CORRECTED VERSION OF RECORD; SEE LAST PAGE OF ARTICLE

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 49, No. 3, August 2024, pp. 2009–2047
ISSN 0364-765X (print), ISSN 1526-5471 (online)
https://pubsonline.informs.org/journal/moor

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

Vijay V. Vazirani[a]

[a] Department of Computer Science, University of California, Irvine, California 92697
Contact: vazirani@ics.uci.edu, https://orcid.org/0000-0002-4106-9077 (VVV)

Received: December 7, 2020
Revised: December 15, 2021; May 16, 2022; May 4, 2023; July 30, 2023; December 12, 2023; March 18, 2024; March 22, 2024
Accepted: March 29, 2024
Published Online in Articles in Advance: May 7, 2024, and updated November 15, 2024

MSC2020 Subject Classifications: Primary: 68Q25, 68W40

https://doi.org/10.1287/moor.2020.0388

Abstract. The Micali–Vazirani (MV) algorithm for finding a maximum cardinality matching in general graphs, which was published in 1980, remains to this day the most efficient known algorithm for the problem. The current paper gives the first complete and correct proof of this algorithm. The MV algorithm resorts to finding minimum-length augmenting paths. However, such paths fail to satisfy an elementary property, called *breadth first search honesty* in this paper. In the absence of this property, an exponential time algorithm appears to be called for—just for finding one such path. On the other hand, the MV algorithm accomplishes this and additional tasks in linear time. The saving grace is the various "footholds" offered by the underlying structure, which the algorithm uses in order to perform its key tasks efficiently. The theory expounded in this paper elucidates this rich structure and yields a proof of correctness of the algorithm. It may also be of independent interest as a set of well-knit graph-theoretic facts.

Funding: This work was supported in part by the National Science Foundation [Grant CCF-2230414].

Keywords: maximum matching problem • efficient algorithms • alternating paths • augmenting paths • blossoms • double depth first search

1. Introduction

The following quote, from the classic book by Lovász and Plummer [22], provides a nice backdrop for the work reported in this paper:

> The concept of an alternating path, although quite simple, is one of the most important in all of matching theory. (Lovász and Plummer [22, p. 12])

For the significance of this notion in the design of efficient algorithms for matching as well as the parallel development of the notion of an augmenting path for flow algorithms, we refer the reader to Ahuja et al. [1] and Lovász and Plummer [22]. The computational importance of *minimum-length* augmenting paths was first recognized by Dinitz [4] in the context of flow theory, and this basic idea gave rise to several efficient maximum flow algorithms; see Ahuja et al. [1]. Independent and simultaneous works by Hopcroft and Karp [15] and Karzanov [19] studied minimum-length augmenting paths in the context of matching and used this notion to give the most efficient algorithm of its time for maximum matching in bipartite graphs; see Section 10 for the improvements obtained in recent years.

Edmonds [7] defined the notion of blossoms and used it to give the first polynomial time algorithm for finding a maximum matching in general graphs. His proof of correctness was built around graph-theoretic facts, which formalize the manner in which augmenting paths traverse blossoms and their complex nested structure.

The most efficient known algorithm for general graph matching is from Micali and Vazirani [24] in 1980.[1] It resorts to finding minimum-length augmenting paths and uses the scheme proposed in Hopcroft and Karp [15] and Karzanov [19]; see Section 1.2. The blossoms that it finds are special, and they are defined in Section 8; in contrast, the blossoms of Edmonds [7] do not take into consideration length information and therefore, inadequate for the purpose of finding minimum-length augmenting paths. The description of this algorithm given in Micali and Vazirani [24], via a pseudocode, is complete and error free.[2] However, the paper did not attempt a proof of correctness.

A proof of correctness was attempted in Vazirani [34]. That paper correctly recognized the fact that an elaborate, new theory of alternating paths and blossoms, from the perspective of minimum-length paths, was called for to give such a proof. As detailed in Section 1.1, although that paper made some important contributions, it had serious shortcomings.

The current paper completes the task started in Vazirani [34] by presenting the pertinent theory in full detail and using it to give the first complete and correct proof of the Micali–Vazirani (MV) algorithm. In Section 1.1, we state the new ideas underlying this proof. Considering the special status of the maximum matching problem within the theory of algorithms, it was not appropriate to leave its most efficient known algorithm in an essentially unproven state, hence the investment of (substantial) effort to produce the current paper, notwithstanding the lapse of considerable time since the publication of the algorithm.

In the case of bipartite graphs, minimum-length alternating paths from an unmatched vertex to a matched vertex can be of one parity only, either even or odd. Consequently, such paths possess an elementary property, called *breadth first search* (BFS) *honesty* in this paper. Let p be a minimum alternating path from unmatched vertex f to v, and let u lie on p. Then, the part of p from f to u is a minimum alternating path from f to u and not any longer. In the presence of this property, a straightforward alternating BFS suffices for executing a phase[3] in linear time.

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

As a result, the following fundamental difficulty arises. For finding a minimum augmenting path in the graph, we need to find arbitrarily long paths to intermediate vertices, even though the latter does admit short paths; see Section 3. As such, this appears to call for an exponential time[4] algorithm. How then does the MV algorithm accomplish this task within the same time as bipartite graphs (i.e., linear time for a phase)?

The theory expounded in this paper shows how the underlying structure offers several different "footholds" for the key tasks that the algorithm needs to perform in order to home in on a solution quickly. The bottom line is that although minimum-length alternating paths are not BFS honest, they are not arbitrarily BFS dishonest. First, Theorem 2 uses the notion of tenacity to carve out an important case in which vertex u must be BFS honest on path p from f to v. Second, even if vertex u is not BFS honest on p, it is BFS honest on p w.r.t. a special vertex, called base(u). In turn, even if base(u) is not BFS honest on p, it is BFS honest on p w.r.t. base(base(u)) = base^2(u) and so on; see Theorems 5 and 6.

Consistent with these simple-looking rules, myriad situations can arise—illustrated via the many examples given in this paper—and the algorithm needs to efficiently handle all of them. Indeed, the structure expounded in this paper lays bare a stark contrast: on the one hand, the extreme complexity of the problem being handled by the algorithm and on the other hand, the simplicity of the algorithm itself.

Matching has had a long and distinguished history within graph theory and combinatorics, spanning more than a century and a half (Lovász and Plummer [22]). Its algorithmic history is equally long, dating back to the midnineteenth-century work of Carl Gustav Jacobi on the bipartite case, as mentioned in Wikipedia [36]. Its exalted status in the theory of algorithms[5] arises from the fact that its study has yielded quintessential paradigms and powerful algorithmic techniques, which form the foundation of the modern theory of algorithms, as we know it today. These include definitions of the classes $\mathcal{P}$ (Edmonds [8]) and $\#\mathcal{P}$ (Valiant [32]), the primal-dual paradigm (Kuhn [20]), the equivalence of random generation and approximate counting for self-reducible problems (Jerrum et al. [17]), characterizing facets of the convex hull of solutions to a combinatorial problem (Edmonds [7]), the canonical paths argument in the Markov chain Monte Carlo method (Jerrum and Sinclair [16]), and the isolation lemma (Mulmuley et al. [27], Wikipedia [37]).

1.1. Overview and Contributions

In this section, we will state the contributions of Vazirani [34] and point out the nature of its shortcomings, because of which the current paper was called for. We will also state the contributions of the current paper and discuss the ideas that led to the current proof.

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

At a high level, the algorithm involves *three main ideas*.

1. *Double depth first search* (DDFS). The new graph search procedure called DDFS is described in Section 2 in a completely self-contained manner; it can be read without reading the rest of the paper.

2. *Marking petals appropriately*. Petals need to be marked appropriately so that paths through them can be found quickly. Petals are defined in Definition 13, and Section 5.2 provides details on marking them. Remark 1 highlights how this way of marking petals helps achieve the promised running time.

3. *Synchronization of events*. This is described in Section 9.1; in particular, see Remark 8.

Section 8 gives the central notions of base of a vertex and base of a blossom; these are essential to the proof of correctness of the algorithm and for clarifying the "footholds" mentioned in Section 1. As described in Section 8.1, in order to define the base of a vertex v, we need to show that the set $B(v)$, defined in Definition 19, is a singleton. However, its proof requires the notion of a blossom and its associated properties. On the other hand, blossoms can be defined only after defining the base of a vertex. Therefore, we are faced with a severe "chicken-and-egg" problem.

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

Once the base of the lowest-tenacity vertices is well defined, the blossoms containing these vertices can be defined, and properties of these blossoms and properties of paths traversing through them can be established. In the induction step, which is carried out in Section 8.3, these facts are established for higher-tenacity vertices.

As mentioned, the heart of the proof involves using DDFS in an appropriate setting; the latter is a graph obtained from the given graph. The graph in which the induction basis is carried out is far simpler than the one in which the induction step is carried out because the latter contains blossoms defined in the previous steps of the induction. Because of this simplicity, the induction basis plays the role of a crucible in which proof techniques can be developed with relative ease for the various claims. Furthermore, as detailed here, we can determine the "correct" order in which these proofs need to be carried out. In the proof of the induction step, we first apply a transformation on previously defined blossoms (see Definition 28), after which the structure of the proofs becomes similar to those in the induction basis, and these proofs are omitted unless they contain a significant new insight.

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

1. For ease of comprehension, DDFS has been described in the simpler setting of a directed, layered graph H. In the algorithm, DDFS is run on the original graph G. However, describing DDFS on G is too cumbersome; this was done in Vazirani [34]. Instead, we provide a mapping from G to H in Section 5.3 via which the reader can easily trace the steps that DDFS executes in G. Interestingly, this cosmetic-looking idea had a far-reaching conceptual consequence as discussed.

2. To further help the reader, in the many illustrative examples given, the distance of vertices from unmatched vertices is proportional to their min levels, such as would be the case in the corresponding layered graph H. Thus, the unmatched vertices belong to the lowest level.

3. Wherever possible, procedures are described in plain English. For example, DDFS is described in English in Section 2; readers who prefer to understand it via a pseudocode can find it in Micali and Vazirani [24] and Vazirani [34].

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

At a high level, Vazirani [34] also accurately identified the interplay between the notions of tenacity, base, blossom and bridge. However, the actual definitions given for some of these notions were incorrect. For example, in Vazirani [34], the central notion of base of a vertex was defined for any vertex of finite tenacity. However, it turns out that there may be vertices of finite tenacity that have no base (e.g., see Example 14). In the current paper, base has been defined only for vertices of *eligible tenacity*; see Definition 17 for this notion. Vazirani [34, theorem 3] "proves" that every vertex of finite tenacity has a well-defined base; it is obviously incorrect. Furthermore, Vazirani [34, theorem 3] is incorrect even for vertices of eligible tenacity.

The errors in the proofs given in Vazirani [34] can be traced back to three reasons; we provide them together with the ways we get around them in the current paper.

1. Vazirani [34] failed to identify the "chicken-and-egg" problem stated and tried to "prove" several facts in a stand-alone manner. As mentioned, the current paper rectifies this problem by carrying out an induction on tenacity in Theorem 3, proving all these fact for lower-tenacity vertices before moving to higher-tenacity vertices.

2. The expository idea of describing DDFS in the simpler setting of a layered graph had an unexpected consequence; it led to two breakthrough ideas for the proof: first, use of the power of DDFS and second, doing the proof in the simpler setting of graphs H'_m and H'_t. As a result of the latter, we avoid dealing with the debilitating

complexity of the original graph $G = (V, E)$, unlike Vazirani [34].

The induction basis is carried out in graph H'_m. Then, the nontrivial definitions of pred_t and pred^*_t, given in Definition 28, help finesse the lower-tenacity blossoms, which were defined in previous induction steps, to yield the graph H'_t in which the induction step is carried out. All the facts established are then "ported" back to G using the mapping between these graphs and G; see Sections 8.2.1 and 8.3.1.

These ideas qualitatively changed[6] the nature of the proofs; instead of dealing with individual alternating paths that can get arbitrarily complicated, leading to incorrect proofs, we now deal with the very precise and potent information provided by the DDFS Certificate, as proven in Theorem 1.

3. The mindset in Vazirani [34] was the following. The culminating fact that needed to be proven was the existence of a bridge of the right tenacity on a max-level path,[7] and a variety of other structural facts needed to be established prior to that. This order turns out to be incorrect. As mentioned, in the current paper, the order is dictated by the proof of the induction basis carried out in graph H'_m. Its simple structure makes transparent the "right" order of implications; see Section 8.2. This is the right order for the induction step as well; see Section 8.3. In particular, the existence of a bridge is the first, and not the last, fact we establish in the induction basis as well as in the induction step.

1.2. Running Time and Related Papers

The simplest scheme for finding a maximum matching is to start with the empty matching, iteratively find augmenting paths, and augment the current matching. When there is no such path, the matching must be maximum; this is shown in Berge [2]. In order to improve the running time, Hopcroft and Karp [15] and Karzanov [19] proposed finding multiple augmenting paths in each iteration as stated in Definition 1.

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

As shown in Hopcroft and Karp [15] and Karzanov [19], $O(\sqrt{n})$ phases suffice[8] for finding a maximum matching. These papers also show how to implement a phase in $O(m)$ time in a bipartite graph, thereby getting a total running time of $O(m\sqrt{n})$. The MV algorithm executes a phase in linear time on the Random Access Model (RAM) model. Its total running time is $O(m\sqrt{n})$ on the RAM model and $O(m\sqrt{n} \cdot \alpha(m, n))$ on the pointer model (see Theorem 8 for details).

We note that small theoretical improvements to the running time, for the case of very dense graphs, have been given in recent years: $O(m\sqrt{n}\log(n^2/m)/\log n)$ (Goldberg and Karzanov [14]) and $O(n^w)$ (Mucha and Sankowski [26]), 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 improves on MV for $m = \omega(n^{1.85})$. However, the latter algorithm involves a large multiplicative constant (additionally, the algorithm requires randomization) in its running time, which comes from the use of fast matrix multiplication as a subroutine, thereby making the small improvement in the exponent not very meaningful.

Prior to Micali and Vazirani [24], Even and Kariv [9] had used the idea of finding augmenting paths in phases to obtain an $O(n^{2.5})$ maximum matching algorithm. However, their algorithm is extremely complicated, and its correctness is hard to ascertain, in particular because there is no journal version of that result.

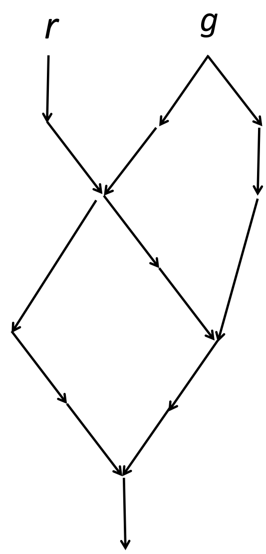
Subsequent to Micali and Vazirani [24], Gabow and Tarjan [11] give an efficient scaling algorithm for finding a minimum weight matching in a general graph with integral edge weights, and at the end of their paper, the authors claim that the unit weight version of their algorithm achieves the same running time as the MV algorithm; see also Gabow [10]. The rest of the history of matching algorithms is very well documented, and it will not be repeated here (e.g., see Lovász and Plummer [22], Vazirani [34]).

2. DDFS

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

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

Figure 1. Layered graph H with r and g in layer l_7.



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

We require that H satisfies the DDFS Requirement.

2.1. DDFS Requirement

Starting from every vertex $v \in V$, there is a path to a vertex in layer l_0.

Vertex v will be called a *bottleneck* if every path from r to l_0 and every path from g to l_0 contains v; v is allowed to be r or g or to be a vertex in layer l_0. Let p be a path from r or g to layer l_0. Because layer numbers on p are monotonically decreasing, if there is a bottleneck, the one having highest level must be unique. It will be called the *highest bottleneck*, and we will denote it by b. If H has a bottleneck, we will say that we are in Case 1. Otherwise, there must be distinct vertices r_0 and g_0 in layer l_0 such that there are vertex-disjoint paths from r to r_0 and g to g_0; this will be called Case 2.

As stated, in the MV algorithm, these two cases correspond to the creation of a new petal and the discovery of a new augmenting path, respectively. In the graph of Figure 1, DDFS will terminate in Case 1, with bottleneck b, as shown in Figure 2. In the graph of Figure 3, which differs from the graph of Figure 1 only in the two edges going from c to g_0, DDFS terminates in Case 2, with disjoint paths from r to r_0 and g to g_0.

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

2.2. The Objective of DDFS

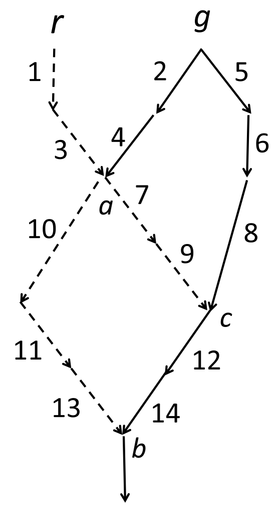
The first objective of DDFS is to determine which of these two cases holds. Additional objectives of DDFS in the two cases are as follows.

