COMBINATORICA
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA **14** (1) (1994) 71–109

A THEORY OF ALTERNATING PATHS AND BLOSSOMS FOR PROVING CORRECTNESS OF THE $O(\sqrt{V}E)$ GENERAL GRAPH MAXIMUM MATCHING ALGORITHM

VIJAY V. VAZIRANI[1]

Received December 30, 1989
Revised June 15, 1993

1. Introduction

Finding a maximum matching in a graph is a classical problem in the study of algorithms. In this paper we present new algorithmically relevant combinatorial structure of matchings. This structure yields the first proof of correctness of the general graph matching algorithm of Micali and Vazirani [14]; this is currently the most efficient known matching algorithm.

Berge's theorem [2], which says that matching M in graph G is a maximum matching if and only if there are no augmenting paths w.r.t. it, gives an iterative schema for finding a maximum matching in G, i.e. successively find augmenting paths. Finding augmenting paths is fairly easy in bipartite graphs; however, not so in general graphs (see [13] for a detailed history of the problem). The first polynomial time algorithm $(O(|V|^4))$ for general graph matching was given by Edmonds [4]. In this paper, Edmonds introduced the notion of *blossom* (an odd length alternating cycle), and showed that by "shrinking" blossoms, one can find augmenting path efficiently. In this seminal paper, Edmonds also introduced the notion of a polynomial time algorithm.

Over the years, faster implementations of Edmonds' algorithm were given by several authors, including Whitzgall and Zahn [16], Balinski [1], Gabow [6], Lawler [12], and Kameda and Munro [10]. In 1972, Hopcroft and Karp [9] proposed finding augmenting paths in *phases*; in each phase a maximal set of disjoint minimum length augmenting paths is found. They showed that only $O(\sqrt{|V|})$ phases are needed, as opposed to $O(|V|)$ iterations in the previously-mentioned schema. They also presented an $O(|E|)$ implementation of a phase in bipartite graphs, thereby giving an $O(\sqrt{|V|}|E|)$ matching algorithm for such graphs, and left the open problem of

[1] Partially supported by an NSF PYI Grant with matching funds from AT& T Bell Labs at Cornell University

AMS subject classification codes (1991): 05 C 70, 05 C 85

obtaining an algorithm having the same efficiently for general graphs. Using the idea of phases, an $O(|V|^{2.5})$ algorithm for general graphs was given by Even and Kariv [5]. The algorithm of Micali and Vazirani achieves the above-stated $O(\sqrt{|V|}|E|)$ running time on a RAM, using the incremental-tree set union algorithm of Gabow and Tarjan [7].

The natural schema for finding minimum length alternating paths is an alternating breadth first search (BFS). As stated in Section 2, this leads to a simple algorithm for the bipartite case; however, there are fundamental difficulties in making this schema work for general graphs. In order to point out the key difficulty, let us first consider ordinary BFS starting at vertex s in graph G to find the *levels* of all vertices. A vertex v having level $i+1$ must have a neighbour, say u, having level i, and while searching out from u, BFS will assign v its correct level. We will say that vertex u is the *agent* that makes BFS find the level of vertex v. So, in ordinary BSF, the agent is local, i.e., it is one of the neighbours of the vertex.

In the case of minimum length alternating paths in general graphs, the situation is more complex — there is a need to define two levels for each vertex v, $minlevel(v)$ and $maxlevel(v)$, corresponding to shortest paths of the two parities. The agent for minlevel is a neighbour, just as in the case of ordinary BFS. However, not so for maxlevel — it could be the case that none of the neighbours of a vertex of maxlevel $i+1$ is of level i. Fortunately, it is possible to salvage the situation: it is possible to identify a different agent — a special edge on the path (a "bridge" of the "correct tenacity"). In the Micali–Vazirani algorithm, this edge triggers off a special search procedure called *double depth first search (DDFS)* that efficiently finds maxlevels of vertices. This agent is not local, and so we need to carefully synchronize events, and mark the graph properly in order to execute a phase in linear time.

For establishing correctness of the algorithm, we need to prove that every maxlevel path has this special edge — this is our main structural theorem. To prove this theorem, we need to identify the combinatorial structure which the algorithm uses as footholds — (Theorems 1 to 7). Central to this structure is a definition of blossoms from the perspective of minimum length alternating paths. However, the structure is very rich, to the extent that considerable preparation is needed before blossoms can even be defined. For this reason, we will give an overview of the structure in Section 2.

The algorithm contains two main ideas: the precise manner in which the various events are synchronized, and the graph searching procedure of double depth first search (DDFS). The correctness of DDFS and the synchronization are also established (in Theorem 8 and 9 respectively).

In the past few years, Gabow and Tarjan [8] and Blum [3] have given general graph maximum matching algorithms having the same running time as the Micali–Vazirani algorithm. It is interesting to observe that besides cardinality matching, for several other matching problems, such as weighted matching, finding a maximum matching in parallel, and approximately computing the number of perfect matchings in dense graphs, the known algorithms for general graphs require additional ideas but achieve the same efficiently as for bipartite graphs. Is this just a coincide, or is there an underlying reason for this?

2. Overview of structural results and the algorithm

Matching algorithms and their proofs of correctness tend to be considerably more involved for general graphs than for bipartite graphs; this is particularly true of the algorithm of Micali and Vazirani, and its present proof of correctness. Is this complexity unavoidable? We will attempt to convince the reader that the answer to this question is "yes" by first sketching the algorithm for the bipartite case and showing fundamental difficulties in making this schema work for the case of general graphs. Eventually, we will indicate how the rich combinatorial structure of minimum length alternating paths and blossoms helps deal with these difficulties, and we will also give an overview of the structural results.

Let us start by giving some standard definitions. Let $G(V,E)$ be a graph. A set $M \subseteq E$ is said to be a *matching* if every vertex of G has at most one edge of M incident at it. M is a *maximum matching* if it is a matching of largest possible cardinality in G. The following terms are defined w.r.t. a matching M in G: edges in M are said to be *matched*, and those in $E \backslash M$ are *unmatched*. A vertex is said to be *matched* if it has a matched edge incident on it, and *unmatched* otherwise; sometimes an unmatched vertex is also referred as a *free* vertex. If (u,v) is a matched edge, then we say that *u is the matched neighbour of v*. A simple path is said to be an *alternating path* if it consists alternately of matched and unmatched edges. An *augmenting path* is an alternating path that starts and ends at (distinct) unmatched vertices.

The significance of an augmenting path p is that it helps obtain a mathing of one larger cardinality than N, namely the matching $M \oplus p$, where $\oplus$ denotes symmetric difference.

As stated in the introduction, we will resort to finding augmenting paths in *phases*, as proposed by Hopcroft and Karp: Start with the empty matching. In each phase, find a maximal set of disjoint minimum length augmenting paths w.r.t. the current matching and augment the matching along these paths. If there are no augmenting paths w.r.t. the current matching, halt.

2.1. The bipartite case

Let $G(U,V,E)$ be a bipartite graph, and let M be a matching in it. Define the *level* of a vertex $x \in U \cup V$ to be the length of the shortest alternating path from an unmatched vertex in U to x. Notice that vertices in U have even levels and those in V have odd levels. Also, among the unmatched vertices in V, the one having the smallest level gives the length of a minimum length augmenting path w.r.t. M. So, let us consider the problem of finding the levels of all vertices. (It turns out this is the core of the problem. As shown below, a small modification to the procedure for finding levels helps find minimum length augmenting paths.) The natural schema for this is an alternating breadth first search (BFS): Assign level 0 to all the unmatched vertices in U, and initialize the search level, i, to 0. Then, iterate on i as follows: if i is even, for all unmatched edges incident at vertices having level i, consider their other endpoints; if the endpoint does not have a level assigned yet, assign it level $i+1$. If i is odd, consider the matched edges incident at vertices having level i, and assign their other endpoints level $i+1$ (notice that these endpoints will not have levels assigned yet.)

The procedure given above clearly works in $O(|E|)$ time, and a straightforward proof by induction on i shows its correctness. We would like to highlight here that the algorithm and the proof of correctness are based on the following deceptively-straightforward-seeming fact: Let p be an alternating path that gives vertex x its level. We shall say that p is a $level(x)$ path. Then, the levels of vertices on p are contiguous, i.e., starting with 0, the levels increase by 1. Another way to put it is that minimum length alternating paths in bipartite graphs are *breadth-first-search honest*: let y be the free vertex in U at which p starts, and let v be any vertex on p. Then the part of p from y to v is a $level(v)$ path.

Another point worth mentioning, though less important, is that while searching from a vertex $u \in U$ along unmatched edge (u,v), if $level(v)$ was already set i.e., $level(v) < level(u)$, we ignored this edge. It is easy to see that this edge is not on a $level(x)$ path for any vertex x.

Finally, let us show how the algorithm given above can be modified to actually find minimum length augmenting paths. Let us say that u is a *predecessor* of v if (u,v) is the last edge on some $level(v)$ path. The alternating BFS can be easily modified to leave at each vertex the list of its predecessors. When the search encounters a free vertex $f \in V$, it can use the predecessor information to find a minimum length augmenting path, p, ending at f: let f be the starting center of activity; at each step, pick an arbitrary predecessor of the current center of activity, and move to it, until a free vertex in U is encountered. Let p be the path so traced. To complete the phase, the algorithm removes p and all edges incident at vertices on p from the graph. In addition, it also keeps removing any vertex that has no predecessors left. Then, the next path found will clearly be disjoint from p. In this manner, a maximal set of disjoint minimum length augmenting paths is obtained.

2.2. Difficulties encountered with non-bipartite graphs

The first point to be noticed in non-bipartite graphs is that a free vertex f may have alternating paths of both parities to a vertex v, e.g., see *Fig. 1*. Moreover, paths of both parity may be useful; neither one can be ignored. For example, in *Fig. 1* either of the edges, (w,v) and (w,x) could potentially lead to an augmentation. Hence, we must find paths of both parities to w. This motivates the following definitions (we have indicated definitions that first appeared in [14]):

Definition [14]. W.r.t. matching M in graph $G(V,E)$ define:
$evenlevel(v)$: Length of the shortest even length alternating path from an unmatched vertex to v, ∞ if no such path exists.
$oddlevel(v)$: Length of the shortest odd length alternating path from an unmatched vertex to v; ∞ if no such path exists.
A path that gives v its evenlevel (oddlevel) will be called an *evenlevel*(v) (*oddlevel*(v)) *path*.

The levels of vertices are marked in *Fig. 1*. Notice that an unmatched vertex with the smallest oddlevel gives the length of the minimum length alternating path in the graph. Once again, we will first consider the problem of finding the even and

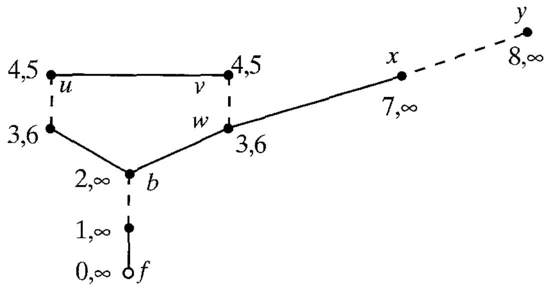


Fig. 1

odd levels of all vertices — this is the core of the problem in the non-bipartite case as well. Clearly, the simple alternating BFS will fail on non-bipartite graphs: if the search is not allowed to come around edge (u, v) then w does not get its evenlevel, and if is, then b gets an incorrect oddlevel.

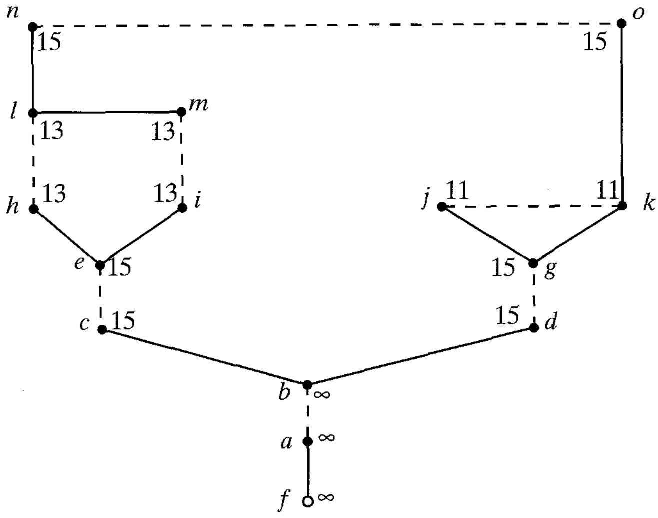


Fig. 2

How should we modify the simple alternating BFS? In order to seek an answer, let us see the difficulties we need to overcome. In *Fig. 2*, consider an *oddlevel*(e) path; *oddlevel*(e) is 11. Notice that levels are not contiguous on this path! Minimum length alternating paths in non-bipartite graphs are *not breadth-first-search honest.* In particular, *evenlevel*(h) is 8, and yet it occurs as the 10^{th} vertex on this path. This leads to the following basic question: In bipartite graphs, a vertex v of level $i+1$ was adjacent to a vertex u of level i. At the proper search level, u triggered off the appropriate mechanism that assigned v its level. In *Fig. 2*, what should trigger off the process that assigns e its oddlevel? i.e., what is the *agent*. In the case of bipartite graphs, the agent that assigns a vertex its level is local; it is one of the neighbours of the vertex.

Another point to be noticed in *Fig. 2* is that e occurs on every *evenlevel*(h) path; moreover it always occurs at an even distance on such a path. In order to find an *oddlevel*(e) path, we had to find an even length path to h that was longer than its evenlevel. This points out another difficulty: it is not sufficient to find alternating paths of minimum length to vertices; we may have to find longer and longer paths, in order to find the levels of other vertices. As such, this seems to require exponential time.

Finally, consider edge (u,v) in *Fig. 12*. It is clearly not useful for giving u its oddlevel, and we had remarked that such edges could be ignored in the bipartite case. However, in *Fig. 12*, edge (u,v) is critical for obtaining an odd path to w. We shall characterize such edges (called *anamolies*), and show how to deal with them.

2.3. Overcoming the difficulties using the structure of minimum alternating paths and blossoms

The reason we can get a linear time algorithm for finding the levels of vertices, despite the difficulties described above, is that minimum length paths have a rich combinatorial structure that can be exploited algorithmically. The central combinatorial notion is that of *blossoms* defined from the perspective of minimum length alternating paths.

