Algorithm Theory

Chapter 6
Graph Algorithms

Part IX:
Maximum Matching in General Graphs

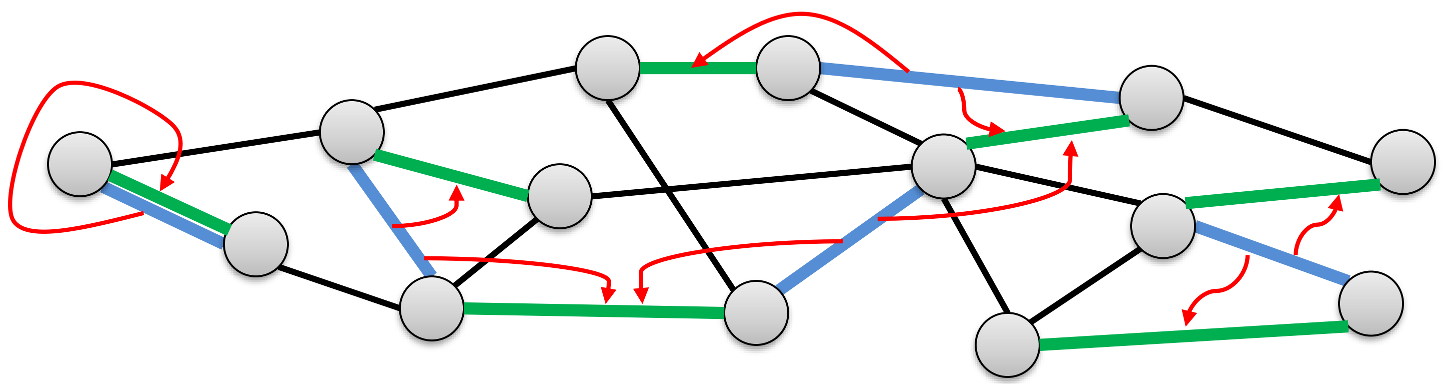
Fabian Kuhn

What About General Graphs

- Can we efficiently compute a maximum matching if G is not bipartite?

- How good is a maximal matching?
  - A matching that cannot be extended…

- **Compare the size of a maximal and a maximum matching**



- Each maximal matching edge is adjacent to ≤ 2 maximum matching edges

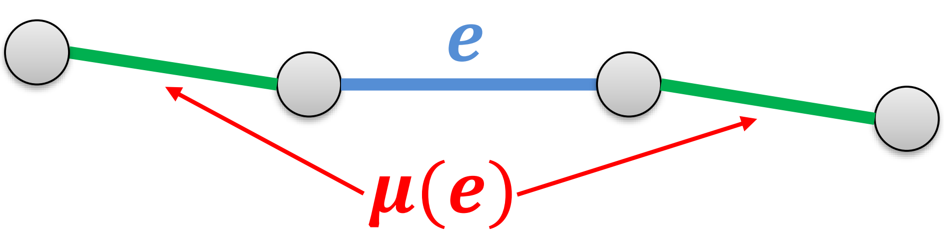
Maximal vs. Maximum Matching

Theorem: For any maximal matching M and any maximum matching M^*, it holds that

$$|M| \geq \frac{|M^*|}{2}.$$

Proof:

- For each edge $e \in M$, let $\mu(e) \subseteq M^*$ be the adjacent edges in M^*



$$\forall e \in M : |\mu(e)| \leq 2$$

- Every edge in M^* is adjacent to some edge of M:

