Washington University in St. Louis

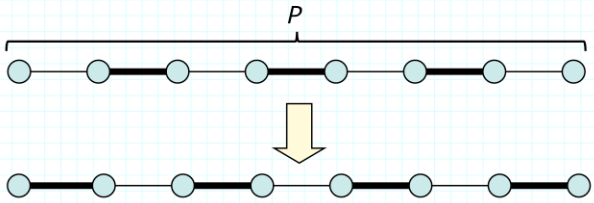
Engineering

# Maximum Size Matchings in General Graphs

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst

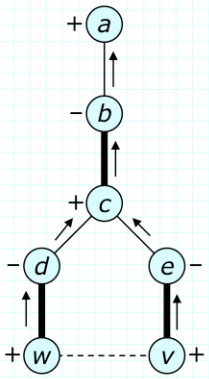
# Recap of Augmenting Path Method




- Search for augmenting path and swap edges on path
- In bipartite graph, find paths by building forest with tree roots at unmatched vertices
- Basic step: choose unexamined edge $\{v,w\}$ with $v$ even and examine it
  - » if $w$ is unreached and matched, make $w$ odd, $mate(w)$ even, $p(w)=v$ and $p(mate(w))=w$
  - » if $w$ is even, stop; path from root of tree containing $v$ to root of tree containing $w$ forms an augmenting path

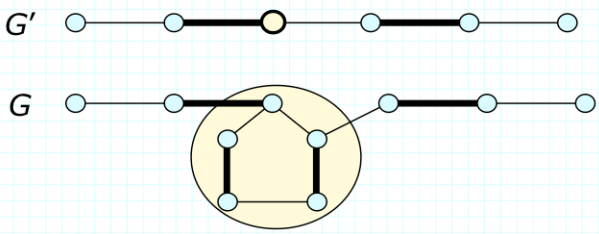
# Blossoms

- In general graphs, path search can fail if an edge is selected which joins two even vertices in same tree
  - » if $\{v,w\}$ is such an edge, the cycle induced by the tree path connecting $v$ and $w$ together with $\{v,w\}$ is called a *blossom*.
  - » handle this case by *shrinking* the blossom, that is, treating the set of vertices if the blossom like a single vertex
- A blossom is an odd-length cycle with one vertex incident to two non-matching edges
  - » this vertex is called the *base* of the blossom
  - » the base is an ancestor of all other vertices in the cycle
  - » base may have a matching edge to its parent or may be unmatched
  - » in either case, the base of the blossom is an even vertex in path search



- *Theorem 9.3*. If *G'* is formed from *G* by shrinking a blossom *b*, then *G'* contains an augmenting path if and only if *G* does

  For now, we prove only that if *G'* contains an augmenting path *P*, then *G* also contains an augmenting path
  - » if *b* is not on *P*, then *P* is an augmenting path in *G*
  - » if *b* is on *P*, we can construct an augmenting path in *G* by expanding *b* and reconnecting the now separated parts of *P* by going around the blossom in the appropriate direction

