First, we shall characterize the paths

$P = [w,A], Q, [u,A]$ with $[u,B] \notin Q$,

found by MDFS. Let $P = e_1, e_2, \ldots, e_t$. Then, the following properties are fulfilled:

1. e_t is a weak back edge.
2. If we start in e_t and consider P backwards, then we see some tree edges followed by a single cross, forward or back edge, followed by a sequence of tree edges, and so on.

Hence, we need after the performance of POP$([u,B])$ the following sets of edges:

$$R_{[u,A]} := \{ [v,B] \in V' \mid ([v,B],[u,A]) \text{ is a weak back edge} \}$$

and for some $[q,A] \in V'$

$$E_{[q,A]} := \{ [v,B] \in V' \mid ([v,B],[q,A]) \text{ is a cross, forward, or back edge} \}.$$

According to invariant 3, during the backward search some subpaths can be skipped over. Therefore, we need set of nodes

$$D_{[q,A]} := \{ [p,A] \in V' \mid L_{[p,A]} = [q,A] \text{ previously} \}.$$

By invariant 3, $D_{[q,A]} \subseteq D_{[q',A]}$ implies $L_{[q,A]} = L_{[q',A]}$. Hence, the knowledge of

$L_{[q',A]}$ and the fact $D_{[q,A]} \subseteq D_{[q',A]}$ implies the knowledge of $L_{[q,A]}$.

We say that $D_{[q,A]}$ is <u>current</u> if

$$D_{[q,A]} \notin D_{[q',A]} \quad \forall [q',A] \in V' \setminus \{[q,A]\}.$$

Invariant 3 $\Rightarrow$

We can compute $L_{[p,A]}$ in the following way:

(1) Compute $[q,A]$ such that
$[p,A] \in D_{[q,A]}$ and $D_{[q,A]}$ current.

(2) If $[q,A]$ does not exist, then $L_{[p,A]} := \emptyset$.
Otherwise

$$L_{[p,A]} := \begin{cases} [q,A] & \text{if PUSH}([q,A]) \text{ has never been performed} \\ \emptyset & \text{otherwise.} \end{cases}$$

As described above, a correct manipulation of the current sets $D_{[q,A]}$ allows the solution of the first subproblem.

Note that by Invariant 2 of Lemma 1.4, every $[p,A] \in V'$ is contained in at most one current set $D_{[q,A]}$.

For the organisation of the backward search, we also need the knowledge if $L_{[p,A]} \neq \emptyset$ previously. This will be realized by the correct update of the following set:

$$L := \{ [p,A] \in V' \mid L_{[p,A]} \neq \emptyset \text{ previously} \}.$$

Now we can give a detailed description of the backward search which will be performed after POP($[u,B]$).

The consideration of those paths

$$P = [w,A], Q, [u,A] \text{ with } [u,B] \notin Q$$

is done in several rounds. In the first round, we construct backwards all paths without any cross, forward, or back edge. In the second round, all paths with exactly one such edge are constructed implicitly, and so on.

In the ith round, we consider the weak back edges $([v,B],[u,A])$ if $i=1$, and we consider those edges $([v,B],[q,A]) \in E_{[q,A]}$ for which $L_{[q,A]} = [u,A]$ is computed in the $(i-1)$th round, if $i > 1$.

Starting in node $[v,B]$, we follow backwards the tree edges as long as node $[u,B]$ is reached. If we reach a node $[p,A] \in L$ then we compute the current $D_{[r,A]}$ such that $[p,A] \in D_{[r,A]}$, and we jump to $[r,A]$ for the continuation of the backward search. According to invariant 3 $L_{[x,A]} = L_{[r,A]}$ and hence, $L_{[x,A]} = [u,A]$ for all $[x,A] \in D_{[r,A]}$.

For the reconstruction of a strongly simple path from s to t constructed by the algorithm, we store in variable $P_{[r,A]}$ that edge in $E_{[q,A]}$ which concludes that block of tree edges containing the tree edge with end node $[r,A]$, for all $[r,A] \in V'$ with $L_{[r,A]} \neq \emptyset$ for the first time. As soon as $P_{[r,A]}$ is defined, the node $[r,A]$ is inserted into L. Hence, the node $[q,A]$ is determined unambiguously by the algorithm.