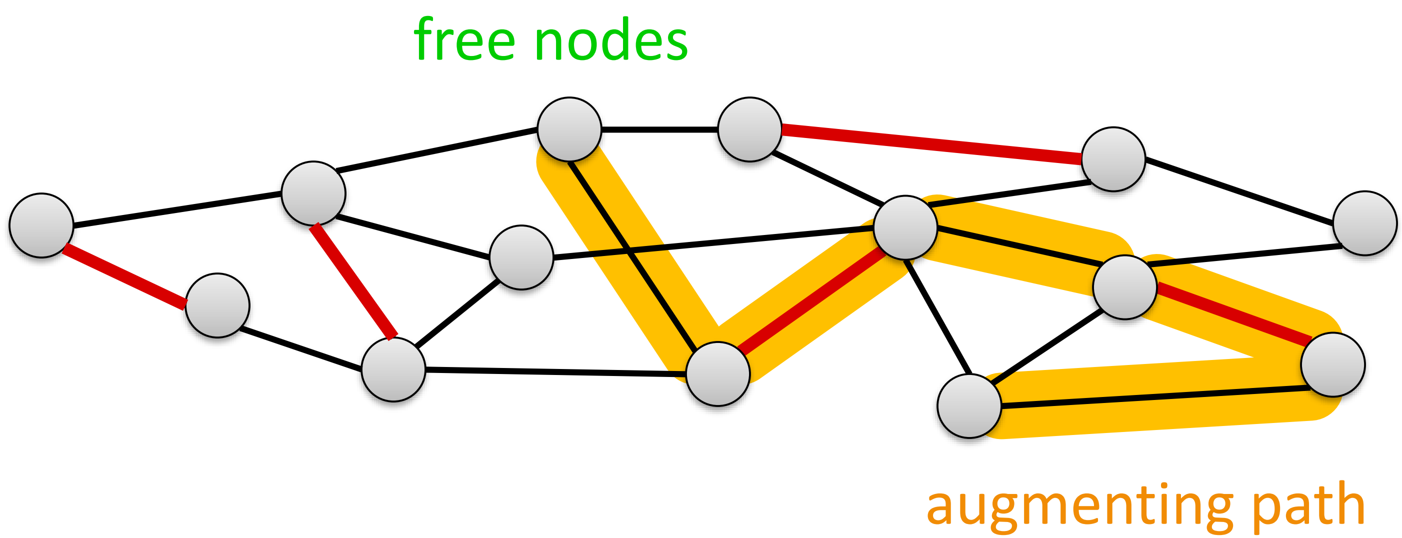
$$|M^*| = \left|\bigcup_{e \in M} \mu(e)\right| \leq \sum_{e \in M} |\mu(e)| \leq 2|M|.$$

Augmenting Paths

Consider a matching M of a graph $G = (V, E)$:

- A node $v \in V$ is called **free** iff it is not matched

Augmenting Path: A (odd-length) path that starts and ends at a free node and visits edges in $E \setminus M$ and edges in M alternatingly.



- Matching M can be improved using an augmenting path by switching the role of each edge along the path

Existence of Augmenting Paths

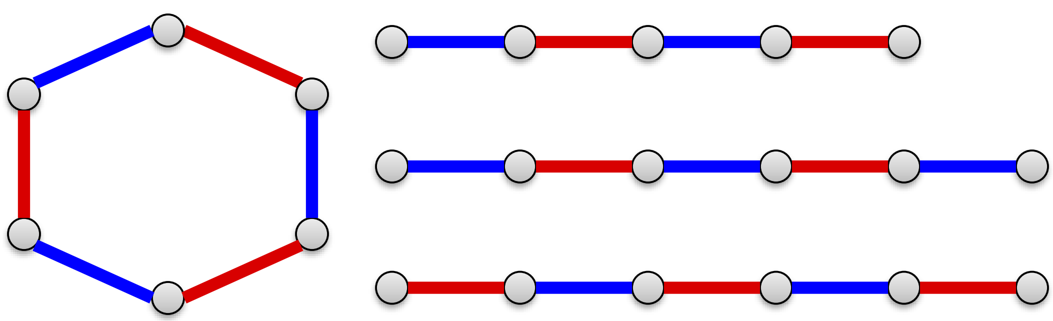
Theorem: A matching M of $G = (V, E)$ is maximum if and only if there is no augmenting path.

Proof:

