13)$.

To summarize, this case happens if (u, v) is 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. The evenlevel of v will be determined by MAX (observe that v is an inner vertex) at search level (tenacity$(v) - 1)/2$ and at this point, tenacity(u, v) is ascertained and the edge is inserted in $Br($tenacity$(u, v))$. The reader can verify that for each bridge in the first five figures, its tenacity gets ascertained by MIN (including the bridge (v, v') in Figure 5).

An important point to note in Figure 6, and in the case described above, is that tenacity$(v) <$ tenacity(u, v), thereby ensuring that bridge (u, v) will be processed by MAX at search level (tenacity$(u, v) - 1)/2$. Indeed, Task 2 in Theorem 8.2 proves that by the end of execution of procedure MIN at search level i, the algorithm would have identified, and determined the tenacity of, every bridge of tenacity $2i + 1$. Hence all such edges would be in the list $Br(2i + 1)$ at this point. At this time, procedure MAX gets executed. It uses DDFS to find the support of each of these bridges. This yields all vertices of tenacity $2i + 1$ and their maxlevels are calculated as indicated in Algorithm 1. If such a vertex, v is inner and has in incident unmatched bridge, say (u, v), then its tenacity is determined and it is inserted in $Br($tenacity$(u, v))$.

Next, let us explain a subtle point which will also lead to further insights into the idea of synchronization of events. This point is raised again in Theorem 8.2 and is illustrated via an example. For certain bridges, of tenacity $2i + 1$ say, the algorithm is able to classify them and determine their tenacity even before search level i. If so, why not conduct DDFS on such a bridge right away instead of waiting until search level i? In the example following Theorem 8.2, we have illustrated how doing so may lead to assigning wrong tenacity to certain vertices. Indeed, the steps of Algorithm 1 have been very precisely synchronized to ensure correct operation. This is also a key new idea underlying the MV algorithm.

3.2 Running DDFS on G

We now describe how DDFS is executed on the given graph, G. For this purpose, the reader is advised to read Section 4 in detail. This section describes DDFS using the layered, directed graph H which is a considerably simplified version of G. The mapping from G to H is specified gradually below. For graph H, we need to specify its vertices, including a layer for each vertex, and its edges. A specific set of vertices of G will form the vertices of H; the layer of any vertex in H will be its minlevel in G.

In the graph of Figure 7, MAX will perform DDFS on bridge (r_1, r_2), of tenacity 9, during search level 4, by starting two DFSs at vertices r_1 and r_2, respectively. We first state a preliminary rule for determining the edges of the corresponding graph H – the preliminary rule will suffice for this first DDFS. If the center of activity of a DFS is at u then it must search along all edges (u, v) where v is a predecessor of u.

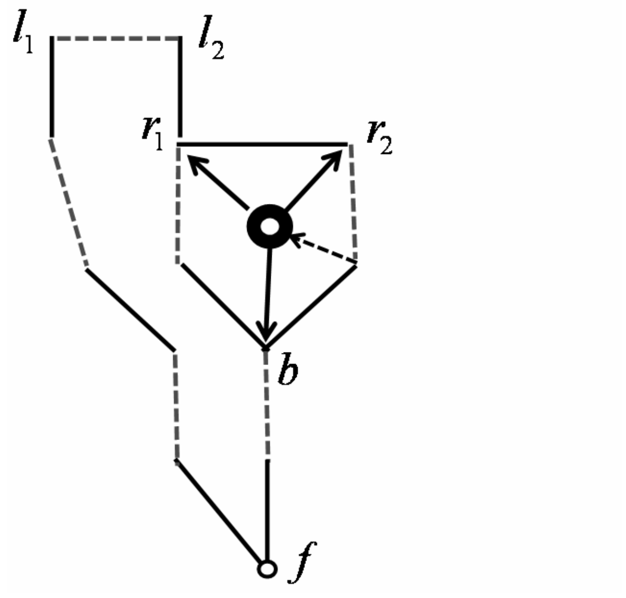

Figure 7: A new petal-node is created after DDFS on bridge (r_1, r_2).

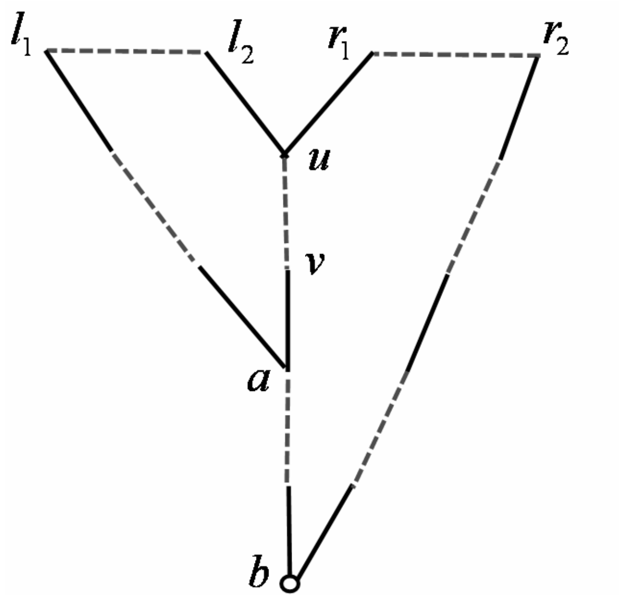

Figure 8: Vertices u and v are in the support of both bridges of tenacity of 11.

Clearly, this DDFS will terminate with the bottleneck b. It will visit the four vertices which constitute the support of bridge (r_1, r_2); observe that b is not in the support of bridge (r_1, r_2). These four vertices form a structure called a *petal*; each call to DDFS ends either with the creation of a new petal or an augmenting path.

The formation of a petal entails the following steps: The algorithm creates a new node, called a *petal-node*; this has the shape of a doughnut in Figure 7. The four vertices of the new petal point to the petal-node; to avoid cluttering Figure 7, only one vertex is pointing to the petal-node. The bottleneck, b, is called the *bud* of the petal. The new petal-node points to the two endpoints of its bridge, r_1 and r_2, and to its bud, b; the reason for the former will be clarified in Section 3.3 and for the latter below.

Petals are intimately connected to the central notion of blossom defined in Section 6; this connection is formalized in Lemma 6.13. We will make several comments below in order to clarify further this key connection. The reader will be able to follow these comments only after going over Section 6. For this reason, we will enclose these comments in square brackets. On a first reading these can be skipped; however, after acquiring the definition of blossoms, the reader is advised to review these comments.

[Observe that the petal created after performing DDFS on bridge (r_1, r_2) is also the blossom $\mathcal{B}_{b,9}$. In general, DDFS may not find an entire blossom but only a part of it. This part has been defined as a petal. As stated in Lemma 6.13, a blossom, in general, is a union of petals.]

Definition 3.1 (The bud of a vertex) If vertex v is in a petal and the bud of this petal is b then will say that the *bud of* v is b, written as $\text{bud}(v) = b$; $\text{bud}(v)$ is undefined if v is not in a petal. Next we define the function $\text{bud}^*(v)$. If v is not in any petal, $\text{bud}^*(v) = v$ else $\text{bud}^*(v) = \text{bud}^*(\text{bud}(v))$. The bud of a petal is always an outer vertex.

[A crucial difference between blossom vs petal and $\text{base}(v)$ vs $\text{bud}(v)$ is that whereas the former notions are defined graph-theoretically and independent of the run of the algorithm, the latter

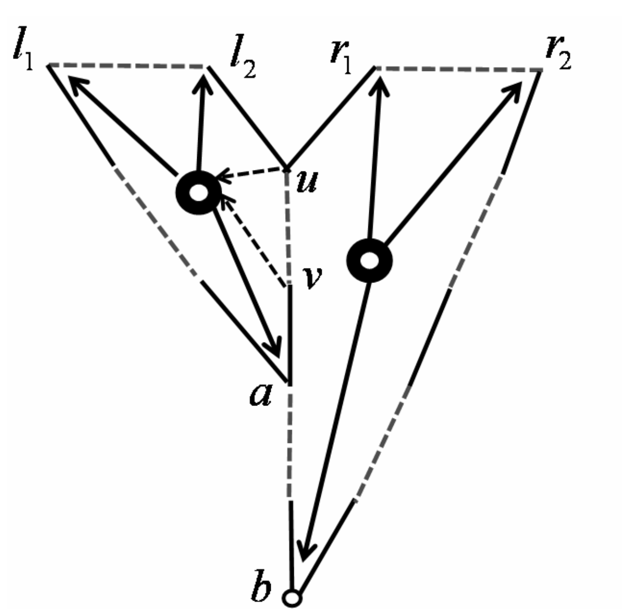

Figure 9: DDFS is performed on the left bridge first, then right.

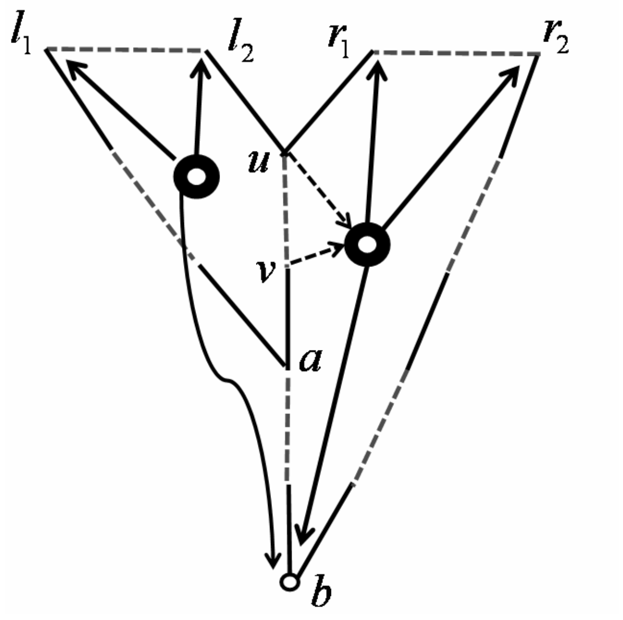

Figure 10: DDFS is performed on the right bridge first, then left.

depend on the manner in which the algorithm breaks ties. Several examples below clarify this. Additionally, bud*(v) not only depends on the particular run of the algorithm, it keeps changing as the algorithm proceeds. At any point, the algorithm uses its latest value.]

In Figure 7, a second DDFS is performed on bridge (l_1, l_2), of tenacity 11, during search level 5. To describe this DDFS, we need to state the complete rule for defining edges of the corresponding graph H: if the center of activity of a DFS is at u and it searches along edge (u, v), where v is a predecessor of u, then DFS must move the center of activity to bud*(v). Thus, when the DFS which starts at l_2 searches along edge (l_2, r_1), it moves the center of activity to b. It does so by following the pointer from r_1 to its petal-node and from the petal-node to the bud of this petal-node. In the process, the center of activity has jumped down more than one layer. The edges of H were allowed to jump down an arbitrary number of layers in order to model this.

This DDFS will end with bottleneck f. The new petal is precisely the support of bridge (l_1, l_2) and consists of the eight vertices of tenacity 11 in Figure 7, which includes b. Once again, a new petal-node is created and these eight vertices point to it. In addition, the petal-node points to l_1, l_2 and to f. [Observe that the blossom $\mathcal{B}_{f,11}$ consists of these eight vertices and the four vertices of blossom $\mathcal{B}_{b,9}$.]

Next consider the graph of Figure 8 which has two bridges of tenacity 11, (l_1, l_2) and (r_1, r_2). Observe that vertices u and v are in the support of both these bridges. Hence, the support of bridges need not be disjoint. Observe also that the base of these vertices is not a but b; note that tenacity(a) = 11. [There is only one blossom in this graph, i.e., $\mathcal{B}_{b,11}$.]

MAX will perform DDFS on these two bridges in arbitrary order during search level 5. Figure 9 shows the result of performing DDFS on (l_1, l_2) before (r_1, r_2). The first DDFS will end with bottleneck a. The new petal is precisely the support of (l_1, l_2), consisting of six vertices of tenacity 11, including u and v. The second DDFS, performed on (r_1, r_2), will end with bottleneck b and the new petal is precisely the difference of supports of (r_1, r_2) and (l_1, l_2), i.e., the remaining eight vertices of tenacity 11, including a. During this DDFS, when the DFS starting at r_1 searches along edge (r_1, u), it realizes that u is already in a petal and it jumps to a, the bud of this petal. This ensures that a vertex is included in at most one petal. In general, at the end of DDFS on bridge (u, v), the new petal will be the support of (u, v) minus the supports of all bridges processed thus far in this search level.

Figure 10 shows the result of performing DDFS on (r_1, r_2) before (l_1, l_2). The first petal is the support of (r_1, r_2), i.e., 10 vertices of tenacity 11, including a, u and v. The second petal is the difference of supports of (l_1, l_2) and (r_1, r_2), i.e., 4 vertices of tenacity 11. [In both cases, the union of the petals found is the blossom $\mathcal{B}_{b,11}$.]

Figure 11 illustrates the second way in which DDFS may end, i.e., instead of a bottleneck, it finds two unmatched vertices; this happens when DDFS is performed on bridge (u, v) of tenacity 11.

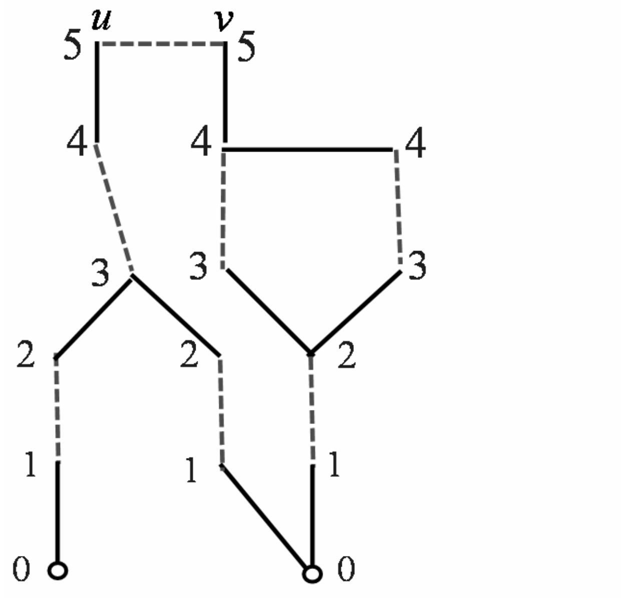

Figure 11: DDFS on the bridge of tenacity 11 ends with the two unmatched vertices.