The implementation of MDFS must be done with attention of the correct manipulation of the sets $D_{[q,A]}$, $R_{[q,A]}$, and $E_{[q,A]}$. The following table describes in terms of the case of MDFS, and in terms of the operation which is performed, how MDFS has to update these sets.

case, operation	set update
Case 1	no update
Case 2.1	$E_{[w,A]} := E_{[w,A]} \cup \{[v,B]\}$
Case 2.2.i	$E_{[w,A]} := E_{[w,A]} \cup \{[v,B]\}$
Case 2.2.ii	$R_{[w,A]} := R_{[w,A]} \cup \{[v,B]\}$
Case 2.3.i	
$L_{[w,A]} \neq \emptyset$	no update
$L_{[w,A]} = \emptyset$	$E_{[w,A]} := E_{[w,A]} \cup \{[v,B]\}$ if $[w,A] \notin L$
Case 2.3.ii	no update

PUSH([u,A])	no update
POP([v,B])	① $D_{[v,A]} := \{ [p,A] \mid$ MDFS has found a path from $[p,A]$ to $[v,A]$ not containing $[v,B] \}$.

In Case 2.1, it is clear that $[w,A] \notin L$ since POP([w,A]) is not performed. In Case 2.2.i, $[w,A] \notin L$ follows directly from $[w,B] \in K$ and Lemma 1.1. Note that in Case 2.3.i, subcase $L_{[w,A]} \neq \emptyset$, we have to store the information that edge $([v,B],[w,A])$ is used. In the implementation, we accomplish this by adding the edge $([v,B],[w,A])$ to node $[v,B]$ in K. Then we obtain the <u>expanded node</u> $\langle([v,B],[w,A]); [v,B]\rangle$. The considerations above lead to the following implementation of the procedure SEARCH.

procedure SEARCH

```
if TOP(K) = t
  then
    reconstruct a strongly simple path P
    from s to t which has been constructed
    by the algorithm
  else
    mark TOP(K) "pushed";
    for all nodes [w, Y] ∈ N[TOP(K)]
      do
        if Y = B
(Case 1)    then
              PUSH([w, B])
              SEARCH
(Case 2)    else
              if [w, A] ∈ K
(Case 2.1)      then
                  E[w,A] := E[w,A] ∪ {TOP(K)}
                else
                  if [w, B] ∈ K
(Case 2.2)          then
                      if [w, A] is marked "pushed"
(Case 2.2.i)            then
                          E[w,A] := E[w,A] ∪ {TOP(K)}
(Case 2.2.ii)           else
                          R[w,A] := R[w,A] ∪ {TOP(K)}
                      fi
```

(Case 2.3) else

if [w,A] is marked "pushed"

(Case 2.3.i) then

if $L_{[w,A]} \neq \phi$

then

expand TOP(k) in k to

$\langle (TOP(k), [w,A]); TOP(k) \rangle$;

PUSH($L_{[w,A]}$);

$L_{[w,A]} := \phi$;

SEARCH

else

if $[w,A] \notin L$

then

$E_{[w,A]} := E_{[w,A]} \cup \{TOP(k)\}$

fi

fi

(Case 2.3.ii) else

PUSH([w,A]);

SEARCH

fi

fi

fi

fi

od;

(*leaf TOP(k) = [v,X] *)

```
if TOP(K) = [v,B] and [v,A] is not
                      marked "pushed"
  then
    L_cur := [v,A];
    D_L_cur := ∅;
    L_def := ∅;
    (* L_def will contain the start nodes
       for the next round.*)
    for all [q,B] ∈ R_[v,A]
      do
        CONSTRL(([q,B],[v,A]), [v,B]);
      od;
      while L_def ≠ ∅
        do
          choose any [z,A] ∈ L_def;
          L_def := L_def \ {[z,A]};
          for all [q,B] ∈ E_[z,A]
            do
              CONSTRL(([q,B],[z,A]),[v,B])
            od
        od
  fi;
  POP;
  POP
fi.
```