- Consider non-max. matching M and max. matching M^* and define

$$F := M \setminus M^*, \qquad F^* := M^* \setminus M$$

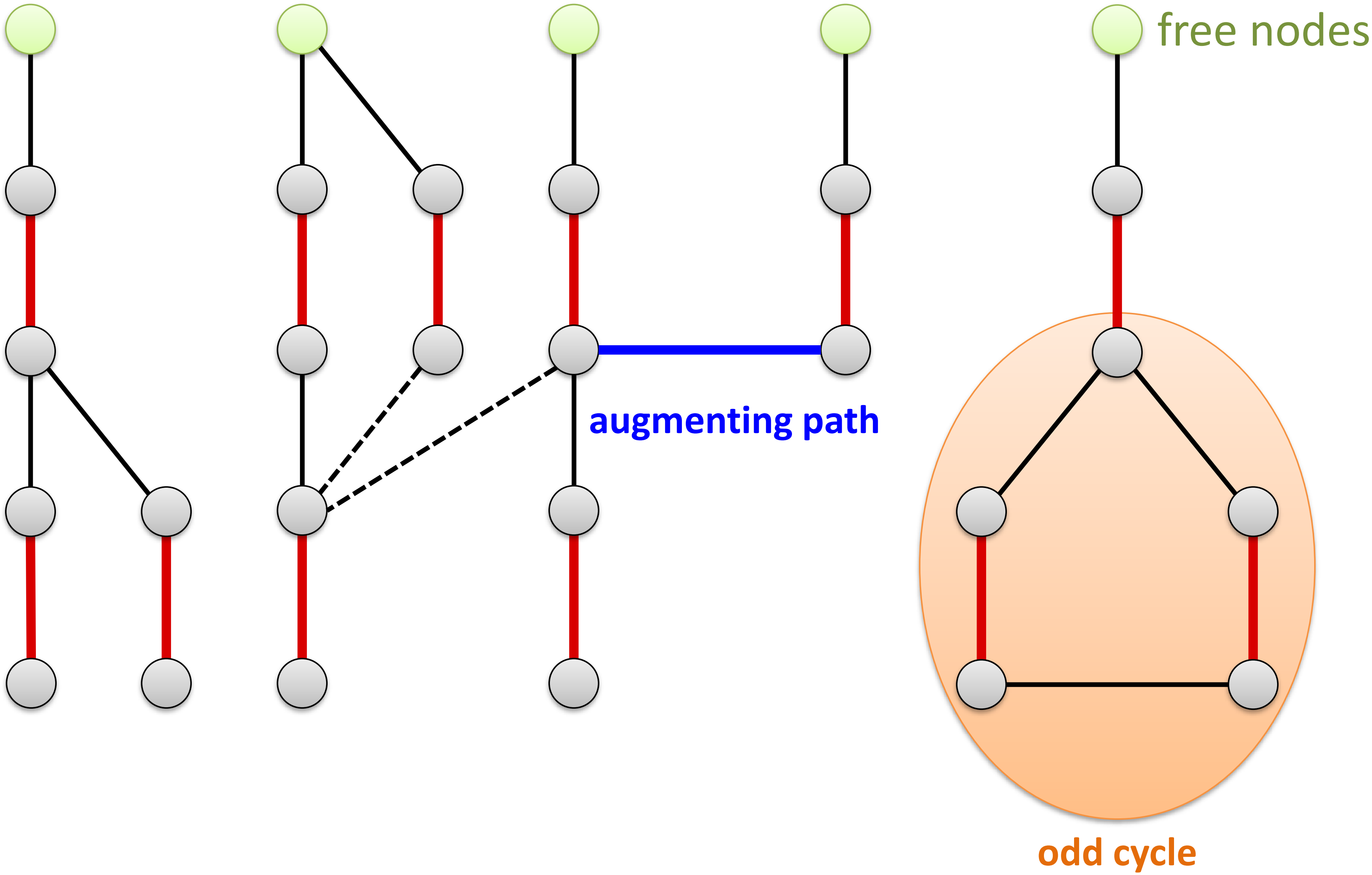
- Note that $F \cap F^* = \emptyset$ and $|F| < |F^*|$
- Each node $v \in V$ is incident to at most one edge in both F and F^*
- $F \cup F^*$ induces even cycles and paths