Case 1. DDFS needs to find the highest bottleneck, b, and partition the vertices in $V_b - \{b\}$ into two sets S_R and S_G, called the *red set* and the *green set* respectively, with $r \in S_R$ and $g \in S_G$. These sets should satisfy the following.

1. There is a path from r to b in $S_R \cup \{b\}$ and a path from g to b in $S_G \cup \{b\}$.
2. There are two spanning trees, T_r and T_g, in $S_R \cup \{b\}$ and $S_G \cup \{b\}$ and rooted at r and g, respectively. Furthermore, DDFS needs to find such a pair of trees.

Case 2. DDFS needs to find distinct vertices r_0 and g_0 in layer l_0 and vertex disjoint paths from r to r_0 and g to g_0. In this case, as soon as DDFS finds these two paths, it halts, even if it has not traversed all edges of E_p

Figure 2. DDFS executed from r and g terminates in Case 1 with bottleneck b. Edge numbers indicate the order of traversal of the edges.



because a new augmenting path has already been found. Let $E_p' \subseteq E_p$ denote the edges that DDFS has actually traversed.

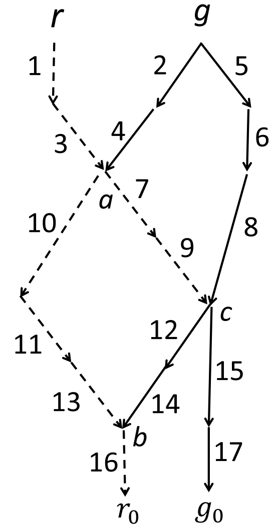
2.3. The Two DFS Trees

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

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

At termination, DDFS provides the following certificate.

Figure 3. DDFS executed from r and g terminates in Case 2.



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

2.4. Running Time

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

2.5. Coordinated Growth of the Two Trees

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

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

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

2.6. When the Two Centers of Activity Meet

As stated, when one of the centers of activity traverses an edge, the edge is assigned to the corresponding tree and is assigned its color.

However, the assigning of color to a vertex is not so straightforward and is not done in a greedy manner. Indeed, a vertex v may first be added to one tree, and later, this decision may be reverted; this happens if C_r and C_g meet at v.[9] We will adopt the convention that when this happens, first C_g backtracks and tries to find an alternative path that is as deep as v. If it fails, then C_g occupies v, and C_r tries to find an alternative path that is as deep as v. If C_r also fails, then it must have backtracked all the way to the root r, and DDFS terminates.

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

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

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

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

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

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

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

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

3. Elementary Definitions and a Fundamental Notion

In Section 3.1, we will present some elementary definitions pertaining to minimum-length augmenting paths. Using these definitions, in Section 3.2, we present the property of breadth first search honesty, because of which an alternating BFS works in bipartite graphs, yielding a linear time algorithm for a phase. The lack of this property in nonbipartite graphs necessitates a much more complex algorithm.

3.1. Elementary Definitions

A matching M in an undirected graph $G = (V, E)$ is a set of edges, no two of which meet at a vertex. Our problem is to find a matching of maximum cardinality in the given graph. Henceforth, all definitions will be w.r.t. a fixed matching M in G. Edges in M will be said to be *matched*, and those in $E - M$ will be said to be *unmatched*. Vertex v will be said to be matched if there is a matched edge incident at it and unmatched otherwise.

An *alternating path* is a simple path whose edges alternate between M and $E - M$ (i.e., matched and unmatched). An alternating path that starts and ends at unmatched vertices is called an *augmenting path*. Clearly, the number of unmatched edges on such a path exceeds the number of matched edges on it by one.[10] 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 Hopcroft and Karp [15] and Karzanov [19]. 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. Hopcroft and Karp [15] and Karzanov [19] show that only $O(\sqrt{n})$ such phases suffice for finding a maximum matching in general graphs. The remaining task is designing an efficient algorithm for a phase.

Definition 3 (Length of Minimum-Length Augmenting Path). Throughout, l_m will denote the length of a minimum-length augmenting path in G; if G has no augmenting paths, we will assume that $l_m = \infty$.

Definition 4 (Even Level and Odd Level of Vertices). The even level (odd level) of a vertex v, denoted by even level(v) (odd level(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 even level(v) (odd level(v)) path. If there is no such path, even level(v) (odd level(v)) is defined to be ∞.

We will typically denote an unmatched vertex by f. The even level is zero, and its odd level is the length of the shortest augmenting path starting at f; if no augmenting path starts at f, odd level(f) $= \infty$. The length of a minimum-length augmenting path w.r.t. M is the smallest odd level of an unmatched vertex.

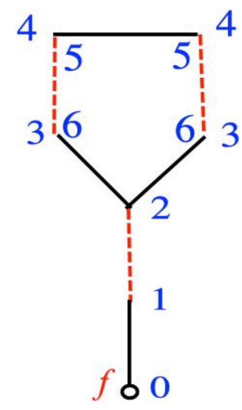
Definition 5 (Max Level and Min Level of Vertices). For a vertex v such that at least one of even level(v) and odd level(v) is finite, max level(v) (min level(v)) is defined to be the bigger (smaller) of the two.

Definition 6 (Outer and Inner Vertices). A vertex v with finite min level is said to be *outer* if even level(v) $<$ odd level(v) and *inner* otherwise.

Definition 7 (Odd and Even w.r.t. p). Let p be an alternating path from unmatched vertex f to v, and let u lie on p. The *length* of path p, denoted by $|p|$, is the number of edges on p. The part of p from f to u will be denoted by $p[f \text{ to } u]$, and $p[f \text{ to } u)$ will denote the part of p from f to the vertex just before u. Other combinations are self-explanatory. We will say that vertex u is *even w.r.t. p* if $|p[f \text{ to } u]|$ is even, and it is *odd w.r.t. p* if $|p[f \text{ to } u]|$ is odd.

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

Figure 4. (Color online) The even levels and odd levels of vertices are indicated; missing levels are ∞.



3.2. The Notion of BFS Honesty

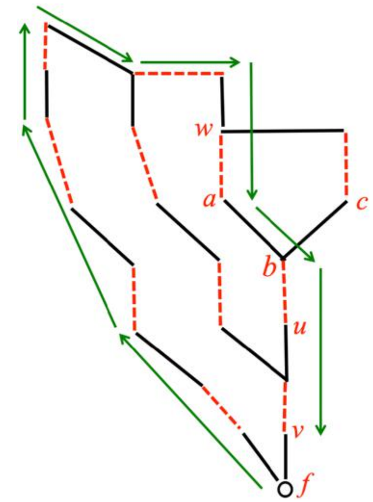
Let p be an alternating path from unmatched vertex f to v. We will say that p is a *minimum alternating path* if $|p| =$ even level(v) ($|p| =$ odd level(v)) if $|p|$ is even (odd).

BFS honesty is the following property. Let p be a minimum alternating path from unmatched vertex f to v, and let u lie on p. Then, $p[f$ to $u]$ is a minimum alternating path from f to u. Bipartite graphs satisfy this property, and as a consequence, a straightforward alternating BFS suffices for finding minimum augmenting paths; see Section 4, or for a complete description, see Vazirani [34, section 2.1].

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

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

Figure 5. (Color online) Vertices w, a, b, and u are not BFS honest on the even-level(v) path shown via arrows.



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

4. Some Essential Definitions

As mentioned in Section 1, in order to implement a phase in linear time in nonbipartite graphs, we need to exploit the elaborate structure offered by minimum-length alternating paths. In this section, we present some facts that are absolutely necessary to describe the MV algorithm. The proof of correctness of the algorithm requires additional structural properties, presented in Section 8.

Definition 8 (Tenacity of Vertices and Edges). Define the tenacity of vertex v tenacity(v) = even level(v) + odd level(v). If (u, v) is an unmatched edge, then tenacity(u,v) = even level(u) + even level(v) + 1, and if it is matched, tenacity(u,v) = odd level(u) + odd level(v) + 1.

The notion of tenacity is central to the structural facts that follow. Clearly, tenacity(f) $\geq l_m$ for an unmatched vertex f; see Definition 3 for the notion of l_m. Furthermore, tenacity(f) $= l_m$ if and only if f participates in a minimum-length augmenting path.

Definition 9 (Minimum Tenacity of a Vertex in G). Throughout, t_m will denote the tenacity of a minimum-tenacity vertex in G.

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

Example 3. In Figure 6, the tenacities of vertices are marked. They are $\alpha = 13$, $\beta = 15$, $\gamma = 17$, and $\delta = \infty$. In Figure 7, the tenacities of edges are marked. They are $\alpha = 13$, $\beta = 15$, and $\gamma = 17$.

Definition 10 (Predecessor, Prop, and Bridge). Consider a min level(v) path, and let (u, v) be the last edge on it; clearly, (u, v) is matched if v is outer and unmatched otherwise. In either case, we will say that u is a predecessor of v and that edge (u, v) is a prop. An edge that is not a prop will be defined to be a bridge.

Definition 11 (The Relations pred and pred*). Let v be a vertex such that min level(v) is finite. If v is an outer vertex, it will have a unique predecessor, namely its matched neighbor; otherwise, it will have one or more predecessors. The relation pred is defined as follows. We will say that u *is* pred v if u is a predecessor of v; we will also write is as u = pred v. The relation pred* is the reflexive, transitive closure of the relation pred. If u is pred$^*v$, we will also write is as u = pred$^*v$.

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

Figure 6. (Color online) The tenacity of vertices is indicated; here, $\alpha = 13$, $\beta = 15$, $\gamma = 17$, and $\delta = \infty$.

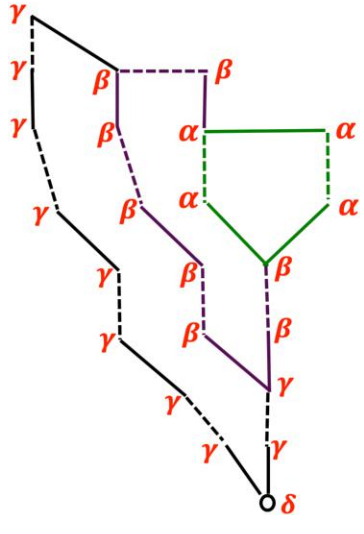
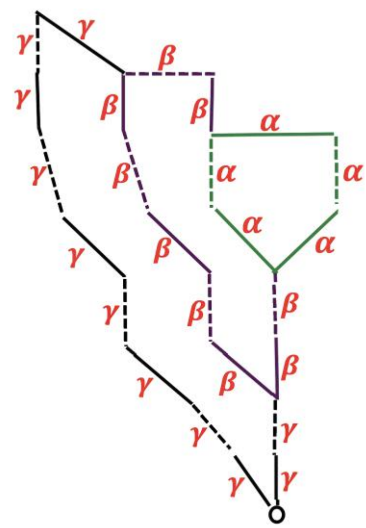


Figure 7. (Color online) The tenacity of each edge is indicated; here, $\alpha = 13$, $\beta = 15$, and $\gamma = 17$.



Definition 12 (The Support of a Bridge). Let (u, v) be a bridge of tenacity $t \leq l_m$. Then, its support is defined to be

$$\text{support}(u, v) = \{w \mid \text{tenacity}(w) = t \text{ and } \exists \text{ a max level}(w) \text{ path containing } (u, v)\}.$$

Example 5. In the graph of Figure 6, the supports of the bridges of tenacity α, β, and γ are the sets of vertices of tenacity α, β, and γ, respectively. In the graph of Figure 8, the tenacity of bridge[11] (u, v) is 13, and its support consists of two vertices: v and its matched neighbor. In Figure 9, the supports of the bridges (w, w') and (u, v) are all vertices of tenacities 11 and 13, respectively. Observe that in Figure 9, f is not in the support of any bridge, and tenacity$(f) = \infty$.

5. A Description of the MV Algorithm

The MV algorithm executes phases as defined in Section 1.2. Each phase starts with the matching, say M, computed in the last phase. A basic task accomplished in a phase is finding the min levels and max levels of all vertices of tenacity $\leq l_m$. For this purpose, the algorithm calls the procedures MIN and MAX iteratively as described.

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

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

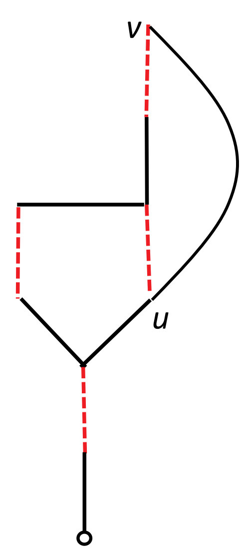
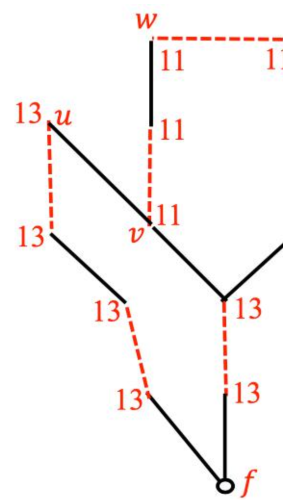


Figure 9. (Color online) Edges (w, w') and (u, v) are bridges, with the latter being an anomaly bridge.



5.1. Procedures MIN and MAX

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

If i is even (odd), MIN searches from all vertices, u, having an even level (odd level) of i along incident unmatched (matched) edges, say (u, v). If edge (u, v) has not been scanned before, MIN will determine if it is a prop or a bridge as follows. If v has already been assigned a min level of at most i, then (u, v) is a bridge. Otherwise, v is assigned a min level 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, which is its matched neighbor, and if i is even, v will have one or more predecessors.

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

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 min level$(v) \geq i+1$ **then**
      min level$(v) \leftarrow i+1$
      Insert u in the list of predecessors of v.
      Declare edge (u, v) a prop.
    Else declare (u, v) a bridge, and if tenacity(u, v) is known, insert (u, v) in Br(tenacity(u, v)).
  End
End
2. **MAX:**
For each edge in $Br(2i+1)$:
  Find its support using DDFS.
  For each vertex v in the support:
    max level$(v) \leftarrow 2i+1-$ min level(v)
    If v is an inner vertex, **then**
      For each edge e incident at v that is not a prop, if its tenacity is known, insert e in Br(tenacity(e)).
    End
  End
End

After MIN is done, procedure MAX calls DDFS with each bridge of tenacity $2i+1$ and finds the support of this bridge. In the process, DDFS finds all vertices, v, having tenacity$(v) = 2i+1$. Because their min levels are at most i, they are already known, and hence, max level$(v) = 2i+1-$ min level(v) can be easily computed. Clearly, if min level(v) is even level(v), then max level(v) will be odd level(v), and if min level(v) is odd level(v), then max level(v) will be even level(v).

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

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

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

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

5.2. Petals and the Way They Are Marked

The notion of a petal is central to the MV algorithm; petals are found by DDFS.

Definition 13 (Petal and Bud). Assume that DDFS is called with a bridge (u, v) of eligible tenacity t and that it terminates in Case 1. Then, the bottleneck b found is called a *bud*. The set of vertices of tenacity t encountered during the current DDFS, which must lie in the support of (u, v), forms a new *petal*. Formally, the petal consists of all vertices in the support of (u, v) minus the supports of all bridges (of tenacity t) processed thus far in the current search level.

Note that b must be an outer vertex and that it is not included in the current petal. Observe that each vertex of eligible tenacity is included in exactly one petal and that the formation of each petal is triggered by a distinct bridge.[12]

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

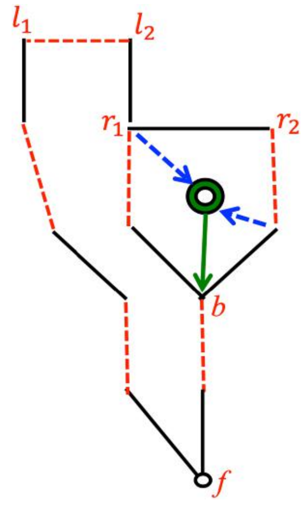
5.2.1. Marking a Petal.

When a new petal is found, the algorithm executes the following steps. It creates a new node, called *petal node*; this has the shape of a doughnut in Figure 10. All vertices of the new petal point[13] to the petal node; b is not in the petal and does not point to the petal node. The new petal node points to the two endpoints of its bridge, r_1 and r_2, and to its bud, b. These pointers will enable the algorithm to

1. skip over this petal in future DDFSs and
2. efficiently find an alternating path through the petal; see Remark 1.

Definition 14 (The Functions bud(v) and bud$^*(v)$). Define a function bud : $V \to V$ as follows. If vertex v is in a petal, then bud$(v) = b$, where the bud of this petal is b, and if v is not in a petal, then bud$(v) = v$. At any point in the execution of the algorithm, the function bud$^*(v)$ is defined recursively as follows. If bud$(v) = v$, then bud$^*(v) = v$; else, bud$^*(v) =$ bud*(bud(v)). Clearly, bud$^*(v)$ will keep changing as the algorithm proceeds.