Algorithmically, the key question is to identify the *agent* that triggers off the process that assigns a vertex its odd or even level. A first cut to this is to distinguish the two levels from a different criterion than parity:

Definition [14]. Define *maxlevel*(v) as the larger of *evenlevel*(v) and *oddlevel*(v), and *minlevel*(v) as the smaller one.

The agents for minlevel and maxlevel are different. The agent for minlevel is local, similar to the bipartite case, i.e., one of the neighbours of the vertex.

Observation. Suppose *minlevel*$(v) = i+1$ and $i+1$ is odd. Then there is a neighbour, u, of v such that *evenlevel*$(u) = i$ and the edge (u,v) is unmatched. Moreover, any *evenlevel*(u) path concatenated with edge (u,v) is an *oddlevel*(v) path. An analogous statement holds if $i+1$ is even.
The second part of the observation follows from the fact that v cannot occur on an *evenlevel*(u) path, since otherwise *minlevel*(v) would be <1.

For describing the agent for maxlevel, we need to introduce the central notion of *tenacity*.

Definition [14]. W.r.t. matching M in graph $G(V,E)$ define *tenacity*(v) = *evenlevel*(v) + *oddlevel*(v).
For edge (u,v),

$$tenacity(u,v) = \begin{cases} evenlevel(u) + evenlevel(v) + 1 & \text{if } (u,v) \text{ is unmatched} \\ oddlevel(u) + oddlevel(v) + 1 & \text{if } (u,v) \text{ is matched} \end{cases}$$

Remark. It is tempting to define *tenacity*(u) to be the length of a shortest alternating walk between two (necessarily distinct) unmatched vertices which contains

v. However, this is not true. For example, vertices v and b in *Fig. 1* will have the same tenacity under this definition.

We also need to distinguish between two types of edges.

Definition. Vertex u is said to be a *predecessor* of v if (u,v) is the last edge on some $minlevel(v)$ path. An edge (u,v) will be called a *prop* if either u is a predecessor of v, or v is a predecessor of u; it will be called a *bridge* otherwise.

The tenacities of vertices are marked in *Fig. 2*. In *Fig. 1*, edge (u,v) is a bridge of tenacity 9; the rest of the edges are props. In *Fig. 2*, only (l,m), (j,k) and (n,o) are bridges. Their tenacities are 13, 11 and 15 respectively.

Finally, we can state the agent that triggers a $maxlevel(v)$ computation — it is a bridge of tenacity $tenacity(v)$. It triggers off a process called *double depth first search (ddfs)* that finds the "blossom" containing v. How do we know that for each vertex v there is such an agent? Let us address this question by giving an intuitive description of the structural results.

2.4. The structural results

The central structural fact proven in this paper is that on any $maxlevel(v)$ path, there is a unique bridge of tenacity $tenacity(v)$. For example, in *Fig. 2*, *fabdgjkonlhe* is a $maxlevel(e)$ path, and (n,o) is the bridge of tenacity 15 on this path. In order prove this fact, we will first need to define blossoms and prove properties of minimum length alternating paths w.r.t. the blossoms.

Consider a vertex v having finite tenacity, and consider the tenacities of all vertices on any $evenlevel(v)$ or $oddlevel(v)$ path. On each path, pick the highest vertex (i.e., furthest from the free vertex) having tenacity $> tenacity(v)$. We will first show that the set picked is a singleton, i.e., there is a unique such vertex. This will be called the *base of* v, and will be denoted as $base(v)$. A base b is always *outer*, i.e., it satisfies $evenlevel(b) < oddlevel(b)$. The bases of various vertices in *Fig. 2* are:

$j,k:g$

$h,i,l,m:e$

$c,d,e,g,n,o:b$

The significance of base lies in the following fact: A path is an $evenlevel(v)$ ($oddlevel(v)$) path iff if consists of an $evenlevel(base(v))$ path concatenated with a minimum even (odd) length alternating path from $base(v)$ to v; the latter path is required to start with an unmatched edge. We shall refer to the latter path as q.

Suppose $b = base(v)$. Then, any $evenlevel(b)$ path in turn contains $base(b)$, and so on. This motivates:

Definition. Define $base^1(v) = base(v)$, and $base^{k+1}(v) = base(base^k(v))$. Also, we will say that $b = base^+(v)$ if $b = base^k(v)$ for some positive integer k. In *Fig. 2*, $base^2(l) = b$.

Blossoms are defined with two parameters: Base, b and tenacity, t, with $tenacity(b) > t$. The blossom with these parameters is the set of vertices v such that $tenacity(v) \leq t$ and $base^+(v) = b$. It is denoted by $B_{b,t}$. Notice that b is not part of this blossom. In *Fig. 2*,

$B_{g,11} = \{j,k\}$

$B_{e,13} = \{h, i, l, m\}$

$B_{b,15} = \{c, d, e, g, h, i, j, k, l, m, n, o\}$

In *Fig. 6*,

$B_{b,22} = \{a, c, u, d, w, w'\}$

$B_{b,15} = \{a, c, u, d, w, w', v, e\}$

Blossoms form a partial order by containment — two blossoms are either disjoint, or one is contained in the other. In the latter case, the first blossom is said to be *nested* in the second. In *Fig. 2*, $B_{g,11} \subseteq B_{b,15}$ and $B_{e,13} \subseteq B_{b,15}$, and in *Fig. 6*, $B_{b,11} \subseteq B_{b,15}$. Notice that in these figures vertices are drawn at heights proportional to their minlevels; this helps reveal the nesting of blossoms.

The significance of blossoms lies in the following: Let $B_{b,t}$ be the blossom with parameters $base(v)$ and $tenacity(v)$. Then, the path q, except for the first vertex b, lies entirely within the blossom $B_{b,t}$. If q were part of a $minlevel(v)$ path, then it would go "directly" from $base(v)$ to v, much the same way as the $evenlevel(base(v))$ path. On the other hand, if q were part of $minlevel(v)$ path, then it must come "around the blossom", i.e., using the bridge of tenacity $tenacity(v)$; furthermore, q consists of disjoint shortest paths from b to one endpoint of the bridge, and from the other endpoint to v (of course, these paths have to start and end with appropriate parity edges).

In general, path q may use blossoms nested within $B_{b,t}$. However, it cannot do so in an arbitrarily complicated manner: it can be shown that q enters and exits from a blossom nested in $B_{b,t}$ at most once, and either the entrance or the exit must be the base of the nested blossom. Furthermore, the part of q, say q', inside the nested blossom is similar to q, i.e., it is a minimum alternating path from the base of the nested blossom to a vertex in the blossom.

Let us illustrate this in *Fig. 2*. The $oddlevel(e)$ path consists of path fab (i.e., an $evenlevel(b)$ path), concatenated with path $bdjkonlhe$ (which is called q above). Path q consists of a minimum odd path from b to o, concatenated with bridge (o, n) concatenated with the reverse of a minimum path from e to n. Path q uses the nested blossoms of $B_{b,15}$; however, it does so in the restricted manner described above.

Let us see at a very high level how the structure described above helps overcome the difficulties. Consider the problem of finding an $oddlevel(e)$ path in *Fig. 2*. At the outset, the problem can be broken into two: finding an $evenlevel(base(e))$ path, and finding the path called q above. The first problem is clearly a smaller version of the original problem. For finding q, we first find disjoint paths from the two endpoints of the bridge (n, o) to e and b respectively, skipping over nested blossoms. Appropriate paths are found in the nested blossoms recursively, and concatenated with these two paths to yield q. On the other hand, if we had to find an $oddlevel(n)$ path, which is a minlevel path, then in order to get q, we first skip over nested blossoms, finding a direct path to b; recursive calls will patch this with appropriate paths through the nested blossoms.

2.5. High level description of algorithm

In this subsection, we will add some more details to the general algorithmic schema presented above. As stated in the introduction, the algorithm if Micali and Vazirani is built on two main ideas: synchronization and the graph searching procedure of DDFS.

Definition. For a bridge (u, v),

$$\begin{aligned} support(u,v) = \{w \mid tenacity(w) = tenacity(u,v), \text{ and} \\ \text{there is a } maxlevel(w) \text{ path containing } (u,v)\} \end{aligned}$$

In *Fig. 2*, $support(n,o) = B_{b,15} - (B_{e,13} \cup B_{g,11})$, i.e., the set obtained on deleting vertices of nested blossoms, which will have lower tenacity, from $B_{b,15}$. For now, it will be useful to take this to be the intuitive meaning of support; later we will refine the picture to deal with vertices that lie in the support of more than one bridge.

The algorithm iterates with parameter i, the search level, starting with $i =$ 0. At each search level, first MIN is executed, followed by MAX. The following is accomplished: At search level i:

MIN: Finds the minlevels of $\{v \in V \mid minlevel(v) = i+1\}$.
Before MAX starts, the following set of bridges would have been found:

$$S_{2i+1} = \{(u,v) \in E \mid (u,v) \text{ is a bridge of tenacity } 2i+1\}.$$

MAX: Finds the maxlevels of $\{v \in V \mid tenacity(v) = 2i+1\}$
These vertices are found as follows:
For each bridge (u, v) in S_{2i+1}, call DDFS to find $support(u, v)$.

Procedure MIN is straightforward. It essentially executes one step of alternating BFS, similar to the bipartite case: from vertices having level i it searches along the appropriate parity edges to find $i+1$ level vertices. MIN also leaves, at each vertex, the list of its predecessors.

On the other hand, DDFS is a more involved search schema. Suppose it is called with bridge (u,v) of tenacity $2i+1$, and let B be the corresponding blossom. DDFS will find the base, b, of this blossom. Consider the process of starting at one of the two endpoints of the bridge, and following predecessors; if the predecessor is in a nested blossom of B, then skip to the base of this blossom. Then, all such paths must go through b; in fact b will be the highest bottleneck for such paths. Furthermore, all vertices encountered on such paths, which are not in nested blossoms, must be of tenacity $2i+1$, and constitute $support(u, v)$.

DDFS needs that all lower tenacity blossoms, B', nested in B are appropriately marked, so it can efficiently reach from any vertex in B' to the base of B'. It grows, in a coordinated manner, two DFS trees rooted at u and v. These trees follow props, skipping over nested blossoms. Whenever the two trees meet, one of them tries to find an alternative path. If they don't succeed, the bottleneck, b, is found.

If instead of finding a bottleneck, the two DFS's reach distinct free vertices, f_1 and f_2, then there is a $2i+1$ length augmenting path containing edge (u, v). At this point, disjoint paths are found from u to f_1 and from v to f_2, skipping over nested blossoms; appropriate paths are found in the nested blossoms recursively. Then, as in the bipartite case, this path, together with all vertices and edges that cannot be on a disjoint path are removed, and the process is continued till a maximal set of such paths is found.

The informal description given in this section, together with Section 9 and 10, can give the reader a fairly detailed idea of the algorithm. However, in order to give a formal description of the manner in which DDFS marks and searches the graph, we will need some structural definitions which will be developed in Section 3 to 8.

The precise manner in which events are synchronized is critical to proving correctness of the algorithm, and synchronization is further explained in Section 12. Special edges, called anamolies, need to be identified for achieving this synchronization; this is described in Section 9.

3. Tenacity and breadth-first-search honesty

In Section 2 we gave examples to show that minimum length paths are not BFS honest in non-bipartite graphs. In this section, we will use the notion of tenacity to prove that these paths are BFS honest to some extent, namely, higher tenacity vertices on the path that are an even (odd) distance from f occur at the distance that defines their evenlevel (oddlevel). Theorem 1 deals with a $minlevel(v)$ path and Theorem 2 with a $maxlevel(v)$ path.

Lemma 1. *Let (u,v) be a matched edge. Then,* $evenlevel(v)=oddlevel(u)+1$ *and* $evenlevel(u)=oddlevel(v)+1$. *Also,* $tenacity(u)=tenacity(v)=tenacity(u,v)$.

Proof. Any alternating path containing both u and v must contain the matched edge (u,v). The first equality follows by observing that any $oddlevel(u)$ path cannot contain v, and therefore concatenating (u,v) to it yields an $evenlevel(v)$ path. Adding $oddlevel(v)$ to both sides of this equality we get $tenacity(v)=tenacity(u,v)$. The remaining equalities follows in a similar manner. ∎

Notation. If p and q are two paths, $p \circ q$ denotes their concatenation, and $|p|$ denotes the length of path p.

Definition. Let p be alternating path starting at an unmatched vertex f, and let u and v be two vertices occurring on p (in either order, u before v or v before u). Then $p[u \text{ to } v]$ denotes the part of p from u to v (both inclusive). Similarly $p[u \text{ to } v)$, $p(u \text{ to } v]$, $p(u \text{ to } v)$ denote the part of p from u to v, including u only, including v only, and excluding both u and v, respectively. We will say that *u occurs at the correct distance on p* if:

(i). if $|p[f \text{ to } u]|$ is even, then $|p[f \text{ to } u]|=evenlevel(u)$

(ii). if $|p[f \text{ to } u]|$ is odd, then $|p[f \text{ to } u]|=oddlevel(u)$.

Similarly, *u occurs at* $minlevel(u)$ *distance on p* if $|p[f \text{ to } u]|=minlevel(u)$, and *u occurs at* $maxlevel(u)$ *distance on p* if $|p[f \text{ to } u]|=maxlevel(u)$.

Vertex u is an *even(odd) vertex w.r.t. p* if $|p[f \text{ to } u]|$ is even (odd), and *v is higher than u on p* if $|p[f \text{ to } v]|>|p[f \text{ to } u]|$.

Theorem 1. *Let p be a minlevel(v) path and let u be a vertex on p such that tenacity$(u) \geq$ tenacity(v). Then u occurs at minlevel(u) distance on p.*

Proof. It is sufficient to prove the theorem for $|p|$ odd, because if $|p|$ is even, consider $p[f \text{ to } v)$ which is a minlevel path to the matched neighbour of v. We will prove that if u does not occur at the correct distance on p then *tenacity*$(u) <$ *tenacity*(v). By Lemma 1, w.l.o.g. we may assume that u is even w.r.t. p. Let q be an *evenlevel*(u) path. We will use q to show that *oddlevel*$(u) <$ *minlevel*(v). Since *evenlevel*$(v) <$ *minlevel*(v) also, this will show that *tenacity*$(u) <$ *tenacity*(v). The situation is illustrated in *Fig. 3*.