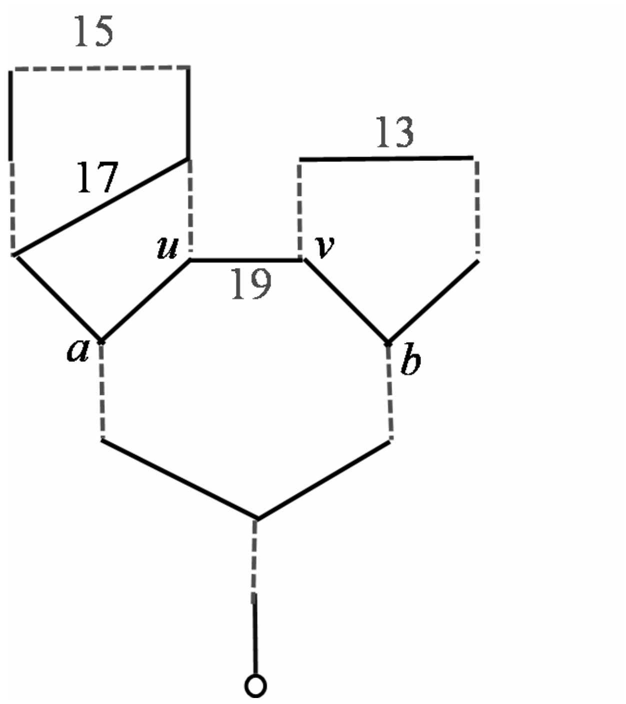

Figure 12: DDFS performed on bridge (u, v) starts the two DFSs at a and b, respectively.

We need to point out one final rule: if DDFS is performed on bridge (u, v), the centers of activity of the two DFSs must start at bud$^*(u)$ and bud$^*(v)$. This rule was vacuous so far, but will be applicable while processing bridge (u, v) in Figure 12. The tenacity of this bridge is 19 and it will be processed by MAX in search level 9. At that point in the algorithm, the bridges of tenacity 15 and 13 would already be processed and u and v will already be in petals, with bud(u) = bud$^*(u) = a$ and bud(v) = bud$^*(v) = b$. Hence the centers of activity of the two DFSs will start at a and b, respectively. Similarly, in Figure 24, when DDFS is performed on bridge (u, v) of tenacity 15, bud$^*(u) = u$ and bud$^*(v)$ will be the unmatched vertex.

All bridges considered so far had non-empty supports; however, this will not be the case in a typical graph. As an example, consider the edge of tenacity 17 in Figure 12. Since it does not assign minlevels to either of its endpoints, it is not a prop and is therefore a bridge. Clearly the support of this bridge is $\emptyset$. DDFS will discover this right away since the bud* of both of its endpoints is a. Clearly, DDFS needs to be run on all bridges, since that is the only way of determining whether the support of a given bridge is empty.

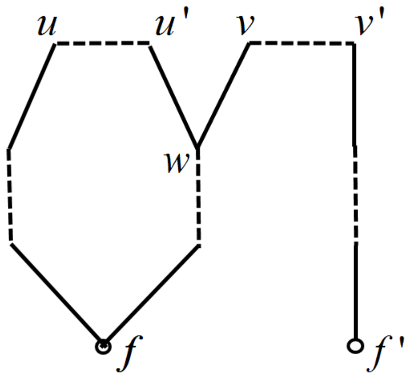

Figure 13: tenacity(u, u') = tenacity$(v, v') = 7 = l_m$.

3.3 Finding the augmenting paths

As stated at the beginning of this Section, during search level j_m, where $l_m = 2j_m + 1$ is the length of a minimum length augmenting path in G, a maximal set of such paths is found; observe that l_m is also the minimum oddlevel of an unmatched vertex. However, not every bridge of tenacity l_m leads to finding an augmenting path, e.g., in Figure 13, suppose DDFS is called with bridge (u, u') before bridge (v, v'). The first DDFS results in finding the bottleneck f. At this point a new petal node, with bud f, is created. In the second DDFS, the DFS started at v goes to its predecessor w, realizes that w is in a petal and jumps to bud$^*(w) = f$. The DFS started at v' follows predecessors and eventually reaches f'. Hence this DDFS terminates with two unmatched vertices and an augmenting path connecting them, of length $l_m = 7$, needs to be found next. This process is described in Section 3.3.1.

Note that besides the above-stated event, i.e., at a certain search level, DDFS ends in two unmatched vertices, another possibility is that G contains two or more unmatched vertices but it has no augmenting paths, i.e., the current matching is maximum (if so, no unmatched vertex will have a finite oddlevel). The algorithm will recognize this when it is done assigning minlevels and maxlevels to as many vertices as it could, i.e., it has explored the unmatched edges incident at all vertices having a finite evenlevel, the matched edge incident at all vertices having a finite oddlevel, and has performed DDFS on all bridges of finite tenacity that it has identified.

3.3.1 Finding one augmenting path

In Figure 14, minlevel$(u) >$ minlevel(v) and hence edge (u, v) is a bridge. DDFS on this bridge terminates with the two unmatched vertices, f_1 and f_2. At this point, the stack of the DFS starting at u (v) contains all the vertices of tenacity l_m that lie on the part of the augmenting path from u to f_1 (v to f_2); observe that the bottom of the latter stack will contain $b = \text{bud}(v)$. All vertices of tenacity less than l_m that constitute such an augmenting path are missing; they

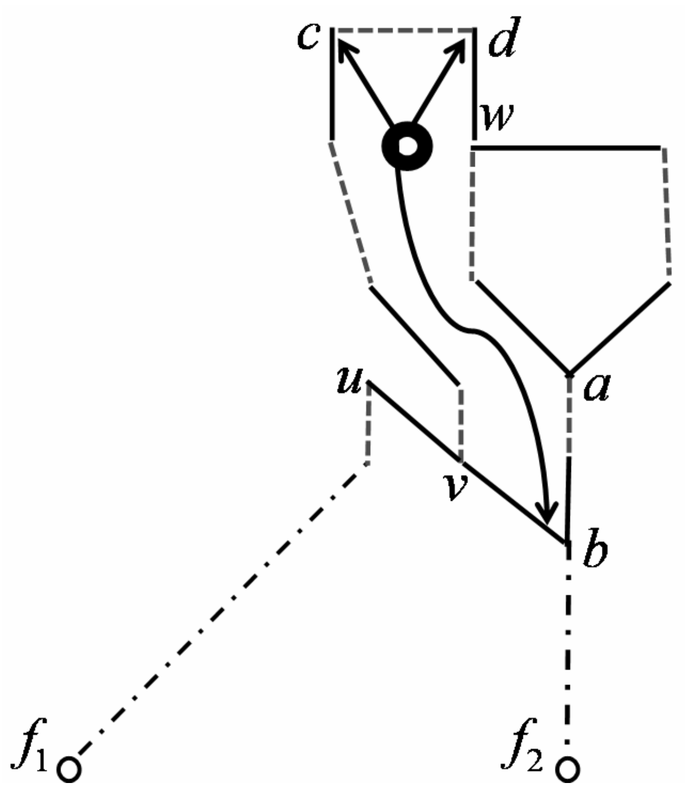

Figure 14: Constructing a minimum length augmenting path between unmatched vertices f_1 and f_2.

lie in petals whose bud*s, which are of tenacity l_m, sit on the two stacks. The procedure given below will find one such complete augmenting path by recursively "opening" the nested petals.

Let us show how to construct the path from v to f_2. Since bud$(v) = b$, we first need to find an evenlevel$(b; v)$ path from v to b. The actions are different depending on whether v is outer or inner; in this case v is inner. Therefore, evenlevel$(b; v) =$ maxlevel$(b; v)$, i.e., the path must use the bridge of this petal, which is (c, d). By jumping from v to its petal-node, the algorithm can get to the endpoints of this bridge. The "red" and "green" colors on the vertices of this petal, as assigned by DDFS (see Section 4), indicate that v was found via the DFS starting at c, say this is the red tree. The algorithm does a DFS on the red edges of this petal, starting from c, and finds a red path to v. Also, it does a DFS from d on the green edges to find a green path to b.

By the rules set above, the DFS that started at d must have skipped to $a =$ bud(w) while searching along edge (d, w). Therefore, the green tree yields the "path" d, a, b. At this point, we need to recursively "open" the petal whose bud is a and find an evenlevel$(a; w)$ path in it. Again, we ask whether w is outer or inner. This time the answer is "outer" and so we simply keep picking predecessors of vertices until we get from w to a. This path is inserted in the right place in the "path" from d to b. The path from b to f_2 is obtained via the same process: following down predecessors and recursively finding paths through any petals that are encountered on the way.

We note that if enough pointers had been kept around, in principle we could have found a path from v to c and a disjoint path from b to d by going "up" the petal. The main difficulty arises when a nested petal is encountered: which edge should be taken out of the nested petal? This issue is finessed completely by the procedure suggested above, i.e., using the petal node, jump to the endpoints, c and d, of the bridge of this petal and find the two paths by going "down" the petal. In this case, if a nested petal is encountered, we simply jump to its unique bud by using the petal node of this nested petal.

3.3.2 Finding a maximal set of paths

Next, we show how to find a maximal set of disjoint minimum length augmenting paths, thereby accomplishing the objective of a phase. After the first path, say p, is found, the next task is to identify the set of vertices that cannot be part of an augmenting path that is disjoint from p, and removing them from the graph. Clearly, the vertices of p will be removed, together with all edges incident at them. As a result, some of the left-over vertices may have no more predecessors. We will prove that such a vertex cannot be on an augmenting path that is disjoint from p. Recursively remove all such vertices until every remaining matched vertex that was assigned a minlevel has a predecessor; of course, unmatched vertices don't have predecessors.

At this point MAX will proceed processing bridges of tenacity l_m. When it encounters another bridge which makes DDFS end with two unmatched vertices, it finds another augmenting path. This continues until all bridges of tenacity l_m are processed. Lemma 8.4 shows that this will result in a maximal set of paths of length l_m.

4 Double depth first search

In order to facilitate wide dissemination, the procedure of double depth first search (DDFS) is expounded in a fully self-contained manner in this section. To make the exposition simpler, we will describe it in the simplified setting of a directed, layered graph H; the MV algorithm executes DDFS on the given graph, as described in Section 3. Additionally, the procedure of DDFS run on the layered graph H, together with its properties derived in this section, will also be used in a crucial way to give constructive proofs of certain facts in the central theorem, Theorem 6.12. As the name indicates, DDFS consists of two highly coordinated depth first searches.

The layered graph: The vertices of H are partitioned into $h+1$ layers, $l_h, \dots l_0$, with l_h being the highest and l_0 the lowest layer. Each directed edge runs from a higher to a lower layer, not necessarily consecutive. Graph H contains two special vertices, r and g, for *red* and *green*, not necessarily at the same layer. Furthermore, H satisfies:

DDFS Requirement: Starting from every vertex v in H, there is a path to a vertex in layer l_0.

A 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. Among the bottlenecks, if there are any, the one having highest level will be called the *highest bottleneck*. If so, we will denote it by b. Let V_b (E_b) be the set of all vertices (edges) that lie on all paths from r and g to b. If there are no bottlenecks, there must be distinct vertices r_0 and g_0 in layer l_0 and disjoint paths from r to r_0 and g to g_0. If so, 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 purpose of DDFS is to find the highest bottleneck if one exists. If so, DDFS needs to partition the vertices in $V_b - \{b\}$ into two sets R and G, with $r \in R$ and $g \in G$. Furthermore it needs to find two trees, T_r and T_g, rooted at r and g, and spanning the vertices of $R \cup \{b\}$ and $G \cup \{b\}$, respectively. If there is no bottleneck, DDFS needs to find the distinct vertices r_0 and g_0 in layer l_0, and vertex disjoint paths from r to r_0 and from g to g_0.

In summary, DDFS yields the following.

DDFS Certificate: In the first case, for every vertex $v \in V_b - \{b\}$, DDFS gives vertex disjoint paths from one of r, g to v and from the other of r, g to b. In the second case, it gives vertex disjoint paths from r to r_0 and from g to g_0, where r_0 and g_0 are distinct vertices in layer l_0.

Finally, the running time of DDFS needs to be a function of the output in the following manner: in the former case, DDFS needs to run in time $O(|E_b|)$ and in the latter case, it needs to run in time $O(|E_p|)$.

The two DFSs and their coordination: The two DFSs maintain their own stacks, S_r and S_g, which start with r and g, respectively. At any point in the search, each stack contains the vertices that have been visited by the corresponding DFS but have not yet backtracked from. Each DFS performs the usual operations, with one important modification. The latter is described below when we give the rules for coordinating the two DFSs. Because edges in H go from higher to lower levels, neither DFS will encounter back edges. The two DFSs start with their center of activity at r and g, respectively. Assume that the center of activity of a DFS is at u and it searches along edge (u, v). If v is not yet visited by either search, it pushes v onto its stack and moves its center of activity to v. In this case, u is declared *parent of* v. If v is already visited by either search, it considers the next unsearched edge incident at u – see below an exception, which is also the important modification mentioned above. If all edges incident at u have already been searched, it pops u from its stack and moves the center of activity to the parent of u.