Figure 10. (Color online) A new petal node is created after DDFS is performed on bridge (r_1, r_2).



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

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

5.3. The Mapping from Graph *G* to *H*

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

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

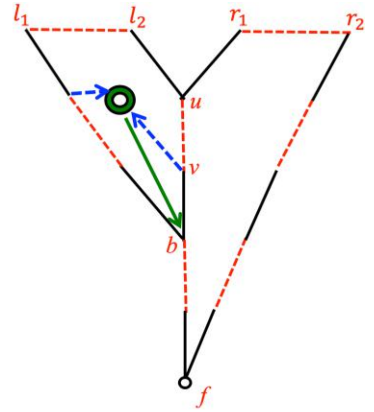
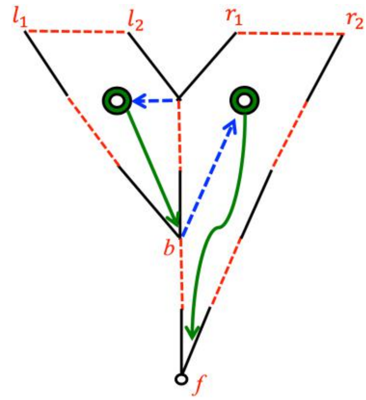


Figure 12. (Color online) The petal and blossom formed when DDFS is performed on bridge (r_1, r_2); their bud and base is f.



Assume that DDFS is called with bridge (r, g). Then, H must have the two vertices $\text{bud}^*(r)$ and $\text{bud}^*(g)$.[14] The rest of H is recursively defined as follows. If min level$(v) > 0$, then corresponding to each predecessor u of v in G, H has the vertex $\text{bud}^*(u)$ and the directed edge $(v, \text{bud}^*(u))$. If min level$(v) = 0$, then $l(v_H) = 0$, and v_H has no outgoing edges. It is easy to confirm that H satisfies the DDFS Requirement. For details, see Sections 8.2.1 and 8.3.1.

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

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

5.4. Finding Augmenting Paths

The MV algorithm will find augmenting paths during search level j_m, where $l_m = 2j_m + 1$ and l_m is the length of a minimum-length augmenting path in the current phase. DDFS on some bridges of tenacity l_m will end in Case 2 (i.e., no bottleneck is found). We note that not every bridge of tenacity l_m leads to an augmentation; DDFS on a bridge of tenacity l_m can end in Case 1 as well (i.e., a bottleneck is found).

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

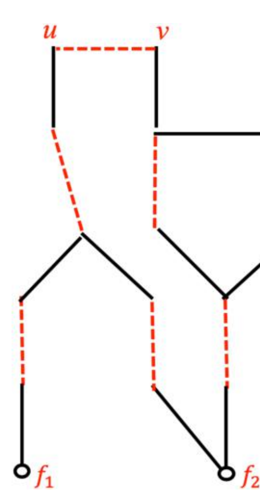
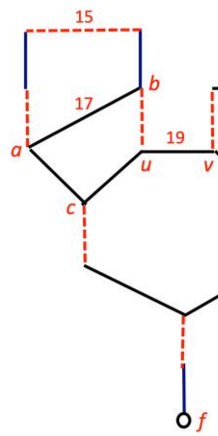


Figure 14. (Color online) The tenacities of all four bridges is indicated. Bridge (a, b) has empty support.



5.4.1. Finding One Augmenting Path. In Figure 15, min level(u) > min level(v), and therefore, edge (u, v) is an anomalous bridge. When DDFS is performed on this bridge, assume that the red DFS trees has root u. Because v is already in a petal with bud*$(v) = b$, the green DFS tree will have root b. The two trees will simply follow predecessors and will terminate at f_1 and f_2, respectively.

A DFS from u in the red tree will yield a path from u to f_1, say p_1. Because v is in a petal with bud*$(v) = b$, the algorithm needs to find an alternating path from v to b, say p_2, starting with a matched edge (i.e., of even length). The construction of this path is described.[15] Additionally, the algorithm needs to find a path, say p_3, from b to f_2 in the green tree of the DDFS performed on bridge (u, v). Then, the complete augmenting path from f_1 to f_2 will be $p_1^{-1} \circ (u,v) \circ p_2 \circ p_3$. Clearly, p_1 and p_3 are easy to find.

We next describe how to find p_2. The algorithm observes that even level(v) = max level(v), and therefore, p_2 must use the bridge of the petal containing v. Using the petal node, the algorithm finds the endpoints of this bridge, namely c and d. It notices that c and v have the same color, say red. Therefore, it looks for a path from c to v in the red tree and a path from d to b in the green tree. For finding the latter path, it jumps from w to bud*(w) at the moment when DDFS was called with the bridge (c, d) because that will be the next node in the green tree. The latter node is a.[16] Next, the algorithm continues searching from a in the green tree and follows predecessors until it reaches b.

To find the complete path from d to b, the algorithm must find a path from a to w in the smaller petal.[17] This time, it observes that even level(w) = min level(w), and therefore, the required path does not use the bridge of the smaller petal. Instead, it is found by doing a DFS in the green tree, assuming that the color of w was green in the DDFS conducted on bridge (w, w'). Then, p_2 is obtained by concatenating the path from v to c with (c, d) with the path from d to b. The latter consists of (d, w) concatenated with the path from w to a concatenated with the path from a to b.

Figure 15. (Color online) Constructing a minimum-length augmenting path between unmatched vertices f_1 and f_2.

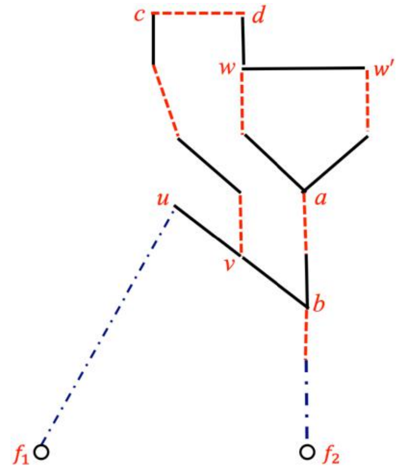
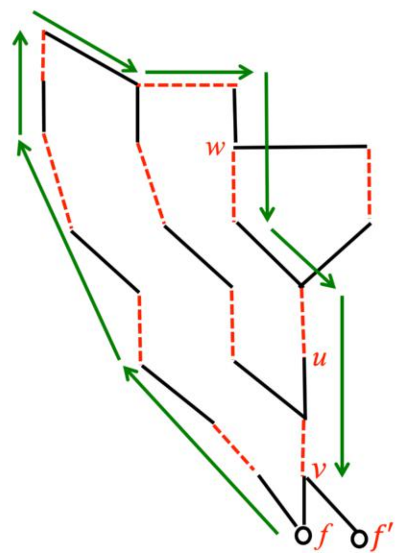


Figure 16. (Color online) The even-level(v) path, which starts at f, followed by edge (v,f') is an augmenting path.



Remark 1. Observe that the process of finding path p_2 from v to b, in Figure 15, critically used the markers left in the petal created by DDFS when called with bridge (c, d). In particular, the pointers from the petal node to the endpoints of this bridge (i.e., c and d) are critical for ensuring linear running time; disjoint paths from c to v and d to b can be easily found by doing DFSs on the two trees rooted at c and d; see Definition 2. In the absence of these markers, the algorithm would need to find disjoint paths from v to c and from b to d; it is unclear how this can be accomplished efficiently.

One last step in the process of finding an augmenting path, which could not be described because it requires Definition 27, is stated in Remark 6 in Section 8.3.1.

Example 11. In Figure 16, we have added one edge to the graph of Figure 5, namely (v,f') and the unmatched vertex f'. The even level(v) path, which starts at f, followed by edge (v,f') yields the unique augmenting path in Figure 16. Therefore, finding even level(v) in Figure 5 was not just an academic matter. However, the MV algorithm will not find this path by following the arrows in Figure 16; instead, it will use the bridges and blossoms as detailed. In particular, the bridge of tenacity $\gamma = 17$, as shown in Figure 7, triggers a DDFS that will find this path by skipping over the blossom of tenacity $\beta = 15$.

5.4.2. Finding a Maximal Set of Disjoint Paths. After the first path, say p, is found, its vertices are removed. As a result, there may be other vertices that cannot be on minimum-length augmenting paths that are disjoint from p. These vertices are recursively removed using the procedure RECURSIVE REMOVE, which works as follows. First, all vertices of p and all edges incident at them are removed. If as a result, there is a matched vertex v that has no more predecessors, it is removed. This process is continued until there are no more such vertices. Finally, all isolated unmatched vertices are removed.

At this point, MAX will process the next bridge of tenacity l_m. When it encounters another bridge that makes DDFS terminate in Case 2, it finds another augmenting path. This continues until all bridges of tenacity l_m are processed. Lemma 24 shows that this will result in a maximal set of paths of length l_m.

6. Relationship Between the Tenacity of an Edge and Tenacities of Its Endpoints

The relationship depends on whether the edge is matched or unmatched and in the latter case, whether it is a prop or a bridge. The answer in each case is significant and will influence the proof of the main theorem, Theorem 3, given in Sections 8.2 and 8.3. It will also help establish facts that give the correct way of synchronizing events in the algorithm, which are presented in Section 9.1.

Lemma 1. *Let* (u, v) *be a matched edge of finite tenacity. Then, even level*(v) = *odd level*$(u) + 1$ *and* tenacity(v) = tenacity(u,v). *Furthermore,* (u, v) *is a bridge if and only if odd level*(u) = *odd level*$(v) = i$, *where* tenacity$(u,v) = 2i + 1$.

Proof. Let p be an odd level(u) path. If v lies on p, then (u, v) must also lie on p because p is an alternating path. If so, p is not a simple odd-length alternating path to u, giving a contradiction. Therefore, v does not lie on p, and hence, $p \circ (u,v)$ is a minimum even-length alternating path to v. Therefore, even level(v) = odd level$(u) + 1$, and similarly, even level(u) = odd level$(v) + 1$. Therefore, tenacity(v) = even level(v) + odd level(v) = odd level$(u) + 1$ + odd level(v) = tenacity(u,v) = even level(u) + odd level(u) = tenacity(u).

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

Remark 2. As a consequence of Lemma 1, in several proofs, restricting attention to only one of the endpoints of a matched edge (u, v) will suffice because even level(v) and odd level(v) fully determine the two levels of u.

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

Proof. We will show that odd level(v) = even level$(u) + 1$. If so, tenacity(v) = even level(v) + odd level(v) = even level(v) + even level$(u) + 1$ = tenacity(u,v), thereby proving the lemma. Because v is getting its min level from the unmatched edge (u, v), min level(v) = odd level(v), and therefore, v is an inner vertex.

Because u is a predecessor of v, there is an odd level(v) path, say q, that ends with the edge (u, v). Now, odd level$(v) = |q|$, and even level$(u) \le |q| - 1$. Therefore, odd level$(v) >$ even level(u).

Let p be an even level(u) path. First, assume that v lies on p. If v is odd w.r.t. p,[18] then odd level$(v) <$ even level(u), a contradiction. Therefore, v is even w.r.t. p. However, then even level$(v) <$ even level(u), implying that even level$(v) <$ odd level(v) and that v is an outer vertex, another contradiction. Therefore, v does not lie on p. Therefore $p \circ (u,v)$ is an odd level(v) path, and therefore, odd level(v) = even level$(u) + 1$ holds, giving the lemma. □

Lemma 3. *Let (u, v) be an unmatched bridge. Then,*

1. tenacity$(v) \le$ tenacity(u,v); *and*
2. *if* tenacity(v) = tenacity(u,v), *then v is an outer vertex.*

Proof.

1. We will show that odd level$(v) \le$ even level$(u) + 1$. If so, adding even level(v) to both sides of this inequality, we will get tenacity$(v) \le$ tenacity(u,v).

Let p be an even level(u) path starting at unmatched vertex f. If v lies on p, there are two cases. If v is odd w.r.t. p, then $p[f \text{ to } v]$ is an odd-length alternating path from f to v, and therefore, odd level$(v) \le |p[f \text{ to } v]| < |p|$ = even level(u).

Next, assume that v is even w.r.t. p. Then, $p[f \text{ to } v] \circ (v,u)$ is an odd-length alternating path from f to u. Clearly, the length of this path is less than that of p, implying that u is an inner vertex. If this path was an odd level(u) path, then v would be a predecessor of u, implying that (u, v) is a prop and contradicting the fact that (u, v) is a bridge. Therefore, odd level$(u) < |p[f \text{ to } v]| + 1$. Let q be an odd level(u) path that starts at unmatched vertex f', where $f = f'$ is allowed. Let w be the first vertex of q that lies on $p[v \text{ to } u]$. If w is odd w.r.t. p, then $q[f' \text{ to } w] \circ p[w \text{ to } u]$ is an even-length alternating path from f' to u of length less than $|p|$ = even level(u), giving a contradiction. Therefore, w is even w.r.t. p. If so, $q[f' \text{ to } w] \circ p[w \text{ to } v]$ is an odd-length alternating path from f' to v of length less than $|p|$. Therefore, odd level$(v) <$ even level(u).

Next, assume that v does not lie on p. Then, $p \circ (u,v)$ is an odd-length alternating path from f to v. Therefore, odd level$(v) \le$ even level$(u) + 1$.

2. Assume that tenacity(v) = tenacity(u,v). Then, odd level(v) = even level$(u) + 1$. If v was an inner vertex, then odd level(v) = min level(v), and the previous equality implies that u is a predecessor of v and that (u, v) is a prop, leading to a contradiction. Therefore, v is an outer vertex. □

Remark 3. Let (u, v) be an unmatched edge, and let u be a predecessor of v. Then, tenacity(u) can be smaller than, equal to, or bigger than tenacity(u,v). The first case is illustrated by prop (r_1, l_2), and the third case is illustrated by the props out of f in Figure 10. Let (u, v) be an unmatched bridge, and assume that tenacity(v) $<$ tenacity(u,v). Then, v can be an inner vertex or an outer vertex. The bridge (a, b) in Figure 14 illustrates both possibilities; although (a, b) has empty support, it is easy to show this for bridges with nonempty support as well.

7. Limited BFS Honesty

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

Definition 15 (Even/Odd w.r.t. p). Let p be an even level(v) or odd level(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.

Definition 16 (BFS Honesty on p). Let p be an even level(v) or odd level(v) path starting at unmatched vertex f, and let u lie on p. We will say that u is *BFS honest on* p if $|p[f \text{ to } u]| = \text{even level}(u)$ (odd level(u)) if u is even (odd) w.r.t. p.

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

Theorem 2. *Let p be an even level(v) or odd level(v) path starting at unmatched vertex f, and let vertex $u \in p$ with* tenacity(u) $\geq$ tenacity(v). *Then, u is BFS honest on p. Furthermore, if* tenacity(u) $>$ tenacity(v), *then* $|p[f \text{ to } u]| = $ *min level*(u).

Proof. Assume without loss of generality that p is an even level(v) path and that u is even w.r.t. p (by Lemma 1). Suppose u is not BFS honest on p, and let q be an even level(u) path (i.e., $|q| < |p[f \text{ to } u]|$). First, consider the case that even level(v) = max level(v), and let r be a min level(v) path. Let u' be the matched neighbor of u. Consider the first vertex of r that lies on $p[u' \text{ to } v]$. If this vertex is even w.r.t. p, then odd level(u) $\leq |r| + |p[u \text{ to } v]|$. Additionally, even level(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 min level(v) $= |r| >$ even level(u) because otherwise, there is a shorter even path from f to v than even level(v). We combine the remaining argument along with the case that even level(v) = min level(v).

Consider the first vertex, say w, of q that lies on $p(u \text{ to } v]$; there must be such a vertex because otherwise, there is a shorter even path from f to v than even level(v). If w is odd w.r.t. p, then we get an even path to v that is shorter than even level(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 even level(v), where $\circ$ denotes the concatenation operator. Again, we get tenacity(u) $<$ tenacity(v), leading to a contradiction.

We next prove the second claim of the theorem. First, consider the case that even level(v) = min level(v), and assume for contradiction that $|p[f \text{ to } u]| = $ max level(u). Then, tenacity(u) $< 2 \cdot$ max level(u) $< 2 \cdot |p[f \text{ to } v]| = 2 \cdot$ min level(v) $<$ tenacity(v), a contradiction.

Therefore, even level(v) = max level(v). As before, let r be a min level(v) path, and consider the first vertex of r that lies on $p[u \text{ to } v]$. If this vertex is even w.r.t. p, then odd level(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 min level(v) $= |r| >$ even level(u) because otherwise, there is a shorter even path from f to v than even level(v). Now, the claim follows because otherwise tenacity(u) $<$ tenacity(v). □

Corollary 1. *Let p be an even level(v) or odd level(v) path, and let u lie on p. If u is not BFS honest on p, then* tenacity(u) $<$ tenacity(v).