Path q must intersect $p(u \text{ to } v]$, because otherwise $q \circ p[u \text{ to } v]$ is a shorter odd path to v. Let v' be the matched neighbour of v, and let $p' = p \circ (v, v')$. Let w be the first vertex of q on $p'(u \text{ to } v']$. If w is odd w.r.t. p', then w must be odd w.r.t. q, and once again by splicing parts of q and p we can get a shorter odd path to v. Therefore, w is even w.r.t. p'. Now, $q[f \text{ to } w] \circ p'[w \text{ to } u]$ is an odd length alternating path from f to u, giving the desired inequality *oddlevel*$(u) < |q| + |p'[u \text{ to } w]| < |p|$.

Hence, if *tenacity*$(u) \geq$ *tenacity*(v), then u must occur at the correct distance, in fact *minlevel*(u) distance on p. ∎

Theorem 2. *Let p be a minlevel(v) path, and u be a vertex on p such that tenacity$(u) \geq$ tenacity(v). Then,*
(i) *if tenacity$(u) =$ tenacity(v), u occurs at the correct distance on p*
(ii) *if tenacity$(u) >$ tenacity(v), u occurs at minlevel(u) distance on p.*

Proof. By Lemma 1, w.l.o.g. we may assume that $|p|$ is even and that u is even w.r.t. p. (Notice that unlike in Theorem 1, if u occurs at the correct distance on p, it does not follow that u occurs at *minlevel*(u) distance. For this reason, we first consider the *minlevel*(v) path in order to establish a relationship between *minlevel*(u) and *minlevel*(v).)

Let q be a *minlevel*(v) path. If q does not intersect $p[u \text{ to } v)$, then *oddlevel*$(u) \leq |q| + |p[v \text{ to } u]|$. Clearly, *evenlevel*$(u) \leq |p[f \text{ to } u]|$. Therefore *tenacity*$(u) \leq$ *tenacity*(v). Since we have assumed that *tenacity*$(u) \geq$ *tenacity*(v), it follows that *tenacity*$(u) =$ *tenacity*(v) and u occurs at the correct distance on p.

Next suppose that q intersects $p[u \text{ to } v)$. Let u' be the matched neighbour of u, and let w be the first vertex of q on $p[u' \text{ to } v)$. Vertex w must be odd w.r.t. p. because otherwise there is a short odd length alternating path to u, showing *tenacity*$(u) <$ *tenacity*(v). Now, $|q[f \text{ to } w]| \geq |p[f \text{ to } w]|$, because otherwise by splicing q and p we can get a shorter even path to v. Therefore $|q| =$ *minlevel*$(v) >$ *evenlevel*(u). Since *tenacity*$(u) \geq$ *tenacity*(v), the minlevel of u must be its evenlevel. It remains to show that u must occur at the correct distance on p. The argument is the same as in Theorem 1: if not, then the *evenlevel*(u) path enables us to prove that *tenacity*$(u) <$ *tenacity*(v). ∎

The next lemma follows from the proof of Theorem 2.

Lemma 2. *Let p be a maxlevel(v) path and u be a vertex on p such that tenacity$(u) >$ tenacity(v). Then minlevel$(v) >$ minlevel(u).*

4. The base of a vertex

In this section we will use the notion of tenacity to define the base of a vertex; this will eventually be the base of the blossom in which the vertex lies.

Definition. Let v be a vertex of finite tenacity, and let p be an *evenlevel*(v) or an *oddlevel*(v) path. The *base of v w.r.t. p*, denoted by *base*(v,p), is the highest vertex on p having tenacity greater than that of v (there must be such a vertex since *tenacity*$(f)=\infty$).

The main result of this section is to show that the base of v is unique, i.e. independent of p: Towards this end we will first show that if p and q are *evenlevel*(v) and *oddlevel*(v) paths respectively, then *base*$(v,p)=$ *base*(v,q). Let q be the highest vertex of *tenacity* $>$ *tenacity*(v) occurring on both p and q. The proof involves showing that all vertices on $p(b$ to $v]$ are of *tenacity* $\leq$ *tenacity*(v). This is done by studying the intersections of p and q; for this we will define flowers, and show that the intersections of p and q form flowers.

Definition. Let p be an alternating path starting at f. An odd length alternating path that meets p only at its endpoints, and starts and ends with unmatched edges is called a *segment w.r.t. p*. A set of segments w.r.t. p satisfying certain conditions form a *flower w.r.t. p, F*. The base (tip) of F is the lowest (highest) vertex of p which is one of these segments. The flower F will be the union of these segments together with p[*base*(F) to *tip*(F)]. The vertices on this part of p are said to be *covered* by F. Following is a recursive definition of the conditions which the set of segments should satisfy:
(i) the set consists of a single segment that starts and ends at even vertices w.r.t. p.
(ii) let F' be a flower and q be a segment one of whose endpoints is covered by F', and the other endpoints is even w.r.t. p. Then the set of segments of F' together with q form a flower.
(iii) let F' and F'' be flowers, and q be a segment whose two endpoints are covered by F' and F'' respectively. Then the sets of segments of F' and F'' together with q form a flower.
The *length of flower F*, denoted by $|F|$, is defined to be the sum of the lengths of the segments of F and $|p$[*base*(F) to *tip*(F)]$|$. *Fig. 4* shows a flower formed by segments q_1, q_2, q_3 and q_4.

The above-stated recursive definition of flower yields the following lemma by a straightforward induction.

Lemma 3. *Let p be an alternating path starting at f. Let v be even (odd) w.r.t. path p and let F be a flower w.r.t. p which covers v. If $v \neq$ base(F), then there is an odd (even) length alternating path in F from base(F) to v of length $\leq |F|$.*

Definition. Let p be an alternating path starting at f, and let q be any other alternating path. Then, a part of q that starts and ends with unmatched edges and meets p only at its endpoints is called a *segment of q w.r.t. p*. A flower w.r.t. p formed by segment of q is said to be a *flower of q*. An alternating path whose first and last edges are matched edges on p (the rest of the path may also intersect p), is called a *section w.r.t. p*.

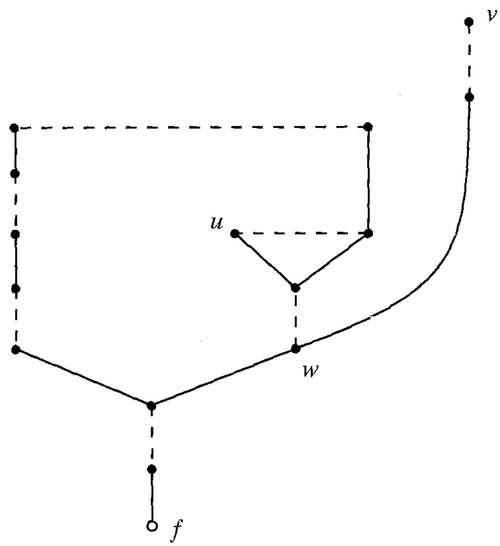


Fig. 3

Lemma 4. *Let p be an alternating path starting at f and q be a section w.r.t. p which starts and ends at vertices s and s' respectively on p. Then at least one of the following must hold:*
(i). *s is even w.r.t. p, or s is covered by a flower of q.*
(ii). *s' is even w.r.t. p, or s' is covered by a flower of q.*

Proof. The proof is by an induction on the number of segments in q. The assertion is obvious in case q consists of no segments. We prove the induction step below.

Suppose s and s' are both odd w.r.t. p. There are two cases. First, suppose there is a segment of q that starts at a vertex above s and ends at a vertex below s. Let q' be the first such segment, and let y and y' be its starting and ending vertices. Since s is not covered by a flower of $q[s \text{ to } y]$, by the induction hypothesis, y is either even w.r.t. p or covered by a flower of $q[s \text{ to } y]$. Now, if y' is even w.r.t. p, s is covered by a flower of $q[s \text{ to } y']$. So, assume that y' is odd w.r.t. p. If s' is covered by a flower of $q[y' \text{ to } s']$, we are done. Otherwise, by the induction hypothesis, y' is covered by a flower of $q[y' \text{ to } s']$. Now using a segment q', s is covered by a flower of q. The last case is illustrated in *Fig. 5*.

Next, suppose there is no such segment q'. Let z be the highest vertex of p on q. Since z is even w.r.t. p, $z \neq s'$. Now, $q[s' \text{ to } s]$ satisfies the first case, and by the proof given above, s' is covered by a flower of q (since clearly s is not). ∎

For the next lemmas, let v be a vertex of finite tenacity, and let p and q be *evenlevel*(v) and *oddlevel*(v) paths respectively. Consider vertices of *tenacity* $>$ *tenacity*(v) which occur on both p and q (f is such a vertex), and let b be the highest such vertex. (Recall that such vertices occur at their minlevel distance on both p and q.) We will first prove that $base(v,p) = base(v,q) = b$ in a simple setting: when there are no separators (see definition below).

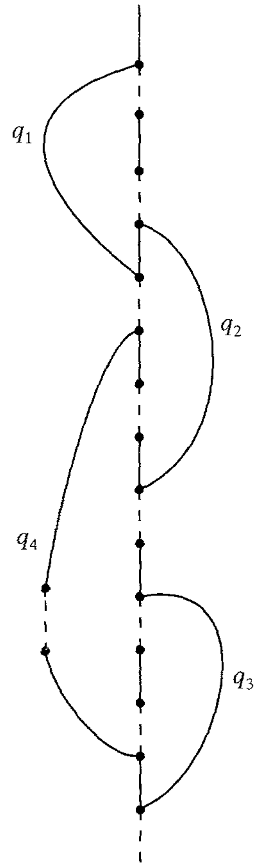


Fig. 4

Definition. We will say that matched edge (w, w') is a *separator w.r.t.* p and q if it occurs on both p and q, and is a cut edge for the subgraph formed by the edges and vertices in $p \cup q$. For example, in *Fig. 9(a)*, (s, s') is a separator w.r.t. *evenlevel*(v) and *oddlevel*(v) paths.

Remark. The matched edge incident at b is a separator w.r.t. p and q. This fact follows from Lemma 7; however, we do not need it for proving Theorem 3.

Lemma 5. *If there are no separators w.r.t. p and q on $p(b$ to $v]$ (and therefore also on $q(b$ to $v]$) then $base(v,p) = base(v,q) = b$.*

We will first need the following definitions:

Definition. Let (w, w') be a matched edge occurring on both $p(b$ to $v]$ and $q(b$ to $v]$, with w' even w.r.t. p. If before traversing (w, w'), q does not meet any vertex of p higher than w', then (w, w') is called a *frontier*. If w' is odd w.r.t. q, (w, w') is called a *backward frontier*. If (w, w') is not a separator w.r.t. p and q, and w'

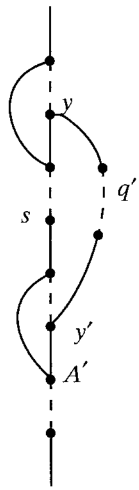


Fig. 5

is even w.r.t. q then (w, w') is called a *forward frontier*. Backward and forward frontiers are illustrated in *Fig. 6* and 7 respectively.

Proof. We will prove that every vertex on $p(b$ to $v]$ has $tenacity \leq tenacity(v)$; the proof for vertices on $q(b$ to $v]$ is similar. For this, it is sufficient to show that for vertex u on $p(b$ to $v]$ which is even w.r.t. p, there is an odd length alternating path from f to u of length $\leq tenacity(v) - |p[f$ to $v]|$.

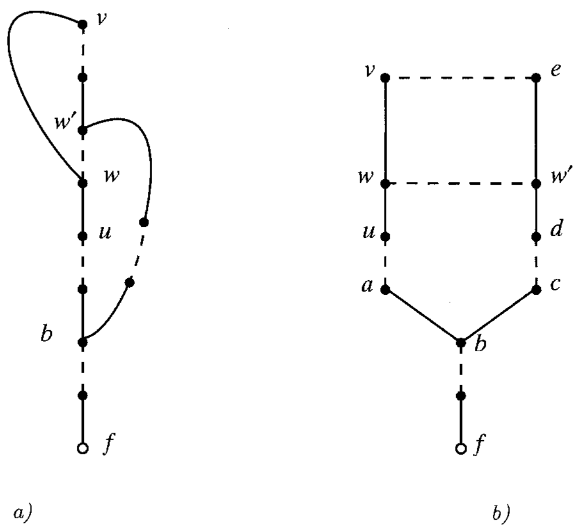


Fig. 6

Let matched edge (w,w') on p be the closest frontier to u such that $|p[f \text{ to } w']| \geq |p[f \text{ to } u]|$, where w' is even w.r.t. p. If (w,w') is a backward frontier, then $q[f \text{ to } w'] \circ p[w' \text{ to } u]$ is the required path (see *Fig. 6*). Next consider that (w,w') is a forward frontier. Clearly $|q[f \text{ to } w']| \geq |p[f \text{ to } w']|$, because otherwise by splicing p and q we can get a shorter even path to v.

We will first prove that w is covered by a flower of $q[w \text{ to } v]$. Consider the last segment of $q[w \text{ to } v]$ which starts below w (say at z) and ends above w. Now, z couldn't be even w.r.t. p because otherwise $p[f \text{ to } z] \circ q[z \text{ to } v]$ is a shorter odd path to v. If z is covered by a flower of $q[w \text{ to } z]$, then this flower must cover w also (because otherwise we can again get a shorter odd path to v using Lemma 3). If z is not covered by a flower of $q[w \text{ to } z]$, by lemma 4, w must be covered by a flower of $q[w \text{ to } z]$. Let b' be the base of this flower, and let r be an even length alternating path from b' to w in this flower.