We next give the rules for coordinating the two DFSs. To meet the running time requirement, we posit that if b is the highest bottleneck in H, then neither DFS will search along any edges below b. This gives our first rule: as soon as the center of activities of the two DFSs are at different levels, the one that is higher must move to catch up. If both are at the same level, we adopt the convention that red moves first. The exception mentioned above happens when one DFS searches along edge (u, v) and v happens to be the center of activity of the other DFS. In this case, v could potentially be a bottleneck and the two DFSs first need to determine whether it is. Furthermore, if v is not a bottleneck, the two DFSs need to determine which tree gets v.

When the two DFSs meet at a vertex: The procedure followed by the two DFSs at this point is the following, independent of which one got to v first. Let us assume that the red and green DFSs reached v via edges (v_r, v) and (v_g, v), respectively. First the green DFS backtracks from v and tries to reach a vertex, say w, with $w \neq v$ and $\text{level}(w) \leq \text{level}(v)$. If green succeeds, red moves its center of activity to v and it adds v to R and edge (v_r, v) to T_r, and DDFS proceeds. If the green fails, its stack, S_g, must be empty. Next, the red DFS backtracks from v and tries to reach a vertex, say w, with $w \neq v$ and $\text{level}(w) \leq \text{level}(v)$. If red succeeds, green moves its center of activity to v; however, it does not push v onto S_g, since it has already backtracked from v. In addition, it adds v to G and edge (v_g, v) to T_g, and DDFS proceeds. If red also fails, then its stack also must be empty and v is indeed the required bottleneck. If so, v is added to both R and G and edges (v_r, v) and (v_g, v) are added to T_r and T_g, respectively, and DDFS halts. If DDFS does not find a bottleneck, then eventually the two DFSs must find distinct vertices in l_0.

Theorem 4.1 *DDFS accomplishes the objectives stated above in the required time.*

Proof : The main difference between a usual DFS and the two DFSs implemented in DDFS arises when the two DFSs meet at a vertex, say v at layer l_j. Observe that once one DFS reaches a vertex at layer l, say, at every future point, the center of activity of at least one DFS will be at level l or lower. Therefore, since both DFSs just moved from higher layers to layer l_j, no other vertices at layer l_j or lower have yet been explored. Therefore, if v is not a bottleneck, there is an alternative path that reaches layer l_j or lower and which has not been explored so far. Since at this point both DFSs will consider all ways of finding such an alternative, they will find one and DDFS will proceed. On the other hand, if v is a bottleneck, there is no such alternative, and after considering all possibilities, both stacks will become empty and v will be declared the bottleneck.

If there is no bottleneck in H, by the arguments given above, the two DFSs will not get stuck at any vertex. Hence, one of them must reach a vertex at layer l_0, say v. Since v is also not a bottleneck, even if the two DFSs meet at v, one of them will find a way of reaching another vertex at layer l_0.

Clearly, the only edges searched by DDFS are those in E_b in the first case and E_p in the second. Furthermore, each such edge is examined at most twice, once in the forward search and once during backtrack. Hence DDFS accomplishes the stated objectives in the required time. $\square$

5 Limited BFS-honesty

In the next four sections, we introduce some crucial graph-theoretic notions and establish structural facts about them; these will be used in the proof of correctness given in Section 8.

Definition 5.1 (Limited BFS-honesty) Let p be an evenlevel(v) or oddlevel(v) path starting at unmatched vertex f and let u lie on p. Then $|p[f \text{ to } u]|$ 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.* We will say that u is BFS-honest w.r.t. p if $|p[f \text{ to } u]| = \text{evenlevel}(u)$ $(\text{oddlevel}(u))$ if u is even (odd) w.r.t. p.

Observe that in the graph of Figures 2 and 3, the vertices a, b and c are BFS-honest on all evenlevel and oddlevel paths to the vertices of tenacity α. (However, the vertices of tenacity α are not BFS-honest on oddlevel(a) and oddlevel(b) paths.) Theorem 5.3 uses the notion of tenacity of vertices to establishes limited BFS-honesty of minimum length alternating paths.

Lemma 5.2 *If (u, v) is a matched edge, then* $\text{tenacity}(u) = \text{tenacity}(v) = \text{tenacity}(u, v)$.

Proof : If (u, v) is a matched edge, $\text{evenlevel}(v) = \text{oddlevel}(u) + 1$ and $\text{evenlevel}(u) = \text{oddlevel}(v) + 1$. The lemma follows. $\square$

As a result of Lemma 5.2, in several proofs given in this paper, it will suffice to restrict attention to only one of the end points of a matched edge.

Theorem 5.3 *Let p be an* evenlevel(v) *or* oddlevel(v) *path starting at unmatched vertex f and let vertex $u \in p$ with* $\text{tenacity}(u) \geq \text{tenacity}(v)$. *Then u is BFS-honest w.r.t. p. Furthermore, if* $\text{tenacity}(u) > \text{tenacity}(v)$ *then* $|p[f \text{ to } u]| = \text{minlevel}(u)$.

Proof : Assume w.l.o.g. that p is an evenlevel(v) path and that u is even w.r.t. p (by Lemma 5.2). Suppose u is not BFS-honest w.r.t. p, and let q be an evenlevel(u) path, i.e., $|q| < |p[f \text{ to } u]|$. First consider the case that $\text{evenlevel}(v) = \text{maxlevel}(v)$, 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$(u) \leq |r| + |p[u \text{ to } v]|$. Additionally, evenlevel$(u) < |p[f \text{ to } u]|$, 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, $q[f \text{ to } w] \circ p[w \text{ to } u]$ is an odd path to u with length less than evenlevel(v). Again we get tenacity$(u) <$ tenacity(v), leading to a contradiction.

We next prove the second claim. The claim is obvious if evenlevel$(v) =$ minlevel(v), so let us assume that 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$(u) \leq |r| + |p[u \text{ to } v]|$. 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). □

As a consequence of Theorem 5.3, if a vertex u on path p fails to satisfy BFS-honesty, then it must have strictly smaller tenacity than tenacity(v). In Theorem 6.12 Statement 5 we will show that even such a vertex needs to satisfy certain other properties.

6 Base, blossom and bridge

A marked difference in the structure of the proof given in [Vaz94] and the current paper is the following. [Vaz94] first proved the existence of base of a vertex and used it to define blossoms. It then established properties of minimum length alternating paths as they traverse through the blossoms. Using these properties, it proved the central fact needed for finding maxlevels of vertices: Every maxlevel(v) path contains a unique bridge having tenacity = tenacity(v). Each of these proofs involved studying intersections of minimum length alternating paths with each other and with other structures via complicated case analyses which are, in retrospect, incorrect at many places.

The key to avoiding this complication, and giving a well-founded proof, is to exploit the fact that a blossom $\mathcal{B}$ contains, for each vertex $v \in \mathcal{B}$, all minimum length alternating paths from the base of $\mathcal{B}$ to v. The complicated intersections of these paths are thereby encapsulated inside the blossom, and the blossom offers, to the rest of the graph, a surprisingly simple interface.

Let v be vertex of tenacity t, with $t_m \leq t < l_m$ 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. Consider all vertices of tenacity $> t$ on p; this set is non-empty since, by the choice of t, it contains f. Among these vertices, define the highest one to be *the base of v w.r.t. p*, denoted $F(p, v)$. Clearly $F(p, v)$ must be even w.r.t. p. Therefore, since tenacity$(F(p, v)) >$ tenacity(v), by Theorem 5.3, minlevel$(F(p, v)) =$ evenlevel$(F(p, v))$. Hence $F(p, v)$ is an outer vertex. Define the set,

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

Next assume that v has evenlevel(v) and oddlevel(v) paths starting at unmatched vertex f. Define the set,

$$B_f(v) = \{F(p, v) \mid p \text{ is an evenlevel}(v) \text{ or oddlevel}(v) \text{ path starting at } f\}.$$

The following claim leads to the central definition of base of a vertex.

Claim 6.1 *Let v be vertex of tenacity t, with $t_m \leq t < l_m$. Then the set $B(v)$ is a singleton.*

It turns out that the proof of Claim 6.1 requires, for the reason stated above, the notion of blossoms, On the other hand, blossoms can be defined only after defining the base of a vertex, which can only be done after this claim is proven. This deadlock is broken via an induction on tenacity: for each value of tenacity, say t, once Claim 6.1 is proven for vertices of tenacity t, their base can be defined, following which, blossoms of tenacity t can be defined. Then properties of these blossoms (laminarity) and properties of paths traversing through these blossoms are established. All these facts are then used in the next step of the induction.

This elaborate induction is given in Theorem 6.12. Since this theorem 6.12 proves many facts all at once, and is very long, we will prove its parts separately. Some of these parts are proven conditionally and are called Propositions below and in Section 7. We will also state some conditional definitions, which are clearly indicated below. We define the condition next.

Definition 6.2 (Claim(t)) For t odd, with $t_m \leq t < l_m$, **Claim(t)** will denote the assumption that Claim 6.1 holds for all vertices v such that $t_m \leq$ tenacity(v) $\leq t$.

For a particular value of t, once **Claim(t)** is proven in the appropriate step of the induction in Theorem 6.12, the conditional lemmas and definitions, for this value of t, will start holding unconditionally. Once Theorem 6.12 is fully proven, the central definitions of base of a vertex and blossom will start holding unconditionally and will be used as such in the rest of the paper.

Conditional Definition 6.3 (Base of a vertex)
For t odd, with $t_m \leq t < l_m$, assume **Claim(t)**. Then, for each vertex v of tenacity $\leq t$, define its base to be the singleton vertex in the set $B(v)$. We will denote it by base(v).

As observed above, $F(p, v)$ is always an outer vertex. Hence the base of a vertex is outer. Furthermore, by Theorem 5.3, base(v) is BFS-honest on every evenlevel(v) or oddlevel(v) path. In the graph of Figures 2 and 3, the base of each vertex of tenacity α is a, tenacity κ is b, and tenacity τ is f, respectively. In Figure 17, b is the base of v, v', w and w'. In Figure 16, f is the base of v, v', w and w'.

Remark: The condition "$t < l_m$" is essential in Claim 6.1 and in Definition 6.3. Figure 15 gives a counter-example. In this graph, $l_m = 3$ and the vertices u and v, both of tenacity 3, have no base, since the evenlevel and oddlevel paths to these vertices do not contain any vertex of tenacity greater than 3.

Conditional Definition 6.4 (Blossom) For t odd with $t_m \leq t < l_m$, assume **Claim(t)**. Blossoms will be defined recursively. Let b be an outer vertex with tenacity(b) $> t$. We will denote the blossom of tenacity t and base b by $\mathcal{B}_{b,t}$. Define $\mathcal{B}_{b,1} = \emptyset$. If $t \geq 3$, let $S_{b,t} = \{v \mid \text{tenacity}(v) = t \text{ and base}(v) = b\}$ and define

$$\mathcal{B}_{b,t} = S_{b,t} \cup \left(\bigcup_{v \in (S_{b,t} \cup \{b\}),\ v \text{ outer}} \mathcal{B}_{v,t-2} \right).$$

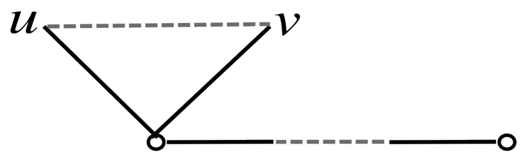

Figure 15: Vertices u and v have no base.

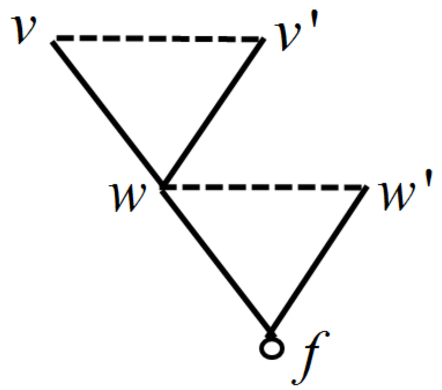

Figure 16: Vertex f is the base of v, v', w, w'.

It is obvious from Definition 7.1 that if $v \in \mathcal{B}_{b,t}$ then tenacity$(v) \leq t$. In particular, observe that $b \notin \mathcal{B}_{b,t}$. We will say that blossom $\mathcal{B}_{b',t'}$ *is nested in* blossom $\mathcal{B}_{b,t}$ if $\mathcal{B}_{b',t'} \subset \mathcal{B}_{b,t}$. From the recursive definition given above, it follows that $t' < t$.

In the graph of Figures 2 and 3, the blossom $\mathcal{B}_{a,\alpha}$ consists of vertices of tenacity α, the blossom $\mathcal{B}_{b,\kappa}$ consists of vertices of tenacity α and κ, and the blossom $\mathcal{B}_{f,\tau}$ consists of vertices of tenacity α, κ and τ. In Figure 17, blossom $\mathcal{B}_{b,7} = \{w, w'\}$ and blossom $\mathcal{B}_{b,11} = \{w, w', v, v'\}$. In Figure 16, blossom $\mathcal{B}_{f,3} = \{w, w'\}$ and blossom $\mathcal{B}_{f,7} = \{w, w', v, v'\}$; clearly, the former is nested in the latter.

Lemma 6.5 *Let v be vertex with $t_m \leq$ tenacity$(v) < l_m$. There is an* evenlevel(v) *path starting at unmatched vertex f if and only if there is an* oddlevel(v) *path starting at f.*

Proof : By Lemma 5.2, it suffices to prove only the forward direction. By the assumption on tenacity(v), v has paths of both parity. Let p be an evenlevel(v) path starting at unmatched vertex f and suppose q is an oddlevel(v) path starting at unmatched vertex $f' \neq f$. We will show that there must also be an oddlevel(v) path starting at unmatched vertex f.

Let u be the lowest vertex w.r.t. q that lies on p as well. If u is even w.r.t. p, then there is an augmenting path from f' to f of length less than t_m. Therefore, u is odd w.r.t. p. Since $q[f' \text{ to } u] \circ p[u \text{ to } v]$ is a valid even alternating path, $|q[f' \text{ to } u]| \geq |p[f \text{ to } u]|$, otherwise we can get a shorter $e(v)$ path than p. If $q(u \text{ to } v] \cap p[f \text{ to } u) = \emptyset$, then $p[f \text{ to } u] \circ q[u \text{ to } v]$ is a valid odd alternating path of length at most $|q|$ and we are done.