8. Base, Blossom, and Bridge

Recall Definitions 3 and 9, which defined l_m and t_m as the length of a minimum-length augmenting path and the tenacity of a minimum tenacity vertex, respectively. As noted earlier, $t_m \leq l_m$, and the case $t_m = l_m$ is trivial. For the rest of the paper, we will deal with the main case, namely $t_m < l_m$; Example 14 explains the importance of this assumption. In Definition 17, we introduce the notion of eligible tenacity. Theorem 3 helps establish the central notions of base and blossom for vertices of eligible tenacity.

Definition 17 (Eligible Tenacity). An odd number t, with $t_m \leq t < l_m$, will be said to be an *eligible tenacity*.

Definition 18 (Higher and Lower on a Path). Let v be a vertex and p be an even level(v) or odd level(v) path; assume it starts at unmatched vertex f. If u and w are two vertices on p and if u is farther away from f on p than w, then we will say that u is *higher* than w and that w is *lower* than u on p.

Definition 19 (The Set $B(v)$). Let v be a vertex of eligible tenacity and p be an even level(v) or odd level(v) path starting at unmatched vertex f. Let t = tenacity(v), and consider all vertices of tenacity $> t$ on p; clearly, this set is nonempty because it contains f. Among these vertices, define the highest one to be *the base of v w.r.t. p*, denoted by $F(p, v)$. Clearly, $F(p, v)$ is even w.r.t. p, and by Theorem 2, it is an outer vertex. Finally, define

$$B(v) = \{F(p, v) \mid p \text{ is an even level}(v) \text{ or odd level}(v) \text{ path}\}.$$

8.1. Central Structural Facts

As mentioned in Section 1.1, the central structural fact needed to prove correctness of the algorithm is the following.

**For every vertex v such that* tenacity(v) = t *and* $t_m \le t \le l_m$, *every* max level(v) *path contains a bridge of tenacity t.*

The proof of this fact requires several other structural facts. Among them, the most important one is the following.

**For a vertex v of eligible tenacity, the set $B(v)$ is a singleton.*

Once this fact is proven, we can define the *base of v* to be the vertex in $B(v)$ and move on to defining the notion of a blossom and proving its properties. However, the proof of this fact is not straightforward because of the following "chicken-and-egg problem," which was also mentioned in Section 1.1. On the one hand, the proof of this fact requires the notion of blossom and its associated properties, and on the other hand, blossoms can be defined only after defining the base of a vertex.

We will break this deadlock by proving this fact via an induction on eligible tenacities. Once this fact is proven for vertices of tenacity $\le t$, the base of vertices of tenacity $\le t$ can be defined. Following this, blossoms of tenacity t can be defined, and properties of these blossoms and properties of paths traversing through these blossoms can be established. These properties are then used to prove this fact for tenacity $t+2$.

8.2. The Induction Basis

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

Lemma 4. *Let (u, v) be an edge of tenacity t_m. Then, (u, v) is a bridge if and only if min level(u) = min level(v) = i, where $t_m = 2i + 1$.*

Proof. If (u, v) is matched, the claim follows by Lemma 1. Next, assume that (u, v) is unmatched. If so, tenacity(u, v) = even level(u) + even level(v) + 1. First, assume (u, v) is a bridge. Because the tenacity of a vertex cannot be less than t_m, by Lemma 3, tenacity(u) = tenacity(v) = tenacity(u,v), and u and v are both outer vertices. Therefore, min level(u) = even level(u), and min level(v) = even level(v). Therefore, odd level(v) = even level(u) + 1, and odd level(u) = even level(v) + 1.

Because $t_m = 2i + 1$, min level(u) = even level(u) $\le i$, and min level(v) = even level(v) $\le i$. Assume one of them has min level $< i$, say u. Then, odd level(v) = even level(u) + 1 $\le i$. This implies that even level(v) $< i$ because v is outer, giving tenacity(v) $< 2i$, a contradiction. Therefore, min level(u) = min level(v) = i.

Next, assume min level(u) = min level(v) = i. Because (u, v) is an unmatched edge of tenacity t_m, even level(u) + even level(v) + 1 = t_m. Therefore, even level(u) = even level(v) = i. Now, because (u, v) is unmatched, u is not a predecessor of v, and v is not a predecessor of u. Therefore, (u, v) is a bridge. □

Lemma 5. *Let v be a vertex of tenacity t_m and p be a max level(v) path. Then, p contains a unique bridge of tenacity t_m.*

Proof. Assume that p starts at unmatched vertex f. Let q be a min level(v) path, and assume it starts at unmatched vertex f', which may or may not be the same as f. If p and q meet only at v, then $f \ne f'$, and $p \circ q$ is an augmenting path of length t_m, leading to a contradiction. Therefore, the intersection of p and q contains vertices in addition to v. Let u be the highest vertex of $p[f$ to $v)$ that is also on q.

Now, there are two cases. If $u = f$, then $f = f'$, and p and q meet at two vertices, namely $u = f$ and v. Next, assume that $u \ne f$. If so, u is matched, and p must contain the matched edge incident at u. Therefore, the vertex u must be even w.r.t. p. By definition, tenacity(u) $\ge t_m$, and therefore, by Theorem 2, u is BFS honest on p as well as q. Therefore, even level(u) = $|p[f$ to $u]|$. If u is odd w.r.t. q, then odd level(u) = $|q[f'$ to $u]|$, thereby implying that tenacity(u) $< |p \circ q| = t_m$, a contradiction. Therefore, u is even w.r.t. q as well, and $|q[f'$ to $u]|$ = even level(u).

Therefore, in both cases, $p[u \text{ to } v] \circ q[v \text{ to } u]$ is an odd-length cycle having two unmatched edges incident at u and is fully matched otherwise. By concatenating $p[f \text{ to } u]$ to an appropriate subpath of this cycle, we can obtain even and odd alternating paths to all vertices of this cycle other than u. For any vertex $w \neq u$ on this cycle, the sum of the lengths of the even and odd paths is t_m. Therefore, these must be minimum-length alternating paths, and tenacity$(w) = t_m$. Furthermore, because this cycle has length at least three, even level$(u) = |p[f \text{ to } u]| < i$.

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

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

8.2.1. The Graphs H_m and H'_m. Proving that the set $B(v)$ is a singleton is not straightforward, even for vertices of tenacity t_m. A major simplification is achieved by using the power of DDFS—in particular, the DDFS Certificate provided on its termination. DDFS is carried out on a special directed, layered graph H_m, which satisfies the DDFS Requirement. We start by defining H_m and a closely related graph, H'_m; the latter is a subgraph of G.

Let $U_m = \{v \in V \mid \text{tenacity}(v) = t_m\}$. Let B_m denote the set of all bridges of tenacity t_m. and let T_m denote the set of endpoints of bridges in B_m.

Lemma 6. *Let $v \in T_m$, and let p be a min level(v) path. Then, all edges on p are props.*

Proof. Assume that p starts at unmatched vertex f, and let u be any vertex on p other than v. Clearly, tenacity(u) $\geq$ tenacity(v), and therefore, by Theorem 2, u is BFS honest on p. Furthermore, $|p[f \text{ to } u]|$ = min level(u) because otherwise, tenacity$(u) <$ tenacity(v). Therefore, p gives min levels to all vertices on it. and hence. all its edges are props. □

Consider each vertex $v \in T_m$, and let p denote an arbitrary min level(v) path. Denote by V_m the union of all vertices on all such paths p for all vertices $v \in T_m$. Furthermore, denote by P_m the union of all edges on all such paths p. By Lemma 6, all edges in P_m are props in the original graph G. Define $H'_m = (V_m, (P_m \cup B_m))$, and define a matching in H'_m as follows; edge $e \in (P_m \cup B_m)$ is matched if and only if e is matched in G. Clearly, H'_m is a subgraph of G.

The next definition is related to Definition 11.

Definition 20 (The Relations pred$_m$ and pred$^*_m$). If $u \in U_m$, $v \in V_m$, and v is a predecessor of u, then we will say that v *is* pred$_m$ *of* u, denoted by $v = \text{pred}_m(u)$. Observe that under this definition, we do not allow tenacity$(u) > t_m$; tenacity(u) *must be* t_m. On the other hand, tenacity$(v) > t_m$ is allowed, and v may even be an unmatched vertex. Next, let $u \in U_m$ and $v \in V_m$, with $u \neq v$, such that there is a path from u to v in H'_m and all vertices on this path are in U_m, except possibly v. Then, we will say that v *is* pred$^*_m$ *of* u, denoted by $v = \text{pred}^*_m(u)$.

Lemma 7. *For every $v \in U_m$, H'_m contains all possible even level(v) and odd level(v) paths that are present in G.*

Proof. We will use the fact that, by construction, H'_m contains all possible min level(u) and max level(u) paths for all vertices $u \in T_m$.

For $v \in U_m$, let p and q be min level(v) and max level(v) paths, respectively, in G, which start at unmatched vertex f. Let (u, u') be the unique bridge of tenacity t_m on q, with u' being higher than u on q; for a definition of "higher on a path," see Definition 18. Then, $q[f \text{ to } u]$ is a min level(u) path, and therefore, it is present in H'_m. Furthermore, $p \circ q[v \text{ to } u']$ is a min level(u') path and is also present in H'_m. Therefore, H'_m contains p and q, hence proving the lemma. □

Next, we will define the directed, layered graph H_m. For each edge $(u, v) \in P_m$, direct it from v to u if u is a predecessor of v. To keep notation simple, we will denote this set of directed edges by P_m as well; the context will easily clarify which graph is being referred to.

We will partition V_m into $i+1$ layers numbered $0, \ldots, i$, where $t_m = 2i+1$ and the layer number of $v \in V_m$ is $l(v)$ = min level(v). Let $H_m = (V_m, P_m)$. Observe that each edge of H_m runs from a layer, say l, to $l-1$ (i.e., it is of unit length). In contrast, the graph H_t, defined in Section 8.3.1, has long edges. It is easy to check that H_m satisfies the DDFS Requirement. We will use the information obtained from DDFS on H_m to find min-level and max-level paths in H'_m. The mapping between the vertices of H_m and H'_m is the obvious one.

Corresponding to each bridge $(u, v) \in B_m$, conduct DDFS in H_m starting at u and v, and denote by $k_{(u,v)}$ the bottleneck found. Clearly, each bottleneck is an outer vertex. Note that two different bridges may have the same bottleneck. Using the DDFS Certificate and the mapping between H_m and H'_m we get Lemma 8.

Lemma 8. *Let bridge* $(u, v) \in B_m$, *and let* $w \in \text{support}(u, v)$. *Then,*

1. $w = \text{pred}^*_m(u)$, $w = \text{pred}^*_m(v)$, *or both; and*
2. $k_{(u,v)} = \text{pred}^*_m(w)$.

8.2.1.1. Procedure Bottleneck. The input to this procedure is a bridge $(u, v) \in B_m$.
Conduct DDFS in H_m, starting at u and v, to find the bottleneck $k_{(u,v)}$.
If tenacity$(k_{(u,v)}) > t_m$, HALT.
Otherwise, there is a bridge of tenacity t_m, say (u', v'), such that $k_{(u,v)} \in \text{support}(u', v')$. Conduct DDFS on bridge (u', v') to find its bottleneck, $k_{(u',v')}$. Clearly, min level$(k_{(u',v')}) <$ min level$(k_{(u,v)})$, and $k_{(u',v')} = \text{pred}^*_m(k_{(u,v)})$. If tenacity$(k_{(u',v')}) = t_m$, repeat this process until a bottleneck, say b, is encountered that has tenacity $> t_m$. This is bound to happen because the min levels of the bottlenecks are decreasing. Eventually, the bottleneck will turn out to be an unmatched vertex, say f, and f satisfies tenacity$(f) \geq l_m > t_m$.

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

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

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

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

Observe that $b \notin S_{b,t_m}$, and if b, b' are two bases in H'_m, then $S_{b,t_m} \cap S_{b',t_m} = \emptyset$. The next lemma is implied by Lemma 1 and the fact that set S_{b,t_m} is defined via the procedure bottleneck.

Lemma 9. *Let* $v \in S_{b,t_m}$, *and let* v' *be the matched neighbor of* v. *Then,* $v' \in S_{b,t_m}$.

Lemma 10. *Let* $v \in S_{b,t_m}$. *Then, every even level*(v) *and odd level*(v) *path in the graphs* H'_m *and* G *consists of an even level*(b) *path followed by an alternating path using vertices of* S_{b,t_m}.

Proof. Let $(u, u') \in B_m$ be a bridge of tenacity t_m whose endpoints are in S_{b,t_m}, and let p be any min level(u) path. By Lemma 8 and the procedure given, p^{-1} starts at u and follows down predecessors until it arrives at $k_{(u,u')}$. If $k_{(u,u')} \neq b$, p^{-1} follows down predecessors until it arrives at b. In either case, the rest of p^{-1} is an even level(b) path followed in reverse. Therefore, p has the structure described in the statement of the lemma.

Next, assume that $v \in S_{b,t_m}$, and let p and q be min level(v) and max level(v) paths in G, respectively. Using the arguments given in Lemma 7, $q \circ p^{-1}$ can be decomposed into min level(u) and min level(v) paths and the unique bridge on q. Now, by the assertion made, p and q also have the structure described in the statement of the lemma. □

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

Definition 21 (The Base of a Vertex of Tenacity t_m and Basal Vertices). For each $v \in U_m$, define base(v) to be the unique vertex, say b, in the set $B(v)$. We will say that the *base* of v is b. Each such vertex b will be called a *basal vertex*. Clearly, b is an outer vertex, and tenacity$(b) > t_m$.

Example 14. In the graph of Figure 17, vertices u and v do not have a base, even though they are of finite tenacity. Clearly, tenacity(u) = tenacity(v) = tenacity(f_1) = tenacity$(f_2) = 3$, and the graph has no vertex of tenacity greater than three. Because $l_m = 3$, u and v are not of eligible tenacity; this explains why they do not have a base. The following question arises. Can f_1 be viewed as the base of u and v, even though its tenacity is the same as that of u and v? A negative answer is easy to see after adding unmatched edge (f_2, v) because then, the graph has min level(v) and max level(u) paths that do not contain f_1.

Figure 17. (Color online) Vertices u and v have no base. The tenacity of bridge (u, v) is three, and so is the tenacity of u, v, f_1, and f_2.

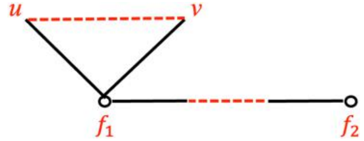
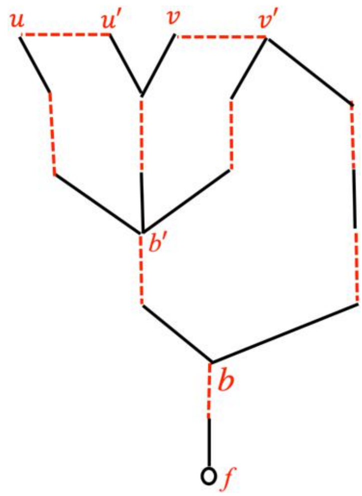


Figure 18. (Color online) The set of vertices having base b forms a blossom of tenacity 15 and $t_m = 15$. Observe that b' is not basal. This blossom contains the endpoints of bridges (u, u') and (v, v').



Definition 22 (Blossom of Tenacity t_m and Base b). Let b be a basal vertex as defined in Definition 21. Then, the *blossom of tenacity* t_m *and base* b is the set $\mathcal{B}_{b,t_m} = \{v \in U_m | \text{base}(v) = b\}$.

Clearly, $\mathcal{B}_{b,t_m} \neq \emptyset$, $\mathcal{B}_{b,t_m} = S_{b,t_m}$, $b \notin \mathcal{B}_{b,t_m}$, and for each vertex, $v \in \mathcal{B}_{b,t_m}$, $b = \text{pred}^*_m(v)$. The next fact follows from Lemmas 1 and 9.

Corollary 3. *Let* (u, v) *be a matched edge of tenacity* t_m. *Then*, u *and* v *have the same base*, *say* b, *and both belong to the same blossom of tenacity* t_m, *namely* $\mathcal{B}_{b,t_m}$.

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

Definition 23 (Shortest Path from a Base to a Vertex of Tenacity t_m). Let $v \in U_m$ and $b = \text{base}(v)$. Then, by an even level$(b;v)$ (odd level$(b;v)$) path, we mean a minimum even-length (odd-length) alternating path in G from b to v that starts with an unmatched edge.

Lemma 11. *Let* $v \in U_m$ *and* $b = \text{base}(v)$. *Let* p *be an even level*(b) *path and* q *be an even level*$(b;v)$ *or odd level*$(b;v)$ *path. Then*, q *meets* p *at* b *only*.

Proof. By Lemma 1, it suffices to prove this lemma for an odd level$(b;v)$ path q. For contradiction, assume that q meets p at vertices besides b. By Lemma 10, odd level$(v) \geq |p| + |q|$. We will define certain subpaths of q as *segments* as follows. Follow along q from b until it meets p, at w say. Then, $q[b \text{ to } w]$ will be called a segment. Subsequent to this, each time q leaves p, at vertex r, say, and meets up p again at s, say, then $q[r \text{ to } s]$ is called a segment. Eventually, q leaves p at y, say, and ends up at v. Then, $q[y \text{ to } v]$ is the last segment.