CONSTRL is a call of the following procedure.

```
procedure CONSTRL(([q,B],[u,A]),[x,B])
  P_cur := ([q,B],[u,A]);
  [z,B] := [q,B];
  while [x,B] is not reached
    do
      for all [y,A] ∉ L meet during the
          backward search from [z,B] to L ∪ {[x,B]}
        do
          D_Lcur := D_Lcur ∪ {[y,A]};
          L := L ∪ {[y,A]};
          P_[y,A] := P_cur;
          L_def := L_def ∪ {[y,A]}
        od;
      if [y,A] ∈ L is met by the backward search
        then
          (* Let D_[r,A] be the current set
             containing [y,A] *)
          D_Lcur := D_Lcur ∪ D_[r,A];
          [z,B] := [r,B]
      fi
  od.
```

The reconstruction of a strongly simple path P from s to t constructed by the algorithm remains to be explained.

Beginning at the end of P, such a path P can be reconstructed by traversing the MDFS-tree T backwards. Note that TOP points to the end of P, and that the father of each node in T is always unique. As long as we traverse tree edges of the algorithm MDFS, we have no difficulty. But every time we meet a node [u,A] which was added to P by an application of Case 2.3.i, we have to reconstruct a subpath [w,A], Q, [u,A] which was joined to P. In this situation, the considered portion of T is the expanded node $\langle([v,B],[w,A]);[v,B]\rangle$; i.e., the structure of T tell us that MDFS has applied Case 2.3.i. It remains to reconstruct Q. Note that $P_{[w,A]}$ contains the non-tree edge of MDFS, which finishs the block containing the tree edge with end node [w,A]. Let

$$P_{[w,A]} = ([v',B],[v'',A]).$$

Then $P^1_{[w,A]}$ denotes [v',B] and $P^2_{[w,A]}$ denotes [v'',A]. As long as [u,A] is met, we reconstruct Q block by block, beginning at node [w,A]. Each block can be reconstructed as the path P itself. These considerations lead to the following procedure for the reconstruction of an augmenting path, constructed by the algorithm.

```
procedure RECONSTRPATH(t, s);
  NODE_cur := t;
  while NODE_cur ≠ s
    do
      if father(NODE_cur) is not expanded
        then
          NODE_cur := father(NODE_cur)
        else
          (* let father(NODE_cur) =
                <([v,B], [w,A]); [v,B]> *)
          RECONSTRQ(NODE_cur, [w,A]);
          NODE_cur := [v,B]
      fi
  od.
```

RECONSTRQ is a call of the following procedure:

```
procedure RECONSTRQ([u,A], [w,A]);
  ST := [w,A];
  RECONSTRPATH(P¹_ST, ST);
  while P²_ST ≠ [u,A]
    do
      ST := P²_ST;
      RECONSTRPATH(P¹_ST, ST)
  od.
```

The correctness of the manipulation of $L_{[w,A]}$, $[w,A] \in V'$, and the correctness of the reconstruction of the M-augmenting path P follow from Lemma 1.4.

Exercise:
Prove the correctness of the manipulation of $L_{[w,A]}$, $[w,A] \in V'$, and the correctness of the reconstruction of the M-augmenting path P.

The procedure RECONSTRPATH resembles standard recursive methods used for the reconstruction of augmenting paths (i.e., see Tarjan's book)

The time and space complexity of our implementation of MDFS remain to be considered.

It is easy to see that the time used by the algorithm MDFS is bounded by $O(n+m)$ plus the total time needed for the manipulation of the sets $D_{[q,A]}$, $[q,A] \in V'$ where $n = |V|$ and $m = |E|$.

If we use linear lists for the realization of the sets $D_{[q,A]}$ with a pointer to the node $[q,A]$ for each element of $D_{[q,A]}$, the execution time for each union is bounded by $O(n)$. Following the pointer corresponding to $[p,A]$, we can find the set containing $[p,A]$ in constant time.