Otherwise, let (w', w) be lowest matched edge of p that is traversed by q, with w even w.r.t. p. If w is odd w.r.t. q, then again we can get an augmenting path from f' to f of length less than t_m.

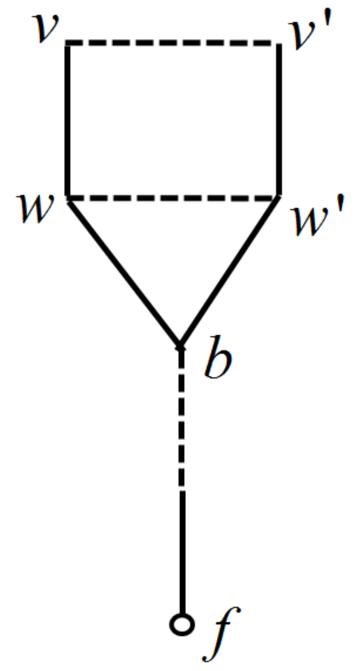

Figure 17: Vertex b is the base of v, v', w, w'.

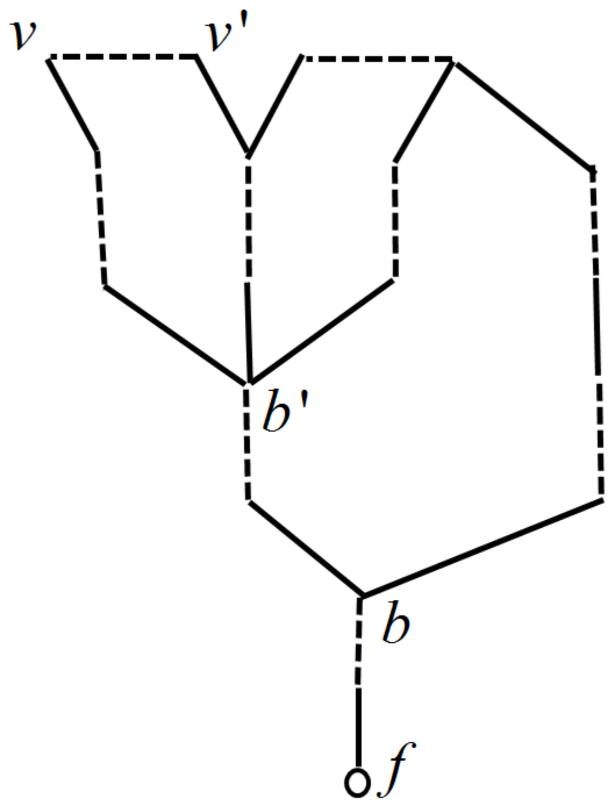

Figure 18: Blossom $\mathcal{B}_{b,15}$ is of minimum tenacity. Note that $H(v, v') = b'$.

In the remaining case, $p[f \text{ to } w] \circ q[w \text{ to } v]$ is a shorter odd alternating path than q, leading to a contradiction. This proves the lemma. □

Blossoms, as defined by Edmonds, also form nested structures. Let us point out an important difference between the two from the viewpoint of nesting. Under Edmonds' definition, an innermost blossom is simply an odd length alternating cycle. In our case, an innermost blossom is, in general, far more elaborate, Figure 18 gives an example of such a blossom. Clearly, a blossom of tenacity t_m must be the innermost blossom in any nesting. The blossom in Figure 18 is in fact of tenacity t_m. Lemmas 6.7 will establish the base case of Claim 6.1, i.e., for tenacity t_m. The complexity of the proof of Lemma 6.7 is natural in the face of the complexity of the innermost blossom.

Let f be an unmatched vertex and t be an odd number with $t_m \leq t < l_m$. We will denote by $V_t(f)$ the set of vertices having tenacity t and having evenlevel and oddlevel paths starting at f. For such a vertex $v \in V_t(f)$, consider all evenlevel(v) and oddlevel(v) paths starting at f, and consider every vertex of tenacity $> t$ that lies on *each* of these paths; in particular, f is such a vertex. Among these vertices, pick the one that is highest and denote it by $A_f(v)$.

The proof of the next lemma is straightforward — the idea behind it is essentially the same as that for Lemma 5.2 — and is omitted.

Lemma 6.6 *Let (u, v) be a matched edge of tenacity t, with $t_m \leq t < l_m$. Then*[5] *$u \in V_t(f)$ if and only if $v \in V_t(f)$. Furthermore, if u and $v \in V_t(f)$, then $A_f(u) = A_f(v)$, $B_f(u) = B_f(v)$ and $B(u) = B(v)$.*

The next lemma proves the base case of the first two statements of Theorem 6.12.

Lemma 6.7 *For every vertex v of tenacity t_m, the following hold:*

Statement 1: *The set $B(v)$ is a singleton.*

Statement 2: *Every* maxlevel(v) *path contains a unique bridge of tenacity t_m.*

Proof : The proof is quite elaborate and has been split into claims, each one establishing a basic fact.

Let f be an unmatched vertex such that $v \in V_{t_m}(f)$. We will first prove:

Statement 1': The set $B_f(v)$ is a singleton, and $B_f(v) = A_f(v)$.

Claim 1 *It suffices to prove Statement 1' and Statement 2 for inner vertices in $V_{t_m}(f)$.*

Proof : From the proof of Lemma 5.2, it is easy to see that the matched neighbor of any outer vertex, say v, is an inner vertex[6], say v'. Furthermore, by Lemma 6.6 if v is in $V_{t_m}(f)$, so is v'. Next observe that every maxlevel(v') path yields a maxlevel(v) path by removing the last edge, i.e, (v, v'), and every maxlevel(v) path yields a maxlevel(v') path by concatenating the edge (v, v'). Hence proving Statement 1' for v' also proves it for v. For Statement 2, by Lemma 6.6, proving it for v' also proves it for v. $\square$

Let v be an inner vertex in $V_{t_m}(f)$. We will next define a graph G_v whose structural properties will lead to a proof of the lemma. Let $A_f(v) = b$. Consider all evenlevel(v) and oddlevel(v) paths starting at f, and denote by G_v all the vertices and edges on these paths. Let $\boldsymbol{\beta} =$ minlevel(b) and $\boldsymbol{\alpha} = (t_m - 1)/2$; clearly $\boldsymbol{\alpha}$ is the maximum possible minlevel of a vertex of tenacity t_m. By definition of t_m, every vertex on an evenlevel(v) or oddlevel(v) path is BFS-honest on it. Graph G_v is a layered graph, similar to graph H defined in Section 4, with one exception (made for the sake of convenient notation), namely it will have edges running between pairs of vertices at layer $\boldsymbol{\alpha}$. G_v has $\boldsymbol{\alpha} + 1$ layers numbered 0 to $\boldsymbol{\alpha}$.

The layer of each vertex in G_v is defined to be its minlevel. Thus layer 0 has only f in it. Clearly, all the props run between adjacent layers of G_v. Furthermore, G_v satisfies the DDFS Requirement (stated in Section 4), that starting from any vertex, there is a path to the lowest layer, i.e., vertex f.

Claim 2 *Let p be an* evenlevel(v) *path in G_v. Then p contains a bridge of tenacity t.*

[5] Observe that by Lemma 5.2, tenacity$(u) =$ tenacity$(v) = t$.

[6] Observe that the reverse is not true, since both endpoints of a matched edge could be inner vertices.

Proof : Clearly, p will contain an edge (u, u') such that minlevel(u) = minlevel$(u') = \boldsymbol{\alpha}$. We will show that (u, u') is a bridge of tenacity t. Any minlevel(u) path concatenated with the edge (u, u') gives a maxlevel(u') path and vice versa. Hence tenacity(u) = tenacity$(u') = t_m$. Furthermore, the predecessors of u and u' are vertices at with minlevel $\boldsymbol{\alpha} - 1$. Hence (u, u') is a bridge. □

We will prove Statement 1' by induction on minlevel(v), for $v \in V_{t_m}(f)$. We will show that for each $l \in [\boldsymbol{\beta} + 1, \boldsymbol{\alpha}]$, all vertices in G_v that have minlevel l must have tenacity t_m, thereby proving that $B_f(v) = \{b\}$. Before proceeding with the induction, let us establish some structural properties which will be used both by the induction basis and the induction step.

Let (u, u') be a bridge of tenacity t_m in G_v. Consider all possible minlevel(u) and minlevel(u') paths and say that vertex w is a *bottleneck* if it lies on all such paths. Denote the highest (in minlevel) bottleneck by $H(u, u')$; clearly, this will be an outer vertex.

Claim 3 *$H(u, u')$ lies on every* oddlevel(v) *path.*

Proof : Suppose not, and consider an oddlevel(v) path that does not use w. This path can be extended to a minlevel(u) or minlevel(u') path, thereby contradicting the fact that $H(u, u')$ is a bottleneck for all such paths. □

Define the set $S(u, u')$ as follows. For each minlevel(u) and minlevel(u') path, include in $S(u, u')$ all vertices that are higher than $H(u, u')$.

Claim 4 *For each vertex $w \in S(u, u')$,* tenacity$(w) = t_m$.

Proof : Clearly, DDFS run on G_v with bridge (u, u') must end with the bottleneck $H(u, u')$, and it will visit each vertex in $S(u, u')$. Therefore, for each $w \in S(u, u')$, the concatenation of the two paths established by the DDFS Certificate, together an evenlevel$(H(u, u'))$ path and the edge (u, u'), yields a maxlevel(w) path, which shows that tenacity$(w) = t_m$. □

Induction basis: Let v be a vertex of minimum minlevel in $V_{t_m}(f)$; clearly v is inner. Let $A_f(v) = b$. defined above.

Claim 5 *Let p be an* oddlevel(v) *path. Then last edge of p is (b, v). Furthermore, there is a bridge (u, u') in G_v such that $H(u, u') = b$.*

Proof : Suppose not, and let the last edge of p be (w, v). Now, an evenlevel(v) path concatenated with edge (v, w) gives an odd alternating path to w thereby proving that tenacity$(w) = t_m$. However, minlevel$(w) <$ minlevel(v), which contradicts the choice of v, thereby proving that the last edge of p is (b, v).

Let (u, u') be a bridge on any evenlevel(v) path, say q. Since $p[b$ to $v]$ is simply the edge (b, v), the only bottleneck on $q[b$ to $v]$ is b. Hence $H(u, u') = b$. □

By Claim 5, for any evenlevel(v) path p, every vertex on $p(b$ to $v]$ has tenacity t_m. Therefore $B_f(v) = A_f(v) = \{b\}$, hence proving the induction basis.

Induction step: Let v be an inner vertex in $V_{t_m}(f)$ with minlevel$(v) = l$, where $l \in (\boldsymbol{\beta}+1, \boldsymbol{\alpha}]$, and assume that Statement 2 holds for every vertex in $V_{t_m}(f)$ having minlevel $< l$. Consider all bridges of tenacity t_m that lie on evenlevel(v) paths. For each one, say (u, u'), consider the bottleneck $H(u, u')$. Among these bottlenecks, let w be one having lowest minlevel. As shown in Claim 3, w lies on all oddlevel(v) paths.

Claim 6 *w lies on every* evenlevel(v) *path.*

Proof : Suppose p is an evenlevel(v) path that does not contain w, and let (u, u') be the bridge on this path. Now there are two cases: either there is an oddlevel(v) path q such that p shares a matched edge (y, y') on $q(w \text{ to } v)$ or there is no such oddlevel(v) path. In the first case, $p[f \text{ to } y] \circ q[y \text{ to } v]$ is an oddlevel(v) path not using w. In the second case, $H(u, u')$ is below w. Both cases lead to contradictions. $\square$

Since w is the highest bottleneck for all oddlevel(v) and evenlevel(v) paths, any vertex $z \in G_v$ with minlevel$(z) >$ minlevel(w), $z \in S(u, u')$ for some bridge (u, u') of tenacity t_m that lies on an evenlevel(v) path. Therefore, by Claim 4, tenacity$(z) = t_m$. Now there are two cases: $w = b$ and $w \neq b$. In the first case, we have established that $B_f(v) = A_f(v) = b$.

Claim 7 *If $w \neq b$, then* tenacity$(w) = t_m$ *and $B_f(w) = A_f(w) = b$.*

Proof : If tenacity$(w) > t_m$, $A_f(v) = w$, leading to a contradiction. Furthermore, since $mn(w) <$ minlevel(v), the induction hypothesis applies to w, and $B_f(w) = A_f(w)$.

We next establish that $A_f(w) = b$. Every evenlevel(w) path, p, can be extended to an oddlevel(v) path and therefore $F(p, w) = b$. Suppose there is an oddlevel(w) path q with $F(q, w) \neq b$. Then either there is an evenlevel(w) path p such that q shares a matched edge (y, y') on $p(b \text{ to } w)$ or there is no such evenlevel(w) path. In the first case, $r = q[f \text{ to } y] \circ p[y \text{ to } w]$ is an evenlevel(w) path such that $F(r, w) \neq b$. In the second case, $q \circ p[w \text{ to } b]$ is an oddlevel(b) path showing that tenacity$(b) = t_m$; here p is any evenlevel(w) path. Both cases lead to contradictions. Therefore $F(q, w) = b$, and hence $A_f(w) = b$. $\square$

Claim 8 *If $w \neq b$, then for every* evenlevel(v) *and* oddlevel(v) *path p, every vertex on $p(b$ to $w)$ is of tenacity t_m.*

Proof : Since w occurs on every evenlevel(v) and oddlevel(v) path p and is BFS-honest on it, p consists of an evenlevel(w) path concatenated with a appropriate path from w to v. Since $B_f(w) = b$, for any evenlevel(w) path q, every vertex on $q(b \text{ to } w)$ is of tenacity t_m. Hence every vertex on $p(b \text{ to } w)$ is of tenacity t_m. $\square$

Therefore we have established that $B_f(v) = A_f(v) = \{b\}$ in second case as well, i.e., $w \neq b$. This completes the proof of Statement 1'. The next claim will enable us to prove Statement 1.