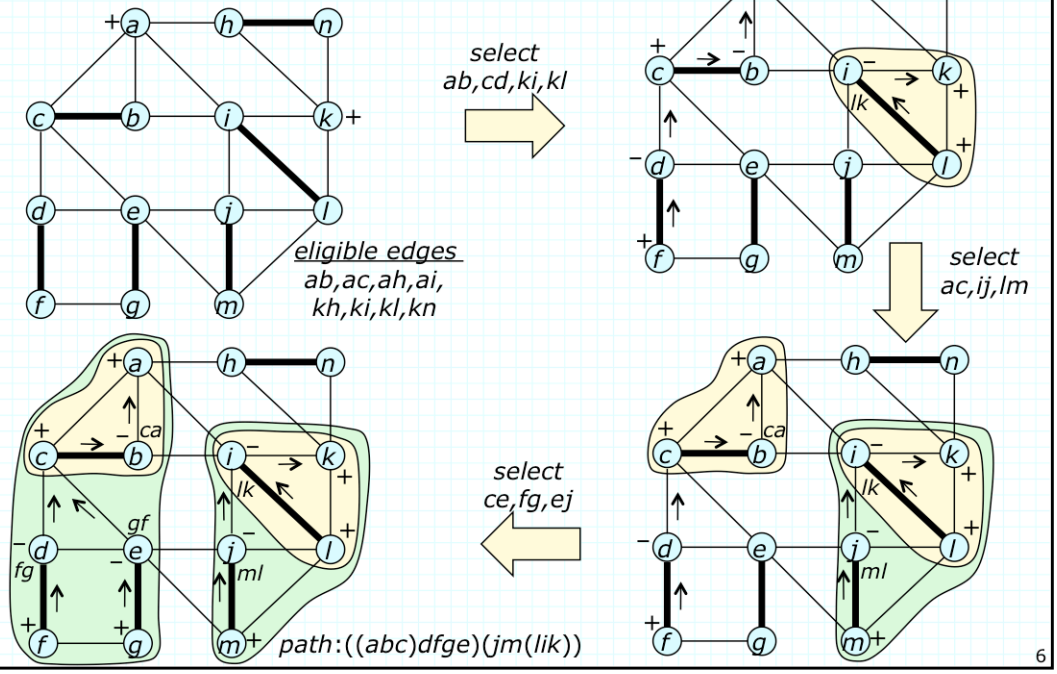



The proof of the other direction follows from Theorem 9.4, below

# Edmond's Algorithm

- Replace each edge $\{x,y\}$ with pair $[x,y]$, $[y,x]$
- Basic step: choose an unexamined edge $[v,w]$ with $v$ even and examine it
  - » if $w$ is unreached and matched, make $w$ odd, $mate(w)$ even, $p(w)=v$ and $p(mate(w))=w$
  - » if $w$ is even and $v$ and $w$ are in same tree, edge $\{v,w\}$ together with tree path joining $v$ and $w$ forms a blossom
    - let $u$ be nearest common ancestor of $v$ and $w$ (in the tree) and condense all descendants of $u$ that are ancestors of either $v$ or $w$ into a shrunken blossom $b$, and let $p(b) = p(u)$; make $b$ even
  - » if $w$ even and $v$,$w$ are in different trees, form augmenting path from root of $v$'s tree to root of $w$'s tree (expanding blossoms)
- A vertex is called *shrunken* if at some point in the algorithm if it has been included in a blossom
  - » both original vertices and blossoms may become shrunken

# Example


select
ab,cd,ki,kl
eligible edges
ab,ac,ah,ai,
kh,ki,kl,kn
select
ac,ij,lm
select
ce,fg,ej
ca
lk
gf
fg
ml
path:((abc)dfge)(jm(lik))

- *Theorem 9.4*. The blossom shrinking algorithm finds an augmenting path ⇔ there is one in the original graph

  *Proof*. (⇒) If the algorithm succeeds then we can find augmenting path in original graph by expanding blossoms in reverse order

  (⇐) Note the following points

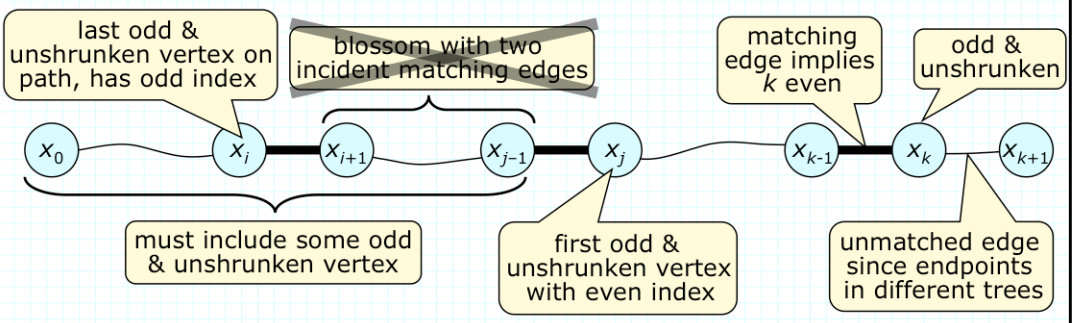
  (1) if $v$ is reached and matched, $mate(v)$ is also reached and in same tree

  (2) if $v$ is a free vertex contained in a blossom $b$, then $b$ is free

  (3) if algorithm fails then any two adjacent vertices that are either shrunken or even are contained in a common blossom when the algorithm halts

  Suppose algorithm fails, but there is an augmenting path $p=[x_0,\ldots,x_{2r+1}]$ and consider the situation just after algorithm halts

  - note that $x_0$ and $x_{2r+1}$ are in different trees; let $k$ be any integer such that $x_{k+1}$ is in different tree from $x_0$ and $x_{k+1}$ is in different tree from $x_k$
  - by point (3) above, either $x_k$ or $x_{k+1}$ is odd and unshrunken
  - assume (w.l.o.g.) that $x_k$ is odd and unshrunken; $mate(x_k)=x_{k-1}$, so $k$ must be even; let $j$ be smallest integer such that $j$ is even and $x_j$ is odd and unshrunken




- there must be an odd, unshrunken vertex among $x_0,\ldots,x_{j-1}$ since if not (by point 3), $x_0,\ldots,x_{j-1}$ would all be contained in a common blossom
  - since this blossom contains a free vertex $x_0$, it must (by point 2) be a free blossom; this implies that matching edge to $x_j$ must also be part of the blossom, contradicting the assumption that $x_j$ is unshrunken
- let $i$ be the largest integer such that $i<j$ and $x_i$ is odd and unshrunken; then $i$ is odd which means $mate(x_i) = x_{i+1}$
- this in turn implies that $x_{i+1},\ldots,x_{j-1}$ are contained in a single blossom that is incident to two matched edges, which is impossible ■

# Implementing Edmond's Algorithm

- Replace each edge $\{u,v\}$ with edges $[u,v]$, $[v,u]$
- Current shrunken graph is represented implicitly using partition data structure
  - »each set corresponds to a vertex in current shrunken graph and contains the original vertices contained in current vertex; define

$$origin(v) = \begin{cases} v & \text{if } v \text{ is an original vertex} \\ origin(base(v)) & \text{if } v \text{ is a blossom} \end{cases}$$

  - »*origin*($v$) initialized to $v$ for all $v$; whenever blossom forms, the canonical vertex $v$ of set representing the blossom has *origin*($v$) set to the origin of the blossom
  - »so, *origin*(*find*($v$)) always equals origin of largest blossom containing $v$ and represents vertex containing $v$ in current graph
  - »if $v'$=*origin*(*find*($v$)) for all vertices $v$ then edge $[x',y']$ is the edge in current graph corresponding to original edge $[x,y]$

- For each odd vertex in blossom, define a *bridge*
  - » if examination of original edge [$v$,$w$] causes an odd vertex $x$ to be shrunk into a blossom, *bridge*($x$)=[$v$,$w$] if $x$ is an ancestor of $v'$ before condensation and *bridge*($x$)=[$w$,$v$], otherwise
- Algorithm: for each $v$ let *origin*($v$)=$v$, and make $v$ even if $v$ is free, unreached otherwise; repeat the following
  - » choose unexamined original edge [$v$,$w$] with $v'$ even and examine it
  - » if $w'$ is unreached and matched, make $w'$ odd, *mate*($w'$) even, $p(w')=v$ and $p(mate(w'))=w'$
  - » if $w'$ is even and $v'$ and $w'$ are in different trees, stop and recover augmenting path
  - » if $w'$ is even and $v'$ and $w'$ are in same tree, let $u$ be *nca*($v'$,$w'$)
    - for every $x$ that is a descendant of $u$ and an ancestor of $v'$, perform *link*(*find*($u$),*find*($x$)) and if $x$ is odd let *bridge*($x$)=[$v$,$w$]
    - for every $x$ that is a descendant of $u$ and an ancestor of $w'$, perform *link*(*find*($u$),*find*($x$)) and if $x$ is odd let *bridge*($x$)=[$w$,$v$]
    - let *origin*(*find*($u$))=$u$

- Edges eligible for selection are kept in a list
- To recover augmenting paths, use bridge information and reversible list data structure
  - » suppose $[v,w]$ is examined by algorithm and $v'$ and $w'$ are in different trees with roots $x$ and $y$
  - » the augmenting path from $x$ to $y$ is $reverse(path(v,x))$ & $path(w,y)$ where *reverse* reverses a list and *path* is defined by

$$path(a,b) = \begin{cases} [a] & \text{if } a = b \\ [a, p(a)] \,\&\, path(p^2(a), b) & \text{if } a \neq b \text{ and } a \text{ is even} \\ [a] \,\&\, reverse(path(c, mate(a))) & \text{if } a \neq b \text{ and } a \text{ is odd} \\ \quad \&\, path(d, b) & \quad \text{and } [c,d] = bridge(a) \end{cases}$$

    - where the first argument of *path*() is a descendant of the second
  - » this allows the augmenting path to be recovered in $O(n)$ time

# Detailed Implementation

In following, *G* assumed to be in digraph form

```
function path augpath(digraph G, matching M)
    vertex u,v,w,x,y;
    mapping state:vertex→{unreached,even,odd};
    mapping matched:vertex→{true,false};
    mapping mate, origin, p:vertex→vertex;
    mapping bridge:vertex→edge;
    partition(V);
    list Q;
    for u ∈ V ⇒ state(u) := even; matched(u) := false; rof;
    for {u,v} ∈ M ⇒
        state(u), state(v) := unreached;
        matched(u), matched(v) := true;
        mate(u) := v; mate(v) := u;
    rof;
```

```
Q := [];
for u∈[1,n] ⇒
    for [u,v]∈out(u) ⇒
        if state(u)=even ⇒ Q := Q & [u,v] fi
    rof;
rof;
do Q ≠ [] ⇒
    [v,w] := Q(1); Q := Q[2..];
    v' := origin(find(v)); w' := origin(find(w));
    if v' ≠ w' and state(w') = unreached ⇒
        x := mate(w');
        state(w') := odd; p(w') := v'; state(x) := even; p(x) := w';
        for [x,y] ∈ out(x): [x,y] ∉ Q ⇒ Q := Q & [x,y] rof;
     | v' ≠ w' and state(w') = even and not sametree(v',w') ⇒
        x := v'; do p(x) ≠ null ⇒ x := origin(find(p(x))); od;
        y := w'; do p(y) ≠ null ⇒ y := origin(find(p(y))); od;
        return reverse(path(v,x)) & path(w,y);
```

```
        | v' ≠ w' and state(w') = even and sametree(v',w') ⇒
            u := nca(v',w'); x := v';
            do x ≠ u ⇒
                origin(link(find(u),find(x))) := u;
                if state(x) = odd ⇒
                    bridge(x) := [v,w];
                    for [x,y]∈out(x): [x,y] ∉ Q ⇒ Q := Q & [x,y] rof;
                fi;
                x := origin(find(p(x)));
            od;
            x := w';
            do x ≠ u ⇒
                origin(link(find(u),find(x))) := u;
                if state(x) = odd ⇒
                    bridge(x) := [w,v];
                    for [x,y]∈out(x): [x,y] ∉ Q ⇒ Q := Q & [x,y] rof;
                fi;
                x := origin(find(p(x)));
            od;
        fi;
    od;
end;
```

# Analysis of Edmond's Algorithm

- Main loop is executed at most $2m$ times since each (directed) edge is put on the queue at most once
  - » similarly, the for-loops that put edges on the queue contribute a total of $O(m)$ to the running time
- Function call *sametree*($v'$,$w'$) returns true if $v'$ and $w'$ are in the same tree, otherwise it returns false
- Function call *nca*($v'$,$w'$) returns nearest common ancestor of $v'$ and $w'$ in current shrunken graph
- Implement *sametree* and *nca* by moving up tree paths (in current shrunken graph) from $v'$ and $w'$ in parallel
  - » stop either on reaching different free vertices or on reaching an ancestor of $v'$ that was visited previously in search from $w'$ or vice versa

- Can perform path searches for *sametree* and *nca* in time $O$(# of vertices visited)
  - » if $v'$ and $w'$ are in different trees it takes $O(n)$ time to complete path searches (this case is used only once)
  - » if $v'$ and $w'$ are in same tree then # of vertices visited is ≤twice # on cycle defining blossom that is about to be condensed
  - » there are $\leq n/2$ blossoms, since formation of a blossom reduces the number of vertices by at least two
  - » hence, number of vertices on all blossom cycles is $\leq 3n/2$ and total time spent executing *sametree* and *nca* is $O(n)$
- By same argument, the total time spent shrinking blossoms is $O(n)$, excluding the partition operations
- Hence, number of partition operations is $O(m)$ and augmenting path can be found in $O(m\alpha(m,n))$ time
- So, can find max size matching in $O(mn\alpha(m,n))$ time

# Exercises

1. Suppose that during the execution of the Edmond's algorithm, $u$ is a vertex on some blossom with base $b$ and edge $[x,y]$ is the bridge of $u$. Assuming $x \neq b \neq y$, is it possible that $x$ is an ancestor of $u$? Why or why not? Is it possible for $u$ to be an ancestor of $x$? Why or why not?

   *X cannot be an ancestor of u. The edge [x,y] was the one that formed the blossom. At the time, all the edges on the blossom were either ancestors of x or ancestors of y. So u might be an ancestor of x, but x cannot be an ancestor of u.*

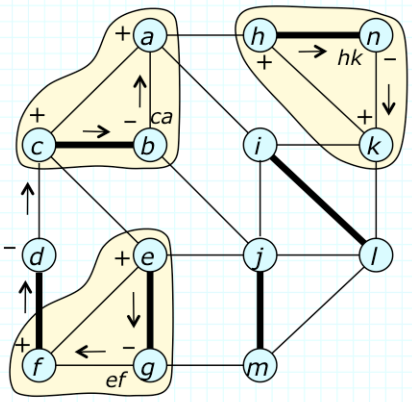
Let $z$ be the nearest common ancestor of $x$ and $y$. Is $z$ even or odd? Can it be matched? Can it be free?

*The nearest common ancestor of x and y is the base of the blossom, so z must be even. Consequently, it may be either matched or free.*

Let $v$ be a vertex on the blossom that is not $z$. Is $v$ incident to a matching edge? Why or why not?

*V must be incident to a matching edge. In general, any vertex that has been reached but is not a tree root must be incident to matching edge, and only the base of a blossom can be a tree root. Since v is in the blossom, it has been reached and since it's not the base of the blossom, it must be matched.*

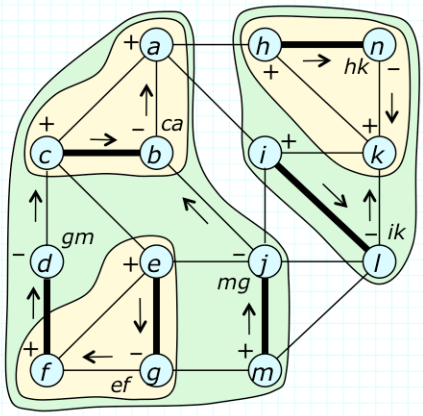
2. The figure below shows an intermediate state in the execution of Edmond's algorithm. Which edges are eligible to be processed next?




*Edges ah, ai, bj, ce, ej, gm, ki and kl are all elgible.*

Show the state after edges *kl, ki, bj* and *gm* have been selected.

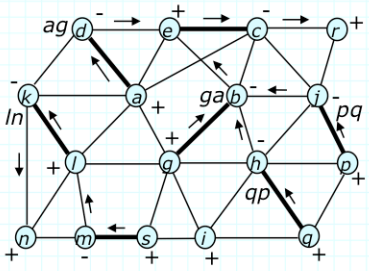
Show the state after edges *kl, ik, bj* and *gm* have been selected.



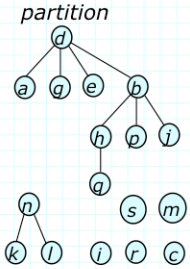

If the edge *jl* is selected next, what augmenting path is returned?
*abcdfegmjlik*

3. The figure below show an intermediate state in the execution of the Edmond's algorithm. The edges shown next to certain vertices correspond to their *bridge* values. The partition data structure is shown at right and the queue of edges to be processed is at the top.

*queue*: *ae, ih, la, kd, ln*




*partition*



In the diagram at right, draw a closed curve around the sets of vertices that have been condensed into a single vertex in the current shrunken graph. If any of these sets contain subsets, corresponding to smaller blossoms, circle the vertices in those smaller blossoms, as well.

What happens when the next edge in the queue is processed by the algorithm? Explain.

*Nothing. The edge ae joins two vertices that are contained in the same vertex in the current shrunken graph, so the algorithm just discards this edge.*

When the next edge, [*i*,*h*] is processed, an augmenting path is found. What is that augmenting path?

*i,h,q,p,j,b,g,a,d,e,c,r*