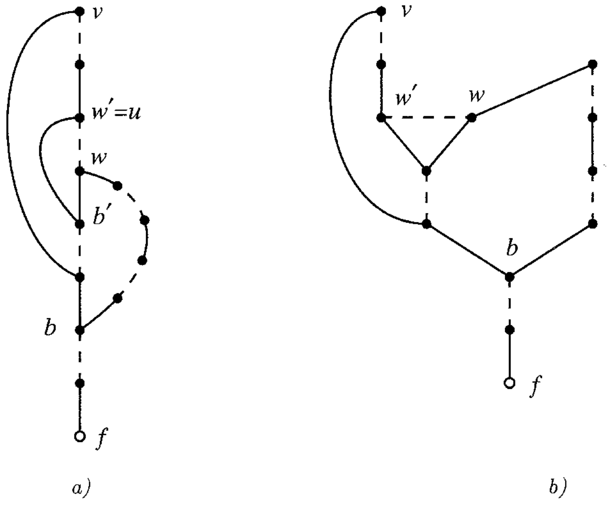


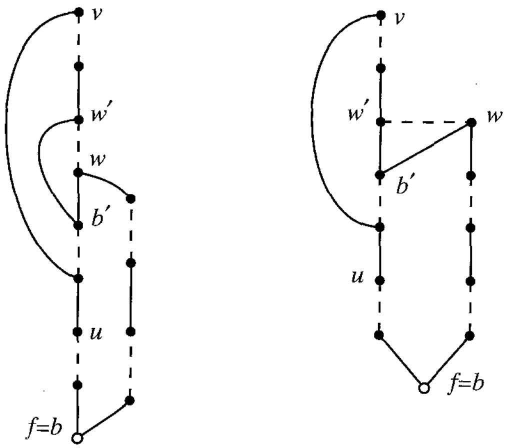
Fig. 7

Now, if u is above b', then $p[f \text{ to } b']$ concatenated with the odd length alternating path from b' to u through the flower gives the odd path to u. Otherwise, $q[f \text{ to } w] \circ r \circ p[b' \text{ to } u]$ is the required path. The first case is illustrated in *Fig. 7* and the second in *Fig. 8*. ∎

Lemma 6. $base(v,p) = base(v,q) = b$.

Proof. The no separators case is proved in Lemma 5. Let (s,s') be the lowest separator on $p(b \text{ to } v]$ and $q(b \text{ to } v]$, with s' even w.r.t. p and q. Clearly, $|p[f \text{ to } s]| = |q[f \text{ to } s]|$, and by the choice of b, $tenacity(s) \leq tenacity(v)$. Let (w,w') be the highest frontier on $p(b \text{ to } s)$, with w' even w.r.t. p. By the proof of Lemma 5, all vertices on $p(b \text{ to } w']$ have $tenacity \leq tenacity(v)$.

For dealing with vertices on $p(w' \text{ to } s)$, first consider the case that s occurs at the correct distance on p (e.g. see *Fig. 9(a)*). Let r be an $evenlevel(s)$ path which shares the most number of vertices with q. If r has no separators on $p(w' \text{ to } s)$,



a) b)

Fig. 8

then by the proof of Lemma 5, all vertices on this part of p have $tenacity \leq tenacity(v)$. Otherwise, let (x, x') be the highest separator, with x' even w.r.t. p. Once again, the proof of Lemma 5 takes care of vertices on $p(x'$ to $s)$. For the remaining vertices, there are some cases to be considered.

The canonical case is when $r[x'$ to $s]$ does not intersect $q[f$ to $s)$. Let u be an even vertex on $p(w'$ to $x']$. We will show that the odd path $p' = q[f$ to $s] \circ r[s$ to $x'] \circ p[x'$ to $u]$ has length $\leq tenacity(v) - |p[f$ to $u]|$, thereby bounding $tenacity(u)$.

Since s occurs at the correct distance on p, $|r[f$ to $s]| \leq tenacity(v) - |p[f$ to $s]|$. Also, $|r[f$ to $x']| \geq |p[f$ to $x']|$, because otherwise we can splice p and r to get a shorter even path to v. Therefore, $|r[x'$ to $s]| \leq tenacity(v) - |p[f$ to $s]| - |p[f$ to $x']|$. Substituting this in $|p'|$ and using $|p[f$ to $s]| = |g[f$ to $s]|$ gives bound.

Next suppose $r[x'$ to $s]$ intersects $q[f$ to $s]$ in vertex y first. If y is even w.r.t. q then by the same argument as above, $q[f$ to $y] \circ r[y$ to $x'] \circ p[x'$ to $u]$ is the required odd path to u. Finally, suppose y is odd w.r.t. q. Then, $|r[f$ to $y]| \geq |q[f$ to $y]|$. Now, $r[y$ to $s]$ must intersect $q[f$ to $y)$, because otherwise r violates the condition that it shares the most number of vertices with q. Let (z, z') be the lowest matched edge of $q[f$ to $y)$ traversed by $r[y$ to $s]$, with z' even w.r.t. q. If z' is even w.r.t. r, then we can get a shorter even path to s by splicing r and q. Otherwise, $q[f$ to $z'] \circ r[z'$ to $x'] \circ p[x'$ to $u]$ is the required odd path to u.

In case s does not occur at the correct distance on p (e.g. see *Fig. 9(b)*), let r be an $oddlevel(s)$ path. Now, r must intersect $p(s$ to $v]$ in an even vertex first. Let this vertex be h, and as before, let (x, x') be the highest separator of r on $p(w'$ to $s)$. Let $r' = r[x'$ to $h] \circ p[h$ to $s]$. Using r' in place of $r[x'$ to $s]$ in the above-stated cases yields the required odd path to u.

Finally, we remark that these arguments apply to vertices between any two consecutive separators on p; the vertices between the highest separator on p and v are dealt with using the proof of Lemma 5. ∎

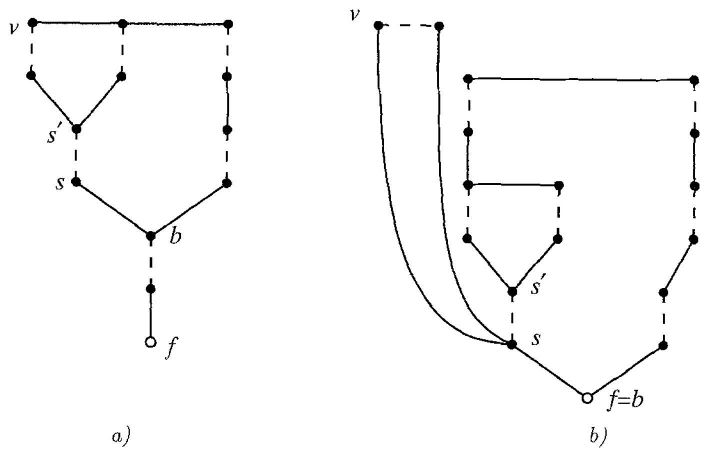


Fig. 9

Theorem 3. *Let v be a vertex of finite tenacity. Then its base is unique, i.e., the set $\{b \mid b = base(v,p)$ for some $evenlevel(v)$ or $oddlevel(v)$ path $p\}$ is a singleton.*

Proof. Let p and q be any $evenlevel(v)$ and $oddlevel(v)$ paths. By lemma 5, $base(v,p) = base(v,q) = b$ (say). Now, by fixing p and varying q over all $oddlevel(v)$ paths and then fixing q and varying p over all $evenlevel(v)$ paths we get required result. ∎

Definition. For a vertex v of finite tenacity define $base(v)$ to be its unique base. Say that a vertex v is *outer* if $evenlevel(v) < oddlevel(v)$, and *inner* if $oddlevel(v) < evenlevel(v)$.

Remark. 1) For a matched edge (u,v), $base(u) = base(v)$, by Lemma 1.
2) For a vertex v of finite tenacity, $base(v)$ is an outer vertex.

Definition. Let v be a vertex of finite tenacity. Define $base^1(v) = base(v)$. Furthermore, for $k \in Z^+$, if $base^k(v)$ is of finite tenacity, then define $base^{k+1}(v) = base(base^k(v))$.

Remark. Notice that $tenacity(base^{k+1}(v)) > tenacity(base^k(v))$, and $evenlevel(base^{k+1}(v)) < evenlevel(base^k(v))$.

Corollary 1. *Let v be a vertex such that $base^{k+1}(v)$ exists, for $k \in Z^+$, and let p be any $evenlevel(v)$ or $oddlevel(v)$ path. Then every vertex on $p(base^{k+1}(v)$ to $v]$ has $tenacity \le tenacity(base^k(v))$.*

5. The significance base

We will use the notion of base to define blossoms in the next section. The other significance of base is that a path is an $evenlevel(v)$ path iff it consists of an $evenlevel(base(v))$ path concatenated with a minimum even-alternating path from

$base(v)$ to v. Thus the two paths can be found *independently*. A similar statements holds for $oddlevel(v)$ paths.

Definition. An *even-alternating path (odd-alternating path)* from u to v, is an alternating path of even (odd) length, starting with an unmatched edge.

Lemma 7. *Let u be a vertex occurring on an $evenlevel(v)$ ($oddlevel(v)$) path p. Suppose u is even w.r.t. p and $tenacity(u) > tenacity(v)$. Let q be an $evenlevel(u)$ path and r be a minimum length even-alternating (odd-alternating) path from u to v. Then q and r meet only at u.*

Proof. Suppose not, and let (w, w') be the lowest matched edge of q traversed by r, with w' even w.r.t. q. If w' is even w.r.t. r, than by splicing parts of q and r, we can get an even (odd) path to v which is shorter than $|q| + |r|$. However, by Theorems 1 and 2, $|p| \geq |q| + |r|$, leading to a contradiction. Suppose w' is odd w.r.t. r. Then $q[f \text{ to } w'] \circ r[w' \text{ to } u]$ is an odd path to u of length $< |p|$. We now get that $tenacity(u) < tenacity(v)$ (in case p is a $minlevel(v)$ path this is obvious, otherwise use Lemma 2). The contradiction proves the lemma. ∎

Remark. We have considered only the case that u is even w.r.t. p. This is so because of the manner in which we defined even-alternating and odd-alternating paths: they always start with an unmatched edge. The reason for this choice will become clear in Theorem 4.

Notice that in general u may not occur on every $evenlevel(v)$ path, e.g. see *Fig. 10*.

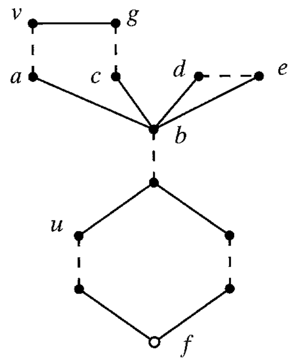


Fig. 10

Theorem 4. *Let v be a vertex of finite tenacity, and let $b = base(v)$. Then, every $evenlevel(v)$($oddlevel(v)$) path consists of an $evenlevel(b)$ path concatenated with a minimum length even-alternating (odd-alternating) path from b to v.*

Proof. Follows from Lemma 7 and Theorem 3.

Definition. Let $b = base^k(v)$, for $k \in Z^+$. Define $evenlevel(v, b)$($oddlevel(v, b)$) to be the length of a shortest even-alternating (odd-alternating) path from b to v. The smaller of these two is called $minlevel(v, b)$, and the larger is called $maxlevel(v, b)$.

Corollaries 2 and 3 follow from Theorem 1 to 4.

Corollary 2. *Let $b = base(v)$, and let p be an $evenlevel(v,b)$ or an $oddlevel(v,b)$ path. Let u be even w.r.t. p with $tenacity(u) = tenacity(v)$. Then, $p[b$ to $u]$ is an $evenlevel(u,b)$ path. Moreover, if p is a $minlevel(v,b)$ path then $p[b$ to $u]$ is a $minlevel(u,b)$ path.*

Corollary 3. *Let v be a vertex such that $base^1(v), base^2(v), \ldots, base^k(v)$ exist. Let p_k be an $evenlevel(base^k(v))$ path, and p_l be an $evenlevel(base^l(v), base^{l+1}(v))$ path, for $1 \le l \le k-1$. Finally, let p_0 be an $evenlevel(v, base^1(v))(oddlevel(v, base^1(v)))$ path. Then $p_k \circ p_{k-1} \circ \ldots \circ p_1 \circ p_0$ is an $evenlevel(v)(oddlevel(v))$ path, and $p_{k-1} \circ \ldots \circ p_1 \circ p_0$ is an $evenlevel(v, base^k(v))(oddlevel(v, base^k(v)))$ path. Conversely, every $evenlevel(v)(oddlevel(v))$ path and every $evenlevel(v, base^k(v))$ $(oddlevel(v, base^k(v)))$ path is of this form.*

6. Blossoms and their significance

In this section we will define blossoms from the perspective of minimum length alternating paths. Theorem 5 gives the central result that all shortest alternating paths from $base(v)$ to v lie in a blossom.

Definition. Let v be a vertex of finite tenacity, and let t be an odd positive integer such that $t \ge tenacity(v)$. Let $k = \min\{j \in Z^+ \mid tenacity((base^j(v)) \ge t\}$, and $l = \min\{j \in Z^+ \mid tenacity(base^j(v)) > t\}$. Define $base_{\ge t}(v) = base^k(v)$, and $base_{>t}(v) = base^l(v)$.

Remark. Let p be an $evenlevel(v)$ or $oddlevel(v)$ path. Then by Corollary 1, every vertex on $p(base_{\ge t}(v)$ to $v]$ is of $tenacity < t$, and every vertex on $p(base_{>t}(v)$ to $v]$ is of $tenacity \le t$.

Definition. Let b be an outer vertex, and t be an odd positive integer such that $t < tenacity(b)$ (b is chosen outer because the base of a vertex is always outer). The *blossom of tenacity t having base b* is the set

$$B_{b,t} = \{v \in V \mid tenacity(v) \le t \text{ and } base_{>t}(v) = b\}.$$

In general the vertices of a blossom may not even be connected by an alternating path. For example, in *Fig. 10*, $B_{b,13} = \{a, c, d, e, v, g\}$.

Remark. 1). If Edmonds' algorithm is modified to 'shrink' sets of vertices in stages: at stage i, shrink all vertices of tenacity $2i+1$, then 'macronodes' obtained at the end of each stage correspond exactly to the blossoms defined above.
2). For matched edge (u,v), u and v belong to the same blossoms.

We will need the following properties of blossoms to prove Theorem 5.

Lemma 8. *Let B_1 and B_2 be two blossoms. Then either they are disjoint or one is contained in the other.*

Proof. Let B_1 be a blossom of tenacity t_1 and base b_1, and B_2 be a blossom of tenacity t_2 and base b_2. Let $t_1 \le t_2$. Suppose $v \in B_1 \cap B_2$. Then $base_{>t_1}(v) = b_1$ and $base_{>t_2}(v) = b_2$, then clearly $B_1 \subseteq B_2$

Consider the case $b_1 \neq b_2$. Then, $base_{>t_2}(b_1) = b_2$. Let u be any vertex in B_1. Then $tenacity(u) \leq t_1$ and $base_{t_1}(u) = b_1$. Therefore, $base_{>t_2}(u) = b_2$. Therefore, $u \in B_2$. Hence $B_1 \subseteq B_2$. ∎

The proof of the following lemma is straightforward.