augmenting path for M

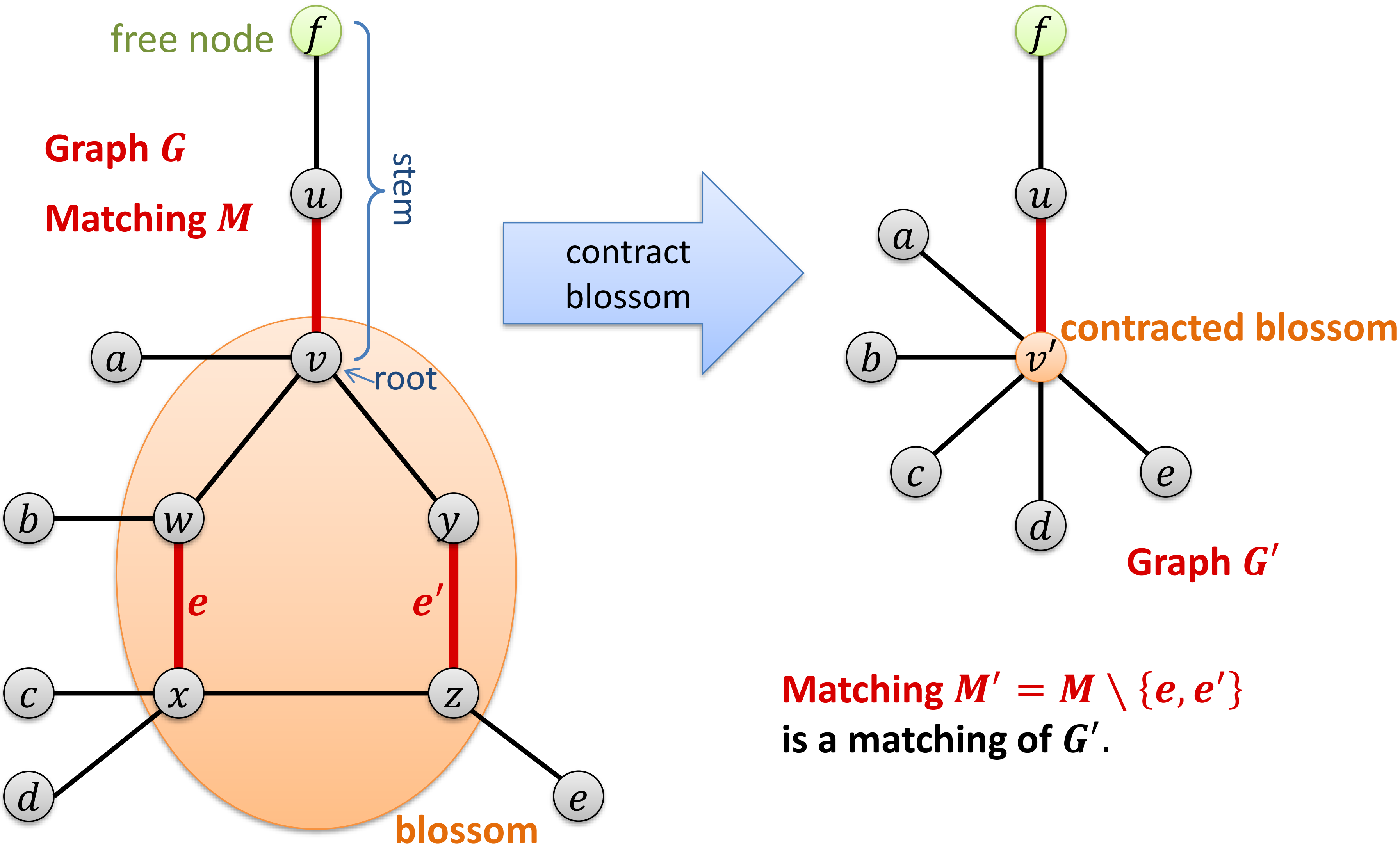
augmenting path for M^* (cannot exist)

Finding Augmenting Paths



Blossoms

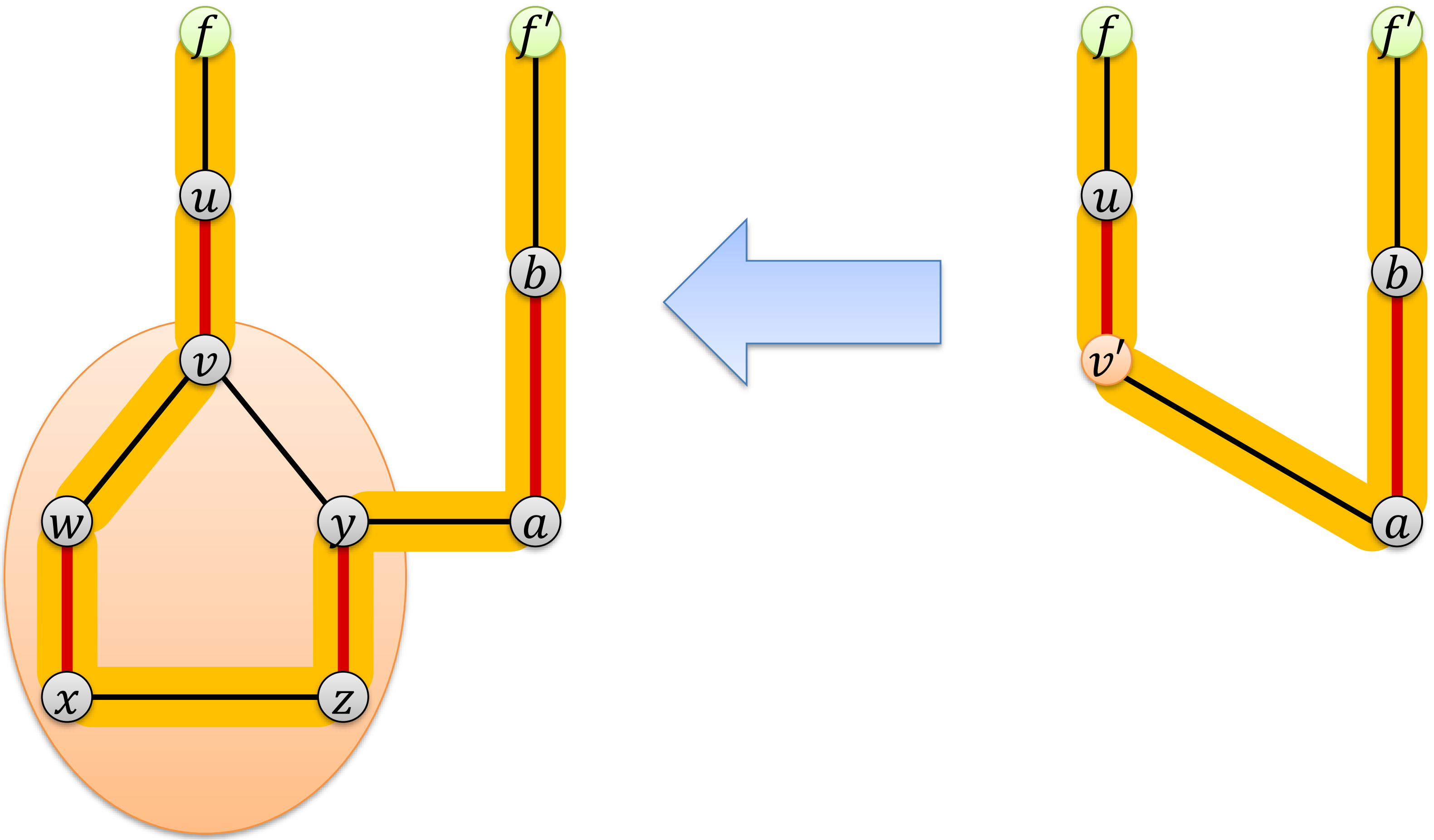
- If we find an odd cycle...



Matching $M' = M \setminus \{e, e'\}$
is a matching of G'.

Contracting Blossoms

Lemma: Graph G has an augmenting path w.r.t. matching M iff G' has an augmenting path w.r.t. matching M'.

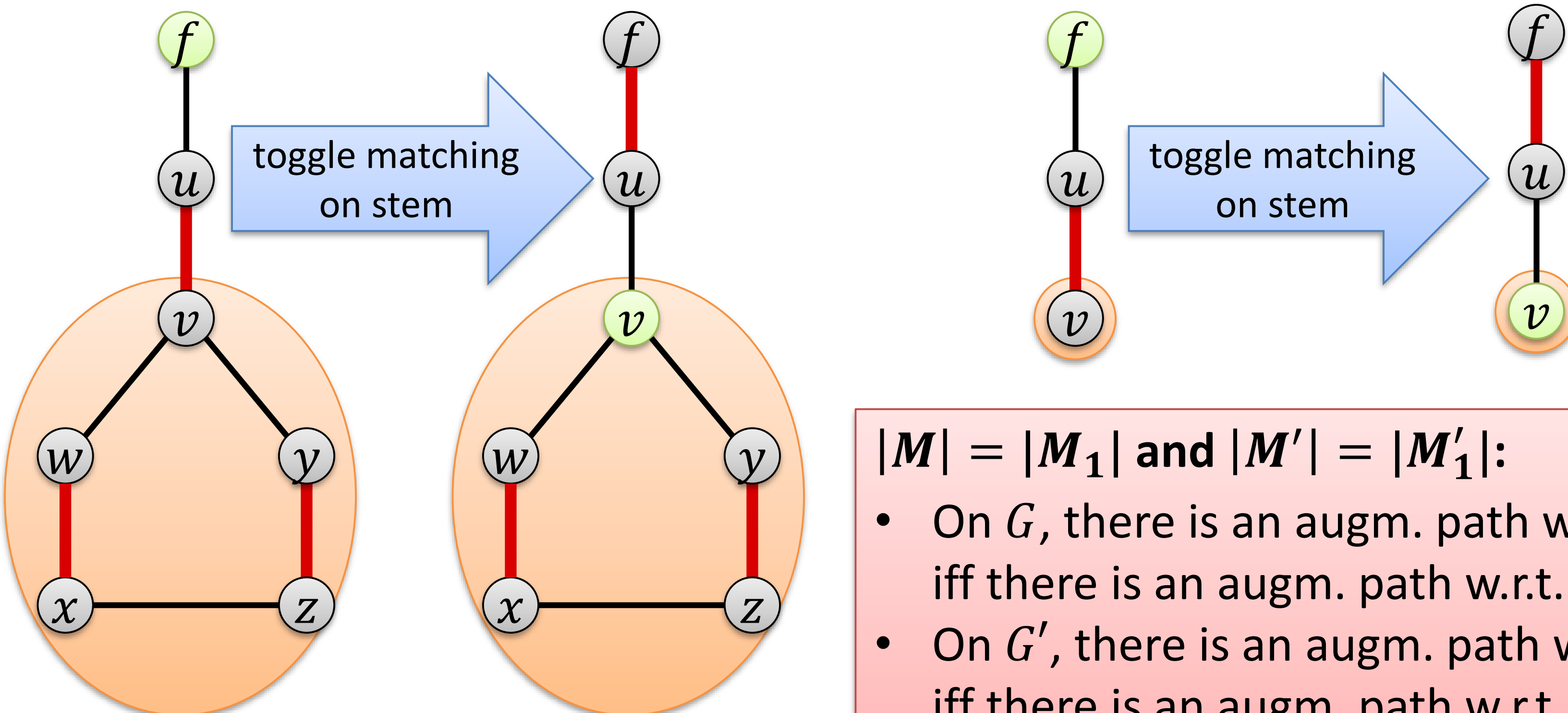


Also: The matching M can be computed efficiently from M'.

Contracting Blossoms

Lemma: Graph G has an augmenting path w.r.t. matching M iff G' has an augmenting path w.r.t. matching M'.

- Obtain matchings M_1 / $M_1{}'$ on G / G' by toggling matching on stem



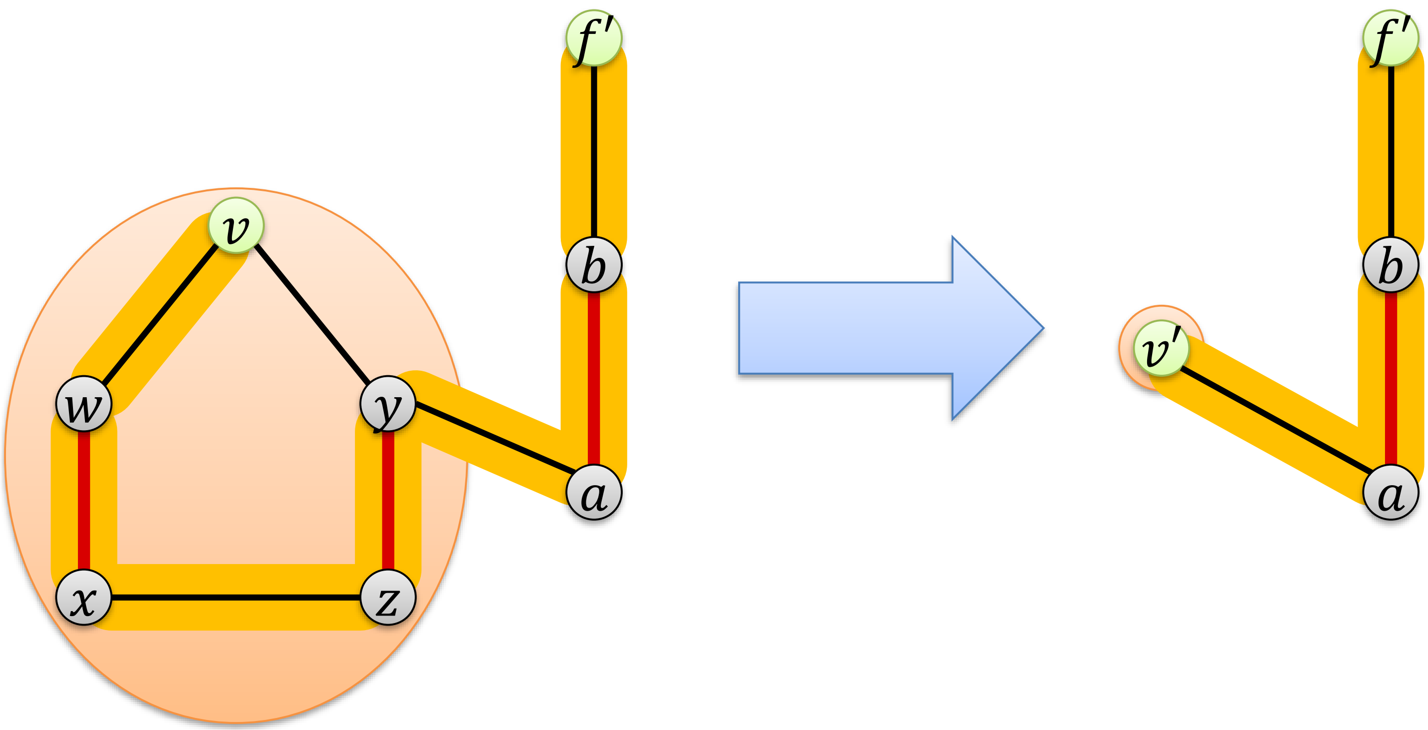
$|M| = |M_1|$ and $|M'| = |M_1'|$:

- On G, there is an augm. path w.r.t. M iff there is an augm. path w.r.t. M_1
- On G', there is an augm. path w.r.t. M' iff there is an augm. path w.r.t. M_1'
- We can w.l.o.g. assume that the root of the stem is a free node.

Contracting Blossoms

Lemma: Graph G has an augmenting path w.r.t. matching M iff G' has an augmenting path w.r.t. matching M'.

- If the root of the blossom is free, any augmenting path w.r.t. M_1 that contains nodes of the blossom can be turned into an augmenting path that ends at the root of the blossom and consists of a part inside the blossom and a part outside it.



Edmond’s Blossom Algorithm

Algorithm Sketch:

1. Build a tree for each free node
2. Starting from an explored node u at even distance from a free node f in the tree of f, explore some unexplored edge $\{u, v\}$:

   1. If v is an unexplored node, v is matched to some neighbor w:
      add w to the tree (w is now explored)

   2. If v is explored and in the same tree:
      at odd distance from root → ignore and move on
      at even distance from root → **blossom found**

   3. If v is explored and in another tree
      at odd distance from root → ignore and move on
      at even distance from root → **augmenting path found**

Running Time

Finding a Blossom: Restart search on smaller graph

Finding an Augmenting Path: Improve matching

Theorem: The algorithm can be implemented in time $O(mn^2)$.

- DFS to find augmenting path or blossom: $O(m)$
- Needs to be repeated each time, when a blossom is found
  - Contraction of blossom reduces number of nodes by at least 2
  - Number of repetitions is $\leq n/2$
- In time $O(mn)$, we can find an augmenting path, if there is one and improve a given non-maximum matching
- Maximum matching has size $\leq n/2$

Matching Algorithms

We have seen:

- $O(mn)$ time alg. to compute a max. matching in *bipartite graphs*
- $O(mn^2)$ time alg. to compute a max. matching in *general graphs*

Better algorithms:

- Best known running time (bipartite and general gr.): $O(m\sqrt{n})$

Weighted matching:

- Edges have weight, find a matching of **maximum total weight**
- The problem can also be solved optimally in polynomial time, both in bipartite graphs and in general graphs
  - Algorithms use maximum matching in unweighted graphs as subroutine