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

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

Lemma 10 and Lemma 11 give Corollary 4.

Corollary 4. *Let* $v \in U_m$ *and* $b = \text{base}(v)$. *Then, the matched neighbor of* v *also lies in the blossom, and every even level*(v) (*odd level*(v)) *path consists of an even level*(b) *path followed by an even level*$(b;v)$ (*odd level*$(b;v)$) *path, where the latter lies in* $\mathcal{B}_{b,t_m} \cup \{b\}$.

Lemma 10 and Corollary 4 give Lemma 12.

Lemma 12. *Let $\mathcal{B}_{b,t_m}$ and $\mathcal{B}_{b',t_m}$ be two blossoms with bases $b \neq b'$. Then, $\mathcal{B}_{b,t_m} \cap \mathcal{B}_{b',t_m} = \emptyset$.*

Finally, we need to prove one more fact, which captures the disciplined manner in which a minimum-length alternating path that intersects a blossom of tenacity t_m, $\mathcal{B}_{b,t_m}$, uses vertices of $(\mathcal{B}_{b,t_m} \cup \{b\})$. This fact will be used in the induction step.

Lemma 13. *Let p be an even level(v) path from unmatched vertex f to an arbitrary vertex v such that there is a blossom $\mathcal{B}_{b,t_m}$ with $p \cap \mathcal{B}_{b,t_m} \neq \emptyset$. Then, the base of this blossom, b, also lies on p, and there is a vertex $u \in (p \cap \mathcal{B}_{b,t_m})$ such that $p[b$ to $u]$ contains all the vertices of $p \cap (\mathcal{B}_{b,t_m} \cup \{b\})$ and $p[b$ to $u]$ is an even level($b;u$) path.*

Proof. For the sake of contradiction, assume that p does not intersect $\mathcal{B}_{b,t_m} \cup \{b\}$ in the manner described. If so, two cases arise.

Case 1. b lies on p, and there is a vertex $u \in (p \cap \mathcal{B}_{b,t_m})$ such that $p[b$ to $u]$ contains all the vertices of $p \cap (\mathcal{B}_{b,t_m} \cup \{b\})$ as well as some vertices not in $\mathcal{B}_{b,t_m}$ (i.e., $p[b$ to $u]$ enters and exits $(\mathcal{B}_{b,t_m} \cup \{b\})$ more than once).

If so, by Corollary 4, $p[b$ to $u]$ is longer than an even level($b;u$) path. Therefore, replacing the former with the latter will result in a shorter even path to v, giving a contradiction.

Case 2. The first vertex of p in $(\mathcal{B}_{b,t_m} \cup \{b\})$ is $x \in \mathcal{B}_{b,t_m}$, the last vertex is $y \in \mathcal{B}_{b,t_m}$, and $p(x$ to $y)$ is arbitrary (i.e., it may visit b as well as other vertices not in $\mathcal{B}_{b,t_m}$). For ease, we will call $p[f$ to $x]$ the *blue path* and $p[y$ to $v]$ the *red path*.

Let q be an even level(b) path starting at unmatched vertex f', say, where $f' = f$ is allowed. Now, there are several cases. If the red path does not intersect q, then by Corollary 4, $q \circ s \circ p[y$ to $v]$ is a shorter even path[19] to v than p, where s is an even level($b;y$) path.

If the red path does intersect q and the last vertex of p on q is even w.r.t. q, say w, then follow q to w, and then, follow by p to v; this is shorter than p. Next, assume that the last vertex of p on q is odd w.r.t. q, say w. Then, we ask whether the blue path intersects $q(w$ to $b]$. If the answer is no, then via Corollary 4, we get that the blue path followed by r^{-1} to b followed by q to w followed by p to v is a shorter even path to v, where r is an even level($b;x$) path.

Case 2′ (The Main Case). Finally, assume that the blue and red paths both intersect q in arbitrary ways and that the last vertex of the red path on q is odd w.r.t. q, say w. Both paths will traverse some matched edges of the path q. We will say that such an edge is blue (red) if the blue (red) path traverses it; furthermore, we will direct the edge in the direction in which the path traverses it. We will say that such an edge is *directed up* if it points toward b and is *directed down* if it points toward f'.

A subpath of $q[w$ to $b)$ is said to be *unicolored* if it does not contain matched edges of both colors; it is allowed to contain uncolored matched edges. A maximal unicolored subpath that starts and ends with colored matched edges is called an *interval*. In each interval, we will *mark* one edge as follows; in a blue interval, mark the lowest[20] edge on the blue path, and in a red interval, mark the highest edge on the red path. The rest of the arguments will only be based on the marked edges and not based on the rest of the intervals. Clearly, on $q[w$ to $b)$, the color of the marked edges must be alternating, with the lowest marked edge being red and directed down.

There are two cases; either all marked red edges are directed down or not. In the latter case, let e_1 be the lowest marked red edge that is directed up, and let e_2 be the marked red edge just below e_1; e_2 is directed down. Let e_3 be the marked blue edge in between e_1 and e_2. Now, if e_3 is directed down, then follow the blue path from f to e_3, then follow q to e_2, and finally, follow the red path all the way to v. However, if e_3 is directed up, then follow the blue path from f to e_3, then follow q to e_1, and finally, follow the red path all the way to v.

Next, we deal with the case that all marked red edges are directed down. Now, we ask what is the color of the highest marked edge, say e. If e is red, then we follow the entire blue path from f to x, then follow r^{-1} to b, then follow q to e, and finally, follow the red path to v, where r is an even level($b;x$) path.

Finally, assume that e is blue. Again two cases arise. If e is directed down, then we follow the blue path from f to e, then follow q to the highest red edge (which is directed down), and finally, follow the red path to v. If e is directed up, then we follow the blue path from f to e, then follow q to b, then follow s to y, and finally, follow the red path to v, where s is an even level($b;y$) path. In all cases, a shorter even path to v is obtained, leading to a contradiction. □

Definition 24 (BFS Honesty on p with Respect to the Base). Let p be an even level(v) or odd level(v) path starting at unmatched vertex f to an arbitrary vertex v, let u lie on p with tenacity(u) = t_m, and let b = base(u). Then, we will say that u is *BFS honest on p w.r.t. b* if $p[b$ to $u]$ is an even level($b;u$) (odd level($b;u$)) path assuming that $|p[b$ to $u]|$ is even (odd). Note that we are allowing b to appear either before or after u on path p.

Lemma 13 and Theorem 2 yield Lemma 14.

Lemma 14. *Let p be an even level(v) path from unmatched vertex f to an arbitrary vertex v. Let $w \in p$ with* tenacity(w) = t_m, *and let b =* base(w). *Then, b also lies on p, and w is BFS honest on p w.r.t. b.*

Proof. By Lemma 13, there is a vertex u on p such that $p[b \text{ to } u]$ contains all vertices in $p \cap (\mathcal{B}_{b,t_m} \cup \{b\})$ and $p[b \text{ to } u]$ is an even level$(b;u)$ path. Let q be an even level(b) path. Then, by Corollary 4, $q \circ p[b \text{ to } u]$ is an even level(u) path. Because w lies on this path and tenacity(w) = tenacity$(u) = t_m$, by Theorem 2, w is BFS honest on this path. Hence, w is BFS honest on p w.r.t. b. □

Note that in Lemmas 13 and 14, we took p to be an even level(v) path; by Remark 2, this is without loss of generality. Observe that Lemma 13 allows b to appear either before or after u on path p. The following question arises. How does this affect whether b and u are BFS honest on p? Lemma 15 provides an answer, and Example 16 illustrates the various cases.

Lemma 15. *Let p be an even level(v) path from unmatched vertex f to an arbitrary vertex v such that there is a blossom $\mathcal{B}_{b,t_m}$ with $p \cap \mathcal{B}_{b,t_m} \neq \emptyset$. Let vertex $u \in (p \cap \mathcal{B}_{b,t_m})$ be such that $p[b \text{ to } u]$ contains all the vertices of $p \cap (\mathcal{B}_{b,t_m} \cup \{b\})$. Then, the following hold.*

1. *If b appears before u on path p, then b and u are either both BFS honest or both not BFS honest on p.*
2. *If b appears after u on path p, then u is not BFS honest on p; furthermore, if b is BFS honest on p, then $p[f \text{ to } b]$ is a max level(b) path.*

Proof.

1. The proof follows from Corollary 4.
2. If b appears after u on path p, then by Corollary 4, u is not BFS honest on p. Furthermore, if b is BFS honest on p, then because $p[f \text{ to } b]$ cannot be a min level(b) = even level(b) path, it must be an odd level(b) = max level(b) path. □

Example 16. In the graph of Figure 23, even level$(u) = 8$; let p be this path. Vertex w appears on the p, tenacity$(w) = t_m = 13$, base$(w) = b$, and b appears before w on p. Observe that b and w are both BFS honest on p. Additionally, w is BFS honest on p w.r.t. b.

In the graphs of Figures 25 and 26, tenacity$(u) = t_m = 11$, and base$(u) = b$. These graphs has three bridges—of tenacities 11, 13, and 15. Observe that even level$(v) = 16$, and let p denote the even level(v) path; the three bridges appear in the order 15, 11, and 13 on p. Now, b appears before u on p. Observe that b and u are both not BFS honest on p; however, u is BFS honest on p w.r.t. b.

In the graph of Figure 23, even level$(c) = 12$; let q be the even level(c) path. Observe that b appears after w on q and that w is not BFS honest on q. However, b is BFS honest on q, and $q[f \text{ to } b]$ is a max level(b) path. Additionally, w is BFS honest on q w.r.t. b.

In the graph of Figure 23, even level$(v) = 16$; let r be the even level(v) path. Observe that b appears after w on r and that b and w are both not BFS honest on r; however, w is BFS honest on r w.r.t. b.

Remark 4. As a result of the induction basis, we have established that every vertex v of tenacity t_m has a unique base, base(v). As stated in Section 8.1, the induction step, in Theorem 3, will enable us to establish an analogous fact about higher-tenacity vertices, eventually establishing it for every vertex of eligible tenacity. As a result, every such vertex w will have a unique base, base(w), which is an outer vertex and satisfies tenacity(base(w)) > tenacity(w). For ease of exposition, Definition 25 assumes that base(w) is well defined for each vertex w of eligible tenacity in order to define the *iterated bases of v*, where tenacity$(v) = t_m$. A more "correct," although more cumbersome, way of doing this would be to define higher and higher iterated bases of v after each induction step.

Definition 25 (Iterated Bases of a Vertex of Tenacity t_m). For $v \in U_m$, let base$(v) = b$. Define the *first iterated base of v* to be b, denoted as follows: base$^1(v) = b$. When base(b), base(base(b)), etc. get defined in the induction step, we can define higher iterated bases of v. Thus, for $k \geq 1$, we will say that base$^{k+1}(v)$ = base(base$^k(v)$), assuming that base$^k(v)$ and base(base$^k(v)$) exist in the graph.

8.3. The Induction Step

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

Hypothesis 1 (Induction Hypothesis). *Let t be an eligible tenacity, with $t_m + 2 \leq t < l_m$. Then, each of the statements in Theorem 3 holds for tenacities in the range $[t_m, t-2]$.*

After proving statement 2 of Theorem 3, certain key notions will become well defined for the case of tenacity t. These definitions are formally stated after the proof of statement 2 of Theorem 3, and they will be used for formally stating and proving the rest of the statements. These include the base of vertices of tenacity t, blossoms of tenacity t, and iterated bases of a vertex.

In the induction step, we will need the following definition regarding the iterated bases of a vertex v such that tenacity(v) $< t$; these bases would already be defined in the previous iterations of the induction.

Definition 26. Let vertex v be such that tenacity(v) $< t$. Define $k(t, v)$ such that $\text{base}^{k(t,v)}(v)$ is the first iterated base of v having tenacity at least t: that is,

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

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

Statement 1. *Every max level(v) path contains a bridge of tenacity t.*

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

Statement 3. *Let $b = \text{base}(v)$. Then, every even level(v) (odd level(v)) path consists of an even level(b) path followed by an even level($b;v$) (odd level($b;v$)) path, where the latter lies in $\mathcal{B}_{b,t} \cup \{b\}$.*[21]

Statement 4. *The blossoms of tenacity t are disjoint, and the set of blossoms of tenacity at most t forms a laminar family.*

Statement 5. *Let p be an even level(u) path from unmatched vertex f to an arbitrary vertex u. Let $w \in p$ with tenacity(w) $= t$, and let $b = \text{base}(w)$. Then, b also lies on p, and w is BFS honest on p w.r.t. b.*

Note that in statement 5 of Theorem 3, we took p to be an even level(v) path; by Remark 2, this is without loss of generality.

Proof of Statement 1 of Theorem 3. The proof is given in Lemma 16 and is very different from the proof of the analogous fact, given in Lemma 5, in the induction basis. The reason is that the former needs to account for blossoms defined in the previous iterations of the induction. □

Lemma 16. *Let v be a vertex of tenacity t and p be a max level(v) path. Then, p contains a bridge of tenacity t.*

Proof. Assume that p starts at unmatched vertex f, and let u be an arbitrary vertex on p. If tenacity(u) $= t$, then by Theorem 2, u is BFS honest on p, and therefore, $|p[f \text{ to } u]|$ is either min level(u) or max level(u). If tenacity(u) $< t$, then by applying Lemma 20 from the previous induction step to p, we get that $x = \text{base}^{k(t,u)}(u)$ lies on p and that $p[x \text{ to } u]$ lies in $\{x\} \cup \mathcal{B}_{x,t-2}$, where $\mathcal{B}_{x,t-2}$ is the blossom of tenacity $t-2$ with base x.

Define sets S_1, S_2, and S as follows:

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

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

$$S = S_1 \cup S_2.$$

For a definition of "higher/lower on a path," see Definition 18. Let w be the lowest vertex of S on p, and let w' be the matched neighbor of w. First, assume that $w \in S_2$; therefore, tenacity(w) $< t$. Let $x = \text{base}^{k(t,w)}(w)$ and $w \in \mathcal{B}_{x,t-2}$. Applying Lemma 17 from the previous induction step, we get that $w' \in \mathcal{B}_{x,t-2}$. Because w' is not the lowest vertex of S on p, we get that w must be odd w.r.t. p. Consider the following two cases.

Case 1. w is even with respect to p.

By the argument given, $w \notin S_2$. Therefore, $w \in S_1$, and hence, tenacity(w) $= t$ and $|p[f \text{ to } w]| = \text{max level}(w) = \text{even level}(w)$. By Lemma 1, tenacity($w'$) $=$ tenacity(w, w') $= t$, and w' is lower than w on p. Because w' is not the lowest vertex of S on p, $w' \notin S$, and therefore, $|p[f \text{ to } w']| = \text{min level}(w')$. Furthermore, because w' is odd w.r.t. p, w' is an inner vertex. Because max level(w) $=$ even level(w), w is also an inner vertex. Therefore, the predecessors of w and w' are given by unmatched edges incident at them. Therefore, neither is w a predecessor of w' nor is w' a predecessor of w. Hence, (w, w') is a bridge of tenacity t.

Case 2. w is odd with respect to p.

Let (u, w) be the unmatched edge on p incident at w. Clearly, u is lower than w on p, and $|p[f \text{ to } u]|$ is even. We will show that (u, w) is a bridge of tenacity t.

First, let us show that w is not a predecessor of u. Suppose tenacity(u) $\geq t$. By Theorem 2, u is BFS honest on p, and because $u \notin S_1$, $|p[f \text{ to } u]| \neq \text{max level}(u)$. Therefore, $|p[f \text{ to } u]| = \text{min level}(u) = \text{even level}(u)$. Furthermore, the predecessor of u is its matched neighbor and not w.

Next, suppose that tenacity(u) $< t$. Let $x = \text{base}^{k(t,u)}(u)$ and $u \in \mathcal{B}_{x,t-2}$. Clearly, the predecessor of u is either x, or it lies in the blossom $\mathcal{B}_{x,t-2}$. Applying Lemma 20 from the previous induction step to p, we get that x lies on p and that the path $p[x \text{ to } u]$ lies in $\{x\} \cup \mathcal{B}_{x,t-2}$. Because $u \notin S_2$, the base of $\mathcal{B}_{x,t-2}$, namely x, is lower than u on p. If $w \in S_1$, tenacity(w) $= t$, and therefore, w does not lie in $\mathcal{B}_{x,t-2}$. If $w \in S_2$, the base of the blossom containing w is higher than w on p, and again, w does not lie in $\mathcal{B}_{x,t-2}$. Therefore, in both cases, w is not a predecessor of u. As stated, by Lemma 20, $p[x \text{ to } u]$ lies in $\{x\} \cup \mathcal{B}_{x,t-2}$. Furthermore, because w does not lie in $\mathcal{B}_{x,t-2}$ and w is not the base of $\mathcal{B}_{x,t-2}$, we get that $p[x \text{ to } u]$ contains all vertices in $p \cap (\mathcal{B}_{x,t-2} \cup \{x\})$. Now, by Lemma 19 of the previous induction step, $p[x \text{ to } u]$ is an even level$(x;u)$ path.[22] Because tenacity$(x) \geq t$, by Theorem 2, x is BFS honest on p. Therefore, $|p[f \text{ to } u]| =$ even level(u) in this case as well.