At most n union operations are performed by MDFS. Hence, the total time used for the manipulation of the sets $D_{[q,A]}$ is bounded by $O(n^2)$. The time needed for the n union operations can be reduced to $O(n \cdot \log n)$ if we use the following standard trick, the so called <u>weighted union heuristic</u>:

We store with each set the number of elements of the set. A union operation is performed by changing the pointer of the smaller of the two sets which are involved and updating the number of elements. Every time when the pointer with respect to an element is changed, the size of the set containing this element is at least twice of its previous set. Hence, for each element, its pointer is changed at most $\log n$ - times. Hence, the total time used for all union operations is $O(n \cdot \log n)$. Altogether, the total time used for the augmentation of one path is $O(m + n \cdot \log n)$. No space more than $O(m+n)$ is needed.

$\rightsquigarrow$

<u>Theorem 1.3</u>

MDFS can be implemented such that it uses only $O(m + n \log n)$ time and $O(m+n)$ space.

If we use for the update of the sets $D_{[q,A]}$ disjoint set union (see book of Tarjan), the total time can be bounded by $O((m+n)\alpha(m,n))$.

α: inverse of the Ackermann function

Note that for each node $[p,A]$ one find operation suffices for the decision of $L_{[p,A]}$.
Further, we can reduce these bounds to $O(m+n)$ using incremental tree set union.

H. N. Gabow, R. E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. Syst. Sci. 30 (1985), 209-221.

1.2 Speeding up matching algorithms

1.2.1 The Theory

The general method for the computation of a maximum matching in a graph G was the following:

Algorithm MAXIMUM MATCHING
Input: undirected graph $G=(V,E)$, matching $M \subseteq E$
Output: A maximum matching M_{max}
Method:

```
while there exists an M-augmenting path
    do
phase { construct such a path P;
        M := M ⊕ P
    od;
M_max := M.
```

Properties:

- During each phase exactly one M-augmenting path is constructed if such a path exists.
- A phase uses $O(n+m)$ time where $n := |V|$ and $m := |E|$.
- total time $\geq$ # phases $\cdot\ O(n+m)$
  $= O(n \cdot (m+n)) = O(n \cdot m)$

Question:

Is it possible to compute during a phase several node disjoint M-augmenting paths simultaneously and to augment these such that the used time is increased only by a small constant factor and the total time is considerably reduced?

To answer this question, we modify the general method as follows:

Algorithm MAXIMUM MATCHING'

Input: undirected graph $G = (V, E)$, matching $M \subseteq E$.

Output: maximum matching M_{max}.

Method:

while there exists an M-augmenting path
do
phase { • construct a maximal set of paths $\{Q_1, Q_2, \dots, Q_t\}$ such that
i) Q_i, Q_j are node disjoint, $1 \le i < j \le t$
ii) Q_i is a shortest M-augmenting path, $1 \le i \le t$.
• $M := M \oplus Q_1 \oplus Q_2 \oplus \dots \oplus Q_t$
od;
$M_{max} := M$.

Reference:

John E. Hopcroft, Richard M. Karp,
An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, SIAM J. COMPUT. 2 (1973), 225 – 231.

Goal: Analysis of the modified general method.

Theorem 1.4 (Hopcroft, Karp 1973)

Let $G = (V, E)$ be an undirected graph and $M, N \subseteq E$ be matchings. If $|M| = r$, $|N| = s$ and $s > r$ then $M \oplus N$ contains at least $s - r$ node disjoint M-augmenting paths.

Proof:

Consider the graph $G' = (V, M \oplus N)$.

M, N matchings $\Rightarrow$

Each node $v \in V$ is incident to at most one edge in $M \setminus N$ and at most one edge in $N \setminus M$.

$\Rightarrow$

Each connected component C of G' is either

a) an isolated node,
b) a cycle of even length with edges alternatively in $M \setminus N$ and in $N \setminus M$,
or
c) a path whose edges are alternatively in $M \setminus N$ and in $N \setminus M$.

Let $C_1, C_2, \ldots, C_e$ be the connected components of G' where

$$C_i = (V_i, E_i).$$

Let

$$\delta(C_i) := |E_i \cap N| - |E_i \cap M|.$$

Then there holds with respect to the three cases:

a) $\delta(C_i) = 0$

b) $\delta(C_i) = 0$

c) $\delta(C_i) \in \{-1, 0, 1\}$

Observe that

$\delta(C_i) = 1 \Leftrightarrow C_i$ is an M-augmenting path.