Lemma 9. *Let v be a vertex such that $base^1(v) = b_1, base^2(v) = b_2, \ldots, base^k(v) = b_k$. Let $tenacity(v) = t_0$, and $tenacity(b_i) = t_i$, for $1 \leq i \leq k$. Let B_i be the blossom of tenacity t_{i-1} having base b_i for $1 \leq i \leq k$. Then $B_1 \subsetneq B_2 \subsetneq \ldots \subsetneq B_k$.*

Proof. $B_i \subseteq B_{i+1}$, for $i = 1, \ldots, k-1$ follows from the definition of blossom. Furthermore, since b_i is in $B_{i+1} - B_i$, proper containment follows.

Lemma 10. *Let $b = base^k(v)$, for $k \in Z^+$. Let p be an $evenlevel(b)$ path, and let $u \neq b$ be even w.r.t. p. Then (u, v) is not an edge in the graph.*

Proof. If (u, v) is an edge, there is an odd path to v of length less then $evenlevel(b)$, giving a contradiction. ∎

Theorem 5. *Let v be a vertex of finite tenacity. Let $b = base(v)$, $t = tenacity(v)$, and $B_{b,t}$ be the blossom having base b and tenacity t. Let p be an $evenlevel(v, b)$ or an $oddlevel(v, b)$ path. Then any vertex on $p(b$ to $v]$ is in $B_{b,t}$.*

Proof. By Lemma 1, it is sufficient to prove the theorem for $|p|$ even. The proof is by induction on $evenlevel(v)$. For the base case, let v be a vertex of finite tenacity having the smallest evenlevel. Then clearly $|p| = 2$, and since the matched neighbour of v is in $B_{b,t}$, the assertion holds. We prove the induction step below.

Suppose $p(b$ to $v]$ contains a vertex not in $B_{b,t}$. Let u be the highest such vertex on p; clearly u is even w.r.t. p. First consider the case $|p[u$ to $v]| > 2$. Let v' be the matched neighbour of v, and let w be the highest vertex on $p(u$ to $v')$. Now, w couldn't be of tenacity t because otherwise by Corollary 2, $p[b$ to $w]$ is an $evenlevel(w, b)$ path, which contradicts the induction hypothesis. Therefore, *tenacity*$(w) < t$. Let $b' = base_{\geq t}(w)$, and let q be an $evenlevel(w, b')$ path. If $b' = b, q \circ (w, v') \circ (v', v)$ is shorter even-alternating path from b to v (using Corollary 3, Lemma 9 and the induction hypothesis). Otherwise, $tenacity(b') = t$ and $b' \in B_{b,t}$. Let r be an $evenlevel(b', b)$ path. Vertex v cannot be on r (if it is even w.r.t. r, we get a shorter path from b to v, otherwise this contradicts Lemma 10) or on $q(b'$ to $w]$ (because these vertices are of $tenacity < t$). Now, by the induction hypothesis, $r \circ q \circ (w, v') \circ (v', v)$ is a shorter path from b to v.

Finally, consider the case $|p[u$ to $v]| = 2$. Let $b' = base_{>t}(u)$, and let q be an $evenlevel(u, b')$ path and r be an $evenlevel(b')$ path. By the induction hypothesis, $q(b'$ to $u]$ is in the blossom having base b' and tenacity t, and before $v \notin q$. As before, v cannot be on r. Since b is not $base^k(u)$ for $k \in Z^+$, $evenlevel(b) + |p[b$ to $u]| > evenlevel(u) = |r| + |q|$. Therefore $r \circ q \circ (u, v') \circ (v', v)$ is a shorter even path to v than that obtained by concatenating an $evenlevel(b)$ path with p. The contradiction proves the induction step. ∎

The following feature of blossoms is being used in the above-stated inductive proof: that p is contained in a blossom, and therefore has a simple interface to the rest of the graph, through the base of the blossom.

7. The nesting of blossoms

The nesting of blossoms is described in Lemma 11. Notice that in *Fig. 2*, the $evenlevel(c,b)$ path either enters or exists from blossoms nested in $B_{b,15}$ at their base (i.e. vertices g and e), and each blossom is used at most once. This is established in Theorem 6. As a consequence, the part of this path in the nested blossom is a minimum length alternating path; this is shown in Corollary 4. These properties of paths w.r.t. the nested blossom structure reveal the reason for BFS dishonesty.

Lemma 11. *Let $B_{b,t}$ be a blossom of tenacity t having base b. Let $C=\{v \mid tenacity(v)=t$ and $base_{>t}=b\}$. For $u\in C\cup\{b\}$, let $B_u=\{v \mid tenacity(v)<t$ and $base_{\geq t}(v)=u\}$. Then $B_{b,t}=C\cup(\bigcup_{u\in C\cup\{b\}} B_u)$.*

Proof. Let $v\in B_{b,t}$. If $tenacity(v)=t$, then $base(v)=b$ and $v\in C$. Suppose $tenacity(v)<t$. Then $base^k(v)=b$, for some positive integer k. If $k=1$, $v\in B_b$. Otherwise, let $u=base^{k-1}(v)$. Clearly, $tenacity(u)\leq t$. If $tenacity(u)=t$, then $u\in C$, and $v\in B_u$. If $tenacity(u)<t$, then $v\in B_b$. Containment in the other direction is obvious. ∎

Definition. Let B_1 and B_2 be two blossoms. If B_1 is a proper subset of B_2 then we will say that B_1 is *nested* in B_2. If furthermore there is no blossom B_3 such that $B_1\subsetneq B_3\subsetneq B_2$ then B_1 is *properly nested* in B_2. The *nesting depth* of blossom B is defined recursively as follows: if B has no properly nested blossoms then its nesting depth is 0. Otherwise, among the blossoms properly nested in B, let B' have the largest nesting depth, and let k be the nesting depth of B'. Then the nesting depth of B is $k+1$.

Lemma 12. *Let $u\in C$ as defined in Lemma 11, and $v\in B_u$. Then,*

$$minlevel(v,b) > evenlevel(u,b), \quad \text{and}$$
$$maxlevel(v,b) < oddlevel(u,b).$$

Proof. The proof follows from Corollary 2, and the fact that $tenacity(v)<tenacity(u)$. ∎

Theorem 6. *Let v be a vertex of tenacity t in blossom $B_{b,t}$, and p be an $evenlevel(v,b)$ or an $oddlevel(v,b)$ path. Then p enters and exits from any blossom properly nested in $B_{b,t}$, say B_u, at most once. If so, p must either enter or exit from the blossom and its base, $B_u\cup\{u\}$ at its base u.*

Proof. By Lemma 1, it is sufficient to prove the theorem for $|p|$ even. The proof is by induction on $evenlevel(v,b)$ for vertices of tenacity t in $B_{b,t}$. For the base case, let v be such a vertex having smallest $evenlevel(v,b)$. Clearly, any vertex of $tenacity<t$ on p must be in the nested blossom B_b. Let w be the highest such vertex on p. Then by Theorem 4, $p(b$ to $w]$ is in B_b, proving the assertion. We prove the induction step below.

Suppose p enters and exits at least one blossom properly nested in $B_{b,t}$ more than once. Let B_u be the last such blossom on p. The proof of the base case shows that $u \neq b$. Let w be the first vertex of p in $B_u \cup \{u\}$, and w' be the last. If either $w = u$ or $w' = u$, then by Theorem 5 we can get a shorter even-alternating path from b to v which uses B_u 'properly'. Otherwise, let q be an *evenlevel*(u, b) path; by Lemma 12, $|q| < |p|$, and therefore by the induction hypothesis q satisfies the condition. By Corollary 2 any vertex of tenacity t must be at its correct distance on q; moreover since u is outer, this must be its minlevel distance. Therefore, q must enter a set $B_{u'} \cup \{u'\}$ at u'. Assume that $p(b$ to $w]$ and $p[w'$ to $v]$ both intersect q; the remaining cases are simpler. A vertex of tenacity t on $q \cap p[w'$ to $v]$ must occur at its maxlevel distance on p. Now, if $B_{u'}$ is a blossom such that u' occurs on q then $p[w'$ to $v]$ must enter $B_{u'}$ at most once (since B_u is the last 'misused' blossom on p), and must exit the set $B_{u'} \cup \{u\}$ at u'. Let x be the first vertex of $p[w'$ to $v]$ which is on q and has tenacity t. By Theorem 5, $q[x$ to $u]$ concatenated with an *evenlevel*(w, u) path is shorter than $p[x$ to $w]$. Therefore, if $p(b$ to $w]$ does not intersect $q[x$ to $u]$, we can get a shorter even path from b to v. Otherwise, let y be the first vertex of $p(b$ to $w]$ on $q[x$ to $u]$. First assume *tenacity*$(y) = t$. If y is odd w.r.t. q, then since $q[y$ to $u]$ concatenated with an *evenlevel*(w', u) path is shorter than $p[y$ to $w']$, we can get a shorter path to v. Otherwise we can use $q[y$ to $x]$ instead of $p[y$ to $x]$. Next suppose *tenacity*$(y) < t$, and $y \in B_{u'}$. Then, $p[x$ to $y]$ is an even length alternating path from x to y. But the shortest such path is obtained by concatenating $q[x$ to $u']$ with an *evenlevel*(y, u') path (this path does not use B_u). Again, this gives a shorter path to v. The contradiction proves the induction hypothesis. ∎

Remark. If p is a *minlevel*(v, b) path then all properly nested blossoms used by p are entered through the base. On the other hand suppose p is *maxlevel*(v, b) path and uses nested blossoms $B_1 \ldots B_k$ in this order. Then, either all of these blossoms are entered through the base or all are exited through the base, or $\exists i$, $1 \leq i < k$ such that $B_1 \ldots B_i$ are entered through the base $B_{i+1} \ldots B_k$ are exited through the base.

Corollary 4. *Suppose p enters (exits) the set $B_u \cup \{u\}$ at u and exits (enters) at w. Then $p[u$ to $w]$ is an evenlevel(w, u) path and therefore evenlevel$(w) =$ evenlevel$(u) + |p[u$ to $w]|$.*

Proof. Follows form Theorem 5 and 6, and the minimality of p.

Remark. Theorem 6 and Corollary 4 are also true for any blossom nested (not necessarily properly) in $B_{b,t}$. This can proven by an easy induction on the nesting depth of $B_{b,t}$.

We can now explain why minimum length alternating paths are BFS dishonest in general graphs. Let p be a *maxlevel*(v, b) path that enters $B_u \cup \{u\}$ at vertex $w \neq u$. By Theorem 4 and 5, every *oddlevel*(w, b) path consists of an *evenlevel*(u, b) path concatenated with an *oddlevel*(w, u) path; the latter path is contained in B_u. Therefore $|p[b$ to $w]| >$ *oddlevel*(w, b). However, this is consistent with Theorems 1 and 2 since *tenacity*$(w) <$ *tenacity*(v). The following corollary complements Theorems 1 and 2 for case *tenacity*$(w) <$ *tenacity*(v) and gives additional constraints that minimum length alternating paths must satisfy despite their BFS dishonesty. It can be proven by an easy induction on $|p|$.

Corollary 5. *Let p be an evenlevel(v) or an oddlevel(v) path, and w be a vertex of finite tenacity on p. Then base(w) is also on p; moreover, $p[$base(w) to $w]$ is an* *evenlevel*$(w, base(w))$ *or an* *oddlevel*$(w, base(w))$ *path, depending on the parity of* $|p[base(w) \text{ to } w]|$.

8. Every maxlevel path contains a bridge

We will prove that every edge on the path from $base(v)$ to v has *tenacity* $\leq$ *tenacity*(v). This is followed by the last structural theorem: that a *maxlevel*(v) path contains a unique bridge of tenacity *tenacity*(v). Theorem 7(b) shows how v can be obtained by searching down from the bridge; this fact will eventually be used in DDFS. Theorem 7(c) gives a relationship between the even and oddlevels of the endpoints of the bridge and *tenacity*(v). This is used in Lemma 15 to prove that bridges are found 'well in time' to make the synchronization work.

Lemma 13. *Let* p *be an* *evenlevel*(v,b) *or an* *oddlevel*(v,b) *path, where* $b = base(v)$. *Then, all edges on* p *have tenacity* $\leq t$, *where* $t = tenacity(v)$.

Proof. By induction on the nesting depth of $B_{b,t}$, for the base case, suppose $B_{b,t}$ has nesting depth 0. Then every vertex on $p(b \text{ to } v]$ has tenacity t. So by Lemma 1, every matched edge on p has tenacity t. Let (u, u') be an unmatched edge on p, with u even w.r.t. p. Since u and u' occur at the correct distance on p, and $tenacity(u') = t$, it follows that $evenlevel(u') = t - evenlevel(u) - 1$. This gives $tenacity(u, u') = t$.

We now prove the induction step. Suppose p enters (exits) the set $B_u \cup \{u\}$ in u and exits (enters) in w. Then, by Corollary 4, and the induction hypothesis, all edges on $p[u \text{ to } w]$ have tenacity$< t$. Of the remaining edges on p, the tenacity of matched edges is t since they are incident on vertices of tenacity t. Finally consider a remaining unmatched edge (z, z'), If both endpoints have tenacity t (or more, since b may be an endpoint), then z and z' occur at the correct distance on p, and the argument of the base case applies. Otherwise suppose $tenacity(z) = t$ and $tenacity(z') < t$. Let $b' = base_{\geq t}(z')$; by Theorem 6, b' must be on p. Now, using Corollary 4 and the fact that b' must be at the correct distance on p, it is easy to see that $tenacity(z, z') = t$. This proves the induction step. ∎

Theorem 7. *Let* v *be a vertex of finite tenacity, say* t, *and let* p *be a* *maxlevel*(v) *path. Then*

(a) *There is unique bridge of tenacity* t *on* p.

Proof. We will first show the existence of such a bridge and then its uniqueness. Let $b = base(v)$, and let set S consist of all vertices on $p[b \text{ to } v]$ which have *tenacity* $\geq t$. By Theorem 2, these vertices occur at the correct distance on p. Partition S into two sets: $S_{\min}(S_{\max})$ consists of vertices of S which occur at their minlevel (maxlevel) distance on p. Notice that $b \in S_{\min}$ and $v \in S_{\max}$. Let w be the highest vertex of p in $S_{\min}$, and let w' be the lowest vertex of p in $S_{\max}$. Notice that all vertices of $S_{\min}$ lie on $p[b \text{ to } w]$ and those of $S_{\max}$ on $p[w' \text{ to } v]$.

First suppose (w, w') is a matched edge. Clearly, $oddlevel(w) = |p[f \text{ to } w]|$ and $oddlevel(w') = t - |q[f \text{ to } w']|$ giving $tenacity(w, w') = t$. Moreover, since the minlevel of both w and w' is odd, (w, w') is not a prop, and is therefore a bridge.

In the remaining case, w and w' are both outer vertices, and all vertices on $p(w \text{ to } w')$, if any, are of *tenacity* $< t$. By Theorem 6, these vertices must be either in B_w or $B_{w'}$, i.e. blossoms of *tenacity* $\leq t-2$ having base w and w' respectively. Let x be the highest vertex of $p(w \text{ to } w')$ in B_w; if there is no such vertex, let $x = w$. Let x' be the first vertex on $p(x \text{ to } w']$. Clearly (x,x') is unmatched; we will show that (x,x') is a bridge of tenacity t.

By Corollary 4, $\mathit{evenlevel}(x) = |p[f \text{ to } x]|$, and $\mathit{evenlevel}(x') = \mathit{evenlevel}(w') + |p[w' \text{ to } x']|$. Also, $\mathit{evenlevel}(w') = t - |p[f \text{ to } w']|$. This gives $\mathit{tenacity}(x,x') = t$. Now, x' is not predecessor of x; if $x = w$, the predecessor of x is its matched neighbour, and otherwise the predecessor of x is in B_w. Similarly x is not a predecessor of x', and so (x,x') is a bridge.

We finally prove uniqueness. Let (u,u') be an edge on $p[b \text{ to } w]$ with u' higher than u. If $\mathit{tenacity}(u') = t$, u is a predecessor of u', and so (u,u') is a prop. Otherwise by Lemma 13, (u,u') is of *tenacity* $< t$. The same applies for an edge (u,u') on $p[w' \text{ to } v]$ with u higher than u'. Also, by Lemma 13, edges on $p[w \text{ to } w']$ other than (x,x'), if any, are of tenacity $< t$. Finally consider edge (u,u') on $p[f \text{ to } b]$, with u' higher than u. If $\mathit{tenacity}(u') \geq t$, u is a predecessor of u'. Otherwise by Corollary 5, $\mathit{base}_{\geq t}(u')$ must be on $[f \text{ to } u')$, and so by Lemma 13, $\mathit{tenacity}(u,u') < t$. ∎

The following definition and extensions of Theorem 7(a) give algorithmically useful facts; their proofs follow from the proof of Theorem 7(a).

Definition. Let v be a vertex of tenacity t. We will say that vertex u is pred_t, of v if either:

(i). u is a predecessor of v and $\mathit{tenacity}(u) \geq t$, or

(ii). there is a predecessor, u' of v, such that

$$\mathit{tenacity}(u') < t, \text{ and}$$
$$u = \mathit{base}_{\geq t}(u').$$

Define $\mathit{pred}^* - t$ to be the reflexive, transitive closure of the relation pred_t.

Remark. If v is outer, then only the matched neighbour of v is pred_t of v. Thus (ii) applies only for inner vertices. In this case, there is an odd-alternating path from u to v of length $\mathit{minlevel}(v) - \mathit{minlevel}(u)$ whose internal vertices are all in the blossom $B_{u,t-2}$ (because a $\mathit{minlevel}(v)$ path consists of an $\mathit{evenlevel}(u')$ path concatenated with the edge (u',u)).

Theorem 7(b). *Let (x,x') be the unique bridge of tenacity t on p. If $\mathit{tenacity}(x) < t$ then let $y_0 = \mathit{base}_{\geq t}(x)$, otherwise let $y_0 = x$. Similarly, if $\mathit{tenacity}(x') < t$ then $z_0 = \mathit{base}_{\geq t}(x')$, otherwise let $z_0 = x'$. Then there is a set of distinct vertices $y_1, \ldots, y_k$, $z_1, \ldots, z_l$ such that*

(i). y_{i+1} is pred_t of y_i, for $0 \leq i < k$, and b is pred_t of y_k, where $b = \mathit{base}(v)$.

(ii). z_{i+1} is pred_t of z_i, for $0 \leq i < l$, and v is pred_t of z_l.

(c): *If (x,x') is matched then $\mathit{oddlevel}(x) = \mathit{oddlevel}(x') = (t-1)/2$. If (x,x') is unmatched then either $\mathit{evenlevel}(x) \leq (t-1)/2$ or $\mathit{tenacity}(x) < t$, and either $\mathit{evenlevel}(x') \leq (t-1)/2$ or $\mathit{tenacity}(x') < t$.*

The following lemma will be used in the proof of Theorem 9.

Lemma 14. *Let v_0 be a vertex of tenacity t, and $v_1, \ldots, v_k$ be distinct vertices such that v_i is pred_t of v_{i-1} for $1 \leq i \leq k$. If $v_1, \ldots, v_{k-1}$ are of tenacity t and $\mathit{tenacity}(v_k) > t$ then $v_k = \mathit{base}(v_0)$.*

Proof. By the remark following the definition of $pred_t$, there is a path p from v_k to v_0 of length $minlevel(v_0) - evenlevel(v_k)$ By the definition of $pred_t$, p is part of a $minlevel(v_0)$ path. Now, since all vertices on p, other than v_k, have $tenacity \leq t$, it follows that $base(v_0) = v_k$. ∎

Remark. Let $G(V,E)$ be a graph and M be a matching in it. Construct a new graph $G'(V',E')$ as follows: on each unmatched vertex of G add a new matched edge, and connect the other endpoints of these edges via unmatched edges to a single new unmatched vertex f. Now, there is an obvious one-to-one correspondence between $evenlevel(v)$ and $oddlevel(v)$ path in G and G', for $v \in V$: simply remove the first two edges of a path in G' to obtain the path in G. (Notice that the first two edges are props, since they give minlevels.) So, the evenlevel and oddlevel increases by 2 and the tenacity by 4 in going from G to G'. Therefore, Theorems 1 and 2 are true for G as well. Henceforth, let m denote the length of a minimum length augmenting path in G. For $v \in V$ of finite tenacity, let us say that *the bases of v is defined* if $base(v) \neq f$ in G'. Notice that following is an alternative characterization of such vertices: on any $evenlevel(v)$ or $oddlevel(v)$ path in G, there is a vertex of $tenacity > tenacity(v)$. (Clearly, if $tenacity(v) < m$, the base of v is defined.) For such vertices, Theorems 3 to 6 hold in G as well because of the above-stated one-to-one correspondence. Finally, Theorem 7 holds for all vertices, v, in G because the first two edges on any $evenlevel(v)$ or $oddlevel(v)$ path in G' are props.

9. Procedure MIN, and the role of anamolies

Procedure MIN finds minlevels of vertices, and it also determines whether an edge is a prop or a bridge. MIN examines an edge (u,v) at most once (MIN marks edges that it examines 'used' so they don't get examined again). If (u,v) is unmatched (matched), MIN examines this edge while searching from the endpoint having smaller evenlevel (oddlevel); ties are broken arbitrarily. Let us assume that (u,v) is examined from v. If at this stage MIN gives u its minlevel then (u,v) is a prop, otherwise it is a bridge.

If the search level i is even, MIN examines unmatched edges incident at vertices having evenlevel i. Suppose $evenlevel(v) = i$, (u,v) is unmatched and has not yet been examined by MIN. The following cases arise (we will assume that at the beginning of the phase, the evenlevels and oddlevels of all vertices are initialized to ∞):

(i). $minlevel(u) = \infty : minlevel(u)$ is set to $i+1$, v is inserted in the set of predecessors of u, and (u,v) is marked a prop.

(ii). $minlevel(u) = i+1 : v$ is inserted in the predecessor list of u, and (u,v) is marked a prop.

(iii). $minlevel(u) \leq i$ and $evenlevel(u)$ is finite: (u,v) is marked a bridge and is inserted in the set of bridges of tenacity $(evenlevel(u) + evenlevel(v) + 1)$. (See *Fig. 11.*)

(iv). $minlevel(v) \leq i$ and $evenlevel(u) = \infty$; v inserted in the set of anamolies of u, and (u,v) is marked a bridge. (We will a give a precise definition of anamolies below and will explain their significance. See *Fig. 12* for an example.)

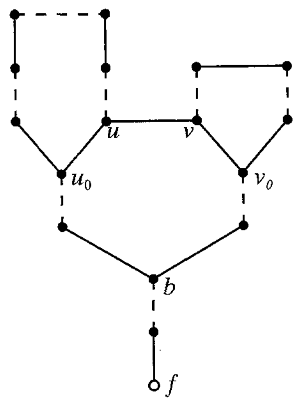


Fig. 11

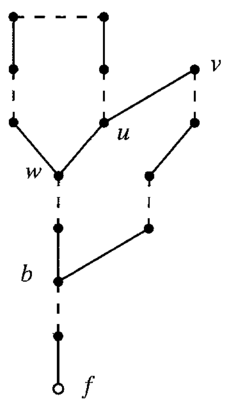


Fig. 12

If the search level i is odd, MIN examines matched edges incident at vertices having oddlevel i. Suppose $oddlevel(v) = i$, and matched edge (u,v) has not yet been examined by MIN. The following cases arise:

(i). $minlevel(u) = \infty : minlevel(u)$ is set to $i+1$, v is inserted in the set of predecessors of u, and (u,v) is marked a prop.

(ii). $minlevel(u)$ is finite: in this case, $oddlevel(u)=i$. Edge (u,v) is marked a bridge and is inserted in the set of bridges of tenacity$(oddlevel(u)+oddlevel(v)+1)$).

Definition. The following definition follows from case (iv). We will say that v *is an anamoly of* u if (u,v) is an unmatched edge, u is inner, and $oddlevel(u) < evenlevel(v) \leq (tenacity(u)-1)/2$. In this case (u,v) is called an *anamoly*.

From the above definition it is clear that an anamoly is a bridge, and that $tenacity(u,v) > tenacity(u)$. Notice that MIN is able to determine the tenacity of all bridges other than anamolies. In case (iv), $tenacity(u,v)$ cannot be determined since $evenlevel(u)$ is not yet found. At search level $(tenacity(u)-1)/2$, MAX will find *evenlevel*(u) and will determine the tenacity of all anamolies of u, including (u,v). This is well in time for calling DDFS with bridge (u,v) since $tenacity(u,v) > tenacity(u)$.

Notice that anamolies were not mentioned in the structure developed in Theorems 1 to 7. Anamolies are an algorithmic convenience, and the above-stated manner of handling them leads to the synchronization mentioned in Section 2.

10. Double depth first search

Procedure MAX uses a new graph searching algorithm: double depth first search (DDFS). We will first describe and prove correctness of this algorithm in the following simple setting:

Definition. A directed graph $H(S,T)$ with distinguished vertices $a,b \in S$ is said to be *layered* if S is partitioned into sets $S_n, S_{n-1}, \dots, S_0$, called *layers*, such that:

(i). every edge goes from a higher numbered layer to a lower numbered layer (not necessarily consecutive), and

(ii). Vertices in $S_n(S_0)$ have positive outdegree (indegree), and those in $S_{n-1} \cup \dots \cup S_1$ have positive indegree as well as outdegree.

If $u \in S_k$, then $level(u) = k$. The distinguished vertices a and b can be at any levels. Vertex $s \in S_l$ is said to be a *bottleneck* if for every vertex u having $level \le l$, every path from a to u and from b to u contains s, and moreover s is the highest leveled such vertex. We will assume that (u,v) represents the directed edge from u to v.

The problem is to determine if there is a bottleneck. Also (we will assume that a and b are initially marked 'L' and 'R' respectively):

(i). if there is a bottleneck, say $s \in S_l$, to find it. Furthermore, to mark 'L' or 'R' (corresponding to left and right) all vertices having $level > l$ which are reachable from a or b. If such a vertex, u, is labeled $L(R)$ then there is a path from $a(b)$ to u consisting of $L(R)$ marked vertices. There also two paths, one from a to s and an other from b to s consisting of L and R marked vertices respectively.

(ii). if not, to find vertices c and $d \in S_0$, and vertex disjoint paths from a to c and b and d.

DDFS accomplishes this task in linear (i.e. $O(|T|)$) time. It simultaneously grows two vertex disjoint DFS trees T_L and T_R rooted at a and b, consisting of L and R marked vertices respectively. DDFS maintains two centers of activity, c_L and c_R which start at a and b respective. As in the usual DFS, if a center of activity moves from u to v, then $p(v) = u$, i.e. u is made the *parent* of v. Also, if a center of activity is at u and all outgoing edges from u have been examined, then the center of activity moves to $p(u)$; this important step is called *backtracking*. A pidgin Algol description of the procedure appears on the next page; for the sake of visual clarity we have used indentation to demarcate statements.

Procedure DDFS

```
        begin
          Mark each vertex 'unvisited' and each edge 'unused';
          c_L ← a;
          c_R ← b;
          barrier ← b

          while not[level(c_L) = level(c_R) = 0] do begin
Lloop:        while [level(c_L) ≥ (c_R)] do begin
                  mark c_L 'L' and 'visited';
                  for each unused edge (c_L, u) do begin
                      mark (c_L, u) 'used';
                      if u = c_R then                        ...  /c_L and c_R meet
                          c_R ← p(c_R);
                          p(u) ← c_L;
                          c_L ← u;
                          goto Rloop;
                      else if u is not visited then
                          p(u) ← c_L;
                          c_L ← u;
                  end;
                  if c_L = a then HALT                       ...  /bottleneck found
                      else c_L ← p(c_L);                     ...  /c_L backtracks
                          goto Lloop;
              end;
Rloop:        while [level(c_R) > level(c_L)] do begin
                  mark c_R 'R' and 'visited'
                  for each unused edge (c_R, u) do begin
                      mark (c_R, u) 'used';
                      if u is not visited then
                          p(u) ← c_R;
                          c_R ← u;
                          goto Rloop;
                  end;
                  if c_R ≠ barrier then c_R ← p(c_R)         ...  /c_R backtracks
                      else barrier ← c_L;                    ...  /barrier updated
                          c_R ← c_L;
                          c_L ← p(c_L);                      ...  /c_L backtracks
                          goto Lloop;
              end;
          end;
        end;
```

We now describe how the two DFS's are coordinated: we will first present and prove a quadratic time version of DDFS, and later make it linear time. If c_L and c_R are at different levels then the higher one grows its tree, and otherwise c_L

grows its tree; the latter choice is arbitrary. The reflexive transitive closure of the relation parent is called *ancestor*. The ancestors of $c_L(c_R)$ give a directed path from a to c_L (b to c_R) consisting of $L(R)$ marked vertices; these vertices will be called the left active (right active) vertices. The most important step is dealing with the situation that c_L and c_R meet at a vertex w. In this case, first c_R attempts (another arbitrary choice) to find a new path from a right active vertex having $level \leq level(c_L)$, by backtracking and finding new outgoing edges from right active vertices in the usual DFS manner. (Clearly, the path found will start at the lowest possible right active vertex.) If c_R succeeds, then w is included in T_L (i.e. marked 'L'), and the search proceeds. Otherwise, c_L attempts to find a path from a left active vertex to a vertex having $level \leq level(w)$. If c_L succeeds, then w is included in T_R (i.e. marked 'R'), and the search proceeds. If c_L also fails, then w is the bottleneck, and DDFS halts. In this case, w is not included in T_L or T_R. In the first two cases, $p(w)$ is appropriately set, depending on whether w is included in T_L or T_R. Finally, DDFS halts if c_L and c_R are both at distinct level 0 vertices.

It easy to check that DDFS maintains the following invariant: any vertex that is visited (i.e. in $T_L \cup T_R$) but not active has been backtracked from. (Notice that there may be right active vertices that have already been backtracked from; however, every left active vertex has not been backtracked from.) It follows that if DDFS halts at vertex s, then every vertex visited, other than s, is backtracked from. Using a straightforward proof by contradiction, one can now show that s is indeed the bottleneck, and furthermore, DDFS would have visited every vertex reachable from a or b and having $level < level(s)$. The remaining requirements follow from the fact that at any point in the algorithm, there is a path from a to c_L (b to c_R) consisting of $L(R)$ marked vertices.

The above-stated algorithm follows any outgoing edge at most once. However, notice that c_R may backtrack from a vertex several times. (Though c_L backtracks from a vertex at most once because any left active vertex is not yet backtracked from.) This leads to an $O(|S|^2)$ running time. The algorithm is made more efficient as follows: initially, a *barrier* is placed at b. When c_L and c_R meet at a vertex, say w, c_R backtracks only up to the barrier. If it fails to find an alternative path, it includes w in T_R and it moves the barrier to w. The right active vertices now include the ancestors of c_R from c_R to the barrier only. The above-stated invariant is still maintained; in addition we have that any active vertex (left or right) is not yet backtracked from. This yields the following:

Theorem 8. *DDFS accomplishes the above-stated tasks, (i) and (ii), in $O(|T|)$ time. More precisely, if it ends with a bottleneck s, then the time taken is $O(|T'|)$, where T' is the set of edges which are on a path from a or b to v.*

11. Using DDFS to find the support of a bridge, and procedure MAX

We will first show how DDFS can be used to find the support of a bridge in an idealized setting. Let (u,v) be a bridge of tenacity $t \leq m$, where m is the length of a minimum length augmenting path in G. Let

$$u_0 = \begin{cases} u & \text{if } tenacity(u) = t \\ base_{\geq t}(u) & \text{o.w.} \end{cases}$$

$$v_0 = \begin{cases} v & \text{if } tenacity(v) = t \\ base_{\geq t}(v) & \text{o.w.} \end{cases}$$

This is illustrated in *Fig. 11*. Let $H(S,T)$ be the directed graph consisting of vertices $S = \{w \in V \mid w \text{ is } pred_t^* \text{ of } u_0 \text{ or } v_0\}$, and edges $T = \{(w,w') \mid w,w' \in S, \text{ and } w' \text{ is } pred_t \text{ of } w\}$. Partition vertices in S into layers according to their minlevels, thereby obtaining a layered graph $H(S,T)$.

Proposition 1. *Suppose DDFS is run on graph $H(S,T)$ with distinguished vertices u_0 and v_0. Then:*

(i). *if DDFS terminates with bottleneck s, then the set of vertices visited by DDFS, other than s, constituted support (u,v).*

(ii). *if DDFS terminates at two unmatched vertices f and f' , then there is a minimum length augmenting path from f to f' containing (u,v). In this case $tenacity(u,v) = m$.*

Proof. (i). Suppose $w \in support(u,v)$. By Theorem 7(b), $w \in S$. If the base of w is defined then let $b = base(w)$, otherwise let b be any unmatched vertex at which a $minlevel(w)$ path starts. By Theorem 7(b), $b \in S$, and there are two disjoint paths in $H(S,T)$, one from u_0 (say) to w and the other from v_0 to b. Since $minlevel(w) > minlevel(b)$, w is above the bottleneck and will be visited.

For the other direction, first note that s must be outer, since inner vertices in S have only one incoming edge. Suppose $w \neq s$ is visited by DDFS; assume w.l.o.g. that w is outer and is marked L, and that u_0 and v_0 are marked L and R respectively. Then there is an L marked path, p_1, from u_0 to w and an R marked path, p_2, from v_0 to s in $H(S,T)$.

Let $x,y \in S$, with x $pred_t$ of y. If x is a predecessor of y then (y,x) is an edge in G. Otherwise, by the remark following Theorem 7(a), there is a path from x to y of length $minlevel(y) - minlevel(x)$ in G. Using this fact (and since the paths mentioned above will be in distinct blossoms), it is easy to see that corresponding to p_1 and p_2 there are disjoint paths in G: p_1' from w to u_0 of length $minlevel(u_0) - minlevel(w)$, and p_2' from s to v_0 of length $minlevel(v_0) - minlevel(s)$. Let p_3' be an $evenlevel(s)$ path. Now, p_3', p_2' and (u,v), together with an $evenlevel(u,u_0)$ path (if $u_0 \neq u$), and $evenlevel(v,v_0)$ path (if $v_0 \neq v$) gives a $maxlevel(u_0)$ path. This path concatenated with p_1' gives an $oddlevel(w)$ path of length $t - minlevel(w)$. This proves that $tenacity(w) = t$ and $w \in support(u,v)$.

(ii). There are two disjoint paths, one from u_0 to f and the other from v_0 to f', in $H(S,T)$. As in (i), these correspond to disjoint paths in G which yield an augmenting path from f to f'. ∎

Remark. The bottleneck s found in case (i) will be the base of a blossom iff $tenacity(s) > tenacity(u,v)$. *Fig. 13(a)* shows an example in which $tenacity(s_1) = tenacity(u_1,v_1)$.

Let the search level be i. MAX imposes an arbitrary ordering on the bridges of tenacity $2i+1$, say $g_1, g_2, \ldots, g_k$, and calls DDFS with the bridges in this order.

For the purposes of efficiency, the working of DDFS is different in two ways from the above-stated idealized setting. We explain the differences below.

Firstly, a vertex may be in the support of more than one bridge. For example vertex w in *Fig. 13(a)* is in support of (u_1,v_1) and (u_2,v_2). Now, w will be visited by MAX only once and will be assigned to one bridge: to (u_1,v_1) if DDFS is called with (u_1,v_1) before (u_2,v_2), and to (u_2,v_2) otherwise. Define $petal(g_i) = support(g_1) - \bigcup_{j<i} support(g_j)$. DDFS finds the petals of $g_1 \dots g_k$, rather than their supports. Notice that unlike blossom and support which are graph-theoretically defined, petals are algorithmically defined since they depend on the ordering imposed on the bridges. Suppose (u_1,v_1) is processed before (u_2,v_2) in *Fig. 13(a)*. The first DDFS ends with bottleneck s_1 (notice that $tenacity(s_1) = tenacity(u_1,v_1)$), and the second with s_2. Define the *base* of a petal to be its bottleneck . Thus $base(petal(u_1,v_1)) = s_1$ and $base(petal(u_2,v_2)) = s_2$. Also, if $w \in petal(u,v)$, define $base(w)$ to be $base(petal(u,v))$. If the newly found petal is non-empty as an implementational convenience, DDFS creates a new node; all the vertices of the petal point to the node, and the node points to the base. This is illustrated in *Fig. 13(b)* and *14*. These figures also show the 'L' and 'R' marks left by DDFS.

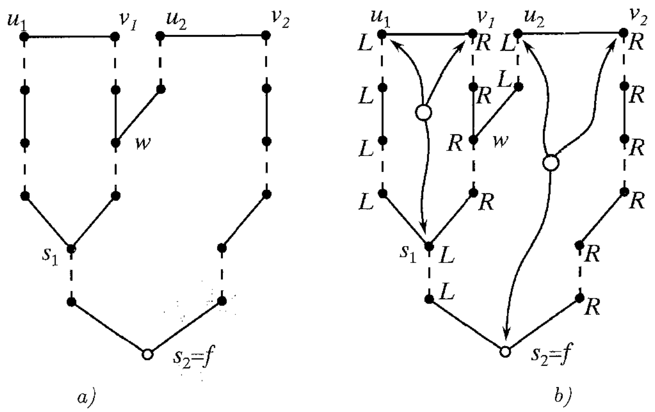


Fig. 13

The second difference is that the graph $H(S,T)$ is not constructed explicity; DDFS is run on graph $G(V,E)$ itself as follows. Suppose the center of activity is at v and u is a predecessor of v. If u is not in any petal, then the center of activity moves to u. Otherwise (in this case $tenacity(u) \le tenacity(v)$), the center of activity moves to $base^*(u)$. The function $base^*(u)$ is defined below:

function $base^*(u)$

if u is not in a petal then return u

else return $base^*(base(u))$.

end;

One more point needs to be mentioned. Suppose DDFS is called with bridge (u,v). If u is in a petal, then the left center of activity starts from $base^*(u)$; similarly for v.

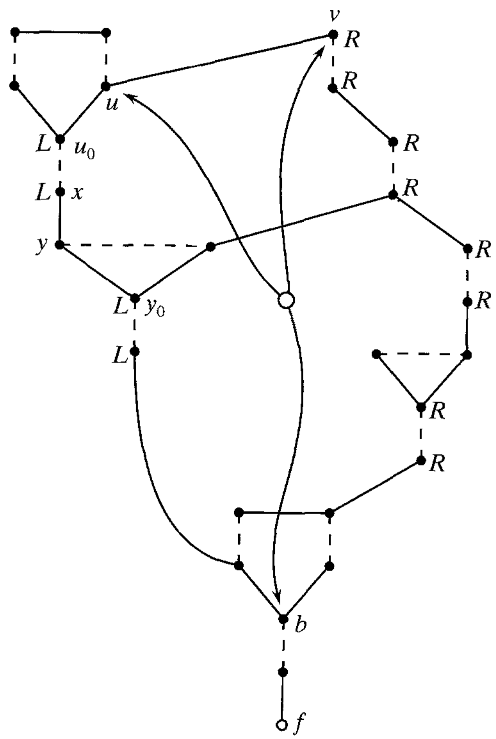


Fig. 14

Having found the vertices of tenacity $2i+1$, MAX determines their maxlevels. For each such vertex v, and for each anamoly u of v, MAX also determines the tenacity of bridge (u, v).

12. Proof of correctness of MIN and MAX

In Theorem 9, we will show why the synchronization works, and we will establish several properties of petals and buds in order to prove the correctness of MIN and MAX.

Theorem 9. *Let m be the length of a minimum length augmenting path in G. Then MIN and MAX correctly find the minlevel of all vertices having minlevel $\leq m$ and the maxlevel of all vertices having tenacity $\leq m$.*

Proof. By induction on i, the search level, we will prove the following stronger statement:

Induction Hypothesis. Let $t = 2i+1$. At the end of search level i:
1) all vertices v s.t. $minlevel(s) \leq i+1$ get their correct minlevel; the remaining vertices have their minlevel set at ∞.

2) all vertices v s.t. $tenacity(v) \le t$ get their correct maxlevel, and are assigned a petal. The petal of vertices of higher tenacity is still undefined, and their maxlevel is set at ∞.
3) for any vertex v s.t. $tenacity(v) \le t$, $base^*(v) = base_{>t}(v)$.

The hypothesis is clearly true for search level 0. Assuming its truth for search levels$< i$, we prove it true for search level i.
(1.) At the beginning of search level i the following holds: if i is even (odd), every vertex u having an evenlevel (oddlevel) of i would have gotten it. (If $minlevel(u) = i$, this follows from (1). If $maxlevel(u) = i$, then $tenacity(u) \le 2i-1$, and the assertion follows from (2)). Suppose $minlevel(v) = i+1$, and let (u,v) be the last edge on a $minlevel(v)$ path. By the above-stated fact, MIN will find v while searching from u. In this manner, MIN finds all the predecessors of v, and is correctly able to distinguish props from bridges.
(2.) This involves proving the following three statements: (i). All bridges of tenacity $2i+1$ are found at the end of execution of MIN during search level i.
(ii). When called with bridge (u,v), DDFS finds $support(u,v)$.
(iii). For every vertex v, every $maxlevel(v)$ path contains a bridge having $tenacity = tenacity(v)$.
Statement (iii) is proven in Theorem 7. Since MIN correctly distinguishes between props and bridges, to prove the first statement, we only need to show that the tenacity of bridges is determined 'well in time'; this is done below in Lemma 15 for bridges having non-empty support.

Let $g_1, g_2 \dots g_k$ be an arbitrary ordering imposed by MAX on the bridges of tenacity $t = 2i+1$. By induction on j, we will prove that when DDFS is called on g_j it finds $petal(g_j)$, thereby proving the second statement.

The induction basis follows from Proposition 1 and induction hypothesis (3), because DDFS is essentially being run on the associated graph $H(S,T)$. We prove the induction step below.

Consider the situation when DDFS is called with bridge g_j. We make the following observations: Suppose vertex v is in petal, and suppose u, a predecessor of v, is not a petal. Then clearly $u = base^*(v)$. Secondly, if vertex v is in a petal and $tenacity(v) = t$, then $bud^*(v)$ is $pred_t^*$ of v. From the first observation it follows that if DDFS arrives at v, it is sufficient to continue search from $bud^*(v)$ only. From the second observation it follows that DDFS visits all vertices in $support(g_j) - \bigcup_{l<j} petal(g_l)$.But by the induction hypothesis, this set is $support(g_j) - \bigcup_{l<j} support(g_l) = petal(g_j)$.
(3.) Let v be a vertex of tenacity t, and $b = bud^*(v)$ at the end of search level i. Clearly, b is $pred_t^*$ of v. Since b is not in the support of any bridge of tenacity t, $tenacity(b) > t$. Now, by Lemma 14, $base(v) = b$. Next suppose $tenacity(v) < t$. Let b' be $bud^*(v)$ at the end of search level $i-1$. By induction hypothesis (3), $b' = base_{\ge t}(v)$. If $tenacity(b') > t$, b' will be $bud^*(v)$ at the end of search level i also. If $tenacity(b') = t$, $b = bud^*(b')$ will be $base^*(v)$ at the end of search level i. By the above-stated argument, $b = base_{>t}(v)$. ∎

Remark. Let b be an outer vertex of $tenacity > t$, and let $B_{b,t}$ be the blossom having base b and tenacity t. Then, hypothesis (3) implies that at the end of search level i, the set $\{v \in V \mid bud^*(v) = b\} = B_{b,t}$.

Lemma 15. *Let (u,v) be a bridge of tenacity $2i+1$ having non-empty support. Assuming the induction hypothesis of Theorem 9, the algorithm will determine the tenacity of (u,v) by the end of execution of MIN during search level i.*

Proof. Suppose (u,v) is matched. By Theorem 7(c), $oddlevel(u)=oddlevel(v)=i$. Therefore, during the execution of MIN during search level i, (u,v) will be examined and its tenacity will be ascertained.

Next suppose (u,v) is unmatched. W.l.o.g assume $evenlevel(u) \leq evenlevel(v)$. Since $tenacity(u,v)=2i+1$, $evenlevel(u) \leq i$. Edge (u,v) will be examined from u at search level $evenlevel(u)$. If $evenlevel(v)$ is determined at this stage, $tenacity(u,v)$ is ascertained. Otherwise, $evenlevel(v)$ must be $>i$. If so, by Theorem 7(c), $tenacity(v) \leq 2i+1$. This implies $evenlevel(v) oddlevel(v)$, i.e. v is inner. Since u is not a predecessor of v, $evenlevel(u)+1 > oddlevel(v)$. So, while searching from u at search level $evenlevel(u)$. MIN will make u an anamoly of v. Now, at search level $(tenacity(v)-1)/2$, i.e. before search level i, $evenlevel(v)$ will be given, and $tenacity(u,v)$ will be ascertained. ∎

Remark. A matched bridge (u,v) has non-empty support, since its support contains u and v. One can extend Lemma 15 to unmatched bridges having empty support using the following easily proven assertions (assume (u,v) is an unmatched edge):
a). if u is not a predecessor of v and v is inner, then $tenacity(v) < tenacity(u,v)$.
b). if v is outer, then $evenlevel(v) \leq (tenacity(u,v)-1)/2$.

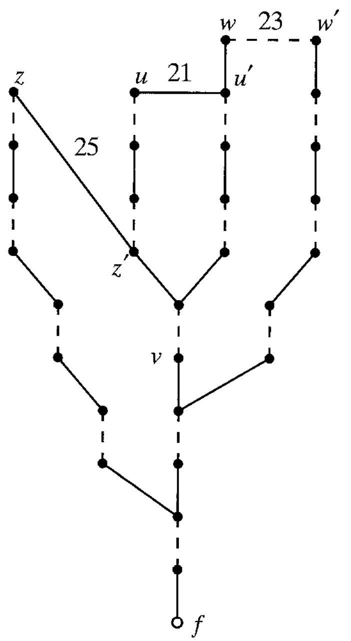


Fig. 15

Figure 15 illustrates the importance of this synchronization. Notice that $tenacity(z,z')=25$ is ascertained at search level 10 while processing bridge (u,y'). So, why not call DDFS with (z,z') at search level 10? If this is done, then the vertices visited may not be in $support(z,z')$, and so DDFS will not be able to ascertain their tenacity. For example, v will be visited, even though $v \in support(w,w')$ and has tenacity 23.

13. Finding minimum length augmenting paths

If the current matching, M, is not maximum then at search level $(m-1)/2$ DDFS will eventually be called with a bridge that is on a minimum length augmenting path, and will end up at two unmatched vertices; m denotes the length of a minimum length augmenting path w.r.t. M. At this point procedure FINDPATH is invoked to find such a path between the two vertices. On the other hand, if the current phase executes for $(|V|-1)/2$ search levels without finding an augmenting path, matching M is maximum, and the algorithm halts.

DDFS marks petals appropriately so that FINDPATH may find a path through the petals efficiently. Suppose DDFS is called with bridge (u,v). Let $u_0 = bud^*(u)$ and $v_0 = bud^*(v)$, and assume that the left and right centers of activity start at u_0 and v_0 respectively. After the new petal P is formed, the following are set (assuming P is non-empty): L-peak$(P) = u$ and R-peak$(P) = v$. As shown in *Figs. 13(b)* and *14* the new petal-node points to L-peak and R-peak.

Furthermore, if $u_0 \neq u$, DDFS sets $exit-bud(u,v) = u_0$. Similarly, if $V_0 \neq v$, $exit-bud(v,u) = v_0$. Suppose DDFS visits vertex x, y is a predecessor of x, y is already in a petal, and $y_0 = bud^*(y)$. Then, DDFS visits y_0, and sets $exit-bud(y,x) = y_0$. In this case, y_0 is $pred_t^*$ of x, where $t = tenacity(u,v) = tenacity(x)$. This is illustrated in *Fig. 14*.

Let us say that u is $bud^+(v)$ if $u = bud^*(v)$ at some point during the execution of the current phase. FINDPATH is called with two vertices, say x and y, as parameters, where y is $bud^+(x)$. It returns an even-alternating path from y to x of length $evenlevel(x) - evenlevel(y)$. Assume x is in petal P, is marked R and has tenacity T. FINDPATH is a recursive procedure; it first finds the 'shell' of the required path.

First suppose x is an outer vertex. FINDPATH simply follows predecessors till it gets to $bud(x)$: suppose it is at vertex $z \in P$ and w is a predecessor of z. If $w \in P$, FINDPATH adds w to the shell. If $w \notin P$, it finds $exit-bud(w,z)$, say w', and adds w and w' to the shell. It then continues search from w'. If $bud(x) \neq y$, FINDPATH continues the above process; notice that $bud(x)$ is outer and y is $bud^+(bud(x))$. Finally, FINDPATH fills in the 'gaps' in the shell (such as (w,w')) by recursively calling FINDPATH (e.g. on parameters w,w'). When all the recursive calls are complete, the path from y to x has been found. For example, the call FINDPATH(o,q) in *Fig. 16* results in the shell $m\ n\ o\ q$. The recursive call to fill the gap is FINDPATH(o,q).

If x is an inner vertex, the process is more involved since the path from y to x will use the bridge of P. FINDPATH first finds the left and right peaks of p, say u and v. Suppose $exit-bud(u,v) = u_0$ and $exit-bud(v,u) = v_0$. FINDPATH now grows a DFS tree rooted at v_0 and consisting only of the R marked vertices of P till it finds x. This yields a shell from x to v_0. It then grows another DFS tree

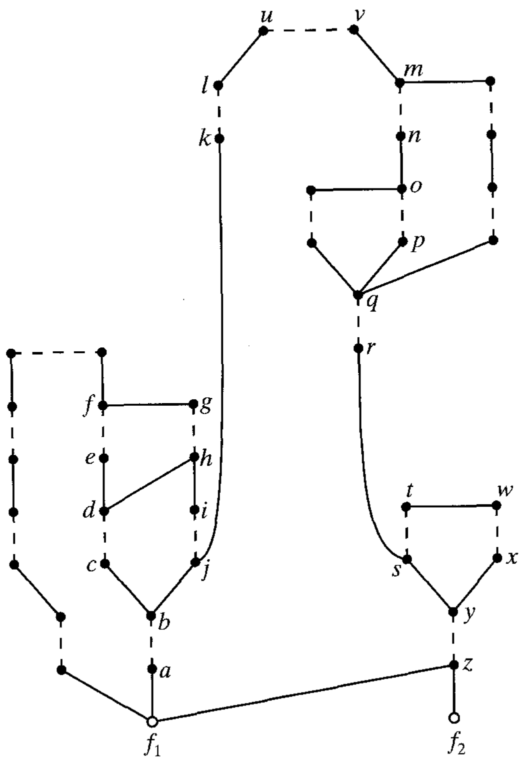


Fig. 16

rooted at u_0 and consisting only of the L marked vertices of P till it finds $bud(x)$. These shells are filled in through recursive calls. Also paths are found from u_0 to u and v_0 to v. These concatenated with (u,v) give an even-alternating path from $bud(x)$ to x. The path from y to $bud(x)$ is found as stated above. For example, the call FINDPATH(j, f_1) in *Fig. 16* results in the two shells $f\ e\ d\ c\ b$ and $g\ h\ i\ j$; these have no gaps. These are concatenated with the path $b\ a\ f_1$ from b to f_1.

Suppose DDFS is called with bridge (u,v) at search level $(m-1)/2$, and ends at unmatched vertices x and y. Suppose $u_0 = bud^*(u)$ and $v_0 = bud^*(v)$. Then, an augmenting path from x to y is found by the following: FINDPATH$(u_0, x) \circ$ FINDPATH(u, u_0). In *Fig. 16*, the augmenting path is obtained by FINDPATH$(u, f_1) \circ ($FINDPATH$(v, f_2))^{-1}$. This results in the recursive calls FINDPATH(j, f_1), FINDPATH(m, q) and FINDPATH(s, y).

After a path p is found, MAX invokes procedure TOPOLOGICAL ERASE. This procedure erases p and all vertices which cannot be in a minimum length augmenting path disjoint form p as follows: TOPOLOGICAL ERASE first erases all vertices of p and all edges incident at these vertices. Then, it successively removes any remaining vertices which have no predecessors left. After this, MAX continues processing the remaining bridges of tenacity m to find augmenting paths in the remaining graph. In this manner it finds a maximal set of disjoint minimum length augmenting paths.

14. Running time of the algorithm

Finally, we analyse the time taken by algorithm to execute one phase. Since MIN examines each edge only once, it takes $O(|E|)$. A vertex belongs to at most one petal, and DDFS examines its predecessor edges once during the formation of this petal. However, DDFS also has to compute *bud** of vertices. This can be accomplished in $O(|E| \propto (|E|, |V|))$ time using the set union algorithm of [15]; here $\propto$ is the inverse Ackerman function. By resorting to the RAM model of computation (in which operations on $O(\log n)$ bit numbers are assumed to take unit time), Gabow and Tarjan [7] have given an incremental tree set union algorithm which gives a linear implementation of *bud** on the RAM model. We leave the open problem of obtaining a linear implementation of *bud** in the pointer machine model of computation (see [11] for a precise definition). One avenue for accomplishing this is to prove the claim in [14] that because of the special structure of blossoms, if *bud** is implemented using only path compression, its cost can be charged to the edges and is linear.

Let us analyse the time taken by FINDPATH in a phase. Suppose vertex v which is in petal P is on a minimum length augmenting path p. Let $b = base_{\geq m}(v)$, where m is the length of p. Clearly, b is also on p. Let $B_{b,m}$ be the blossom having base b and tenacity m. Clearly $P \subseteq B_{b,m}$. Now, a minimum length alternating path from any unmatched vertex to a vertex in $B_{b,m}$ must use b. Therefore, the vertices in $B_{b,m} - p$ cannot be on a minimum length augmenting path disjoint from p, and will be deleted by TOPOLOGICAL ERASE. Therefore, FINDPATH does at most two DFS's in petal P.

Let (u, v) be a prop, with u predecessor of v. Define $petal(u, v)$ to be $petal(u)$, and define the size of a petal to be the number of edges in it. By the above-stated remarks, the work done by DDFS in a petal is linear in its size. Therefore, the total time taken by FINDPATH in a phase is $O(|E|)$. It is easy to see that TOPOLOGICAL ERASE also takes linear time. This proves that the running of the algorithm on the RAM model is $O(\sqrt{|V|}|E|)$.

Acknowledgements. A very special thanks to: Silvio Micali for memorable times spent discovering this algorithm; to Richard Karp for sharing his perspective on graph algorithms and for introducing me to matching theory; to Manuel Blum for his encouragement and enthusiasm for this work; and to Steve Mitchell and Umesh Vazirani for spending several days verifying the proofs. Thanks also to the numerous other people who saw parts of this proof at various points during its development.

References

[1] M. L. Balinski: Labelling to obtain a maximum matching, in *Combinatorial Mathematics and its Application*, Eds.: R. C. Bose and T. A. Downing, University of North Carolina Press, 1969, 585–602.

[2] C. Berge: Two theorems in graph theory, *Proc. Natl, Acad. Sci.* **43** (1957), 842–844.

[3] N. Blum: A new approach to maximum matchings in general graphs, Proceedings of the 17*th* International Colloquium on Automata, *Languages and Programming* (1990), 586–597.

[4] J. Edmonds: Paths, trees, and flowers, *Canad. J. Math.* **17** (1965), 449–467.

[5] S. Even, and O. Kariv: An $O(n^{2.5})$ algorithm for maximum matching in general graphs, *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science* (1975), 100–112.

[6] H. Gabow: An efficient implementation of Edmonds' algorithm for maximum matching on graphs, *JACM* **23** (1976), 221–234.

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

[8] H. N. Gabow, and R. E. Tarjan: Faster scaling algorithms for general graph matching problems, technical report CU–CS–432–89, University of Colorado at Boulder.

[9] J. E. Hopcroft, and R. M. Karp: An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, *SIAM J. Comput.* **2** (1973), 225–231.

[10] T. Kameda, and I. Munro: A $O(|V||E|)$ algorithm for maximum matching of graphs, *Computing* **12** (1974), 91–98.

[11] D. E. Knuth: *The art of computer Programming, Vol.* 1: *Fundamental Algorithms*, 2nd ed., Addison–Wesley, Reading, MA, 1973.

[12] E. L. Lawler: *Combinatorial Optimization: Network and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[13] L. Lovász, and M. Plummer: *Matching Theory*, Academic Press, Budapest, Hungary, (1986).

[14] S. Micali, and V. V. Vazirani: An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs, *Proc. 21st Annual IEEE Symposium in Foundation of Computer Science*, 1980, 17–27.

[15] R. E. Tarjan: Efficiency of a good but not linear set union algorithm, *J. Assoc. Comput. Mach.* **22** (1975), 215–225.

[16] C. Whitzgall, and C. Zahn: Modification of Edmonds' maximum matching algorithm, *J. Res. Nat. Bur. Standards* Sect. B **69** (1965), 91–98.

Vijay V. Vazirani

Department of Computer Science & Engg.
Indian Institute of Technology
New Delhi 110016, India
`vazirani@cse.iitd.ernet.in`