Next, let us show that u is not a predecessor of w. If tenacity$(w) = t$, then $w \in S_1$, and $|p[f \text{ to } w]| =$ max level(w). Therefore, the predecessor of w is its matched neighbor and not u. If tenacity$(w) < t$, then $w \in S_2$ and $w \in \mathcal{B}_{y,t-2}$, where $y = \text{base}^{k(t,w)}(w)$. Therefore, the predecessor of w lies in the blossom $\mathcal{B}_{y,t-2}$; again, u is not a predecessor of w. Hence, (u, w) is a bridge.

Finally, we will show that tenacity$(u,w) = t$, thereby completing the proof. We will consider two cases. First, assume that tenacity$(w) = t$. By Theorem 2, $|p[f \text{ to } w]| =$ odd level$(w) =$ even level$(u) + 1$; the last equality follows from the fact that $|p[f \text{ to } u]| =$ even level(u). Now,

$$\text{tenacity}(u,w) = \text{even level}(u) + \text{even level}(w) + 1 = \text{even level}(u) + (t - \text{odd level}(w)) + 1 = t.$$

Next, assume that tenacity$(w) < t$. If so, $w \in S_2$, and $y = \text{base}^{k(t,w)}(w)$ is higher than w on p. Let q be a min level(v) path from f to v. Because tenacity$(v) = t$, $|p| + |q| = t$. Because tenacity$(y) \geq t$, by Theorem 2, y is BFS honest on p. Therefore, $p[f \text{ to } y]$ is an odd level(y) path. Also, $q \circ p[v \text{ to } y]$ is an even path from f to y, and therefore, its length is at least even level(y).

Now, $|p[f \text{ to } y]| + |q \circ p[v \text{ to } y]| = |p| + |q| = t \geq$ odd level(y) + even level(y) = tenacity$(y) \geq t$, thereby implying that tenacity$(y) = t$ and $q \circ p[v \text{ to } y]$ is an even level(y) path. Because $p[y \text{ to } w]$ contains all vertices in $p \cap (\mathcal{B}_{y,t-2} \cup \{y\})$, by Lemma 19 of the previous induction step, $p[y \text{ to } w]$ is an even level$(y;w)$ path. Together with the previous assertion, we get that $q \circ p[v \text{ to } w]$ is an even level(w) path. Finally,

$$\text{tenacity}(u,w) = \text{even level}(u) + \text{even level}(w) + 1 = |p[f \text{ to } u]| + |q \circ p[v \text{ to } w]| + 1 = |p| + |q| = t. \quad \square$$

Remark 5. As was done in Lemma 5, Lemma 16 can be strengthened to show uniqueness of the bridge as well. Because we will not need this fact, we will not prove it.

8.3.1. The Graphs H_t and H_t'.

For proving the induction step, we will define graphs H_t and H_t', which are analogous to H_m and H_m' defined in Section 8.2.1. The main difference is that whereas all edges in the directed graph H_m are of unit length, those in H_t can be longer. The long edges help "jump" over lower-tenacity blossoms.

Recall that $U_t = \{v \in V \mid \text{tenacity}(v) = t\}$. Let B_t denote the set of all bridges of tenacity t, and let W_t denote the endpoints of all bridges in B_t.

Definition 27. For $v \in W_t$, define

$$v^* = \begin{cases} v & \text{if tenacity}(v) = t, \\ \text{base}^{k(t,v)}(v) & \text{if tenacity}(v) < t. \end{cases}$$

Note that $k(t, v)$ is defined in Definition 26. We now explain the second case of the definition of v^* (i.e., if tenacity$(v) < t$). In this case, v is in a blossom of tenacity $t-2$, which must have been defined in the previous induction step. Now, v^* is meant to be the base of this blossom; it is given by $\text{base}^{k(t,v)}(v)$.

Let $W_t^* = \{v^* \mid v \in W_t\}$. For a vertex $v^* \in W_t^*$, let p denote an arbitrary min level(v^*) path. Let V_t denote the set of all vertices of tenacity at least t on all min level(v^*) paths p for all vertices $v^* \in W_t^*$.

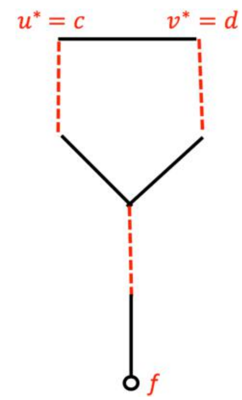
The next definition is related to Definition 20 and is motivated by statement 5 of Theorem 3 and the induction hypothesis.

Definition 28 (The Relations pred_t and pred_t^*). Let $v \in V_t$ with tenacity$(v) = t$, and let u be a predecessor of v; clearly, u may be unmatched. Define

$$\text{pred}_t(v;u) = \begin{cases} u & \text{if tenacity}(u) \geq t, \\ \text{base}^{k(t,u)}(u) & \text{if tenacity}(u) < t. \end{cases}$$

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

Figure 19. (Color online) Graph H'_t for $t = 19$ corresponding to the graph of Figure 14.



Next, we define directed graph H_t and undirected graph H'_t. Analogous to the definitions of graphs H_m and H'_m given in Section 8.2.1, the vertex sets of both H_t and H'_t are the same, namely V_t. The edge set of H_t, E_t, is defined as follows. For vertices $w, v \in V_t$, if $w = \text{pred}_t(v)$, then there is a directed edge $(v, w) \in E_t$. This edge is of unit length if $\text{pred}_t(v)$ is obtained from the first case of Definition 28, and it is longer if $\text{pred}_t(v)$ is obtained from the second case of Definition 28. In contrast, graph H_m, defined in Section 8.2.1, has unit-length edges only. Define directed graph $H_t = (V_t, E_t)$.

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

$$B_t^* = \{(u^*, v^*) | (u, v) \in B_t\}.$$

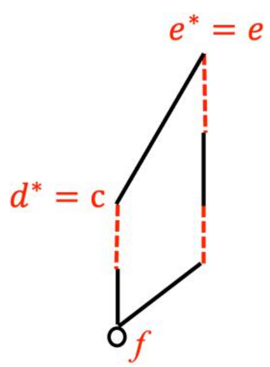
Define undirected graph $H'_t = (V_t, (B_t^* \cup E'_t))$, where the edge set E'_t is obtained from E_t by making each edge undirected. An edge $e \in E'_t$ is matched in H'_t if and only if the corresponding edge is present in G and is matched in G.

Example 17. Figures 19 and 20 illustrate the graphs H'_t corresponding to the graph of Figure 14 (and Figure 26) for $t = 19$ (and $t = 15$).

Remark 6. In Definition 27, for $v \in W_t$, if tenacity$(v) < t$, then $v^* = \text{base}^{k(t,v)}(v)$. This case adds an extra step in the process of finding an augmenting path, stated in Section 5.4.1; this step could not be described in that section because of lack of definition of v^*. We will explain this in the context of the graph of Figure 26. In order to find an even level(v) path, the algorithm needs to find a path from d to a in the blossom $\mathcal{B}_{f,15}$. Because $d^* \neq d$, it realizes that a part of this path, namely the path from d to $d^* = c$, needs to be found in the nested blossom $\mathcal{B}_{c,13}$.

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

Figure 20. (Color online) Graph H'_t for $t = 15$ corresponding to the graph of Figure 26.



Lemmas 7 and 8 carry over, and so does procedure bottleneck. The output of a run of this procedure on input $(u,v) \in B_t$ is a vertex b. As in the induction basis, this vertex is very special and is called a *base*; a formal definition is given. Clearly, b is an outer vertex. In general, H'_t will have a number of bases.

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

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

Clearly, $b \notin S_{b,t}$.

Proof of Statement 2 of Theorem 3. Lemma 10 also carries over, and so does Corollary 2, thereby proving this statement. □

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

Definition 29 (The Base of a Vertex of Tenacity t and Basal Vertices). For each $v \in U_t$, define base(v) to be the unique vertex, say b, in the set $B(v)$. We will say that the *base* of v is b. Each such vertex b will be called a *basal vertex*. Clearly, b is an outer vertex, and tenacity(b) $> t$.

We can now define the iterated bases of a vertex of tenacity at most t; this is done in Definition 30. Once the entire induction step is proven, this definition holds for all vertices of eligible tenacity.

Definition 30 (Iterated Bases of a Vertex of Tenacity at Most t). For $v \in U_t$, let base(v) $= b$. Define the *first iterated base of* v to be b, denoted as follows: $\text{base}^1(v) = b$. Next, consider a vertex u with tenacity(u) $< t$ and such that the induction hypothesis has established that $\text{base}^k(u) = v$, for $k \in \mathbb{Z}_+$. Then, $\text{base}^{k+1}(u) = b$ (i.e., $\text{base}^{k+1}(u) = \text{base}(\text{base}^k(u))$).

Example 18. In the graph of Figure 23, the iterated bases of vertex w are base(w) $= b$, $\text{base}^2(w) = b'$, and $\text{base}^3(w) = f$. Clearly, tenacity(w) $<$ tenacity(b) $<$ tenacity(b') $<$ tenacity(f).

We next come to the key definitions of blossom of tenacity t and the nesting of blossoms.

Definition 31 (Blossom of Tenacity t and Base b). Let b be a basal vertex with tenacity(b) $> t$. Let $T_{b,t} = \{v \in U_t \,|\, \text{base}(v) = b\}$; observe that $T_{b,t} = S_{b,t}$. Then, the *blossom of tenacity t and base b* is the set

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

In the expression given for $\mathcal{B}_{b,t}$, if for $v \in (T_{b,t} \cup \{b\})$, $\mathcal{B}_{v,t-2} \neq \emptyset$, then we will say that $\mathcal{B}_{v,t-2}$ is a *nested blossom of* $\mathcal{B}_{b,t}$. We will assume that the latter relation is transitively closed (i.e., if set A is a nested blossom of B and B is a nested blossom of C, then A is a nested blossom of C). Clearly, if $T_{b,t} = \emptyset$, then $\mathcal{B}_{b,t} = \mathcal{B}_{b,t-2}$.

The next fact is analogous to Corollary 3.

Lemma 17. *Let (u,v) be a matched edge of tenacity t. Then, u and v have the same base, say b, and both belong to the same blossom of tenacity t, namely $\mathcal{B}_{b,t}$.*

Example 19. Figures 21 and 22 show two graphs with nested blossoms. Although the two "look different," in both graphs, vertex b is the base of vertices u, u', v, and v'. The tenacities of bridges (u,u') and (v,v') are 7 and 11, respectively. The set $T_{b,11} = \{v,v'\}$, and the blossoms are $\mathcal{B}_{b,7} = \{u,u'\}$, $\mathcal{B}_{b,9} = \{u,u'\}$, and $\mathcal{B}_{b,11} = \{u,u',v,v'\}$.

Definition 32 (Shortest Path from a Base to a Vertex of Tenacity t). Let $v \in U_t$ and $b = \text{base}(v)$. Then, by an even level $(b;v)$ (odd level$(b;v)$) path, we mean a minimum even-length (odd-length) alternating path in G from b to v that starts with an unmatched edge.

We will extend Definition 32 to define a *shortest path from an iterated base*, say b, to a vertex v of tenacity t. We will denote this path also by even level$(b;v)$ (odd level$(b;v)$) depending on whether the path is even (odd) in length.

Proof of Statement 3 of Theorem 3. Claims analogous to Lemma 10 and Lemma 11 hold, thereby proving this statement; it is analogous to Corollary 4 in the induction basis. □

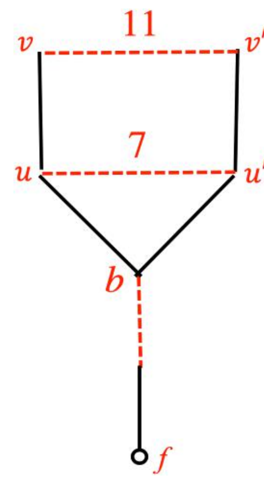
A claim analogous to Lemma 12 also holds.

Lemma 18. *Let $\mathcal{B}_{b,t}$ and $\mathcal{B}_{b',t}$ be two blossoms with bases $b \neq b'$. Then, $\mathcal{B}_{b,t} \cap \mathcal{B}_{b',t} = \emptyset$.*

By Definition 31, if $\mathcal{B}_{b,t}$ and $\mathcal{B}_{b',t'}$ are two blossoms with $t > t'$, then either they are disjoint, or the former contains the latter; note that we are allowing $b = b'$. This gives Corollary 5.

Corollary 5. *The set of blossoms of tenacity at most t forms a laminar family.*

Figure 21. (Color online) Vertex b is the base of u, u', v, and v'. The tenacities of bridges (u, u') and (v, v') are indicated.



Proof of Statement 4 of Theorem 3. Lemma 18 and Corollary 5 prove this statement. □

Definition 33 (BFS Honesty on p with Respect to an Iterated Base). Let p be an even level(v) path starting from unmatched vertex f to an arbitrary vertex v, and let u lie on p with tenacity(u) $\le t$. Let $b = \text{base}^k(u)$ be any one of the iterated bases of u that is well defined at this stage of the induction. Then, we will say that u is *BFS honest on p w.r.t. b* if $p[b \text{ to } u]$ is an even level($b;u$) (odd level($b;u$)) path assuming $|p[b \text{ to } u]|$ is even (odd). Note that we are allowing b to come either before or after u on p.

Proof of Statement 5. We will state a fact that is analogous to Lemma 13; its proof is also analogous and is omitted. □

Lemma 19. *Let p be an even level(u) path from unmatched vertex f to an arbitrary vertex u such that there is a blossom $\mathcal{B}_{b,t}$ with $p \cap \mathcal{B}_{b,t} \neq \emptyset$. Then, the base of this blossom, b, also lies on p, and there is a vertex $y \in (p \cap \mathcal{B}_{b,t})$ such that $p[b \text{ to } y]$ contains all vertices in $p \cap (\mathcal{B}_{b,t} \cup \{b\})$ and $p[b \text{ to } y]$ is an even level($b;y$) path.*

One difference between Lemma 13 and Lemma 19 is that in the former, $b = \text{base}(u)$, whereas in the latter, b may be any iterated base of y that is defined at this stage. Even so, statement 5 of Theorem 3 follows from Lemma 19, along the lines of Lemma 14. The reason is that the only iterated base of vertex w, occurring in statement 5 of Theorem 3, that is defined at this stage is base(w) because tenacity(w) = t.

This completes the proof of statement 5 of Theorem 3. Next, let us integrate this statement over all the induction steps until the current one to get the following fact about iterated bases and nested blossoms. It will be used in the proof of Lemma 16 in the next induction step.

Figure 22. (Color online) Vertex b is the base of u, u', v, and v'. The tenacities of bridges (u, u') and (v, v') are indicated.

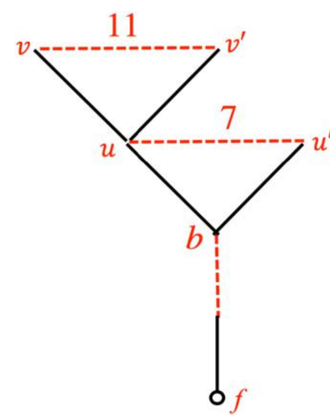
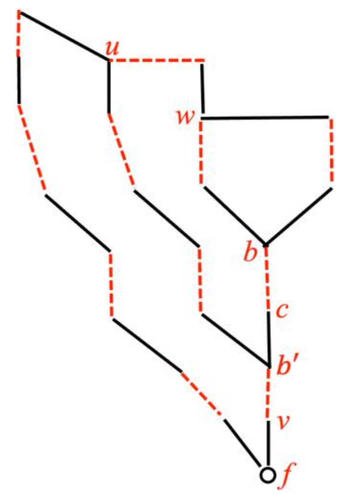


Figure 23. (Color online) The iterated bases of vertex w are $\text{base}(w) = b$, $\text{base}^2(w) = b'$, and $\text{base}^3(w) = f$, and those of vertex u are $\text{base}(u) = b'$ and $\text{base}^2(u) = f$.



Lemma 20. *Let p be an even level(v) path from unmatched vertex f to an arbitrary vertex v. Let vertex $u \in p$ with* $\text{tenacity}(u) \le t$, *and let $l = k(t+2, u)$. Let the iterated bases of u,* $\text{base}^1(u), \ldots, \text{base}^l(u)$, *be $x_1, \ldots, x_l$, respectively. For $1 \le i \le l$, let* $s_i = \text{tenacity}(x_i)$, *and let $\mathcal{B}_{x_i, s_i - 2}$ be the blossom of tenacity $s_i - 2$ with base x_i. Then,*

1. *the iterated bases of u, $x_1, \ldots, x_l$, lie on p; and*
2. *for $1 \le i \le l$, the path $p[x_i$ to $u]$ lies in $\{x_i\} \cup \mathcal{B}_{x_i, s_i - 2}$.*

Remark 7. In Lemma 20, u need not be BFS honest on p w.r.t. x_i for $1 < i \le l$; see Theorem 6 and Example 21.

Lemma 21, stated here, is analogous to Lemma 15. The various cases discussed in this lemma can be illustrated for any eligible tenacity $t > t_m$, similar to the way it was done in Example 16. However, because the examples become large, we have illustrated only a subset of the cases in Example 20.

Lemma 21. *Let p be an even level(v) path from unmatched vertex f to an arbitrary vertex v such that there is a blossom $\mathcal{B}_{b,t}$ with $p \cap \mathcal{B}_{b,t} \ne \emptyset$. Let vertex $u \in (p \cap \mathcal{B}_{b,t})$ be such that $p[b$ to $u]$ contains all the vertices of $p \cap (\mathcal{B}_{b,t} \cup \{b\})$. Then, the following hold.*

1. *If b appears before u on path p, then b and u are either both BFS honest or both not BFS honest on p.*
2. *If b appears after u on path p, then u is not BFS honest on p; furthermore, if b is BFS honest on p, then $p[f$ to $b]$ is a max level(b) path.*

This completes the proof of Theorem 3.

Example 20. In the graphs of Figures 23 and 24, vertex $u \in \mathcal{B}_{b',15}$. Consider the even level(v) paths in both these graphs. On these paths, u appears before b'. Furthermore, u is not BFS honest, and b' is BFS honest on these paths.

In the graph of Figure 24, consider the odd level(w) path. b' appears before u on this path, and b' and u are both BFS honest on this path.

Finally, in the graph of Figure 26, consider the even level(v) path, p. This path goes through the blossom $\mathcal{B}_{c,13}$; it enters the blossom at vertex d. d appears before c on this path, and whereas d is not BFS honest, c is BFS honest on this path.

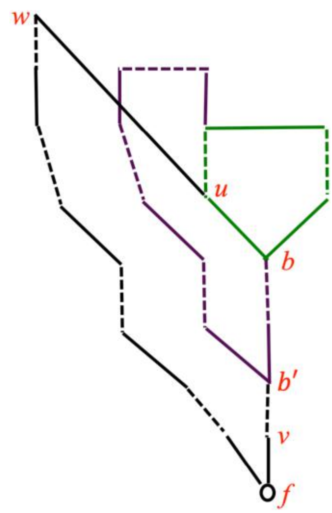
As observed in Example 14, vertices of tenacity l_m may have no base, and as a result, statements 2–5 of Theorem 3 do not hold for them. However, statement 1 of Theorem 3 does hold and needs to be proven; in particular, DDFS on the corresponding bridge will reveal an augmenting path if one exists. This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

Theorem 4. *For every vertex v of tenacity l_m, every max level(v) path contains a bridge of tenacity l_m.*

8.4. BFS Honesty and Iterated Bases

The properties established in Sections 8.2 and 8.3 help prove that the MV algorithm correctly finds the first minimum-length augmenting path in a phase; in particular, the information left in the graph, such as pointers, is

Figure 24. (Color online) Let p be the even-level(v) path. Vertex u is BFS honest on p w.r.t. b and b'; however, u is not BFS honest on p w.r.t. f.



critical to accomplishing this task. After finding an augmenting path, the algorithm removes it and all vertices that cannot be present on a disjoint minimum-length augmenting path.

This raises the following question. Does the remaining graph have the required structural properties and information to enable the algorithm to find successive augmenting paths? In this section, we prove additional properties that provide a positive answer to this question. These properties also show the sense in which minimum-length alternating paths are not arbitrarily BFS dishonest, as stated in Section 1.

Definition 34 (The Number of Iterated Bases of a Vertex of Eligible Tenacity). Let v be a vertex of eligible tenacity. As noted earlier, the tenacities of its iterated bases keep increasing (i.e., tenacity(base(v)) < tenacity(base^2(v)) < tenacity(base^3(v))...). Let l be the smallest number such that tenacity(base^l(v)) $\geq l_m$; clearly, such a number exists because the tenacity of unmatched vertices is at least l_m. Because base^l(v) is not a vertex of eligible tenacity, by definition it does not have a base. We will say that v *has exactly* l *iterated bases*.

Theorem 5. *Let v be a vertex of eligible tenacity, and let p be an even level(v) or odd level(v) path, starting at unmatched vertex f. Assume that v has exactly l iterated bases. Then, the following hold.*

1. *All l iterated bases of v lie on p; moreover, they occur in the order* base^l(v), base^{l-1}(v), ..., base(v) *on p.*
2. *Each iterated base is BFS honest on p.*
3. *v is BFS honest on p w.r.t. each iterated base.*

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

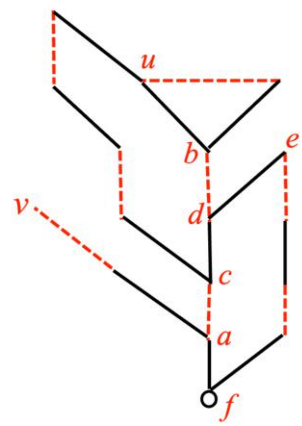
Let base(v) = b. By statement 3 of Theorem 3, every even level(v) (odd level(v)) path consists of an even level(b) path concatenated with an even level($b;v$) (odd level($b;v$)) path. Therefore, p[f to b] is an even level(b) path, hence establishing the basis of the induction.

Assume that the claim is true for k, where $1 \leq k < l$, and let base^k(v) = u. Then, p[f to u] is an even level(u) path. Let base(u) = w; clearly, base^{k+1}(v) = w. Again, by statement 3 of Theorem 3, w lies on p[f to u], and p[f to w] is an even level(w) path, hence establishing the induction step (Figure 25). □

Theorem 6. *Let v be a vertex of eligible tenacity and p be an even level(v) or odd level(v) path, starting at unmatched vertex f. Let u be a vertex of eligible tenacity that is on p, and assume that u has exactly l iterated bases. Then, the following hold.*

1. *If u is BFS honest on p, then the l iterated bases of u satisfy the three conditions stated in Theorem 5.*
2. *If u is not BFS honest on p, then*
   a. *All l iterated bases of u lie on p.*
   b. *u is BFS honest on p w.r.t.* base(u), *and for $1 \leq k < l$,* base^k(u) *is BFS honest on p w.r.t.* base^{k+1}(u).
   c. base^l(u) *is BFS honest on p.*

Figure 25. (Color online) The even level(v) is finite; the reader is encouraged to find such a path.



Proof.

1. Because $p[f \text{ to } u]$ is a minimum alternating path to u, the three conditions of Theorem 5 apply.
2. Next, assume that u is not BFS honest on p. Even so, the first and second facts follow using Lemmas 13 and 19 and an easy induction on k. Because tenacity($\text{base}^l(u)$) $\geq l_m$ and tenacity(v) $< l_m$, by Theorem 2, $\text{base}^l(u)$ is BFS honest on p, giving the third fact. □

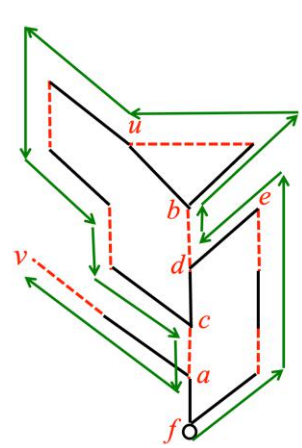
Example 21. In the graph of Figure 23, let p be the even level(v) path; it contains vertices u and w, which are both not BFS honest on p. The iterated bases of w are b, b', and f, and all three lie on p. Of these, b' and f are BFS honest on p, and b is not BFS honest on p; furthermore, each iterated base is BFS honest w.r.t. the next higher base, as proved in Theorem 6. Additionally, w is BFS honest w.r.t. b', w is not BFS honest w.r.t. f, and b is not BFS honest w.r.t. f on p. The iterated bases of u are b' and f. Both lie on p, both are BFS honest on p, and b' is BFS honest on p w.r.t. f.

In the graph of Figure 26, the even level(v) path, p, is indicated. The length of p is 16, and it contains vertex u. The iterated bases of u are b, c, and f, and all three lie on p. Of these, c and f are BFS honest on p, and b is not BFS honest on p. Furthermore, each iterated base is BFS honest w.r.t. the next higher base; however, u is not BFS honest w.r.t. c, u is not BFS honest w.r.t. f, and b is not BFS honest w.r.t. f on p.

9. Proof of Correctness and a Postmortem

As stated in Section 1.1, one of the main ideas behind the MV algorithm is precise synchronization of events; this is described and proved in Section 9.1. Section 9.3 proves that the MV algorithm correctly executes a phase and also establishes the running time of the algorithm. Finally, Section 9.4 provides a postmortem by raising and answering

Figure 26. (Color online) The even-level(v) path, p, is indicated. Vertices b and u are not BFS honest on p, even though b occurs before u on p.



the following question. "Why is it essential to formalize such an elaborate purely graph-theoretic structure for proving correctness of the MV algorithm?"

9.1. Synchronization of Events

Theorem 7. *Let t be an odd number with* $t_m \le t \le l_m$ *(i.e., t is either an eligible tenacity or* $t = l_m$*). The following hold.*

1. *Algorithm 1 finds* $Br(t)$*, the set of bridges of tenacity t, by the end of execution of procedure MIN at search level i, where* $t = 2i + 1$.
2. *For each vertex v such that* tenacity$(v) = t$*, Algorithm 1 assigns min level*(v) *and max level*(v) *correctly.*

Proof. We will show by strong induction on i, for $i = 0$ to $(l_m - 1)/2$, that at search level i, Algorithm 1 will accomplish the following tasks.

Task 1. Procedure MIN assigns a min level of $i+1$ to exactly the set of vertices having this min level. It also identifies all props that assign a min level of $i+1$.

Task 2. By the end of the 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 max levels to all vertices having tenacity $2i+1$.

This will establish both statements of the theorem.

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

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

We next prove that every vertex v with min level$(v) = i+1$ will be assigned its min level in this search level and that every prop that assigns a min level of $i+1$ will be classified as a prop. Let min level$(v) = i+1$, let p be a min level(v) path, and let (u, v) be the last edge on p. Clearly, (u, v) is a prop, and every prop that assigns a min level of $i+1$ is of this type. Now, u must be BFS honest on p. If not, then v must occur on a shorter path to u, contradicting min level$(v) < i+1$. If $|p[f \text{ to } u]| = i = $ max level(u), then tenacity$(u) < 2i+1$. Otherwise, $|p[f \text{ to } u]| = i = $ min level(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 min level of v is set to either ∞ or $i+1$.[23] In either case, v will be assigned a min level 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 1, tenacity$(u) = $ tenacity$(v) = $ tenacity(u,v), and u and v are both inner. Therefore, odd level$(u) = $ odd level$(v) = i$. Hence, during search level i, MIN will determine that (u, v) is a bridge, will determine 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$(u,v) = 2i+1$. By Lemma 3, if tenacity$(v) = $ tenacity$(u,v) = 2i+1$, then v is an outer vertex, and even level$(v) \le i$. Therefore, the algorithm has already determined even level(v). On the other hand, if tenacity$(v) < 2i+1$, then both its levels were determined by the end of the previous search level. By Lemma 3, these are the only two cases.

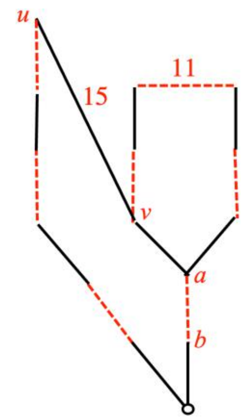
Therefore, in both cases, tenacity(u,v) will be ascertained by the end of execution of procedure MIN at search level i, and $Br(2i+1)$ will be the set of all bridges of tenacity $2i+1$.

Task 3. Let tenacity$(v) = t$. By statement 1 of Theorem 3 or by Theorem 4, depending on whether t is an eligible tenacity or $t = l_m$, v lies in the support of a bridge of tenacity $2i+1$, and by task 2 of the proof of Theorem 7, this bridge is in $Br(2i+1)$ at the start of MAX in search level i. Therefore, DDFS will ascertain max level(v) in this search level. □

Example 22. This example gives an insight into the idea of synchronizing of events. In Figure 27, the algorithm determines that (u, v) is a bridge of tenacity 15 at search level 6. However, according to Algorithm 1, DDFS should be performed on (u, v) at search level 7. The question arises as follows: "Why wait until search level 7; why not perform DDFS on (u, v) when procedure MAX is run at search level 6?" In the graph of Figure 27, no mistakes will be made, provided that the algorithm assigns tenacities of 15 vertices to a and b (i.e., the same as the tenacity of bridge (u, v)).

However, in the graph of Figure 28, when DDFS is performed on the bridge of tenacity 13 at search level 6, a and b will be assigned tenacities of 13; these are their correct tenacities in the graph of Figure 28. Next, assume

Figure 27. (Color online) Can DDFS be performed on bridge (u, v) at search level 6?



that bridge (u, v) is also processed at search level 6, and assume that it is processed before the bridge of tenacity 13; this is consistent with the arbitrary order in which bridges are processed. If so, a and b will be wrongly assigned tenacities of 15.

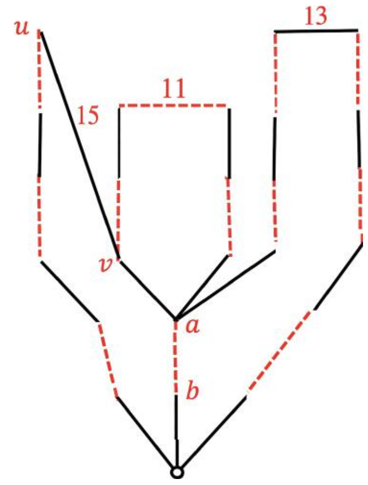
The synchronization of events imposed by the MV algorithm helps avoid such errors. Even though the tenacity of bridge (u, v) is determined at search level 6, DDFS on it is performed at search level 7, consistent with its tenacity. As a result, when the bridge of tenacity 13 is processed at search level 6, a and b are assigned their correct tenacities (i.e., 13). When processing bridge (u, v), DDFS will skip over the blossom of tenacity 13 and will not encounter a and b.

Remark 8. The proper synchronization of events, described here, is central to guaranteeing that the MV algorithm can accomplish its task in linear time. In the absence of synchronization, if DDFS is executed for a bridge of tenacity t and it finds a vertex v, the only guarantee is that tenacity$(v) \leq t$, and determining its exact tenacity would be a time-consuming task. However, in the presence of synchronization, tenacity$(v) < t$ is not possible.[24] Therefore, in the presence of synchronization, if DDFS finds a vertex v, it is sure that tenacity$(v) = t$.

9.2. Relationship Between Graph-Theoretic and Algorithmic Notions

As stated in Section 5.2, the notions of petal and bud are intimately related to the notions of blossom and base; whereas the former is an algorithmic notion, the latter is graph theoretic. This relationship is formally established in Lemma 22. At the end of search level i, once MAX is done processing all bridges of tenacity $t = 2i + 1$, all blossoms of tenacity t can be identified via this lemma; its proof is straightforward and is omitted.

Figure 28. (Color online) If so, vertices a and b will get wrong tenacities.



Lemma 22. *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* $T_{b,t}$ *defined in Definition* 31 *is*

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

Furthermore, the 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.

9.3. Execution of a Phase and Proof of Running Time

The proofs given in Section 9.1 show that the MV algorithm correctly finds one minimum-length augmenting path in the given graph with an initial matching. Lemma 24 shows that it correctly finds a maximal set of such paths as well; it critically uses Lemma 23. Finally, Theorem 8 concludes with a proof of the running time.

Lemma 23. *For a blossom* $\mathcal{B}_{b,t}$, *if* b *is removed, then procedure RECURSIVE REMOVE will remove all vertices of this blossom.*

Proof. We will prove the lemma by induction on the tenacity t of the blossom. The basis follows easily because for each $v \in \mathcal{B}_{b,t_m}$, $b = \text{pred}^*_m(v)$, and therefore, RECURSIVE REMOVE will remove the vertices of this blossom in order of increasing min level.

Next, assume that the statement is true for all blossoms of tenacity $t-2$, where $t_m < t < l_m$. Consider a blossom $\mathcal{B}_{b,t}$. For each vertex, $v \in T_{b,t}$, $b = \text{pred}^*_t(v)$; see Definition 31. Moreover, the base of each blossom of tenacity $t-2$ nested in $\mathcal{B}_{b,t}$ is a vertex from $\{b\} \cup T_{b,t}$. Once b is removed, the vertices of $T_{b,t}$ will be removed in order of increasing min levels together with nested blossoms of tenacity $t-2$; the latter follows by the induction hypothesis. Hence, $\mathcal{B}_{b,t}$ will be fully removed. □

Lemma 24. *The procedures given in Section* 5.4 *will find a maximal set of disjoint minimum-length augmenting paths in* G.

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

The crux of the matter is how do we guarantee that the remaining graph will "look like" a graph in which the first path is found (i.e., it has all the pointers and properties needed). The theorems of Section 8.4 guarantee that if a vertex $v \in p$, then each of its iterated bases is on p and will be removed. By Lemma 23, once the base of a blossom is removed, RECURSIVE REMOVE will remove all its vertices, and therefore, there will be no "half-eaten" blossoms in the graph when the next path needs to be found. The lemma follows. □

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

Proof. In a phase, each of the procedures MIN, MAX, finding augmenting paths, and RECURSIVE REMOVE examine each edge a constant number of times. The only other operation performed by the algorithm is that of computing bud* during DDFS. This can be implemented on the pointer model using the set union algorithm of Tarjan [31], which will take $O(m \cdot \alpha(m,n))$ time per phase. Alternatively, it can be implemented on the RAM model using the linear time algorithm for a special case of set union (Gabow and Tarjan [12]); this will take $O(m)$ time per phase. Because $O(\sqrt{n})$ phases suffice for finding a maximum matching (Hopcroft and Karp [15] and Karzanov [19]), the theorem follows. □

Remark 9. A question arising from Theorem 8 is whether there is a linear time implementation of bud* in the pointer model. Micali and Vazirani [24] had claimed, without proof, that path compression by itself suffices to achieve this. They stated that because of the special structure of blossoms, a charging argument could be given that assigns a constant cost to each edge. This claim is left as an open problem. See a related open problem in Section 10.

Finally, we note that because the MV algorithm consists of simple operations involving no hidden constants, especially if implemented using the set union algorithm (Tarjan [31]), it is not only fast in theory but also, fast in practice. Over the years, several researchers have produced very fast implementations.

9.4. The Role of Graph-Theoretic Structural Properties in the MV Algorithm

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

Assume that min level$(v) = i + 1$ and max level$(v) = j + 1$ so that tenacity$(v) = i + j + 2 = t$, where t is an eligible tenacity or $t = l_m$. It is easy to see that there must be a neighbor, say u, of v such that even level$(u) = i$ or odd level$(u) = i$, depending on the parity of i. Therefore, one step of breadth first search, while searching from u, will lead to assigning v its correct min level. We will say that u is the *agent that assigns v its min level*.

In contrast, none of the neighbors of v may have j as one of its levels. For instance, vertex b in the graph of Figure 5 has max level(b) = odd level$(b) = 11$. Observe that even level(a) = even level$(c) = 8$ and even level$(u) = 12$ (i.e., none of the neighbors of b have an even level of 10). The following questions arise.

1. What is the agent that assigns v its max level, and does it exist for each "relevant" vertex v?
2. Can this agent be found for each relevant vertex v?
3. How does this agent help assign v its max level?

This paper provides very precise answers to all these questions. The agent that assigns v its max level is a *bridge* whose tenacity equals tenacity(v). Statement 1 of Theorem 3 and Theorem 4 prove that every max level(v) path contains such a bridge. Thus, in a sense, this was the central structural fact that needed to be proved.

"Relevant" vertices are those whose tenacity is eligible or is l_m. Theorem 3 proves that for each such vertex v, it is in the support of a bridge whose tenacity is tenacity(v), and Theorem 7 proves that such a bridge will be found "well in time." The answer to the third question is execute the procedure of DDFS on the endpoints of this bridge.

These structural properties suffice for finding the first minimum-length augmenting path. However, after its removal, could it be that the graph is left with "half-eaten blossoms," which simply do not support finding the next path via the same process as the first one, even though a path exists? Lemma 24, which is based on additional properties established in Section 8.4, shows that subsequent paths do not use "half-eaten blossoms." Indeed, the procedure RECURSIVE REMOVE will remove all of them. Hence, subsequent paths are found in the same way as the first one.

10. Discussion

Matching is one of the "big three" problems in combinatorial optimization along with linear programming and flow. Spectacular progress in this field has led to improved running times of the last two as well as other fundamental problems in the last three decades (e.g., see Schrijver [29] as well as recent papers). Recent improvements in flow algorithms have also led to improved running times for the maximum matching problem in bipartite graphs: $O(m^{10/7})$ (Madry [23]), $O(m^{4/3})$ (Liu and Sidford [21]), and $\tilde{O}(m + n^{1.5})$ (van den Brand et al. [33]). Very recently, an almost linear time $O(m^{1+o(1)})$ algorithm was obtained (Chen et al. [3]); however, it is currently unclear if this theoretical running time translates into very fast implementations for use in practice. A concerted effort has been made to improve the running time for general graph matching as well, but so far, the MV algorithm has stood the test of time.

As is well known, the general graph matching problem has numerous applications. We single out a particularly interesting and important one—to the kidney exchange matching market, which was first studied in Roth et al. [28]; see also the scientific background (Economic Sciences Prize Committee of the Royal Swedish Academy of Sciences [6]) for the 2012 Nobel Prize in Economics awarded to Alvin Roth and Lloyd Shapley.

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

Next, assume that a number of incompatible pairs are specified, $(A_1, B_1), \ldots, (A_n, B_n)$, and for every two pairs, we know whether they are consistent. Then, the rudimentary problem[25] of finding the maximum number of disjoint consistent pairs reduces to maximum matching as follows. Let $G = (V, E)$ be a graph with $V = \{v_1, \ldots, v_n\}$, where v_i represents the pair (A_i, B_i) and $(v_i, v_j) \in E$ if and only if the two pairs (A_i, B_i) and (A_j, B_j) are consistent. Clearly, a maximum matching in G will yield the answer.

In addition to the profound influence that matching has had on the theory of algorithms (see Section 1), it has also played a significant role in game theory and economics; we describe this next. The matching game forms one of the cornerstones of cooperative game theory (e.g., see Moulin [25]), and its special case, the assignment game, forms a paradigmatic setting for studying the quintessential solution concept of the core[26] of a game (Shapley and Shubik [30]).

Several matching-based problems form essential ingredients in the area of online and matching-based market design. Besides the application of general graph matching to kidney exchange (described here), two other variants of matching, namely stable matching (Gale and Shapley [13]) and online bipartite matching (Karp et al. [18]), lie at the core of this area (e.g., see Echenique et al. [5]). The seminal 1962 paper of Gale and Shapley [13], on stable matching initiated this area. With the advent of the internet and mobile computing, it underwent a resurgence, leading to the launching of highly impactful new matching markets (e.g., the digital ads marketplaces, Uber, Lyft, Airbnb, Upwork, and Match.com). The online bipartite matching problem has emerged as a paradigm for this area because of the online decision-making feature of these marketplaces.

Finally, here is an open question. Is there a linear time implementation of a phase of MV on the pointer model? Remark 9 states the approach mentioned in Micali and Vazirani [24] for addressing this question. The following approach may be easier. Is it the case that the MV algorithm, implemented using the set union algorithm of Tarjan [31], actually runs in linear time per phase? The running time of $O(m\sqrt{n}\cdot\alpha(m,n))$, reported in Theorem 8, is based on imprecise assumptions that the number of calls to this data structure is $O(m)$ and that the number of elements manipulated is $O(n)$. More precise bounds on these two quantities are the number of calls to DDFS (i.e., the number of bridges of eligible tenacity) and the number of buds of petals,[27] respectively. Using these facts and structural properties established in this paper, can one show that the total time devoted to set union in a phase is bounded by $O(m)$?

Acknowledgments
The author thanks Rohith Gangam and Ruta Mehta for diligently helping to verify this proof and Silvio Micali for embarking on a year-long journey that led to the discovery of this algorithm. The author also thanks the two referees for providing in-depth critiques of this paper, covering all aspects of the exposition and the proof.

Endnotes

1 Section 10 puts this fact in context by comparing with recent improvements in running times of other combinatorial optimization problems.

2 For ease of comparison of the pseudocode given in Micali and Vazirani [24] with the description of the algorithm presented in the current paper, we note that a key algorithmic structure, called "petal" in the current paper, was called "blossom" in Micali and Vazirani [24].

3 See Section 1.2 for a definition.

4 Recall that the problem of finding long paths, such as Hamiltonian paths, is NP hard.

5 See Section 10 for the role played by matching in game theory and economics.

6 In a sense, the current proof owes its existence to a chance event: our attempt at simplifying the exposition of DDFS.

7 This viewpoint is not unreasonable (e.g., see Section 9.4).

8 As is standard, n denotes the number of vertices and m denotes the number of edges in the given graph.

9 Observe that either of the trees could have arrived at v first. This happens despite our convention that C_r keeps ahead of C_g; the reason is that C_g may have used a long edge to arrive at v before C_r. In Figure 2, this happens when the two trees meet at vertex c.

10 Observe that if $M=\emptyset$, then any edge is an augmenting path of length 1.

11 This bridge is very unusual; on the one hand, neither endpoint of a bridge is a predecessor of the other, and on the other hand, in the case of this bridge, one of its endpoints $u=\text{pred}^*v$, as per Definition 11.

12 Not every bridge will lead to the formation of a petal (e.g., see Example 10).

13 To avoid cluttering up Figure 10, only two vertices are pointing to the petal node.

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

15 In the language of Definition 23, p_2^{-1} is an even-level$(b;v)$ path.

16 Observe that $\text{bud}^*(w)$ right after DDFS is performed on bridge (c, d) is b.

17 Again, in the language of Definition 23, this is an even-level$(a;w)$ path.

18 For this notion, see Definition 7.

19 For clarity, in subsequent cases, we will describe the path constructed in plain English as follows. Follow q to b, then s to y, then the red path to v, where s is an even-level$(b;y)$ path.

20 See Definition 18.

21 Note that $\mathcal{B}_{b,t}$, namely a blossom of tenacity t and base b, is defined in Definition 31 after proving statement 2 of Theorem 3.

22 Observe that the conclusion of Remark 7 does not apply here.

23 The latter case happens if even level$(u)=i$ and v has been reached earlier in this search level while searching along an edge (u',v) with even level$(u')=i$.

24 If tenacity$(v)<t$, v would have already been assigned to a petal, and the current DDFS would skip it.

[25] Solution concepts proposed by economists are more involved, taking into account issues such as incentive compatibility, more general exchanges, etc.; see Echenique et al. [5].

[26] The recent paper by Vazirani [35] rectifies the fact that the core of the general graph matching game is empty via the notion of an approximate core.

[27] Observe that it suffices to maintain the set union structure over the set of buds of petals only. If during DDFS, a vertex v, in a previously constructed petal, is encountered, then the algorithm will follow pointers from v to its petal node to bud(v). Next, the set union algorithm will find bud*(v). The work done to go from v to bud(v) is charged to the edge that led to v, and the rest of the work is charged to the set union algorithm.

References

[1] Ahuja RK, Magnanti TL, Orlin JB (1995) *Network Flows: Theory, Algorithms and Applications* (Prentice Hall, Hoboken, NJ).
[2] Berge C (1957) Two theorems in graph theory. *Proc. Natl. Acad. Sci. USA* 43(9):842–844.
[3] Chen L, Kyng R, Liu YP, Peng R, Gutenberg MP, Sachdeva S (2022) Maximum flow and minimum-cost flow in almost-linear time. *2022 IEEE 63rd Annual Sympos. Foundations Comput. Sci. (FOCS)* (IEEE, Piscataway, NJ), 612–623.
[4] Dinitz EA (1970) Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Doklady* 11:1277–1280.
[5] Echenique F, Immorlica N, Vazirani VV, eds. (2023) *Online and Matching-Based Market Design* (Cambridge University Press, Cambridge, UK).
[6] Economic Sciences Prize Committee of the Royal Swedish Academy of Sciences (2012) Stable allocations and the practice of market design. Accessed June 24, 2020, https://www.nobelprize.org/uploads/2018/06/advanced-economicsciences2012.pdf.
[7] Edmonds J (1965) Maximum matching and a polyhedron with 0,1-vertices. *J. Res. National Bureau Standards B* 69B:125–130.
[8] Edmonds J (1965) Paths, trees, and flowers. *Canadian J. Math.* 17(3):449–467.
[9] Even S, Kariv O (1975) An $O(n^{2.5})$ algorithm for maximum matching in general graphs. *16th Annual Sympos. Foundations Comput. Sci.* (IEEE Computer Society, Piscataway, NJ), 100–112.
[10] Gabow HN (2017) The weighted matching approach to maximum cardinality matching. *Fundamenta Informaticae* 154(1–4):109–130.
[11] Gabow HN, Tarjan RE (1985) A linear-time algorithm for a special case of disjoint set union. *J. Comput. System Sci.* 30:209–221.
[12] Gabow HN, Tarjan RE (1991) Faster scaling algorithms for general graph matching problems. *J. ACM* 38:815–853.
[13] Gale D, Shapley LS (1962) College admissions and the stability of marriage. *Amer. Math. Monthly* 69(1):9–15.
[14] Goldberg AV, Karzanov AV (2004) Maximum skew-symmetric flows and matchings. *Math. Programming Ser. A* 100:537–568.
[15] Hopcroft J, Karp RM (1973) An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.* 2:225–231.
[16] Jerrum MR, Sinclair A (1989) Approximating the permanent. *SIAM J. Comput.* 18:1149–1178.
[17] Jerrum MR, Valiant LG, Vazirani VV (1986) Random generation of combinatorial structures from a uniform distribution. *Theoret. Comput. Sci.* 43:169–188.
[18] Karp RM, Vazirani UV, Vazirani VV (1990) An optimal algorithm for on-line bipartite matching. *Proc. Twenty-Second Annual ACM Sympos. Theory Comput.* (Association for Computing Machinery, New York), 352–358.
[19] Karzanov AV (1973) An exact estimate of an algorithm for finding a maximum flow, applied to the problem on representatives. *Problems Cybernetics Seminar Combinatorial Math.* (Sovetskoe Radio, Moscow), 66–70.
[20] Kuhn HW (1955) The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.* 2:83–97.
[21] Liu YP, Sidford A (2020) Faster energy maximization for faster maximum flow. *Proc. 52nd Annual ACM SIGACT Sympos. Theory Comput.* (Association for Computing Machinery, New York), 803–814.
[22] Lovász L, Plummer MD (1986) *Matching Theory* (North-Holland, Amsterdam).
[23] Madry A (2013) Navigating central path with electrical flows: From flows to matchings, and back. *2013 IEEE 54th Annual Sympos. Foundations Comput. Sci. (FOCS)* (IEEE, Piscataway, NJ), 253–262.
[24] Micali S, Vazirani VV (1980) An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. *21st Annual Sympos. Foundations Comput. Sci.* (IEEE, Piscataway, NJ), 17–27.
[25] Moulin H (2014) *Cooperative Microeconomics: A Game-Theoretic Introduction*, vol. 313 (Princeton University Press, Princeton, NJ).
[26] Mucha M, Sankowski P (2004) Maximum matchings via Gaussian elimination. *45th Annual IEEE Sympos. Foundations Comput. Sci.* (IEEE, Piscataway, NJ), 248–255.
[27] Mulmuley K, Vazirani UV, Vazirani VV (1987) Matching is as easy as matrix inversion. *Combinatorica* 7(1):105–113.
[28] Roth AE, Sönmez T, Ünver MU (2005) Pairwise kidney exchange. *J. Econom. Theory* 125(2):151–188.
[29] Schrijver A (1986) *Theory of Linear and Integer Programming* (John Wiley & Sons, New York).
[30] Shapley LS, Shubik M (1971) The assignment game I: The core. *Internat. J. Game Theory* 1(1):111–130.
[31] Tarjan RE (1975) Efficiency of a good but not linear set union algorithm. *J. ACM* 22:215–225.
[32] Valiant LG (1979) The complexity of computing the permanent. *Theoret. Comput. Sci.* 8:189–201.
[33] van den Brand J, Lee Y-T, Nanongkai D, Peng R, Saranurak T, Sidford A, Song Z, Wang D (2020) Bipartite matching in nearly-linear time on moderately dense graphs. *2020 IEEE 61st Annual Sympos. Foundations Comput. Sci. (FOCS)* (IEEE, Piscataway, NJ), 919–930.
[34] Vazirani VV (1994) 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.
[35] Vazirani VV (2022) The general graph matching game: Approximate core. *Games Econom. Behav.* 132:478–486.
[36] Wikipedia (2023) Harold W. Kuhn. Accessed August 7, 2018, https://en.wikipedia.org/wiki/Harold_W._Kuhn.
[37] Wikipedia (2023) Isolation lemma. Accessed August 7, 2018, https://en.wikipedia.org/wiki/Isolation_lemma.

CORRECTION

In this article, "A Theory of Alternating Paths and Blossoms from the Perspective of Minimum Length" by Vijay V. Vazirani (*Mathematics of Operations Research*, vol. 49, no. 3, pp. 2009–2047), the last sentence of the fourth paragraph of Section 1.2 on page 2012 has been corrected.