Consider

$$\textcircled{1} := \sum_{i=1}^{e} \delta(C_i)$$

Then there holds

$$\begin{aligned} \textcircled{1} &= |N \setminus M| - |M \setminus N| \\ &= |N| - |M| \\ &= s - r. \end{aligned}$$

Hence, there are at least $s-r$ connected components C_i of G' such that $\delta(C_i) = 1$.

These components are node disjoint and each is an M-augmenting path.

■

Remark:

Note the similarity to the proof of Berge's theorem.

Corollary 1.1 (Berge)

M is a maximum matching iff G contains no M-augmenting path.

Corollary 1.2

Let $G = (V, E)$ be an undirected graph, $M \subseteq E$ be a matching, $|M| = r$, s be the size of a maximum matching, and $s > r$. Then there is an M-augmenting path of length

$$\leq 2 \cdot \left\lfloor \frac{r}{s-r} \right\rfloor + 1.$$

Proof:

Let N be a maximum matching.

Theorem 1.1 $\Rightarrow$

$G' = (V, N \oplus M)$ contains at least $s - r$ node disjoint M-augmenting paths.

Since $|M| = r$ there exists at least one M-augmenting path P which contains

$$\leq \left\lfloor \frac{r}{s-r} \right\rfloor$$

edges from M.

Hence,

$$|P| \leq 2 \cdot \left\lfloor \frac{r}{s-r} \right\rfloor + 1$$

∎

Next we show that after the augmentation of a shortest M-augmenting path the length of augmenting paths increases monotonely.

Theorem 1.5

Let $G = (V, E)$ be an undirected graph, $M \subseteq E$ be a matching, P be a shortest M-augmenting path and P' be an $(M \oplus P)$-augmenting path. Then

$$|P'| \geq |P| + 2|P \cap P'|.$$

Proof:

Let $N := M \oplus P \oplus P'$. Then N is a matching of size $|M| + 2$.

Theorem 1.1 $\Rightarrow$

$M \oplus N$ contains two node disjoint M-augmenting paths P_1 and P_2.

Since $M \oplus N = P \oplus P'$ there holds:

$$|P \oplus P'| \geq |P_1| + |P_2|.$$

P shortest M-augmenting path $\Rightarrow$

$$|P_1| \geq |P| \quad \text{and} \quad |P_2| \geq |P|.$$

Hence,

$$|P \oplus P'| \geq 2 \cdot |P|.$$

Furthermore, there holds

$$|P \oplus P'| = |P| + |P'| - 2|P \cap P'|.$$

Hence, $|P| + |P'| - 2|P \cap P'| \geq 2|P|$

$$\Leftrightarrow \quad |P'| \geq |P| + 2|P \cap P'|.$$

$\square$

Let us consider the following scheme of computation:

- Start with the matching $M_0 = \phi$.
- Compute a sequence
$$M_0, M_1, M_2, \ldots, M_i, \ldots$$
of matchings where
$$M_{i+1} = M_i \oplus P_i$$
for a shortest M_i-augmenting path P_i.

Theorem 1.5 implies the following Corollary:

Corollary 1.3

$$|P_i| \leq |P_{i+1}| \quad \text{for all } i.$$

Goal:

Proof that all paths of equal length with respect to the computation scheme above can be augmented simultaneously.

Corollary 1.4

For all i, j with $i \neq j$ and $|P_i| = |P_j|$ there holds: P_i and P_j are node disjoint.

Proof:

Assume that there exist i, j, $i \neq j$:
$|P_i| = |P_j|$ and P_i, P_j are not node disjoint.

Then there exists k and ℓ such that

a) $i \leq k < \ell \leq j$

b) P_k and P_ℓ are not node disjoint

c) $\forall\, t : k < t < \ell$ there hold:

P_k, P_t are node disjoint and
P_t, P_ℓ are node disjoint.