Claim 9 *Let f and f' be unmatched vertices. For every vertex $v \in V_{t_m}(f) \cap V_{t_m}(f')$, the following holds: $B_f(v) = B_{f'}(v)$.*

Proof : Suppose $B_f(v) \neq B_{f'}(v)$, and let $b = B_f(v)$ and $b' = B_{f'}(v)$, with minlevel$(b) \leq$ minlevel(b'). All evenlevel(v) and oddlevel(v) paths from f use b and not b', and those from f' use b' and not b. In particular, any evenlevel(b) path is vertex disjoint from any evenlevel(b') path.

Let $S_1 \subseteq V_{t_m}(f)$ be the set of vertices[7] whose evenlevel and oddlevel paths contain b, and $S_2 \subseteq V_{t_m}(f')$ be the set of vertices whose evenlevel and oddlevel paths contain b'. Let S be the set of vertices of minimum minlevel in $S_1 \cap S_2$; clearly they are all inner. First consider the case that there is $v \in S$ that is adjacent to b or b', say the former. Then an oddlevel(v) path containing b concatenated with an evenlevel(v) path containing b' will be a simple alternating path and hence an augmenting path from f to f' of length t_m, contradicting the assumption $t_m < l_m$.

Pick any $v \in S$ and let (w, v) be the last edge on an oddlevel(v) path that contains b. Let p be an evenlevel(v) path that contains b'. Then p concatenated with the edge (v, w) yields an oddlevel(w) path that contains b'. Now applying Lemma 6.5 we get that $w \in S_1 \cap S_2$. Since minlevel$(w) <$ minlevel(v), we get a contradiction. Hence $B_f(v) = B_{f'}(v)$. $\square$

Claim 9 immediately implies that $B(v)$ is a singleton, thereby proving Statement 1. It further implies that $S_1 = S_2$ and hence Statement 2 follows from Claim 2. $\square$

Remark: It is easy to construct an example in which $u, v \in V_{t_m}(f)$ and $B_f(u) \neq B_f(v)$.

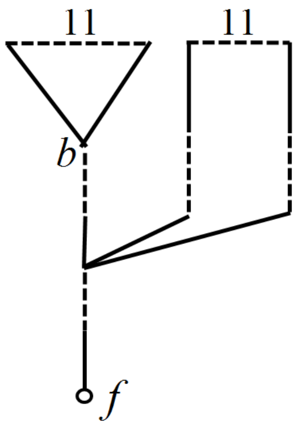

Figure 19: $\mathcal{B}_{b,11}$ is a proper subset of $V_{11}(f)$.

Conditional Definition 6.8 (Iterated bases of a vertex)
For t odd, with $t_m \leq t < l_m$, assume **Claim(t)**.
Let v be a vertex such that tenacity$(v) = t' \leq t$. The following are the iterated bases of v:
Define base$^1(v)$ = base(v), and for $k \geq 1$, if tenacity(base$^k(v)) \leq t$, then define base$^{k+1}(v)$ = base(base$^k(v)$).

In the graph of Figures 2 and 3, base$^1(v) = a$, base$^2(v) = b$ and base$^3(v) = f$.

Conditional Definition 6.9 (Shortest path from an iterated base to vertex)
For t odd, with $t_m \leq t < l_m$, assume **Claim(t)**.
Let v be a vertex such that tenacity$(v) \leq t$, and let $k \in \mathbf{Z}^+$ such that tenacity(base$^k(v)) \leq t$. Let

[7]As shown in Figure 19, $S_1 \subset V_{t_m}(f)$ is possible.

$\text{base}^{k+1}(v) = b$. Then by an evenlevel$(b; v)$ (oddlevel$(b; v)$) path we mean a minimum even (odd) length alternating path from b to v that starts with an unmatched edge.

Proposition 6.10 *For t odd with $t_m \leq t < l_m$, assume* **Claim(t)**.
Let v be a vertex such that tenacity$(v) = t$ *and* base$(v) = b$. *Then every* evenlevel(v) *(*oddlevel(v)*) path consists of an* evenlevel(b) *path concatenated with an* evenlevel$(b; v)$ *(*oddlevel$(b; v)$*) path.*

Proof : Let p be an evenlevel(b) path starting at unmatched vertex f and q be an evenlevel$(b; v)$ path. If their concatenation is longer than evenlevel(v) then q must intersect p below b. Let (w, w') be the lowest matched edge of p used by q, where w' is even w.r.t. p. Now, using the same arguments as those in the proof of Theorem 5.3, one can show that if w is odd w.r.t. q then there is an even path from f to v that is shorter than evenlevel(v) and if w' is odd w.r.t. q then there is a short enough odd path from f to b which gives tenacity$(b) \leq t$. □

The next lemma can be proven unconditionally and will be needed critically in Theorem 6.12.

Lemma 6.11 *Let v be vertex of tenacity t, with $t_m \leq t < l_m$, let p be a* minlevel(v) *path, and let $b = F(p, v)$. Let u be a vertex on $p[b$ to $v]$ satisfying* tenacity$(u) < t$, *edge (u, w) lies on p with w higher than u, and* tenacity$(w) = t$. *Then u is BFS-honest w.r.t. p.*

Proof : Assume w.l.o.g. that p is an evenlevel(v) path (by Lemma 5.2). Clearly, u is even w.r.t. p and w is odd w.r.t. p. Suppose u is not BFS-honest w.r.t. p, and let q be an evenlevel(u) path, i.e., $|q| < |p[f \text{ to } u]|$.

Consider the first vertex, say z, 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 z is odd w.r.t. p then again we get an even path to v that is shorter than evenlevel(v). Hence z must be even w.r.t. p. Then, $q[f \text{ to } z] \circ p[z \text{ to } w]$ is an even path to w with length less than minlevel(v). Furthermore, since w is odd w.r.t. p, oddlevel$(w) <$ minlevel(v). Hence tenacity$(w) <$ tenacity(v), leading to a contradiction. □

Theorem 6.12 *Let t be an odd number with $t_m \leq t < l_m$, and let v be a vertex of tenacity t. The following hold:*

Statement 1: *The set $B(v)$ is a singleton.*

Statement 2: *Every* maxlevel(v) *path contains a unique bridge of tenacity t.*

Statement 3: *The blossoms of tenacity at most t form a laminar family.*

Statement 4: *Let* base$(v) = b$ *and $u \in \mathcal{B}_{b,t}$. Then every* evenlevel(u) *(*oddlevel(u)*) path consists of an* evenlevel(b) *path concatenated with an* evenlevel$(b; u)$ *(*oddlevel$(b; u)$*) path. Moreover, every* evenlevel$(b; u)$ *and* oddlevel$(b; u)$ *path lies in $\mathcal{B}_{b,t} \cup \{b\}$. Furthermore, every edge on every* evenlevel$(b; u)$ *and* oddlevel$(b; u)$ *path is of tenacity $\leq t$.*

Statement 5: *Let* base$(v) = b$, *p be an* evenlevel(v) *or* oddlevel(v) *path, and u lie on $p(b$ to $v]$. Then* base(u) *lies on $p[b$ to $v]$.*

Additionally, for every vertex v of tenacity l_m, every maxlevel(v) *path contains a unique bridge of tenacity l_m.*

Proof :
Induction basis: Statements 1 and 2 are proven in Lemma 6.7. As a result base$(v) = b$, say, and blossom $\mathcal{B}_{b,t_m}$ are unconditionally defined. If d and d' are distinct vertices of tenacity $> t_m$ then the blossoms $\mathcal{B}_{d,t_m}$ and $\mathcal{B}_{d',t_m}$ are disjoint sets, since a vertex cannot have both d and d' as its base, leading to a proof of Statement 3 holds. Observe that vertex u in Statement 4 must have tenacity t_m. The first part of this statement follows from Proposition 6.10. For the second part, by Lemma 6.7, every vertex $w \neq b$ on an evenlevel$(b; u)$ or oddlevel$(b; u)$ path has tenacity t_m and base b, i.e., lies in $\mathcal{B}_{b,t_m}$. The third part is easy to see from the structural properties of graph G_v established in Lemma 6.7. Hence Statement 4 holds. Vertex u in Statement 5 must also have tenacity t_m, and hence its proof follows from Statement 4.

Induction step: Let t be an odd number with $t_m < t < l_m$, and assume that the theorem holds for tenacity $< t$.

Claim 1 **Statement 2** *holds.*

Proof : Let p be a maxlevel(v) = evenlevel(v) path and q be a minlevel(v) path; clearly $|p| + |q| = t$. By Theorem 5.3, each vertex u of tenacity $\geq t$ on p is BFS-honest w.r.t. p. Among these let us partition the vertices having minlevel $\geq \boldsymbol{\beta}$ into two sets: T_1 (T_2) consists of vertices u such that $|p[f \text{ to } u]| =$ minlevel(u) ($=$ maxlevel(u)). Clearly $b \in T_1$ and $v \in T_2$, hence both sets are non-empty. Let a be the vertex in T_1 having the largest minlevel and c be the vertex in T_2 having the smallest maxlevel. Now there are three cases.

Case 1: a and c are adjacent on p and (a, c) is a matched edge. Since $a \in T_1$ and $c \in T_2$, tenacity$(a) \geq t$ and tenacity$(c) \geq t$. The concatenation of p and q can be viewed as the concatenation of an even and an odd path from f to a, giving tenacity$(a) \leq t$. Combined with the previous statement we get tenacity$(a) = t$. By Lemma 5.2 we get tenacity(a) = tenacity(c) = tenacity$(a, c) = t$. Since p assigns minlevel to a and maxlevel to c, it must be the case that minlevel$(a) = \boldsymbol{\alpha}$ and maxlevel$(c) = \boldsymbol{\alpha} + 1$. Furthermore, since tenacity(a) = tenacity$(c) = t$, we get minlevel$(c) = \boldsymbol{\alpha}$ and maxlevel$(a) = \boldsymbol{\alpha} + 1$, i.e., a and c are both inner. Therefore (a, c) is not a prop, and hence it is a bridge of tenacity t.

Case 2: a and c are adjacent on p and (a, c) is an unmatched edge. By arguments analogous to the previous case, it is easy to see that tenacity(a) = tenacity(c) = tenacity$(a, c) = t$, and that a and c are both outer. Therefore (a, c) is not a prop, and hence it is a bridge of tenacity t.

Case 3: In the remaining case, a and c are not adjacent on p, i.e., there are vertices of tenacity $< t$ between a and c on p. Let (c, c') be the unmatched edge on p. There are two cases:

Case 3a: (c, c') is a prop. If so, c' lies in the blossom $\mathcal{B}_{c,t-2}$. Let (d, d') be the first edge on $p[c \text{ to } f]$ that "comes out of this blossom," i.e., $d \in \mathcal{B}_{c,t-2}$ and $d' \notin \mathcal{B}_{c,t-2}$. Clearly (d, d') is unmatched. Now there are two cases. If $d' = a$, then (d, a) must be a bridge. The reason is that a, which gets its minlevel from p must be outer, and all predecessors of d lie inside $\mathcal{B}_{c,t-2} \cup \{c\}$.

Now, the concatenation of p and q can be viewed as the concatenation of an evenlevel(d) path, an evenlevel(a) path and the edge (d, a) – the only aspect requiring justification is the first path, which follows from Statement 5 of the induction hypothesis applied to vertex $d \in \mathcal{B}_{c,t-2}$. Hence (a, d) is a bridge of tenacity t.

If $d' \neq a$, tenacity$(d') < t$ and so d' lies in a blossom of tenacity $t-2$, say $\mathcal{B}_{e,t-2}$, having base e. Now, by Statement 5 of the induction hypothesis applied to $d' \in \mathcal{B}_{e,t-2}$ we get that every shortest even path from d' to f must use e. Hence the e lies on $p[d'$ to $a]$. Furthermore, tenacity$(e) \geq t$. Therefore $e = a$. Finally, by analogous arguments to the previous case, the concatenation of p and q can be viewed as the concatenation of an evenlevel(d) path, an evenlevel(d') path and the edge (d, d'), which shows that (d, d') is a bridge of tenacity t.

Case 3b: (c, c') is a bridge. By similar arguments to the previous case we get that $c' \in \mathcal{B}_{a,t-2}$ and that (c, c') is a bridge of tenacity t.

Finally, we show that none of the remaining edges on p is a bridge of tenacity t. Consider an edge (d, e) on $p[f$ to $a]$, with d below e on p. If tenacity$(e) \geq t$ then e is BFS-honest on p and (d, e) is a prop. If $t(d) < t$ then d lies in a blossom of tenacity $t-2$ and hence by Statement 5 of the induction hypothesis, tenacity$(d, e) < t$. A similar argument holds for the edges on $p[c$ to $v]$. This completes the proof of Statement 2. □

Let f be an unmatched vertex such that $v \in V_t(f)$. The structure of the proof of the next Claim is along the lines of the proof of Statement 1' in Lemma 6.7. Therefore, in the proof given below, our emphasis is on providing only the new ideas needed.

Claim 2 *The following holds:*

Statement 1': *The set $B_f(v)$ is a singleton, and $B_f(v) = A_f(v)$.*

Proof : Once again it suffices to consider inner vertices only, see Claim 1. Let v be an inner vertex in $V_t(f)$. We will define a graph G_v whose structural properties will lead to a proof of Statement 1'. Let $A_f(v) = b$. Let $\boldsymbol{\beta} =$ minlevel(b) and $\boldsymbol{\alpha} = (t_m - 1)/2$; clearly $\boldsymbol{\alpha}$ is the maximum possible minlevel of a vertex of tenacity t. G_v is a layered graph, similar to graph H defined in Section 4, which was used for defining the procedure of DDFS.

The edges of G_v go from higher layers to lower layers and are not distinguished as matched or unmatched. Graph G_v has $\boldsymbol{\alpha} + 1$ layers, which are numbered from 0 to $\boldsymbol{\alpha}$, with not necessarily all layers having vertices. As in Lemma 6.7, we will make one exception to edges not running within the same layer: the two end points of certain bridges of tenacity t may both lie in layer $\boldsymbol{\alpha}$. Even so, we will add an edge connecting them.

The manner in which G_v is obtained from the original graph $G = (V, E)$ is specified below. Its vertices will be a suitably chosen subset of V. The layer number of each vertex $w \in G_v$ is minlevel(w); in particular, layer 0 will contain only f. Each edge also has a specified length; for edges not having layer $\boldsymbol{\alpha}$ as one of the end points, the length of the edge is the difference of the layer numbers of its end points.

We will show that G_v satisfies the DDFS Requirement (stated in Section 4), namely starting from any vertex, there is a path to the lowest layer. Additionally, we will prove the following correspondence between paths in G_v and alternating paths in $G = (V, E)$.

Correspondence of paths between G_v and G: Corresponding to each simple path in G_v from u in layer l to w in layer l', with $l > l'$, such that each edge of the path goes from a higher to a lower layer, there is a simple alternating path of the same length in $G = (V, E)$. Furthermore, if the DDFS Guarantee gives disjoint paths from a to u and c to w, for some bridge (a, c) of tenacity t, then there are vertex-disjoint simple alternating paths from a to u and c to w in $G = (V, E)$ of the same lengths.

Consider all evenlevel(v) and oddlevel(v) paths starting at f. By Theorem 5.3, every vertex of tenacity $\geq t$ on such a path is BFS-honest on it. Let S denote all such vertices. The vertex set of G_v is $S \cup S'$, where S' will be defined below. Its edge set is $E' \cup E''$, where E' is a specially chosen subset of E, and E'' are additional edges defined below. The length of each edge in E' is unit and for edges in E'', the length is specified below. For each pair of vertices $u, w \in S$, if $(u, v) \in E$ and minlevel(u) $\neq$ minlevel(v), then (u, v) is included in E'. In addition E' will contain all bridges of tenacity t, as pointed out below.

Next we define the edges of E''. Intuitively, these edges will replace "sub-paths that lie inside blossoms of tenacity $t - 2$." Let $w \in S$ and let p be a minlevel(w) path that starts at f; clearly p is part of an evenlevel(v) or oddlevel(v) path. Let a and c be vertices of tenacity $< t$ on p, with a lower than c. Let a' immediately precede a and c' immediately succeed c on p. We will say that $p[a \text{ to } c]$ is a *maximal contiguous stretch of vertices of tenacity* $< t$ *on* p if tenacity(a') $\geq t$, tenacity(c') $\geq t$ and all vertices on $p[a \text{ to } c]$ are of tenacity $< t$. By Lemma 6.11, c is BFS-honest w.r.t. p and by Lemma 5.2, $|p[f \text{ to } c]|$ must be even. Hence by Statement 4 of the induction hypothesis, $p[a' \text{ to } c] = \text{evenlevel}(a'; c)$ and it lies in $\mathcal{B}_{a',t-2} \cup \{a'\}$.

Now, in lieu of the path $p[a' \text{ to } c']$, the direct edge (a', c') is added to E'', and its length is defined to be $|p[a' \text{ to } c']|$. Observe that the length equals the difference in the layer numbers of c' and a'. Thus edge (a', c') of graph G_v represents the path evenlevel($a'; c$) $\circ$ (c, c') in the original graph $G = (V, E)$. This operation is performed on every relevant sub-path of every evenlevel(v) and oddlevel(v) path.

Finally we add vertices and edges to G_v corresponding to each bridge of tenacity t; the precise addition depends on the case in Claim 1 satisfied by this bridge. In Case 1 and 2, we only need to add the edge (a, c).

In the first case within Case 3a, we add vertex d to S' and assign it layer $\boldsymbol{\alpha}$. We also add edge (c, d) to E'' and (d, a) to E'. The length of (c, d) is $|p[c \text{ to } d]| = \text{evenlevel}(c; d)$ and it corresponds to an evenlevel($c; d$) path in $G = (V, E)$.

In the second case within Case 3a, we add d and d' to S', both at layer $\boldsymbol{\alpha}$ and we add (d, d') to E'. We also add edges (c, d) and $(d'a)$ to E''. These correspond to an evenlevel($c; d$) path and an evenlevel($a; d'$) path in $G = (V, E)$, respectively, and their lengths are defined to be the lengths of these paths.

In case 3b, we add c' to S', at layer $\boldsymbol{\alpha}$, together with edge (c, c') in E'. We also add edge (c', a) to E''; it corresponds to an evenlevel($a; c'$) path in $G = (V, E)$ and its length is defined to be the length of this path.

This completes the description of graph G_v. It is easy to verify that G_v satisfies the DDFS Requirement (stated in Section 4), namely starting from any vertex, there is a path to the lowest layer, i.e., vertex f. We now prove that the correspondence of paths between G_v and $G = (V, E)$ holds. For this, observe that if (a, c) and (d, e) are edges in E'' on four distinct vertices, then they correspond to two paths that lie in two distinct blossoms of tenacity $t - 2$. By Statement 3 of the induction hypothesis, i.e., laminarity of blossoms, these two blossoms are vertex disjoint, thereby implying that the required paths through them are also vertex disjoint.

We note that the remaining ideas needed to complete the proof of Statement 1' are identical to those used for proving Statement 1' in Lemma 6.7. □

Again, Claim 9 extends Statement 1' to Statement 1. At this point, the base of every vertex of tenacity t, and blossoms of tenacity t are unconditionally defined. Hence Statement 3 follows from Proposition 7.6.

The first part of Statement 4 follows from Proposition 6.10. The second part follows from the fact that all vertices of tenacity $< t$ on evenlevel(v) and oddlevel(v) paths lie in blossoms of tenacity $t-2$, which by the definition of blossoms will be nested inside $\mathcal{B}_{b,t}$. The additional vertices referred to in the third part are those of tenacity $< t$ in $\mathcal{B}_{b,t}$. Such a vertex u lies in a blossom $\mathcal{B}_{d,t-2}$, where d is either b or d is a vertex of tenacity t in $\mathcal{B}_{b,t}$. The first case, following by Statement 4 of the induction hypothesis, and in the second case, b is an iterated base of u and an evenlevel($b;d$) path concatenated with an appropriate path in $\mathcal{B}_{d,t-2}$, which is guaranteed by Statement 4 of the induction hypothesis, yields the required path. The structure of G_v readily implies the third part of Statement 4, hence proving this statement fully.

If tenacity(u) $= t$, Statement 5 is obvious. Next assume tenacity(u) $< t$. Let w be the first vertex on $p[u$ to $v]$ having tenacity t and let w' be the preceding vertex on this path; clearly tenacity(w') $< t$. Also, let a be the last vertex on $p[b$ to $u]$ of tenacity $\geq t$. By Lemma 6.11, w' is BFS-honest on p and by Statement 4 of the induction hypothesis, $p[a$ to $w']$ is an $e(a;w')$ path. Now, by Statement 5 of the induction hypothesis, base(u) lies on $p[a$ to $w']$, thereby completing the proof of Statement 5. This also completes the proof of the induction step.

To establish the last claim made in the theorem, let v be a vertex of tenacity l_m. Observe that the arguments made in Claim 1 for proving Statement 2 do not hinge on proving any of the subsequent statements for that value of tenacity. Hence the proof of this claim will work for vertices of tenacity l_m as well. □

As stated in Section 3.2, petals are intimately connected to blossoms. This relationship is formally established in Lemma 6.13. At the end of search level $i = (t-1)/2$, i.e., once MAX is done processing all bridges of tenacity t, all blossoms of tenacity t can be identified as follows. The proof of this lemma is straightforward and is omitted.

Lemma 6.13 *Let* tenacity(v) $= t$, *and at the end of search level* $i = (t-1)/2$, *assume that* bud*(v) *is* b. *Then* base(v) $= b$ *and the set* $S_{b,t}$ *defined in Definition 7.1, for blossom* $\mathcal{B}_{b,t}$ *is precisely* $\{u \mid$ tenacity(u) $= t$ *and* bud*(u) $= b\}$. *Furthermore, blossom* $\mathcal{B}_{b,t}$ *consists of the union of all petals whose* 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.*

Observe that if 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.

7 Blossoms form a laminar family

In this section, we will assume that **Claim(t)** holds for t odd and $t_m \leq t < l_m$, and we will show that the set of blossoms of tenacity at most t forms a laminar family. This will prove Statement 3 in the induction step in Theorem 6.12. Of course, once Theorem 6.12 is established, the laminarity of all blossoms will hold unconditionally.

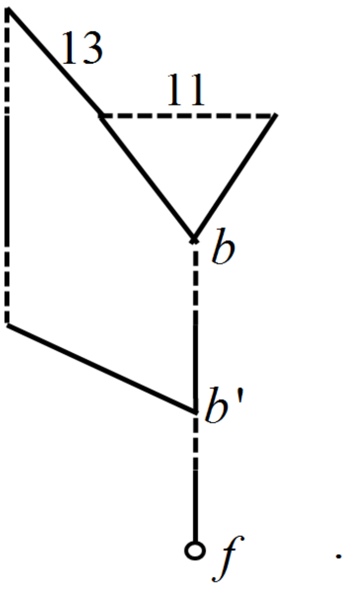

Figure 20: $\mathcal{B}_{b,11} \subset \mathcal{B}_{b',13}$

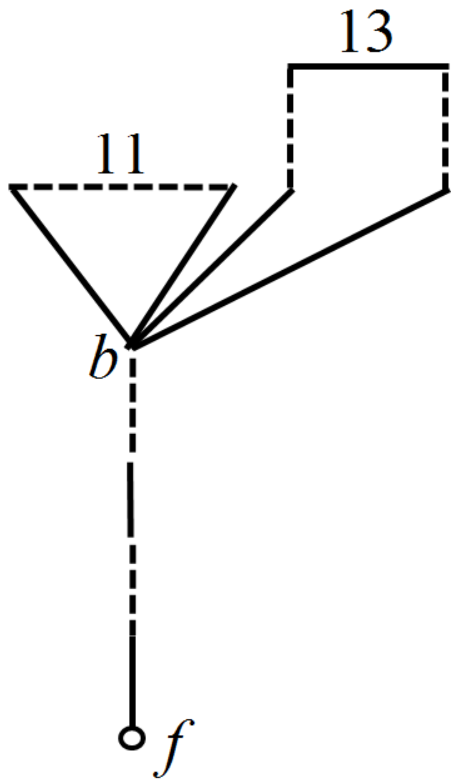

Figure 21: $\mathcal{B}_{b,11} \subset \mathcal{B}_{b,13}$

First let us present an attempt at constructing a counter-example. The subtlety of reason due to which the counter-example fails should indicate that the proof will be non-trivial. In Figure 20, $\mathcal{B}_{b,11} \subset \mathcal{B}_{b',13}$ and in Figure 21, $\mathcal{B}_{b,11} \subset \mathcal{B}_{b,13}$. In Figure 22, we have tried to "combine" the blossoms so that $\mathcal{B}_{b,11}$ is contained in both $\mathcal{B}_{b,13}$ and $\mathcal{B}_{b',13}$ thereby giving a counter-example to laminarity. However, observe that in the process of "combining" the blossoms, the tenacity of b reduces from ∞ to 13, so that $\mathcal{B}_{b,13}$ is not a valid blossom anymore.

Definition 7.1 (Nesting depth of blossoms) Since blossoms were defined recursively, so is their nesting depth. Let b be an outer vertex and t be an odd number such that tenacity$(b) > t$ and $t < l_m$. Define the nesting depth of blossom $\mathcal{B}_{b,1}$ to be $N(\mathcal{B}_{b,1}) = 0$. Define the nesting depth of blossom $\mathcal{B}_{b,t}$ to be

$$N(\mathcal{B}_{b,t}) = 1 + \left(\max_{v \in (S_t \cup \{b\}),\ v \text{ outer}} N(\mathcal{B}_{v,t-2}) \right)$$

if $S_{b,t} \neq \emptyset$ and $N(\mathcal{B}_{b,t-2})$ otherwise.

In the graph of Figures 2 and 3, the nesting depths of these blossoms $\mathcal{B}_{a,\alpha}$, $\mathcal{B}_{b,\kappa}$ and $\mathcal{B}_{f,\tau}$ are 1, 2 and 3, respectively.

Lemma 7.2 *Let $v \in \mathcal{B}_{b,t}$. Then $\exists k$ such that $1 \leq k \leq N(\mathcal{B}_{b,t})$ and $b = \text{base}^k(v)$. Furthermore, all the vertices* $\text{base}(v), \ldots, \text{base}^{k-1}(v)$ *belong to $\mathcal{B}_{b,t}$.*

Proof : By induction on the nesting depth of blossom $\mathcal{B}_{b,t}$. If $N(\mathcal{B}_{b,t}) = 1$, by definition,

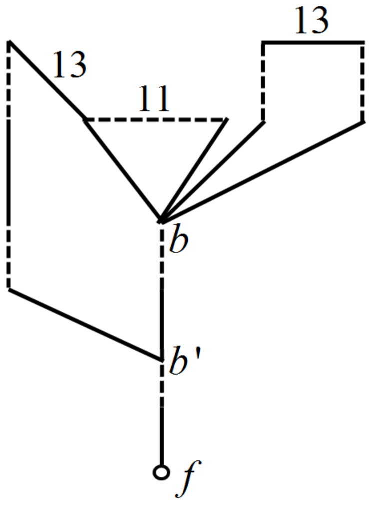

Figure 22: Observe that tenacity$(b) = 13$.

$b = \text{base}(v)$. To prove the induction step, suppose $N(\mathcal{B}_{b,t}) = l + 1$. Now, if $v \in S_{b,t}$, i.e., tenacity$(v) = t$, then base$(v) = b$ and we are done.

Otherwise, $\exists u \in S_{b,t} \cup \{b\}$ such that $v \in \mathcal{B}_{u,t-2}$. Clearly $N(\mathcal{B}_{u,t-2}) \leq l$ and either $u = b$ or base$(u) = b$. By the induction hypothesis, $\exists! k$ such that $l \geq k \geq 1$ and $u = \text{base}^k(v)$. If $u = b$, $b = \text{base}^k(v)$, and by the induction hypothesis, base$(v), \ldots, \text{base}^{k-1}(v)$ belong to $\mathcal{B}_{b,t-2}$ and hence also to $\mathcal{B}_{b,t}$.

If $u \neq b$, $b = \text{base}^{k+1}(v)$ and $k + 1 \leq l + 1$. Now, by the induction hypothesis, base(v), $\ldots, \text{base}^{k-1}(v)$ belong to $\mathcal{B}_{u,t-2}$. Hence base$(v), \ldots, \text{base}^k(v)$ belong to $\mathcal{B}_{b,t}$. □

Lemma 7.3 *Let $t \leq t' <$ tenacity(b) and $t' < l_m$, and let $\mathcal{B}_{b,t}$ and $\mathcal{B}_{b,t'}$ be two blossoms with the same base b. Then $\mathcal{B}_{b,t} \subseteq \mathcal{B}_{b,t'}$.*

Proof : The proof is by induction on $t' - t$. The base case, i.e., $t = t'$, is obvious. Assume the induction hypothesis that $\mathcal{B}_{b,t} \subseteq \mathcal{B}_{b,t'-2}$. Now, by Definition 7.1 it is straightforward that $\mathcal{B}_{b,t'-2} \subseteq \mathcal{B}_{b,t'}$. Hence $\mathcal{B}_{b,t} \subseteq \mathcal{B}_{b,t'}$. □

Lemma 7.4 *Let $\mathcal{B}_{b,t}$ be a blossom with base b and tenacity $t < l_m$, and v be a vertex satisfying* base$^k(v) = b$ *for some $k \geq 1$. If $t \geq$ tenacity(base$^{k-1}(v)$) then $v \in \mathcal{B}_{b,t}$.*

Proof : The proof is by induction on k. In the base case, i.e., $k = 1$, base$(v) = b$. Let tenacity$(v) = r$, clearly $r \leq t$. By Definition 7.1, $v \in \mathcal{B}_{b,r}$ and by Lemma 7.3, $\mathcal{B}_{b,r} \subseteq \mathcal{B}_{b,t}$. Hence $v \in \mathcal{B}_{b,t}$.

For the induction step, let base$^{k-1}(v) = u$, tenacity$(u) = r \leq t$. By the induction hypothesis, $v \in \mathcal{B}_{u,r-2}$. Since base$(u) = b$, by Definition 7.1, $\mathcal{B}_{u,r-2} \subseteq \mathcal{B}_{b,r}$ and by Lemma 7.3, $\mathcal{B}_{b,r} \subseteq \mathcal{B}_{b,t}$. Hence $v \in \mathcal{B}_{b,t}$. □

Lemma 7.5 *Let $\mathcal{B}_{b,t}$ and $\mathcal{B}_{b',t'}$ be two blossoms such that $b \in \mathcal{B}_{b',t'}$. Then $\mathcal{B}_{b,t} \subset \mathcal{B}_{b',t'}$.*

Proof : By Lemma 7.2, there is a $k \geq 1$ such that $b' = \text{base}^k(b)$ and $\text{base}^1(b), \ldots, \text{base}^{k-1}(b) \in \mathcal{B}_{b',t'}$. Clearly, $t' \geq \text{tenacity}(\text{base}^{k-1}(b))$. To prove the statement, we will apply induction on k.

For the base case, i.e., $k = 1$, let $\text{tenacity}(b) = r$. Clearly, $t < r \leq t'$ and $\mathcal{B}_{b,t} \subset \mathcal{B}_{b,r-2}$. By Definition 7.1, $\mathcal{B}_{b,r-2} \subset \mathcal{B}_{b',r}$, where the containment is proper since b is not in the first blossom but it is in the second one. By Lemma 7.3, $\mathcal{B}_{b',r} \subseteq \mathcal{B}_{b',t'}$. Hence $\mathcal{B}_{b,t} \subset \mathcal{B}_{b',t'}$.

For the induction step assume $\text{base}^{k+1}(b) = b'$. Let $\text{base}^k(b) = v$, and let $\text{tenacity}(v) = r$. Since $v \in \mathcal{B}_{b',t'}$, $r \leq t'$. Clearly, $\text{tenacity}(\text{base}^{k-1}(b)) \leq r - 2$. Therefore, by Lemma 7.4, $b \in \mathcal{B}_{v,r-2}$. Furthermore, since $\text{base}^k(b) = v$, by the induction hypothesis, $\mathcal{B}_{b,t} \subset \mathcal{B}_{v,r-2}$.

Since $\text{base}(v) = b'$, by Definition 7.1, $\mathcal{B}_{v,r-2} \subseteq \mathcal{B}_{b',r}$. Since $r \leq t'$, $\mathcal{B}_{b',r} \subseteq \mathcal{B}_{b',t'}$. Hence, $\mathcal{B}_{b,t} \subset \mathcal{B}_{b',t'}$. □

Proposition 7.6 *For t odd with $t_m \leq t < l_m$, assume* **Claim(t)**.
The set of blossoms of tenacity at most t forms a laminar family, i.e., two such blossoms are either disjoint or one is contained in the other.

Proof : Let $t' \leq t$ and $t'' \leq t$. Suppose v lies in blossoms $\mathcal{B}_{b,t'}$ and $\mathcal{B}_{b',t''}$. If $b = b'$, we are done by Lemma 7.3. Next assume that $b \neq b'$. Then by the first claim in Lemma 7.2, $b = \text{base}^k(v)$ and $b' = \text{base}^l(v)$, for some k and l. Since $b \neq b'$, $k \neq l$. Let us assume $k < l$. By the second claim in Lemma 7.2, $b = \text{base}^k(v) \in \mathcal{B}_{b',t''}$. Finally, by Lemma 7.5, $\mathcal{B}_{b,t'} \subset \mathcal{B}_{b',t''}$. Observe that none of the lemmas used assumed existence of blossoms of tenacity $> t$, hence the proposition follows. □

8 Proof of correctness and running time

We first need to prove that vertex will be assigned its correct minlevel and maxlevel. This is done by an induction on the search level in Theorem 8.2. The proof for minlevels is straightforward.

Lemma 8.1 *Let (u, v) be a bridge with* $\text{tenacity}(u, v) \leq l_m$. *Then the following hold.*

- *If (u, v) is matched then u and v are both inner vertices.*
- *If (u, v) is unmatched then if u is outer,* $\text{tenacity}(u) \leq \text{tenacity}(u, v)$, *and if u is inner,* $\text{tenacity}(u) < \text{tenacity}(u, v)$.

Proof : First assume that (u, v) is matched. Since (u, v) is not a prop, neither endpoint of this edge assigns a minlevel (of even parity) to the other. Therefore, the minlevel of both u and v must be odd, and hence they are inner vertices.

Next assume that (u, v) is unmatched. Now, there are three cases:

Case 1: u and v are both outer vertices.
We will first establish that $\text{evenlevel}(u) = \text{evenlevel}(v)$. Suppose $\text{evenlevel}(u) < \text{evenlevel}(v)$. Then $\text{evenlevel}(u) + 1 < \text{evenlevel}(v)$. Since an $\text{evenlevel}(u)$ path concatenated with edge (u, v)

gives an odd alternating path to v, we get that oddlevel$(v) \leq$ evenlevel$(u) + 1 <$ evenlevel(v), thereby contradicting the assumption that v is outer.

Hence evenlevel$(u) =$ evenlevel$(v) = i$, say. Clearly, oddlevel$(v) \geq i+1$. Since an evenlevel(u) path concatenated with edge (u, v) gives an odd alternating path to v of length $i+1$, oddlevel$(v) = i+1$. Similarly, oddlevel$(u) = i + 1$. Hence tenacity$(u) =$ tenacity$(v) =$ tenacity$(u, v) = 2i + 1$.

Case 2: u and v are both inner vertices.
Since minlevel(u) is odd and since (u, v) is not a prop, evenlevel$(v) + 1 >$ oddlevel(u). Therefore evenlevel$(u)+$evenlevel$(v)+1 >$ evenlevel$(u)+$oddlevel(u) and hence tenacity$(u) <$ tenacity(u, v). Similarly tenacity$(v) <$ tenacity(u, v).

Case 3: u is outer and v is inner.
Since minlevel(v) is odd and since (u, v) is not a prop, evenlevel$(u) + 1 >$ oddlevel(v). This implies that evenlevel$(u) +$ evenlevel$(v) + 1 >$ evenlevel$(v) +$ oddlevel(v) and hence tenacity$(v) <$ tenacity(u, v). Since an evenlevel(v) path concatenated with edge (u, v) gives an odd alternating path to u, we get that oddlevel$(v) \leq$ evenlevel$(v) + 1$, and hence tenacity$(u) \leq$ tenacity(u, v). □

Remark: : In the proof of Lemma 8.1, Case 3, if oddlevel$(v) =$ evenlevel$(v) + 1$ (and hence tenacity$(u) =$ tenacity(u, v)), the bridge (u, v) will have non-empty support; in particular, it contains u and its matched neighbor. However, if oddlevel$(v) <$ evenlevel$(v) + 1$, bridge (u, v) will have empty support.

Theorem 8.2 *For each vertex v such that* tenacity$(v) < l_m$*, Algorithm 1 assigns* minlevel(v) *and* maxlevel(v) *correctly.*

Proof : The case $l_m = 1$ is straightforward and involves finding a maximal matching in G. Henceforth we will assume that $l_m \geq 3$. 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.

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 $2i + 1$.

The base case, $i = 0$, is obvious: MIN will assign an oddlevel of 1 to each neighbor of each unmatched vertex. Clearly, no edge can have tenacity 1.

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 ∞ if and only if 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 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 minlevel$(v) = i + 1$ will be assigned its minlevel in this search level, and every prop that assigns a minlevel of i will be classified as a prop. Let minlevel$(v) = i+1$, let p be a minlevel(v) 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 is of this type. Now, u must be BFS-honest w.r.t. p: If not, then v must occur on a shorter path to u, contradicting minlevel$(v) < i+1$. Now, if $|p[f \text{ to } u]| = i =$ maxlevel(u) then tenacity$(u) < 2i+1$. Otherwise, $|p[f \text{ to } u]| = i =$ 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 ∞ or $i+1$[8]. 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 tenacity$(u, v) = 2i+1$. By Lemma 5.2, tenacity$(u) =$ tenacity$(v) =$ tenacity(u, v), and by Lemma 8.1, u and v are both inner. Therefore, oddlevel$(u) =$ 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 tenacity$(u, v) = 2i+1$. We will consider the three cases given in Lemma 8.1. In Case 1, at search level i, MIN will determine that (u, v) is a bridge of tenacity $2i+1$.

In Case 2, assume that tenacity$(u) \geq$ tenacity(v); of course tenacity$(u) <$ tenacity(u, v). At search level (tenacity$(v)-1)/2$, MAX will assign evenlevel(v), and at search level (tenacity$(u)-1)/2 < i$, while assigning evenlevel(u), MAX will be able to determine that (u, v) is a bridge of tenacity $2i+1$.

In Case 3, since u is outer, tenacity$(u) \leq$ tenacity$(u, v) = 2i+1$. Therefore evenlevel$(u) =$ minlevel$(u) \leq i$, and hence it will be assigned by MIN at search level $\leq i$. MIN will also determine that (u, v) is a bridge. Since v is inner, tenacity$(v) <$ tenacity$(u, v) = 2i+1$. Hence evenlevel$(v) =$ maxlevel(v) will be assigned by MAX at search level (tenacity$(v) - 1)/2 < i$. Of these two operations, the one that happens later will determine the tenacity of bridge (u, v) and will insert it in $Br(2i+1)$. Clearly, in either case, this will happen by the end of execution of procedure MIN at search level i.

Task 3: Theorem 6.12 Statement 2 shows that every vertex of tenacity $2i+1$ lies in the support of a bridge of tenacity $2i+1$, and by Task 2, all such bridges are in $Br(2i+1)$ at the start of MAX in search level i. These two facts together with the following gives a proof for the current task.

In a run of MAX, consider the point at which DDFS is called with bridge $(u, v) \in Br(2i+1)$. Let S be the set of vertices of tenacity $2i+1$ found by MAX so far. We next prove:

Claim 8.3 *The set of vertices found by DDFS at this point is* support$(u, v) - S$.

By the induction hypothesis, every vertex of tenacity $< 2i+1$ is already in a petal. Therefore, as DDFS follows down predecessor edges starting from u and v, if any such vertex is encountered, DDFS will skip to the bud* of this petal [9]. Every vertex in S is also in a petal, hence the same applies to it.

Let $w \in$ support$(u, v) - S$. Then there is a maxlevel(w) path containing (u, v), say p; assume p starts at unmatched vertex f. Assume v is higher than u on p. Then w is reachable from v by

[8]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$(u') = i$.

[9]The importance of this subtle point, which is related to the idea of "precise synchronization of events" is explained below with the help of Figures 23 and 24.

following predecessor edges and skipping currently formed petals on the way. The path $p[f \text{ to } u]$ gives DDFS a disjoint way of reaching "below" w. Hence DDFS will find w. $\square$

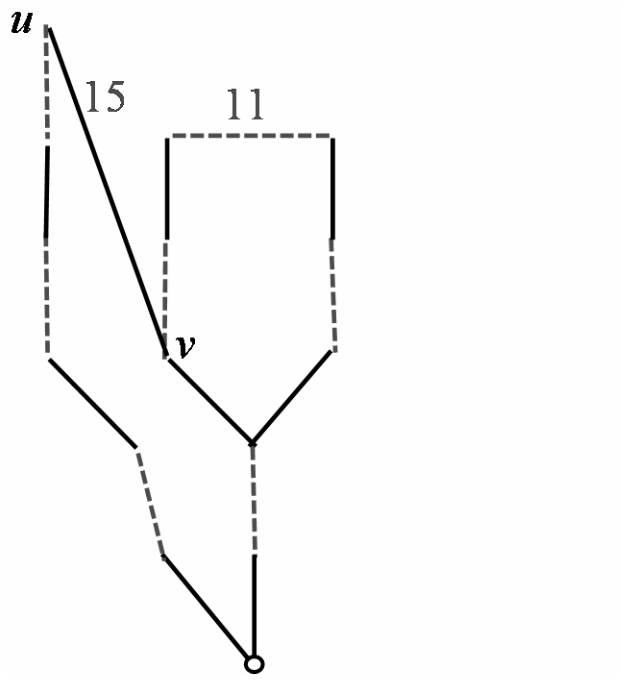

Figure 23: Can DDFS be performed on bridge (u, v) at search level 6?

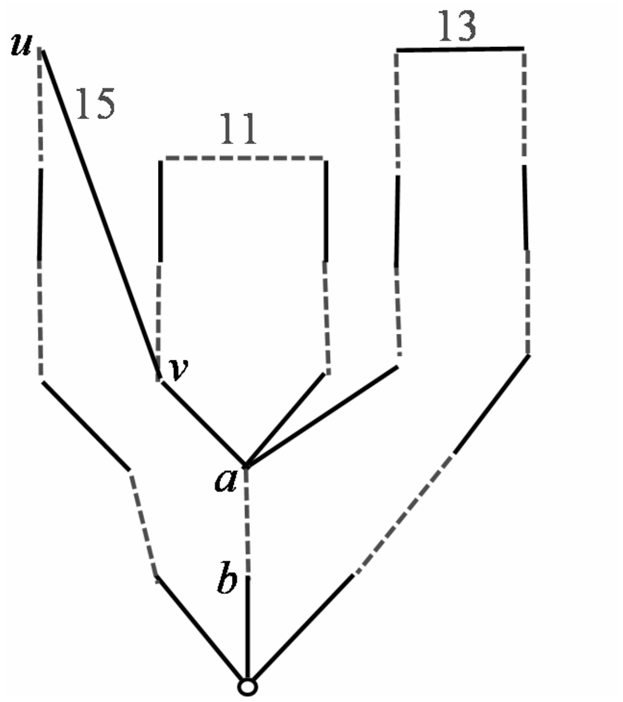

Figure 24: If so, vertices a and b will get wrong tenacities.

In Figure 23, the algorithm determines that (u, v) is a bridge of tenacity 15 at search level 6. However, according to Algorithm 1, DDFS has to 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?" To clarify this, let us change the algorithm so it runs DDFS on an edge as soon as its tenacity and its status as a bridge have been determined. Assume further that among the various bridges ready for processing, ties are broken arbitrarily[10].

Now consider the enhanced graph of Figure 24, in which vertices a and b are clearly in the support of the bridge of tenacity 13 and hence have tenacity 13. Assume that when MAX is run at search level 6 the bridge of tenacity 15 is processed first. Since the tenacities of vertices a and b are not set yet, DDFS will visit them and assign them a of tenacity 15, which would be incorrect. Observe that the correctness of MAX crucially depends on assigning tenacities to each edge of tenacity less than $2i + 1$ before processing bridges of tenacity $2i + 1$, i.e., the precise manner in which events are synchronized in Algorithm 1.

Lemma 8.4 *The procedures given in Section 3.3 will find a maximal set of disjoint minimum length augmenting paths in G.*

Proof : If at search level i, DDFS ends with two unmatched vertices, it must be the case that $l_m = 2i + 1$. Since in each blossom the alternating path found by the procedure given in Section

[10]By making the example given in Figure 24 slightly bigger, one can easily ensure that there are no such ties.

3.3.1 satisfies the properties established in Theorem 6.12, in particular in Statement 4, it finds a minimum length augmenting path, p, between the two unmatched vertices.

It should be easy to see that the vertices identified by the procedure of Section 3.3.2 cannot be part of a minimum length augmenting path that is disjoint from p and removing them is valid. However, in removing these vertices, the procedure may have left the graph in such a state that the next path cannot be found. This remark applies primarily to blossoms that have lost some of their vertices: does DDFS Guarantee still hold despite this loss?

The structural properties established in Theorem 6.12 render the answer to this question surprisingly simple. Assume that vertex $v \in p$ is in a blossom and let $\mathcal{B}$ be the maximal such blossom, with base b. Clearly b will be removed from the graph and the iterative procedure that removes all vertices having no predecessors will end up removing all of $\mathcal{B}$. $\square$

Theorem 8.5 *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 α is the inverse Ackerman function.*

Proof : Through arguments made so far, it should be clear that each of the procedures of MIN, MAX, finding augmenting paths, and removing vertices after each augmentation will examine each edge a constant number of times in each phase. The only operation that remains is that of computing bud* during DDFS. This can either be implemented on the pointer model by using Tarjan's set union algorithm [Tar75], which will take $O(m \cdot \alpha(m,n))$ time per phase, or on the RAM model by using Gabow and Tarjan's linear time algorithm for a special case of set union [GT85][11], which will take $O(m)$ time per phase. Since $O(\sqrt{n})$ phases suffice for finding a maximum matching [HK73, Kar73], the theorem follows. $\square$

A question arising from Theorem 8.5 is whether there is a linear time implementation of bud* in the pointer model. [MV80] had claimed that path compression suffices to achieve this. They had claimed, without proof, that because of the special structure of blossoms, a charging argument could be given that assigns a constant cost to each edge. This claim could not be verified at the time of writing [Vaz94], so it was left as an open problem in that paper. This problem has recently been resolved in the negative. [PV14] give an infinite family of graphs on which path compression in a phase takes time $\Omega(m\alpha(m,n))$.

Next, let us consider the question, "What is the best way of implementing bud* computations in practice?" To answer this, let us compare an implementation based on the set union datastructure [Tar75] and an implementation that uses only path compression. In the latter case, there is no need to build and maintain a separate datastructure on the side: each bud simply maintains and updates a pointer to the lowest bud it has reached. In the former case, not only is a separate datastructure needed, but the "trees" obtained in it will, in general, destroy the natural tree structure of nesting of petals. Additionally, in the latter case, one would expect the unions to be close to balanced anyway in practice. Considering the implementation effort and computational overhead of the former approach, we believe the latter one is superior. Using [TL84], the worst case running time for path compression in a phase in the second case is bounded by $O(m \log(n))$.

[11]Since [GT85] does not give a detailed explanation of how the idea is applicable to the MV algorithm, a new paper clarifying this has been recently written [Gab13]. Observe that [GT85] appeared after [MV80].

9 Equivalence of definitions

Below we establish equivalence of the two definitions of blossoms. Let us start by providing the definition of blossom as given in [Vaz94]; we will denote such a blossom of tenacity $t < l_m$ and base b by $\mathcal{B}_{b,t}{}^o$. Let v be a vertex with tenacity$(v) \leq t$. We will say that an outer vertex b is base$_{>t}(v)$ if for some positive k, base$^k(v) = b$, tenacity$(b) > t$, and tenacity(base$^{k-1}(v)) \leq t$. Then

$$\mathcal{B}_{b,t}{}^o = \{v \mid \text{tenacity}(v) \leq t \text{ and base}_{>t}(v) = b\}.$$

Theorem 9.1 *The two definitions of blossom are equivalent, i.e.,* $\mathcal{B}_{b,t} = \mathcal{B}_{b,t}{}^o$.

Proof : Let $v \in \mathcal{B}_{b,t}$. We will show that $v \in \mathcal{B}_{b,t}{}^o$ by considering the following three cases. The set $S_{b,t}$ is defined in Definition 7.1.

1. $v \in S_{b,t}$. In this case, tenacity$(v) = t$ and base$(v) = b$, and therefore base$_{>t}(v)) = b$. Hence $v \in \mathcal{B}_{b,t}{}^o$.
2. $v \in \mathcal{B}_{b,t-2}$. In this case, tenacity$(v) < t$ and for some $k \geq 1$, base$^k(v) = b$. Clearly, base$^{k-1}(v) \in \mathcal{B}_{b,t-2}$ and therefore base$_{>t}(v) = b$. Hence $v \in \mathcal{B}_{b,t}{}^o$.
3. $v \in \mathcal{B}_{u,t-2}$ and $u \in S_{b,t}$. In this case, tenacity$(v) < t$, tenacity$(u) = t$, base$(u) = b$, and for some $k \geq 1$, base$^k(v) = u$. Therefore, base$^{k+1}(v) = b$ and base$_{>t}(v)) = b$. Hence $v \in \mathcal{B}_{b,t}{}^o$.

Next, let $v \in \mathcal{B}_{b,t}{}^o$. Once again we will consider three cases to show that $v \in \mathcal{B}_{b,t}$.

1. tenacity$(v) = t$. In this case, base$(v) = b$ and therefore $v \in S_{b,t}$. Hence $v \in \mathcal{B}_{b,t}$.
2. tenacity$(v) < t$, for some $k \geq 1$, base$^k(v) = b$ and tenacity(base$^{k-1}(v)) < t$. In this case, $v \in \mathcal{B}_{b,t-2}$. Hence $v \in \mathcal{B}_{b,t}$.
3. tenacity$(v) < t$, for some $k \geq 1$, base$^k(v) = b$ and tenacity(base$^{k-1}(v)) = t$. Let base$^{k-1}(v) = u$. Then, tenacity$(u) = t$ and base$(u) = b$. Therefore, $u \in S_{b,t}$ and $v \in \mathcal{B}_{u,t-2}$. Hence $v \in \mathcal{B}_{b,t}$.

□

10 Epilogue

In summary, the main new task to be accomplished in non-bipartite graphs, beyond bipartite graphs, is finding maxlevels of vertices. Procedure MIN finds minlevels of vertices via an alternating BFS. This procedure and its proof of correctness are just as straightforward as finding minimum length alternating paths in bipartite graphs; in particular, the "agent" that is responsible for assigning a vertex its minlevel is one of its neighbors.

What is the "agent" that is responsible for assigning a vertex its maxlevel? The answer is far from straightforward and is established in Statement 2 of Theorem 6.12. In a sense, our motivation for arriving at the definitions of base of a vertex and blossom, and establishing structural facts about them, was precisely to prove this theorem; observe that the algorithm can be stated without these definitions. However, once these structural facts were found, it became clear that the MV algorithm was "walking" on precisely this structure — and it was also the key to a conceptual description of the algorithm. We hope this viewpoint will help with a better understanding of the paper.

11 Acknowledgments

My biggest thanks go to Silvio Micali for the start of a special friendship and an extraordinary collaboration that led to the discovery of this algorithm. The origins of the structure persented in this paper lie in the intuition we developed while working together during 1979-80.

For help with the current paper, I am particularly indebted to Bruno Loff and Ruta Mehta. Bruno made me aware of some subtle, though serious, issues in an earlier version of this paper. That led to fundamental changes and much cleaner proofs. As I worked out details of the new proofs Ruta, in her characteristic thorough style, went over details and identified several pitfalls.

Conversations with Laci Lovasz and Bob Tarjan made me realize that [Vaz94] was highly inadequate and a fresh attempt at the proof was called for. Thomas Hansen, Alexey Pechorin, Leen Torenvliet and Uri Zwick provided valuable comments on the paper. I wish to thank them all.

References

[Blu90] N. Blum. A new approach to maximum matchings in general graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 586–597, 1990.

[Edm65a] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards. Section B*, 69:125–130, 1965.

[Edm65b] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Ege31] J. Egerváry. On combinatorial properties of matrices. *Mat es Fizikai Lapok*, 38:16–28, 1931.

[EK75] S. Even and O. Kariv. An $o(n^{2.5})$ algorithm for maximum matching in general graphs. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 100–112, 1975.

[Gab13] H. N. Gabow. Set-merging for the MV matching algorithm. Unpublished manuscript, 2013.

[GK04] A. V. Goldberg and A. V. Karzanov. Maximum skew-symmetric flows and matchings. *Math. Program., Ser. A*, 100:537568, 2004.

[GS62] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[GT85] H. N. Gabow and R. E Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. System Sci.*, 30:209–221, 1985.

[GT91] H. N. Gabow and R. E Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38:815–853, 1991.

[HK73] J. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[JS89] M.R. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18:1149–1178, 1989.

[JVV86] M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[Kar73] A. V. Karzanov. An exact estimate of an algorithm for fnding a maximum flow, applied to the problem on representatives. *Problems in Cybernetics*, 5:66–70, 1973. Announced at the Seminar on Combinatorial Mathematics (Moscow, 1971).

[Kon31] D. Konig. Graphs and matrices. *Mat es Fizikai Lapok*, 38:116–119, 1931.

[Kuh55] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[Lof14] B. Loff. An implementation of the Micali Vazirani maximum cardinality matching algorithm. https://bitbucket.org/brunoloff/micali-vazirani-matching-algorithm/wiki/Home, 2014.

[LP86] L. Lovász and M.D. Plummer. *Matching Theory.* North-Holland, Amsterdam–New York, 1986.

[MS04] M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *IEEE Annual Symposium on Foundations of Computer Science*, 2004.

[MV80] S. Micali and V. V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *IEEE Annual Symposium on Foundations of Computer Science*, 1980.

[MVV87] K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[PV14] S. Pettie and V. V. Vazirani. In prepartion, 2014.

[Tar75] R. E Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[TL84] R. E Tarjan and J. Van Leeuwen. Worst case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, 1984.

[Val79] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Vaz94] V. V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.