$\Rightarrow$

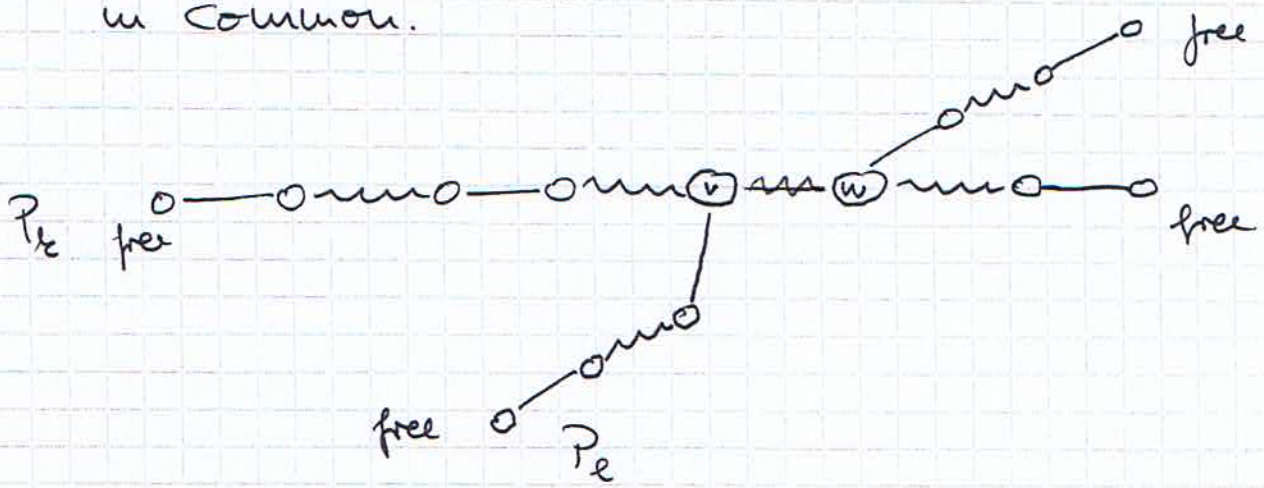
P_ℓ is an $(M_k \oplus P_k)$-augmenting path

Theorem 1.2 $\Rightarrow$

$$|P_\ell| \geq |P_k| + 2|P_k \cap P_\ell|$$

Since $|P_\ell| = |P_k|$ there holds $P_k \cap P_\ell = \emptyset$.

But P_k and P_ℓ have at least one node v in common.



$\Rightarrow$

The edge $(v,w) \in (M_k \oplus P_k)$ is an edge in P_k and in P_ℓ.

$\Rightarrow$ $P_k \cap P_\ell \neq \emptyset$ a contradiction.

$\Rightarrow$

P_k and P_e are node disjoint and therefore, P_i and P_j also.

■

Theorem 1.6

Let $G = (V, E)$ be an undirected graph and s be the size of a maximum matching of G. Then the number of phases performed by the algorithm MAXIMUM MATCHING' is

$$\leq 2\lfloor \sqrt{s} \rfloor + 2.$$

Proof:

Let $r = \lfloor s - \sqrt{s} \rfloor$. Note that with respect to our scheme of computation

$$|M_r| = r.$$

Corollary 1.2 $\Rightarrow$

$$|P_r| \leq 2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1$$

$$\leq 2\lfloor \sqrt{s} \rfloor + 1$$

Note that the length of an augmenting path is always odd.

$\Rightarrow$

The algorithm uses for the construction of M_r at most $\lfloor \sqrt{s} \rfloor + 1$ phases.

Since $s \leq n$, we obtain the following corollary:

Corollary 1.5

Let $G = (V, E)$ be an undirected graph. A maximum matching of G can be computed in $O(\sqrt{n} \cdot T_{phase})$ time where T_{phase} is the time used for a phase.

For bipartite graphs, Hopcroft and Karp have described an elegant, simple $O(m+n)$ implementation of an entire phase. First they perform a breadth-first search (BFS) on G_M with start node s until the target node t is reached, they have obtained a layered and directed graph $\overrightarrow{G}_M$ for which the paths from s to t correspond exactly to the shortest M-augmenting paths in G. Using DFS, they find a maximal set of disjoint M-augmenting paths. Whenever an M-augmenting path is found, the path and all incident edges are deleted and the DFS is continued. BFS and DFS take $O(m+n)$ time. Hence, $T_{phase} = O(m+n)$ for bipartite graphs

Exercise

Work out the matching algorithm of Hopcroft and Karp for bipartite graphs.

With respect to general graphs, the following question suggests itself: